|| JAI SRI GURUDEV ||

Sri Adichunchanagiri Shikshana Trust (R.)

BGS INSTITUTE OF TECHNOLOGY

(Affiliated to VTU, Belagaum; Approved by AICTE, New Delhi and Recognized by Govt. of Karnataka)

BG Nagara - 571 448 (Bellur Cross), Nagamangala Taluk, Mandya District

Certificate

This is to certify that Mr/~~Ms~~..... Jeevan. R .....

*USN:4BW..*17CS029..... *has satisfactorily completed the course of experiments in* Machine Learning..... *Laboratory (Course Code...*17CSL76.....*) prescribed by the Visvesvaraya Technological University, Belagavi for* .....VII..... *Semester, BE.*Computer Science *Engineering, of this College in the year* 2020 - 2021

Record Marks : 29 Test Marks : 09

IA Marks : 38

Date : 16/01/2021

H.J. Prasad

Staff Incharge

B. K. Rag 16/1/21

Head of the Department

Name of the Student : Jeevan . R Class : A Sem. VI

Expt. No.	Date	Title of Experiment	Page No.	Marks Obtained					Sign. of the staff
				a	b	c	d	Total	
1	5/11/20	Find-S Algorithm	01	04	10	05	10	29	
2.	7/11/20	Candidate - Elimination algorithm	02	04	10	05	10	29	
3.	12/11/20	ID3 algorithm	04	04	10	05	10	29	
4.	14/11/20	Back propagation algorithm	10	04	10	05	10	29	
5.	19/11/20	Naive Bayesian classifier	13	04	10	05	10	29	
6.	21/11/20	Naive Bayesian classifier using Built in Java/API	18	04	10	05	10	29	
7.	28/11/20	Bayesian network covidary medical DOA.	21	04	10	05	10	29	
8.	28/11/20	K-mean algorithm	25	04	10	05	10	29	
9.	31/12/20	K-nearest Neighbor algo	27	04	10	05	10	29	

INDEX

Name of the Student: Jeevan.R Class: A Sem.

Expt. No.	Date	Title of Experiment	Page No.	Marks Obtained a	b	c	d	Total	Sig the
10	5/12/20	Locally weighted Regression Algorithm	30	04	10	05	10	29	

Input : find.csv

Example	Sky	Airtemp	Humidity	Wind	Water	Forecast	Enjoy
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Output :

['Sunny' 'Warm' '?' 'Strong' '?' '?']

Name of Experiment: Find-S Algorithm
Date: 05/11/20
Experiment No: 1
Page No: 01

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 numpy as np
import pandas as pd
data = pd.DataFrame(data = pd.read_csv(finde
concepts = np.array(data.iloc[:, 0:-1])
target = np.array(data.iloc[:, -1])
def learn(concepts, target):
    specific_h = concepts[0].copy()
    for i, h in enumerate(concepts):
        if target[i] == 'Yes':
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
    return specific_h
```

Input : find.csv

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	warm	Normal	Strong	warm	Same	Yes
2	Sunny	warm	High	Strong	warm	Same	Yes
3	Rainy	cold	High	Strong	warm	Change	No
4	Sunny	warm	High	Strong	cool	Change	Yes

Output :

Final G :

[['Sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

Name of Experiment: Candidate-Elimination Algo Date: 07/11/20

Experiment No. 2. Page No. 02

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

```
import numpy as np
import pandas as pd
data = pd.DataFrame(data = pd.read_csv
                    ('find.csv'))
concepts = np.array(data.iloc[:,0:,-1])
target = np.array(data.iloc[:,-1])
def learn(concepts, target):
    specific_h = concepts[0].copy()
    general_h = [["?' for i in range(len(specific_h))
                 for i in range(len(specific_h))]
    for i,h in enumerate(concepts):
    #checking if the hypothesis has a positive
      target
        if target[i] == "Yes":
            for x in range(len(specific_h)):
    #change values in S & G only if values
      change
```

BGSIT

```
            if h[x] != specific_h[x]:
                specific_h[x] = '?'
                general_h[x] = '?'
    #checking if the hypothesis has a negative target
    if target[i] == 'No':
        for x in range(len(specific_h)):
    #for negative hypothesis change values in G
            if h[x] != specific_h[x]:
                general_h[x][x] = specific_h[x]
            else:
                general_h[x][x] = '?'
    #find indices where we have empty rows, meaning
     those that are unchanged
    indices = [i for i, val in enumerate(general_h)
                if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h.
```

Seen

10/10

Input: playTennis.csv

Outlook	Temp	Humidity	Wind	Decision
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:

{'ROOT': {'Outlook': {'overcast': 'Yes',
'Rain': {'Wind': {'strong': 'NO', 'weak': 'Yes'}},
'Sunny': {'Humidity': {'High': 'NO', 'Normal': 'Yes'}}}}}

Name of Experiment: ID3 Algorithm
Date: 12/11/20
Experiment No: 3
Page No: 04

2) 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 pandas as pd
from pprint import pprint
data = pd.read_csv("playTennis.csv")
data_size = len(data)
treenode = []
tree = {"ROOT": data}

def total_entropy(data, col):
    mydict = {}
    for elem in data[col]:
        if elem in mydict.keys():
            mydict[elem] += 1
        else:
            mydict[elem] = 1
    total = sum(mydict.values())
    E = 0
```

BGSIT

```
    for key in mydict.keys():
        E += entropy(mydict[key], total)
    return E

def entropy(num, denom):
    return -(num/denom) * np.log2(num/denom)

def get_sorted_data(data, column):
    sort = {}
    for column_name in get_attributes(data, column):
        sort[column_name] = data.loc[data
            [column] == column_name]
    return sort

def get_attributes(data, column):
    return data[column].unique().tolist()

def InfoGain(total_entropy, sorted_data,
             entropy_by_attribute):
    length = data_size
    total = 0
    for col, df in sorted_data.items():
        total += (len(df)/length) * entropy_by_
                 attribute[col]
```

```
    return total_entropy - total

def get_entropy_by_attribut(sortid_data):
    entropies = {}
    for key, df in sortid_data.items():
        entropies[key] = total_entropy(df, 'Decision')
    return entropies

def drop_node(data, column):
    return data.drop(column, axis=1)

def id3(tree):
    for branch, data in tree.items():
    #make sure it's a DataFrame
    if not isinstance(data, pd.DataFrame):
        continue
#Fetch column names so you can use them to
# iterate later
    columns = data.columns
# Calculate the Entropy for the entire dataset
    total_entropy_for_data = total_entropy(data.
                                values, -1)
# If only one column is left, it means we're
  done.
```

Name of Experiment _____ Date

Experiment No. _____ Page No. 01

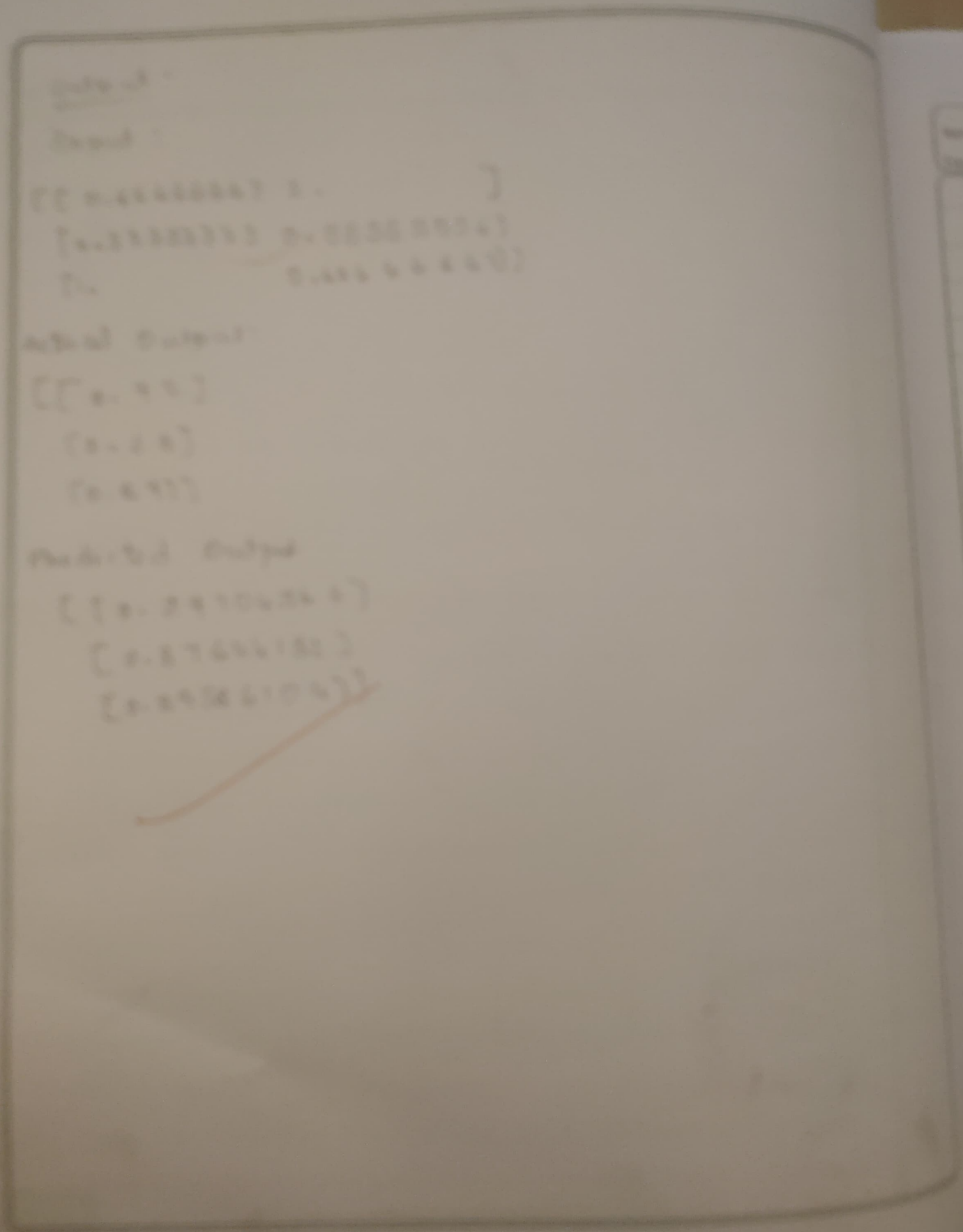
```
        if len(columns) == 1:
            break
    # Keep track of information gain to choose the attribute
      with maximum info gain.
        info_gain_list = []
    # Now iterate over each column to choose the
      attribute with maximum info gain
    #calculate information gain w.r.t o/p column
        for i in range (0, len (data.columns) -1):
            # sort the rows wrt o/p
            sorted_rows = get_sorted_data (data, cloumn[i])
    # Calculate the entropy wrt to each attribute based
      on sorted columns.
            entropy_by_attribute = get_entropy_by_
                                   attribute (sorted_rows)
    # get the info gain
            info_gain_list.append (info_gain)
    # save it
            info_gain = InfoGain (total_entropy_for_
                        data, sorted_rows, entropy_by_attribute)
    # Find index of max info gain
        node = info_gain_list.index (max( info_gain_list))
    # sort the data into branches based on the new
      node
```

BGSIT

```
        branches = get_sorted_data(data, columns[node])
#if we've reached the end of iteration, just
 assign the value,
# else drop the sorted column
        for attr, df in branches.items():
            if (total_entropy(df, columns[-1]) == 0):
                branches[attr] = df.iloc[0, -1]
            else:
                branches[attr] = df.drop(columns[node],
                                         axis = 1)
#Keep track of node already done
        treenodes.append(columns[node])
#add the new branches to the tree
        child = {columns[node] : {}}
        tree[branch] = child
        tree[branch][columns[node]] = branches
#ID3
        id3(tree[branch][columns[node]])

x = id3(tree)
pprint(tree, depth = 5)
```

Seen

10/10



Name of Experiment: Back propagation method
Date: 12/11/20
Experiment No: 4
Page No: 10

4) Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate dataset

```
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 sigmod(x):
    return 1/(1+np.exp(-x))
#Derivative of sigmoid function
def derivative_sigmoid(x):
    return x*(1-x)
#variable initialization
epoch = 7000 #setting training iterations
lr = 0.1 #setting learning rate
inputlayer_neurons = 2 #no of features in dataset
hiddenlayer_neurons = 3 # no of hidden layer neurons
output_neurons = 1 #no of neurons of output layer
```

BGSIT

Name of Experiment _____ Date

Experiment No. _____ Page No. 11.

```
#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))

#for i in range(epoch):
    #forward Propagation
    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 = derivative_sigmoid(output)
    d_output = EO* outgrad
```

BGSIT

```
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
wh += X.T.dot(d_hiddenlayer) *lr

print("Input : \n" + str(X))
print("Actual Output : \n" + str(y))
print("Predicted Output : \n", output)
```

Seen 10/10

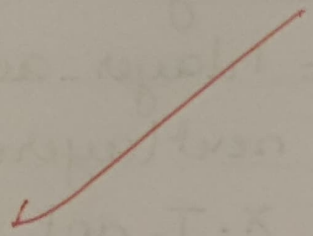
BGSIT

Input : 5 pima-indians-diabetes.data.csv

output :

split 768 rows into train = 514 and test = 254 rows

Accuracy : 0.39370078740157478



Name of Experiment: Naive Bayesian classificatio

Experiment No. 5

Date: 19 11 20

Page No. 13

5) Write a program to implement the naive 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:
```

Name of Experiment _____ Date

Experiment No. _____ Page No. 14

```
def separateByClass(dataset):
    seperated = {}
    for i in range(len(dataset)):
        vector = dataset[i]
        if (vector[-1] not in seperated):
            seperated[vector[-1]] = []
        seperated[vector[-1]].append(vector)
    return seperated

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
```

BGSIT

```
def summarizeByClass(dataset):
    separated = separateByClass(dataset)
    summaries = {}
    for classValue, instances in separated.items():
        summaries[classValue] = summarize(instances)
    return summaries

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 = '5_pima-indians-diabetes.data.csv'
    splitRatio = 0.67
```

Name of Experiment _____ Date

Experiment No. _____ Page No. 17

```
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: {0} %'.format(accuracy))
main()
```

Seen

10/10

Input : 6pg.csv

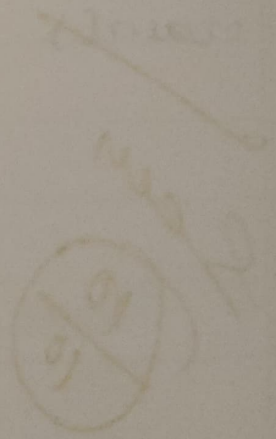
output :

Accuracy metrics
Accuracy of the classifier is 0.4.

confusion matrix
[[1, 2]
 [1 17]]

Recall and Precision

0.5
0.3333333333333



Name of Experiment: Naive Bayesian classifier Date: 21/10/20

Experiment No. 6 Page No. 18

6. Assuming a set of documents that need to be classified, use the naive 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('6pg.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 vectorizer 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 metrices
from sklearn import metrices
print('Accuracy metrices')
print('Accuracy of the classifier', metrics.
      accuracy_score(ytest, predicted)).
print('confusion matrix')
print(metrics.confusion_matrix(ytest, predicted))
```

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

Seen
10/10

Name of Experiment: Bayesian network

Experiment No: 7

Date: 26/11/20

Page No: 21

1) 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 class/ API.

```
import bayespy as bp
import numpy as np.
import csv
from colorama import init
from colorama import Fore, Back, Style
init()

ageEnum = {'SuperSeniorCitizen': 0, 'SeniorCitizen': 1,
    'MiddleAged': 2, 'Youth': 3, 'Teen': 4}
genderEnum = {'Male': 0, 'Female': 1}
familyHistoryEnum = {'Yes': 0, 'No': 1}
dietEnum = {'High': 0, 'Medium': 1, 'Low': 2}
lifeStyleEnum = {'Athlete': 0, 'Active': 1, 'Moderate':
        2, 'Sedetary': 3}
cholestrolEnum = {'High': 0, 'Borderline': 1, 'Normal': 2}
heartDiseaseEnum = {'Yes': 0, 'No': 1}
```

BGSIT

Output:

Index age : {'SuperSeniorCitizen' : 0, 'SeniorCitizen'
'middleAged' : 2, 'Youth' : 3, 'Teen' : 4}

Index Gender : {'male' : 0, 'female' : 1}

Index family history : {'Yes' : 0, 'No' : 1}

Index dietPlan : {'High' : 0, 'medium' : 1, 'low' :

Index Life : {'ALL' : 0 'Active' : 1, 'moderate'

'Sedetary' : 3}}

```
# reading csv file
with open('heart_disease_data.csv') as csvfile:
    # creating a csv reader object
    lines = csv.reader(csvfile)
    dataset = list(lines)
    data = []
    for x in dataset:
        data.append([ageEnum[x[0]], gendaEnum[x[1]],
        familyHistoryEnum[x[2]], dietEnum[x[3]],
        lifeStyleEnum[x[4]], cholesterolEnum[x[5]],
        heartDiseaseEnum[x[6]]])

data = np.array(data)
N = len(data)

# Group assignment probabilities.
p_age = bp.nodes.Dirichlet(1.0*np.ones(5))
# Group assignments for nodes.
age = bp.nodes.Categorical(p_age, plates=(N,))
age.observe(data[:,0])

p_gender = bp.nodes.Dirichlet(1.0*np.ones(2))
gender = bp.nodes.Categorical(p_gender, plates=(N,))
gender.observe(data[:,1])
```

```
p_family_history = bp.node.Dirichlet(1.0*np.ones(2))
familyhistory = bp.node.Categorical(p_familyhistory,
                                    plates=(N,))
familyhistory.observe(data[:,2])

p_diet = bp.nodes.Dirichlet(1.0*np.ones(3))
diet = bp.nodes.Categorical(p_diet,plates=(N,))
diet.observe(data[:,3])

p_lifestyle = bp.nodes.Dirichlet(1.0*np.ones(4))
lifestyle = bp.nodes.Categorical(p_lifestyle,
                                 plates=(N,))
lifestyle.observe(data[:,4])

p_cholestrol = bp.nodes.Dirichlet(1.0*np.ones(3))
cholestrol = bp.nodes.Categorical(p_cholestrol,
                                  plates=(N,))
cholestrol.observe(data[:,5])

p_heartdisease = bp.nodes.Dirichlet(np.ones(2),
                                    plates=(5,2,2,3,4,3))
heartdisease = bp.nodes.MultiMixture([age,gender,
    familyhistory,diet,lifestyle,cholestrol],
    bp.nodes.Categorical,p_heartdisease)
```

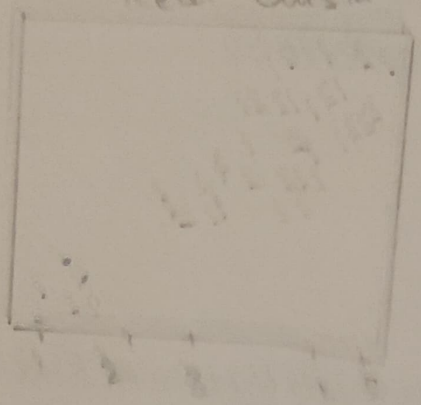
```
heart disease.observe(data[:,6])
p_heart disease.update()

m=0
while m==0:
    printf('\n')
    res = bp.model.MultiMixture([int(input
    ('Enter Age: y' + str(ageEnum))),
    int(input('Enter Gender:' + str(genderEnum))),
    int(input('Enter Family History:' +
    str(familyHistoryEnum))), int(input('
Enter dietEnum:' + str(dietEnum))), int(
input('Enter LifeStyle:' + str(lifeStyleEnum))),
int(input('Enter cholesterol :' + str(cholesterolEnum))),
bp.model.Categorical, p_heart disease).get-
moments()[0][heartDiseaseEnum['No']]
print('Probability(HeartDisease) =' + str(res))
m = int(input('Enter for Continue:0. Exit:1'))
```

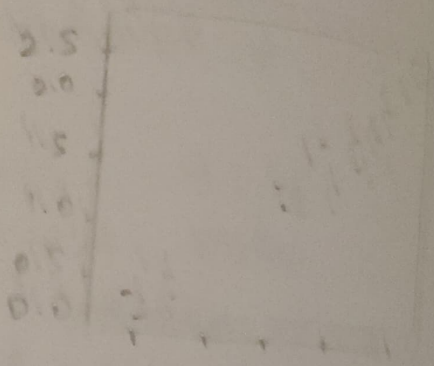
Seen
10/10

Input : Iris. data

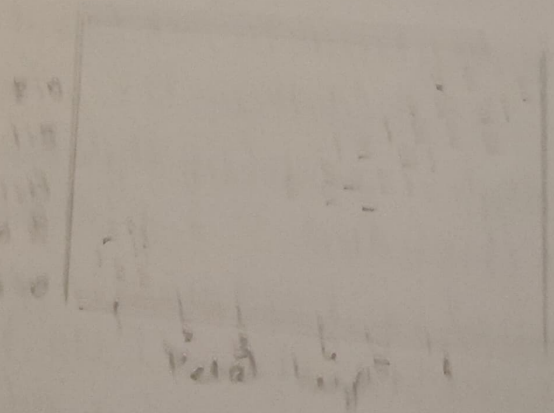
output

Real cluster





K-means clustering



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 class/API in the program.

```
import matplotlib.pylot as plt
from sklearn import datasets
from sklearn.cluster
import KMeans
import pandas as pd
import numpy as np

#import some data to play with
iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length', 'Sepal_Width',
             'Petal_Length', 'Petal_Width']
Y = pd.DataFrame(iris.target)
y.columns = ['Targets']
```

Name of Experiment

Date

Experiment No

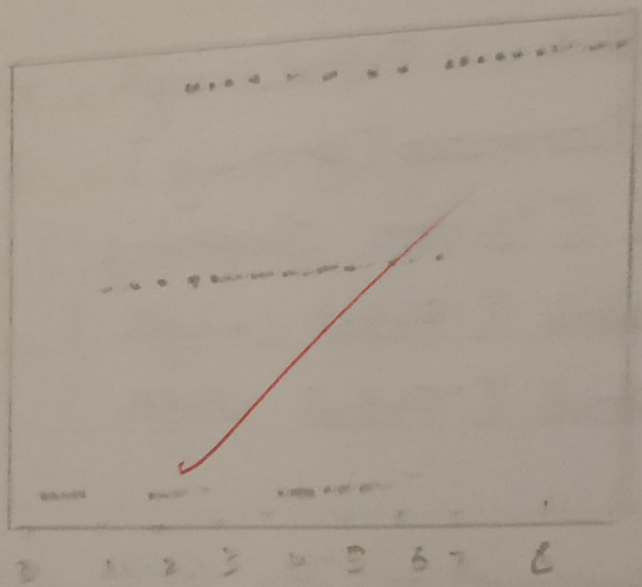
Page No 26

```
#Build the K means model
model = Kmeans(n_clusters = 3)
model.fit(x) #model.labels_ : Give cluster no
  for which samples belongs to

#Visualise the clustering results
plt.figure(fig.size = (14,14))
colormap = np.array(['red', 'lime', 'black'])

#Plot the original classifications using Petal features
plt.subplot(2,2, 1)
plt.scatter(x.Petal_Length, x.Petal_width,
        c= colormap[y.Target], s = 40)
plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal width')

#General EM for GMM
from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
scalar.fit(x)
xsa = scalar.transform(x)
xs = pd.DataFrame(xsa, columns = x.columns)
```

BGSIT

```
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)
gmm_y = gmm.predict(xs)

plt.subplot(2, 2, 3)
plt.scatter(x.Petal_Length, x.Petal_Width,
    c=colormap[gmm_y], s=40)
plt.title('GMM Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

print('Observation: The GMM using EM
algorithm based clustering matched the
true labels more closely than the
Kmeans.').
```

Exercise
10/10



Name of Experiment: K-nearest Algorithm
Date: 6 3 12 20
Experiment No: 9
Page No: 28

9. Write a program to implement K-nearest Neighbour algorithm to classify the iris dataset. Print both correct and wrong predictions. Java/Python ML library class can be used for this problem

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

#Read dataset to pandas dataframe
dataset = pd.read_csv('iris.csv')

x = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 4].values

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test =
        train_test_split(x, y, test_size = 0.20)
plt.plot(x_train, y_train, 'b.', x_test, y_test,
        'r.')
```

BGSIT

Name of Experiment

Date

Experiment No.

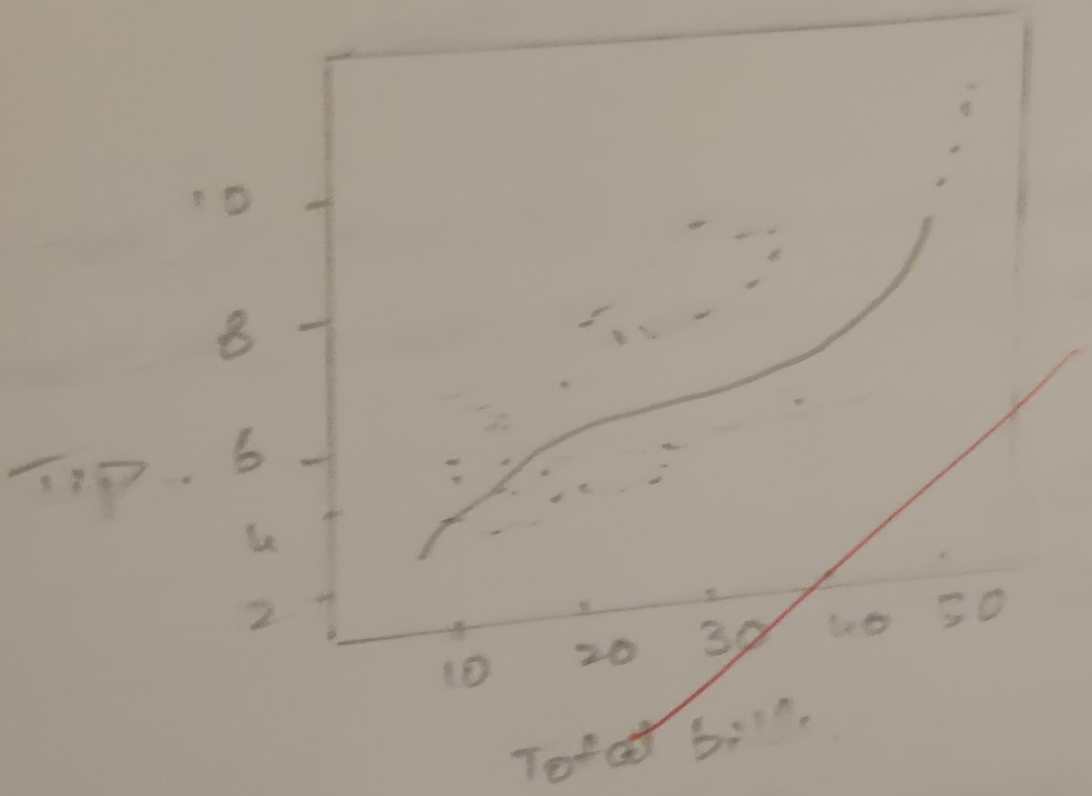
Page No. 54

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train, y_train)

accuracy = classifier.score(x_test, y_test)
accuracy1 = classifier.score(x_train, y_train)

print(accuracy)
print(accuracy1)

example = np.array([[7,7,2,6,6,9,2,3]])
example = example.reshape(1,-1)
print(example)
```

5/5

10/10

Output



Name of Experiment: Locally weight Regression Algorithm Date: 05 12 20

Experiment No. 10. Page No: 30

10) Implement the non-parametric Locally weighted Regression algorithm in order to fit data point. Selection appropriate data set for your experiment and draw graph

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

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

def localweight(point, xmat, ymat, k):
    wei = kernal(point, xmat, k)
    w = (x.T*(wei*x)).I*(x.T*
            (wei*ymat.T))

return w.
```

Name of Experiment _____ Date

Experiment No. _____ Page No. 31

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

def graphplot(x, ypred):
    sortindex = x[:,1].argsort(0)
    #argsort - index of the smallest
    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();

#load data points
data = pd.read_csv('data10_tip1.csv')
```

BGSIT

```
bill = np.array(data.total_bill)
#we use only Bill amount and Tips data.
tip = np.array(data.tip)
mbill = np.mat(bill)
#.mat will convert nd array is converted
# 2D array.
mtip = np.mat(tip)
m = np.shape(mbill)[1]
one = np.mat(np.ones(m))
x = np.hstack((one.T, mbill.T))
# 244 rows, 2 cols.
ypred = localWeightRegression(x, mtip, 8)
#increax k to get smooth curve
graphPlot(x, ypred).
```

Std

10/10

Bayesian.

Jeevan R
4BW17CS029
CSE B

Op Sys

ssuming a set of document that need to classified, use the naive Bayesian classifier model to rform this task. Built-in java classes/API can be used to rite the program. Calculate the accuracy, precision and ecall for your data set.

```
import numpy as np
import pandas as pd
msg = pd.read_csv('6pg.csv', names=['message', 'label'])
X = np.array([[2.9], [1.5], [3.6]], dtype = float)
Y = np.array([[8.6], [8.9], [7.5]], dtype = float)
X = X / np.amax(X, axis=0)
Y = Y / 100
epoch = 7000
lr = 0.1          #learning rate
input_neurons = 2
hidden_neurons = 3
output_neurons = 1
x = msg.message
y = msg.labelnum
print(x)
print(y)
from sklearn.model_selection import train_test_split
wh = np.uniform.classify(size=(input_neurons, hidden_neurons))
bh = np.uniform.classify(size=(1, hidden_neurons))
wout = np.uniform.classify(size=(hidden_neurons, output_neurons))
bout = np.uniform.classify(size=(1, output_neurons))
for i in range(epoch):
```

```
hinpl = np.dot(x_t, wh)        from sklearn.feature_extraction.text
hinp = bh + hinpl
hout = accurati(hinp)

binpl = np.dot(hout, wout)
binp = hout + binpl
bout = derivatiue_accurati(binp)

EO = hout + bout.
h_out = EO * bout
d_output = EO + h_out

print(xtest.shape)
print(xtrain.shape)
print(ytest.shape)
print(ytrain.shape)

def accurati(x):
    return 1/(1 + np.exp(-x))

def derivativel_accurati(x)
    return x * (1 - x)

from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(xtrain_dtm, ytrain)
predicted = clf.predicted(xtest_dtm)
from sklearn import metrics.
print("Input :" + str(x))
print("Precision:" + str(4))
print("Actual Output :" + str(output))
print("Accuracy Metrics")
print('Accuracy of the classifier is', metrics.accuracy_score(ytest, predicted))
print(metrics.confusion_matrix(ytest, predicted))
print("Recall and Precision")
print(metrics.recall_score(ytest, predicted))
print(metrics.precision_score(ytest, predicted))
```

The dimensions of the dataset (18, 2)

Name: message, dtype: object

0	1
1	1
2	1
3	1
4	0
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,)

['amazy', 'an', 'awesome', 'bad', 'best', 'box', 'can', 'dance', 'deal', 'do', 'enemy', 'fun', 'good', 'great', 'have', 'he', 'hodiday', 'horrible', 'how', 'is', 'like', 'locality', 'love', 'my']

Accuracy metrics.

Accuracy of the classifier is 0.6

Confusion matrix

[[1 2]

[0 2]]

Recall and Precision

1.0

0.5