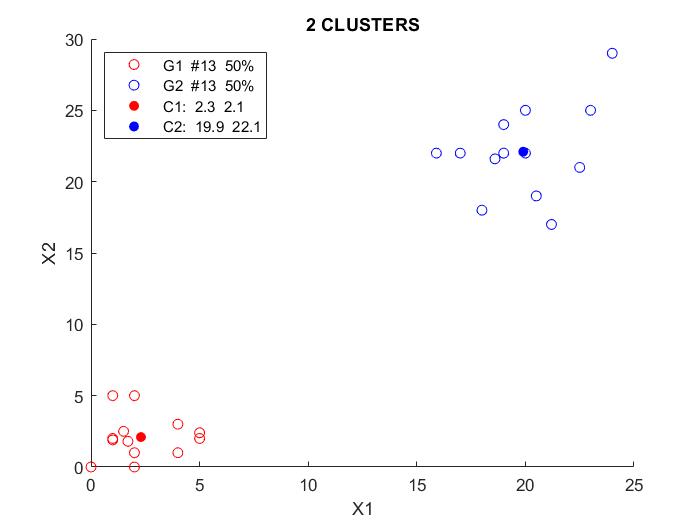
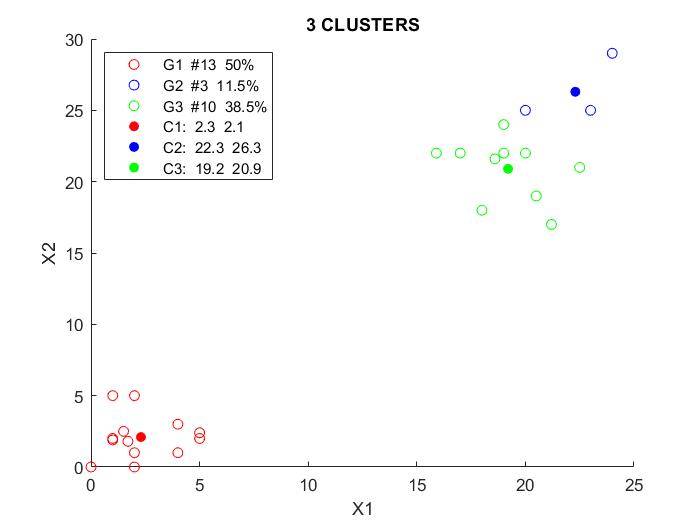
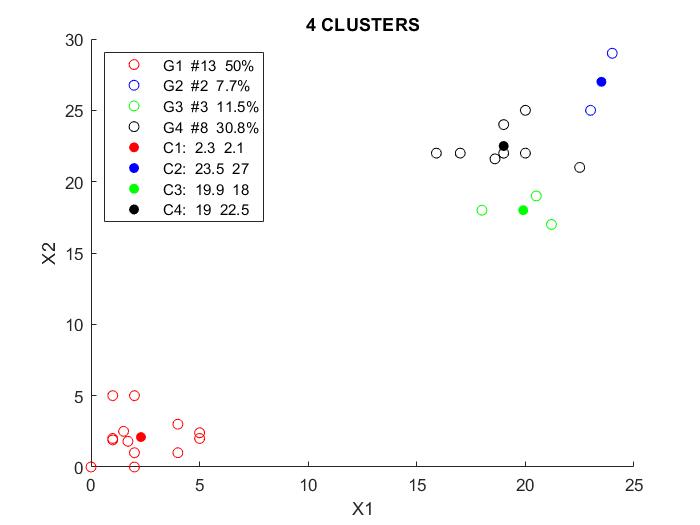
* **Problem 1: KNN Clustering Analysis - Part 1**

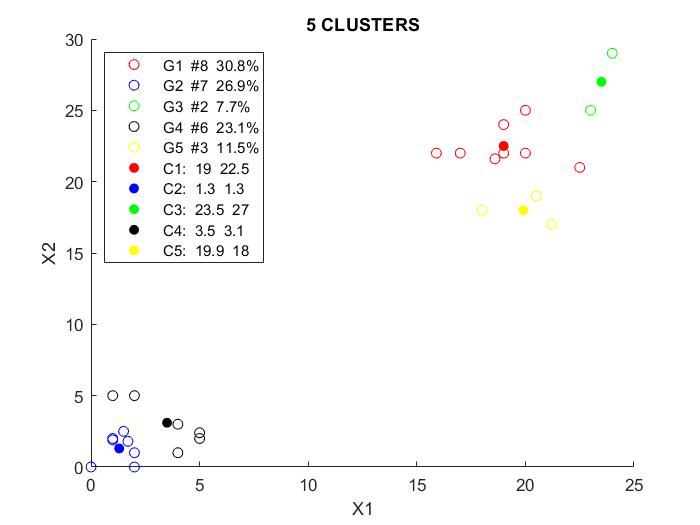
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CLUSTERS | CENTROIDS OF CLUSTER GROUP: | | | | | | | | | |
| **G1** | | **G2** | | **G3** | | **G4** | | **G5** | |
| C1 | C2 | C1 | C2 | C1 | C2 | C1 | C2 | C1 | C2 |
| **2** | 2.3 | 2.1 | 19.9 | 22.1 | **\*** | **\*** | **\*** | **\*** | **\*** | **\*** |
| **3** | 2.3 | 2.1 | 22.3 | 26.3 | 19.2 | 20.9 | **\*** | **\*** | **\*** | **\*** |
| **4** | 2.3 | 2.1 | 23.5 | 27.0 | 19.9 | 18.0 | 19.0 | 22.5 | **\*** | **\*** |
| **5** | 19 | 22.5 | 1.3 | 1.3 | 23.5 | 27.0 | 3.5 | 3.1 | 19.9 | 18.0 |

* + **Plots:**





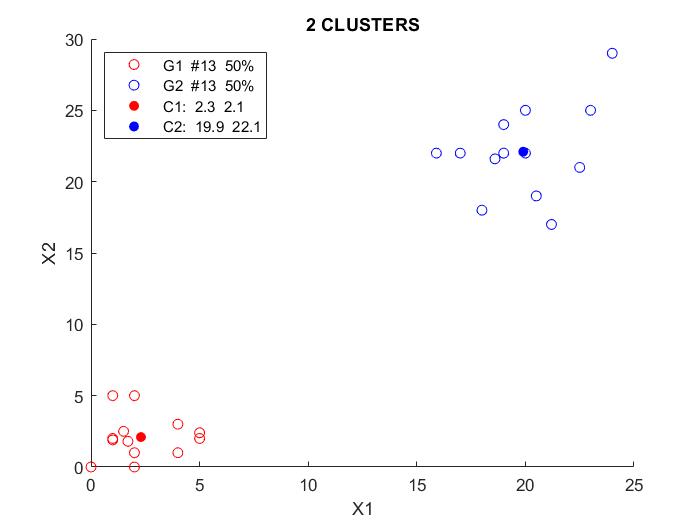




* **Problem 1: KNN Clustering Analysis - Part 2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Clusters with Initial Centroids | Cluster Centroids | | | |
| **G1** | | **G2** | |
| C1 | C2 | C1 | C2 |
| **Using KNN++** | 2.3 | 2.1 | 19.9 | 22.1 |
| **With (5,5) & (8,8)** | 2.3 | 2.1 | 19.9 | 22.1 |

* + **Plots:**



* MATLAB Code

clc;clearvars;close all;clear figure;

infile = 'clust\_data.xlsx';

%% Reading data

datatable = readtable(infile);

headers = datatable.Properties.VariableNames; headers(:,end)=[];

data = datatable.Variables; clear datatable;

%% Calling Clustering function

no\_of\_clusters = 2;

intial\_centroids=[5,5;8,8];

%clusters = KNNplus\_clustering(data,no\_of\_clusters);

clusters = KNNplus\_clustering(data,no\_of\_clusters,intial\_centroids);

%% Plot the final clusters

color = ['r','b','g','k','y'];

lgd\_txt=strings(1,no\_of\_clusters\*2);

plots=gobjects(no\_of\_clusters\*2,1);

centroids=round(clusters.centroids,1);

K=no\_of\_clusters;

for i = 1:K

Xc= centroids(i,1);

Yc= centroids(i,2);

X = data(clusters.cluster\_nos==i,1);

Y = data(clusters.cluster\_nos==i,2);

no= strcat('#',string(clusters.numbers(i)));

pe= strcat(string(clusters.percentage(i)),'%');

lgd\_txt(i)=strcat('G',string(i),{' '},no,{' '},pe);

tx= string(centroids(i,1));

ty= string(centroids(i,2));

lgd\_txt(i+K)=strcat('C',string(i),':',{' '},tx,{' '},ty);

hold on

plots(i)=scatter(X,Y,color(i));

plots(i+K)=scatter(Xc,Yc,color(i),'filled');

end

title\_text = strcat(string(no\_of\_clusters),{' '},'CLUSTERS');

xlabel('X1');ylabel('X2');title(title\_text);

legend(plots,lgd\_txt,'Location','northwest');

hold off

-------------------------------------------------------------------------

function clusters = KNNplus\_clustering(data,no\_of\_clusters,varargin)

%data = (data-mean(data))./std(data);

[nI,nF] = size(data);

%% Parameters defination

K = no\_of\_clusters;

if K==1

clusters.cluster\_nos = ones(nI,1);

clusters.numbers = nI;

clusters.percentage = 100;

clusters.centroids = mean(data);

diff = data-clusters.centroids;

clusters.variance = 0.5\*sum(diff.\*diff,'all')/nI;

return;

end

dist\_old = 10^6;

termination\_threshold = 10^-8;

%% Parameters Initiazation

cluster\_nos = zeros(nI,1);

numbers = zeros(K,1);

percentage = zeros(K,1);

threshold = 100;

iterations = 0;

%% Initial centroids

if isempty(varargin) %for default, use KNN++ approach

centroids = zeros(K,nF);

centroids(1,:) = data(randi(nI),:);

dist=zeros(nI,1);

for i=2:K

for m = 1:nI

instance = data(m,:);

delta = centroids(1:i-1,:)-instance;

euclid = sqrt(sum(delta.\*delta,2));

dist(m) = min(euclid);

end

[~,n]=max(dist);

centroids(i,:)=data(n,:);

end

elseif length(varargin)==1

[nC,nDc] = size(varargin{1});

if or(nC~=K,nDc~=nF)

disp('clustering unsuccessful:centroid dimensions do not match');

return;

else

centroids = varargin{1};

end

else

disp('clustering unsuccessful:too many inputs');

return;

end

%% Clustering

while threshold > termination\_threshold

% Finding Euclidean distance of data points

dist\_new=0;

if iterations~=0

for j=1:K

instances = data(cluster\_nos==j,:);

centroids(j,:)= mean(instances,1);

end

end

for i=1:nI

instance = data(i,:);

delta = (centroids-instance);

euclid = sqrt(sum(delta.\*delta,2));

[min\_dist,cluster\_nos(i)]=min(euclid);

dist\_new = dist\_new+min\_dist;

end

% Finding new centroids

threshold = (abs(dist\_old-dist\_new)/dist\_old);

dist\_old = dist\_new;

iterations = iterations+1;

end

disp(iterations);

clusters.cluster\_nos = cluster\_nos;

clusters.numbers = zeros(K,1);

clusters.percentage = zeros(K,1);

clusters.centroids = centroids;

clusters.variance = zeros(K,1);

for j=1:K

instances = data(cluster\_nos==j,:);

delta = instances-centroids(j,:);

clusters.numbers(j) = size(instances,1);

clusters.percentage(j)= round(clusters.numbers(j)/nI\*100,1);

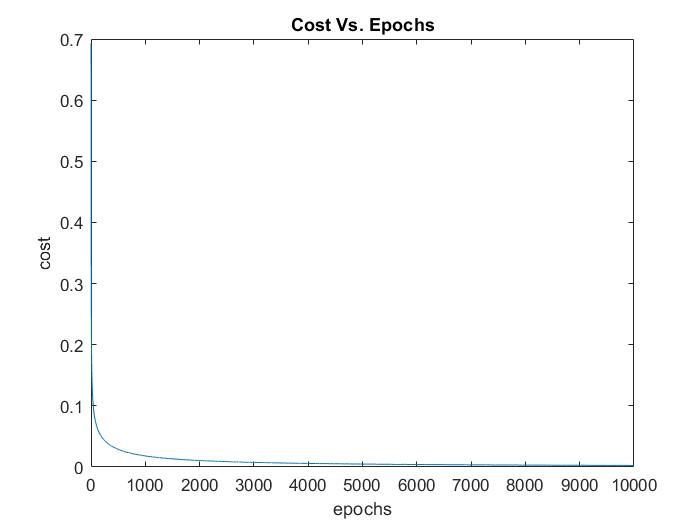
clusters.variance(j) = 0.5\*sum(delta.\*delta,'all')/clusters.numbers(j);

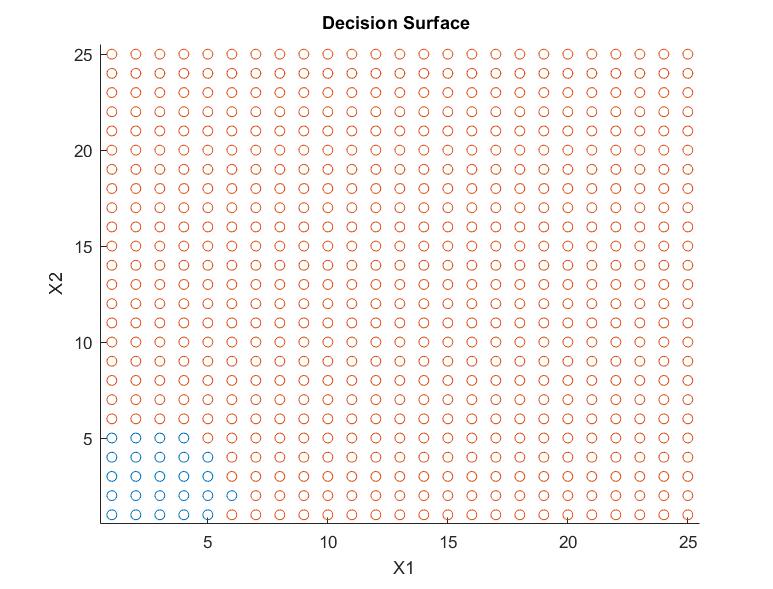
end

end

-------------------------------------------------------------------------

* **Problem 2: Radial Basis Neural Network**





* MATLAB Code

clc;clearvars;close all;

infile = 'Clust\_data\_RBFNN.xlsx';

datatable = readtable(infile);

headers = datatable.Properties.VariableNames; headers(:,end)=[];

training\_data = datatable.Variables; clear datatable;

network\_architecture.learning\_rate = 5;

network\_architecture.max\_epoch = 10000;

network\_architecture.receptors = [1,1]; %no. of receptors for each class

trainedNeuralNetwork = RadialBasisClassifier(network\_architecture,training\_data);

cost = trainedNeuralNetwork.cost;

max\_epoch = network\_architecture.max\_epoch;

plot(1:max\_epoch,cost);

title('Cost Vs. Epochs');xlabel('epochs');ylabel('cost');

no\_of\_class = length(network\_architecture.receptors);

data=zeros(25\*25,2);k=1;

for i=1:25

for j=1:25

data(k,:)= [i j];

k=k+1;

end

end

predicteds=predictoutput\_rbfnn(trainedNeuralNetwork,data);

figure;

for k=1:no\_of\_class

class\_instances=data(predicteds==k,:);

scatter(class\_instances(:,1),class\_instances(:,2));

hold on

end

-------------------------------------------------------------------------

function trainedNeuralNetwork = RadialBasisClassifier(network\_architecture,data)

trainedNeuralNetwork.network\_architecture=network\_architecture;

learning\_rate = network\_architecture.learning\_rate;

max\_epoch = network\_architecture.max\_epoch;

receptors = network\_architecture.receptors;

no\_of\_receptors = sum(receptors);

no\_of\_class = length(receptors);

%%data = transpose(data);

inputs = data(:,1:end-1);

targets = data(:,end); clear data;

[no\_of\_instances,no\_of\_features] = size(inputs);

cost = zeros(max\_epoch,1);

weights = zeros(no\_of\_class,no\_of\_receptors+1);

n=1;

centroids=zeros(no\_of\_receptors,no\_of\_features);

variance=zeros(no\_of\_receptors,1);

for class\_num = 1:no\_of\_class

class\_data = inputs(targets==class\_num,:);

no\_of\_clusters = receptors(class\_num);

m = no\_of\_clusters+n-1;

clusters = KNNplus\_clustering(class\_data,no\_of\_clusters);

centroids(n:m,:)= clusters.centroids;

variance(n:m,:) = clusters.variance;

n = m+1;

end

phi=zeros(no\_of\_instances,no\_of\_receptors);

for r=1:no\_of\_receptors

diff=inputs-centroids(r,:);

phi(:,r)=exp(-0.5\*sum(diff.\*diff,2)/variance(r));

end

phi = transpose([ones(no\_of\_instances,1),phi]);

indices = sub2ind([no\_of\_class,no\_of\_instances],targets,(1:no\_of\_instances)');

for epoch = 1:max\_epoch

netinputs = weights\*phi;

predicteds = softmax\_activation(netinputs);

cost(epoch,1)= sum(-log(predicteds(indices)))/no\_of\_instances; %cross entropy

delta = predicteds; clear predicted;

delta(indices)=delta(indices)-1;

gradient = delta\*transpose(phi)/no\_of\_instances;

weights = weights - learning\_rate\*gradient;

end

trainedNeuralNetwork.weights=weights;

trainedNeuralNetwork.cost =cost;

trainedNeuralNetwork.receptors.centroids=centroids;

trainedNeuralNetwork.receptors.variance=variance;

end

-------------------------------------------------------------------------

function activations = softmax\_activation(netinputs)

activations = exp(netinputs);

activations = activations./sum(activations);

end

-------------------------------------------------------------------------

function predicteds = predictoutput\_rbfnn(trainedNeuralNetwork,inputs)

receptors = trainedNeuralNetwork.network\_architecture.receptors;

weights = trainedNeuralNetwork.weights;

centroids = trainedNeuralNetwork.receptors.centroids;

variance = trainedNeuralNetwork.receptors.variance;

no\_of\_receptors = sum(receptors);

no\_of\_instances = size(inputs,1);

phi=zeros(no\_of\_instances,no\_of\_receptors);

for r=1:no\_of\_receptors

diff=inputs-centroids(r,:);

phi(:,r)=exp(-0.5\*sum(diff.\*diff,2)/variance(r));

end

phi = transpose([ones(no\_of\_instances,1),phi]);

netinputs = weights\*phi;

activations = transpose(softmax\_activation(netinputs));

[~,predicteds] = max(activations,[],2);

end

-------------------------------------------------------------------------