# **CHAPTER 24: Clustering Methods.**



Clustering methods belong to the class of *Unsupervised Machine Learning* methods.

Imagine we are given a *training set* of points {x[1],...,x[m]}, where x[i] ∈ R2:

load('separated\_clusters\_of\_points.mat')

X = data(:,1:2);y = data(:,3);

figure; scatter(X(:,1),X(:,2))

xlabel('x\_1'),ylabel('x\_2')

box on, grid on, hold on

and we want to train a model of the distribution of these points capable of recognizing the presence of *clusters* in the distribution. In a situation of *supervised learning* we would be given a *label* for each point, representing the *cluster* to which the point belongs.

scatter(X(:,1),X(:,2),20,y,'filled')

Then, we could use a *multiclass SVM classifier* (CHAPTER 22 - SPECIAL TOPICS: Support Vector Machines) to identify the regions of the R2 space that identify each cluster:

t = templateSVM('Standardize',true,'KernelFunction','gaussian');

Mdl = fitcecoc(X,y,'Learners',t,'FitPosterior',true, 'ClassNames',{'1','2','3'}, 'Verbose',2);

Here we set up a grid of equally spaced points covering the same range of coordinates of the points in the training set:

xMax = [9 6]; xMin = [-1 0];

x1Pts = linspace(xMin(1),xMax(1)); x2Pts = linspace(xMin(2),xMax(2));

[x1Grid,x2Grid] = meshgrid(x1Pts,x2Pts);

Here we contour the posterior probabilities

[~,~,~,PosteriorRegion] = predict(Mdl,[x1Grid(:),x2Grid(:)]);

figure;

contourf(x1Grid,x2Grid,reshape(max(PosteriorRegion,[],2),size(x1Grid,1),size(x1Grid,2)));

h = colorbar;

h.YLabel.String = 'Maximum posterior'; h.YLabel.FontSize = 15;

hold on

gh = gscatter(X(:,1),X(:,2),y,'rmb','\*xd',8);

gh(2).LineWidth = 2; gh(3).LineWidth = 2;

title('Points coordinates and Maximum Posterior')

xlabel('x\_1'); ylabel('x\_2') 

axis tight; legend(gh,'Location','NorthEast')

hold off

In contrast, in a situation of *unsupervised learning* there are no labels, and we are asked to assign each point to a different cluster. In this case we generate the labels as the *indices* of the clusters.

The most popular method to accomplish this task is the *K-means* clustering algorithm, which is based on an iterative procedure in 3 steps. Step 1 initializes the algorithm (function *kMeansInitCentroids*). Step 2 (function *findClosestCentroids*) and Step 3 (function *computeCentroids*) are repeated inside an inner loop until convergence:

STEP 1: choose *K*, the *number of clusters* to identify, and set arbitrary *coordinate centroids* for the clusters.

K = 3;

figure;scatter(X(:,1),X(:,2))

xlabel('x\_1'),ylabel('x\_2')

box on; grid on; hold on

centroids = [1 1;3 3;5 5];

h = scatter(centroids(:,1),centroids(:,2),160,[1 0 0;0 1 0;0 0 1],'s','filled')



STEP 2: assign points to a cluster based on their *Euclidian distance* from the centroids.

idx = findClosestCentroids(X, centroids)

idx\_mat = [1 0 0;0 1 0;0 0 1]

idx\_mat = idx\_mat(idx,:)

h1 = scatter(X(:,1),X(:,2),20,idx\_mat,'filled')



STEP 3: calculate new centroids as the *mean of the points* initially assigned to each cluster.

centroids = computeCentroids(X, idx, K)

delete(h)

delete(h1)

h = scatter(centroids(:,1),centroids(:,2),160,[1 0 0;0 1 0;0 0 1],'s','filled')



STEP 2: assign points to a cluster based on their *Euclidian distance* from the centroids. idx =

findClosestCentroids(X, centroids)

idx\_mat = [1 0 0;0 1 0;0 0 1]

idx\_mat = idx\_mat(idx,:)

h1 = scatter(X(:,1),X(:,2),20,idx\_mat,'filled')



STEP 3: calculate new centroids as the *mean of the points* initially assigned to each cluster.

centroids = computeCentroids(X, idx, K)

delete(h)

delete(h1)

h = scatter(centroids(:,1),centroids(:,2),160,[1 0 0;0 1 0;0 0 1],'s','filled')



STEP 2: assign points to a cluster based on their *Euclidian distance* from the centroids.

idx = findClosestCentroids(X, centroids)

idx\_mat = [1 0 0;0 1 0;0 0 1]

idx\_mat = idx\_mat(idx,:)

scatter(X(:,1),X(:,2),20,idx\_mat,'filled')



These steps can be conveniently incorporated in a single function (*runkMeans*), which can be run *only once* with *k* centroid guesses*, or recursively*, selecting the centroids with the best *cost function*:

Random initialization

close all

K = 3;

max\_iters = 10;

centroids = kMeansInitCentroids(X, K)

[centroids, idx] = runkMeans(X, centroids, max\_iters, 'plot\_progress')

Cost function

xc\_diff = X - centroids(idx,:);

J = mean(diag(xc\_diff\*xc\_diff'))

Recursive initialization

close all

ntrials = 100;

[m,n] = size(X);

centroids\_mat = zeros(3,2,ntrials);

idx\_mat = zeros(m,ntrials);

J\_mat = zeros(ntrials,1);

for i = 1:ntrials

K = 3;

max\_iters = 10;

centroids = kMeansInitCentroids(X, K)

[centroids, idx] = runkMeans(X, centroids, max\_iters)

% Cost function

xc\_diff = X - centroids(idx,:);

J = mean(diag(xc\_diff\*xc\_diff'));

% Arrays

centroid\_mat(:,:,i) = centroids;

idx\_mat(:,i) = idx;

J\_mat(i) = J;

end

[J\_best J\_best\_idx] = min(J\_mat)

best\_centroids = centroid\_mat(:,:,J\_best\_idx)

best\_idx = idx\_mat(:,J\_best\_idx)

Plot the point as a scatter plot

figure;

scatter(X(:,1),X(:,2))

xlabel('x\_1'),ylabel('x\_2')

box on,grid on

hold on

color\_mat = [1 0 0;0 1 0;0 0 1]

scatter(best\_centroids(:,1),best\_centroids(:,2),160,color\_mat,'s','filled')

scatter(X(:,1),X(:,2),20,color\_mat(best\_idx,:),'filled')

The *K-means* clustering algorithm can also be used to segment a continuous spread of data.

load('adjacent\_clusters\_of\_points.mat')

Plot the point as a scatter plot

figure;

scatter(X(:,1),X(:,2))

xlabel('x\_1'),ylabel('x\_2')

box on,grid on

hold on

*K-means* with evenly spaced centroids

close all

K = 4;

max\_iters = 10;

cen\_x = linspace(min\_xy(1),max\_xy(1),K);

cen\_y = linspace(min\_xy(2),max\_xy(2),K);

centroids = [cen\_x' cen\_y'];

[centroids, idx] = runkMeans(X, centroids, max\_iters, 'plot\_progress')

*K-means* by random initialization

close all

K = 3;

max\_iters = 10;

centroids = kMeansInitCentroids(X, K)

[centroids, idx] = runkMeans(X, centroids, max\_iters, 'plot\_progress')

*K-means* by recursive random initialization

close all

ntrials = 100;

[m,n] = size(X);

centroids\_mat = zeros(3,2,ntrials);

idx\_mat = zeros(m,ntrials);

J\_mat = zeros(ntrials,1);

for i = 1:ntrials

K = 3;

max\_iters = 10;

centroids = kMeansInitCentroids(X, K)

[centroids, idx] = runkMeans(X, centroids, max\_iters)

% Cost function

xc\_diff = X - centroids(idx,:);

J = mean(diag(xc\_diff\*xc\_diff'));

% Arrays

centroid\_mat(:,:,i) = centroids;

idx\_mat(:,i) = idx;

J\_mat(i) = J;

end

[J\_best J\_best\_idx] = min(J\_mat)

best\_centroids = centroid\_mat(:,:,J\_best\_idx)

best\_idx = idx\_mat(:,J\_best\_idx)

Plot the point as a scatter plot

figure;

scatter(X(:,1),X(:,2))

xlabel('x\_1'),ylabel('x\_2')

box on,grid on

hold on

color\_mat = [1 0 0;0 1 0;0 0 1]

scatter(best\_centroids(:,1),best\_centroids(:,2),160,color\_mat,'s','filled')

scatter(X(:,1),X(:,2),20,color\_mat(best\_idx,:),'filled')

MATLAB has its own *kmeans* function, but also more complex functions (i.e., *clusterdata, cluster*) for clustering.

Load well separated clusters of points in 2D space

clear,clc,close all

load('separated\_clusters\_of\_points.mat')

X = data(:,1:2);

figure;

scatter(X(:,1),X(:,2))

xlabel('x\_1'),ylabel('x\_2')

box on,grid on

hold on

Using *kmeans*

k = 3;

[idx,C] = kmeans(X,k)

scatter(X(:,1),X(:,2),20,idx,'filled')

scatter(C(:,1),C(:,2),160,[1 0 0;0 1 0;0 0 1],'s','filled')

Using *clusterdata*

idx = clusterdata(X,'Linkage','ward','SaveMemory','on','Maxclust',3);

scatter(X(:,1),X(:,2),40,idx,'filled')

xlabel('x\_1'),ylabel('x\_2')

Using *fitgmdist* and *cluster* with mixture of gaussian distributions. This is a *generative* methods that assumes the clusters contain points that have a *gaussian n-dimensional* distribution.

gm = fitgmdist(X,3);

idx = cluster(gm,X);

figure;

gscatter(X(:,1),X(:,2),idx);

xlabel('x\_1'),ylabel('x\_2')

legend('Cluster 1','Cluster 2','Cluster 3','Location','best');

grid on;hold on

gmPDF = @(x,y) arrayfun(@(x0,y0) pdf(gm,[x0 y0]),x,y);

fcontour(gmPDF,[-1 9 -1 6])



Load adjacent clusters of points in 2D space

clear, clc, close all

load('adjacent\_clusters\_of\_points.mat')

figure;

scatter(X(:,1),X(:,2))

xlabel('x\_1'),ylabel('x\_2')

box on,grid on

hold on

Using *kmeans*

k = 3;

[idx,C] = kmeans(X,k)

scatter(X(:,1),X(:,2),20,idx,'filled')

scatter(C(:,1),C(:,2),160,[1 0 0;0 1 0;0 0 1],'s','filled')

Using *clusterdata* with *ward* linkage

idx = clusterdata(X,'Linkage','ward','SaveMemory','on','Maxclust',3);

scatter(X(:,1),X(:,2),40,idx,'filled')

xlabel('x\_1'),ylabel('x\_2')

box on;grid on

Using *clusterdata* with *centroid* linkage

figure;

scatter(X(:,1),X(:,2))

xlabel('x\_1'),ylabel('x\_2')

box on,grid on

hold on

idx2 = clusterdata(X,'Linkage','centroid','SaveMemory','on','Maxclust',3);

scatter(X(:,1),X(:,2),40,idx2,'filled')

xlabel('x\_1'),ylabel('x\_2')

box on;grid on

Using *fitgmdist* and *cluster* with mixture of gaussian distributions. In this case the assumption of a gaussian distribution of points inside each cluster reveals a completely different clustering.

gm = fitgmdist(X,3);

idx = cluster(gm,X);

figure;

gscatter(X(:,1),X(:,2),idx);

xlabel('x\_1'),ylabel('x\_2')

legend('Cluster 1','Cluster 2','Cluster 3','Location','best');

grid on;hold on

gmPDF = @(x,y) arrayfun(@(x0,y0) pdf(gm,[x0 y0]),x,y);

fcontour(gmPDF,[0 8 2 8])

