CLP Lab 7 Report

Albert Aparicio Isarn
albert.aparicio.isarn@alu-etsetb.upc.edu
Héctor Esteban Cabezos
hect.esteban@gmail.com

1. Programación de método de clasificación

El script `main_2_1` contiene el código del ejercicio 1. En la sección 4.1 se puede ver su código fuente.

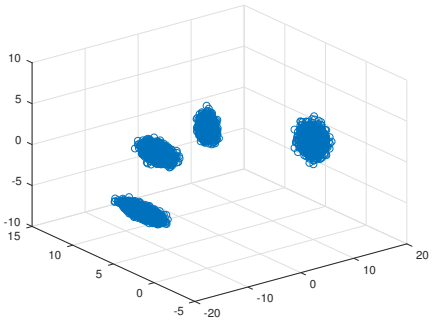
La sección 2 de este ejercicio corresponde a la función `CLP_Kmeans`, cuyo código fuente se muestra en la sección 4.5.

A continuación se muestran los resultados de este ejercicio. En la figura 1 se muestra la base de datos creada por `CLP_Generate` (código fuente en la sección 4.4) y el resultado de la clasificación con 4, 9 y 10 *clusters*.

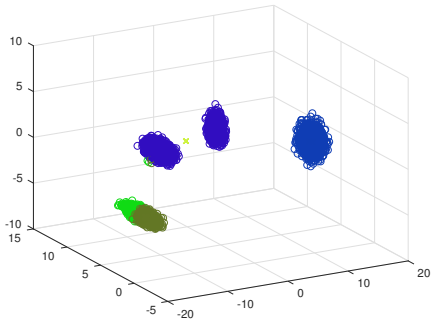
En los resultados de la clasificación de la figura 1 se pueden ver también algunas de las evoluciones de los centroides. No se ven todas porque algunos centroides quedan "ocultos" dentro de la nube de puntos del *cluster*.

El cálculo de $Trace\left(S_W^{-1}S_B\right)$ es el que más información da sobre el rendimiento del clasificador. Cuando la clasificación empeora, el valor de éste cálculo disminuye notablemente ($K = 4$ en la figura 2c), mientras que J y $Trace\left(S_T^{-1}S_W\right)$ aumentan en menor medida o se mantienen.

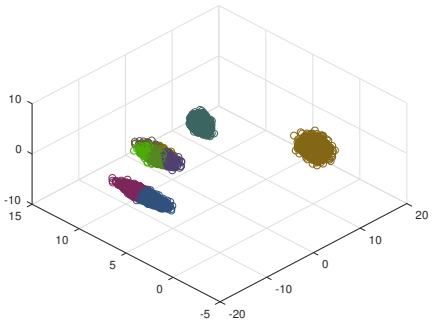
Si se usa la menor K que da mayor valor de $Trace\left(S_W^{-1}S_B\right)$, se consigue el número de clústeres óptimo para esa base de datos.



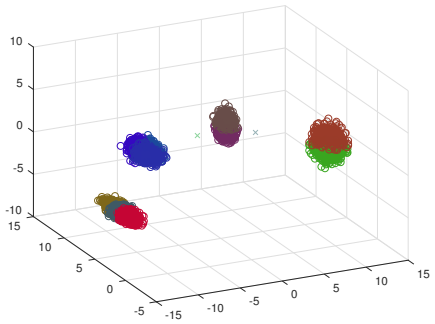
(a) Base de datos sintética



(b) 4 *clusters*

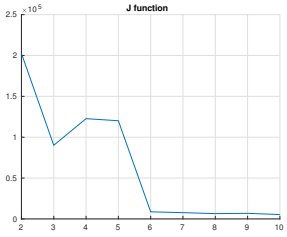


(c) 9 *clusters*

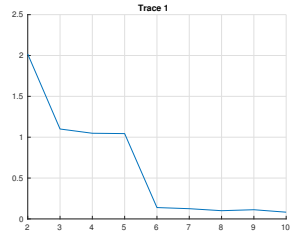


(d) 10 *clusters*

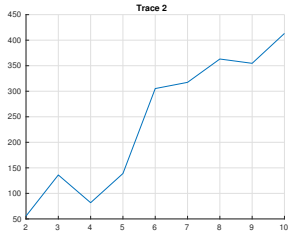
Figura 1: Clasificación de la base de datos sintética



(a) Evolución de la función de coste (J)



(b) Evolución de la métrica $Trace\left(S_T^{-1}S_W\right)$



(c) Evolución de la métrica $Trace\left(S_W^{-1}S_B\right)$

Figura 2: Evolución de las diversas métricas usadas en la clasificación

2. Cuantificación de imágenes

El script `main_2_2` contiene el código del ejercicio 2. En la sección 4.2 se puede ver su código fuente.

La recuantificación de la imagen se ha hecho con $K = 7$ colores.

En las figuras 3a y 3b, respectivamente, se puede ver el resultado de la clasificación.

(a) Canales de la imagen original (b) Canales de la imagen recuantificada

Figura 3: Recuantificación de la imagen *Lena* con $K = 7$ colores

En la figura 4 se pueden ver los *clusters* de la image original y de su versión recuantificada.

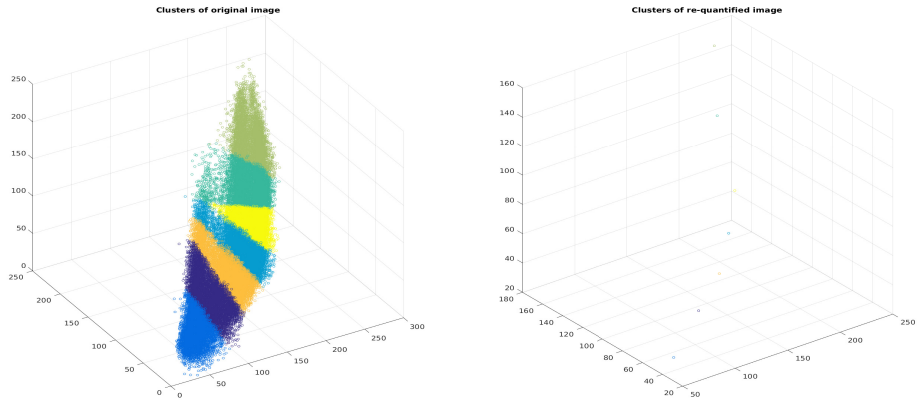


Figura 4: *Clusters* de la imagen original y recuantificada

La evolución de los valores de la función de coste J se muestra en la figura 5.

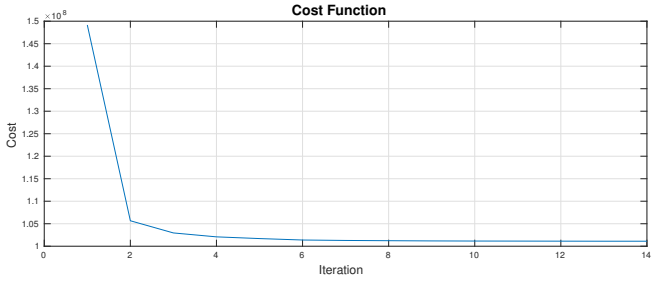


Figura 5: Evolución de los valores de la función de coste J

.

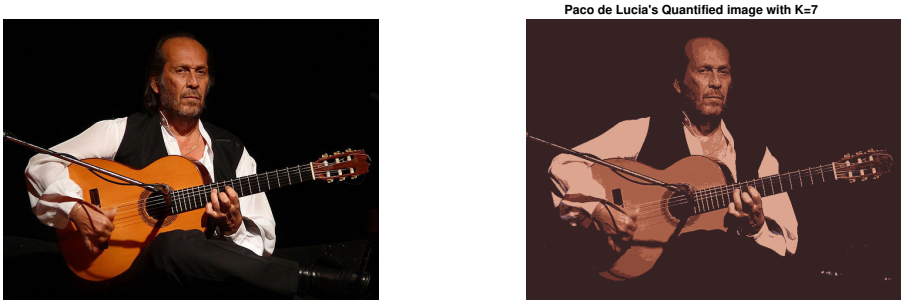
En la figura se puede ver como el valor de la función desciende rápidamente, de manera que a partir de la iteración 6, la clasificación prácticamente no varia. Las iteraciones siguientes se han calculado para cumplir con la condición del clasificador, que dicta que se continue iterando hasta que $\Delta J \leq th = 0{,}0005$.

Los valores de $Trace\left(S_T^{-1}S_W\right)$ y $Trace\left(S_W^{-1}S_B\right)$, junto al cálculo de los bits necesarios para el almacenamiento de las imágenes, son los siguientes:

```
>> main_2_2
La medida de tr(St\Sw) es: 1.42
La medida de tr(Sw\Sb) es: 37.7702
We need 6291456 bits to store the Lena image
We need 3145728 bits to store the quantified image
```

Estos valores concuerdan con la optimización de la clasificación; tenemos un valor pequeño para $Trace\left(S_T^{-1}S_W\right)$ y un valor grande para $Trace\left(S_W^{-1}S_B\right)$.

Con los centroides obtenidos de la imagen *Lena* con $K = 7$, se ha cuantificado una imagen del guitarrista Paco de Lucía. Los resultados se muestran en la figura 6.



(a) Imagen original (b) Imagen recuantificada

Figura 6: Recuantificación de la imagen *Paco* con los centroides obtenidos con la imagen *Lena*

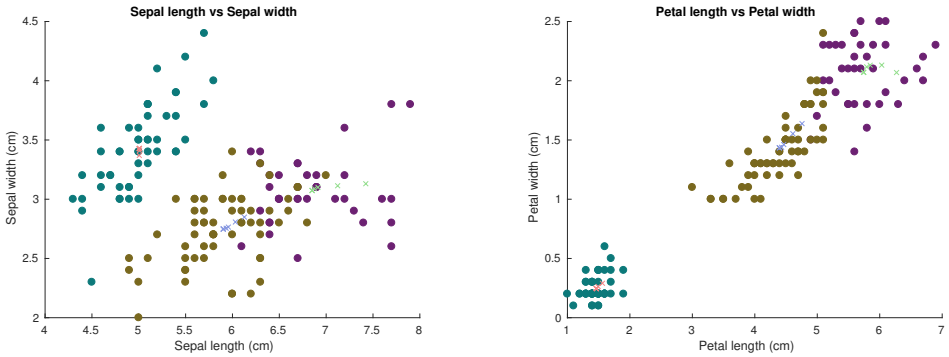
3. Identificación de clústeres en una base de datos propia

El script `main_2_3` contiene el código del ejercicio 3. En la sección 4.3 se puede ver su código fuente.

Para este ejercicio se ha usado la base de datos **Iris Plants Database** (http://archive.ics.uci.edu/ml/datasets/Iris). Esta base de datos contiene información de la longitud y anchura de los sépalos y pétalos de varias espécies de flores del género *Iris*.

En la descripción de la base de datos se especifica que los 150 vectores están distribuidos en 3 clases. Una de ellas es linealmente separable de las otras dos, mientras que las otras dos restantes, *NO* son linealmente separables.

En la figura 7 se pueden ver los *clusters* clasificados. Claramente se muestra una clase separada de las otras dos.



(a) *Clusters* de las características de los sépalos

(b) *Clusters* de las características de los pétalos

Figura 7: Resultado de la clasificación de la base de datos **Iris Plants Database**

Para evaluar el rendimiento del clasificador, como esta base de datos es etiquetada, podemos comprobar el índice de aciertos. Sin embargo, esto no es una tarea simple.

El problema radica en que los centroides del clasificador se asignan aleatoriamente. Esto hace que no podamos saber qué especie de *Iris* corresponde a cada centroide. Por tanto, no podemos medir cuántos vectores han sido clasificados correctamente comparando las etiquetas.

Para poder evaluar el rendimiento del clasificador, hemos aplicado un producto matricial sobre el vector de etiquetas resultante de la clasificación, y lo hemos representado con un código de colores. La figura 8 muestra esta representación.

Esta forma de mostrar las etiquetas permite una evaluación visual aproximada de cuán bien trabaja el clasificador. La lógica tras esta representación es:

- Si todas las etiquetas se detectan correctamente, la figura resultante mostrará 9 (3^2) rectángulos de distintos colores, segun el código elegido.

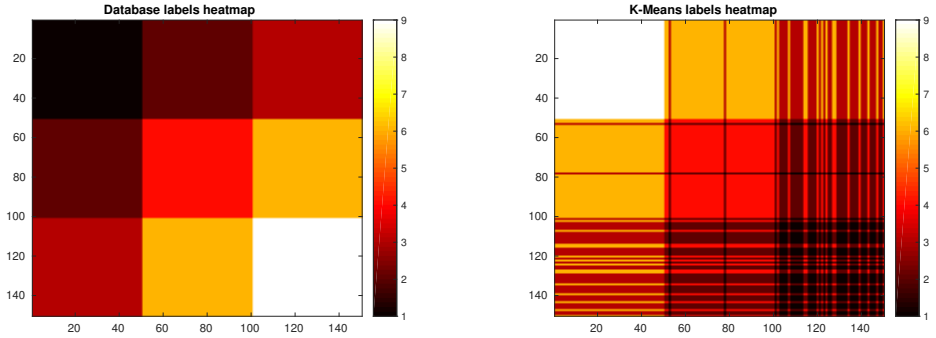
- Si hay vectores mal clasificados, éstos se mostrarán en el gráfico en forma de líneas de distinto color al del rectángulo en el que deberian pertenecer.

Esta forma de evaluar la clasificación funciona porque en la base de datos, los vectores están ordenados de tal manera que los vectores de una misma clase ocupan posiciones correlativas.

En la figura 8a se muestran las etiquetas de la base de datos. Este gráfico muestra los 9 rectángulos perfectos. En la figura 8b se muestran las etiquetas de la clasificación. Como indicábamos anteriormente, hay líneas de distinto color al rectángulo que las rodea, indicando vectores mal clasificados.

El rectángulo de la parte superior izquierda de la figura representa la clase linealmente separable de las otras dos, ya que solo muestra un solo vector mal clasificado.

Los colores no se corresponden entre los dos gráficos, debido a la asignación aleatoria de los centroides mencionada anteriormente.



(a) Etiquetas de la base de datos (b) Etiquetas resultado de la clasificación

Figura 8: Representación de las etiquetas de la base de datos y de la clasificación.

4. Código fuente

A continuación se encuentra el código fuente generado para la resolución de este laboratorio

4.1. main_2_1

```
%% Exercise 2.1 script of the K-Means classifier
clear
close all

% Switch to activate or deactivate the plotting of all classifiers
plot_clusters = 1;

%% Section 1
% Generate DB

% Initialize parameters
L = 4;
N = 10000;
d = 3;
th = 0.0005;

% Compute a priori probabilities for each cluster
probabilities = rand(L,1);
probabilities = probabilities./sum(probabilities);

% Generate DB samples
[DB, Nnew] = CLP_Generate(L,N,d,probabilities);

% Draw clusters
scatter3(DB(1,:), DB(2,:), DB(3,:))%, hold on

%% Section 3
% Classify with K-Means clustering

% Preallocate results from the classifier
J = cell(9,1);
minimized_J = zeros(9,1);

trace1 = zeros(9,1);
trace2 = zeros(9,1);

Sw = zeros(d,d,9);
Sb = zeros(d,d,9);

for K=2:10
    [Centroides, Labels, n , J{K-1}, trace1(K-1), trace2(K-1), ...
        Sw(:,:,K-1), Sb(:,:,K-1)] = CLP_Kmeans(DB(1:d, :),K, d, th);

    minimized_J(K-1) = J{K-1}(end);
```

```
    if plot_clusters
        % Plot DB with color labeling
        figure, hold on
        for i=1:K
            c = rand(1,3);
            scatter3(DB(1,Labels==i), DB(2,Labels==i), DB(3,Labels==i),
    ...
                'MarkerEdgeColor', c/sum(c))
            scatter3(Centroides(1,i,:), Centroides(2,i,:), Centroides(3,i
    ,:),...
                'x', 'MarkerEdgeColor', 1 - c/sum(c))
        end
        grid on
        hold off
    end
end

%% Section 4
% Analyze classification metrics

figure, hold on
plot(2:10, minimized_J)
grid on
title('J function')
hold off

figure, hold on
plot(2:10, trace1)
grid on
title('Trace 1')
hold off

figure, hold on
plot(2:10, trace2)
grid on
title('Trace 2')
hold off

% Trace 2 is great, Trace 1 not good, because it decays constantly. It
    does so
% because as we increase the number of clusters, each of them is more
    compact,
% so the "within" metric improves.
%
% The J function behaves sometimes roughly like Trace 1
%
% We want to minimize Trace 1 and maximize Trace 2
%
% Trace 2 increases greatly when different clusters are classified as
    such,
% while Trace 1 does slightly decrease in the same situation.
%
```

```
% Because of this, Trace 2 is the best metric
```

../src/main_2_1.m

4.2. main_2_2

```
%% Exercise 2.2 script of the K-Means classifier
clear
close all

% Read Lena image
imageclp = imread('images/lena.jpg');

original_red = reshape(imageclp(:,:,1),1,[]);
original_green = reshape(imageclp(:,:,2),1,[]);
original_blue = reshape(imageclp(:,:,3),1,[]);
image_rgb = [original_red ; original_green ; original_blue ];

%% Section 1
% Requantify image with a k-means algorithm

d = 3; % RGB represents 3 dimensions
K = 7; % Compute 7 centroids
th = 0.0005; % Variation threshold for the classifier

% 1st centroids column represents all dimensions of red's centroids
[Centroids_rgb, Labels_rgb, n_rgb , J_rgb, trace1_rgb, trace2_rgb, ...
    Sw_rgb(:,:,K-1), Sb_rgb(:,:,K-1)] = CLP_Kmeans(image_rgb(1:d, :),K,d,
    th);

% Reconstruct re-quantified image

% Take the centroid of the last iteration
Centroides_definitiu = Centroids_rgb(:,:,end);

% Transpose the matrix to adapt it to the shape of the centroids matrix
Labels_rgb = Labels_rgb';

% Pre-allocate the matrix of the result image
vector_image = zeros([1, size(image_rgb')]);

for i = 1:length(original_red)
    % Assign the corresponding centroid to each pixel, according to its
    label
    vector_image(1,i,:) = Centroides_definitiu(:,Labels_rgb(1,i));
end

% Reshape back into a 3-channel image
requantified_lena = uint8(reshape(vector_image, size(imageclp)));

%% Section 2
% Represent the separate components of the original and re-quantified
    image
```

```
% Original image
figure
subplot(2,2,1)
imshow(imageclp), title('Original image')
subplot(2,2,2)
imshow(imageclp(:,:,1)), title('Original Red channel')
subplot(2,2,3)
imshow(imageclp(:,:,2)), title('Original Green channel')
subplot(2,2,4)
imshow(imageclp(:,:,3)), title('Original Blue channel')

% Re-quantified image
figure
subplot(2,2,1)
imshow(requantified_lena), title('Re-quantified image')
subplot(2,2,2)
imshow(requantified_lena(:,:,1)), title('Re-quantified Red channel')
subplot(2,2,3)
imshow(requantified_lena(:,:,2)), title('Re-quantified Green channel')
subplot(2,2,4)
imshow(requantified_lena(:,:,3)), title('Re-quantified Blue channel')

% Show clusters of original image
figure
subplot(1,2,1)
scatter3(original_red, original_green, original_blue, 10, Labels_rgb)
title('Clusters of original image')
% Show clusters of re-quantified image
subplot(1,2,2)
scatter3(vector_image(:,:,1), vector_image(:,:,2), vector_image(:,:,3),
    ...
    10, Labels_rgb)
title('Clusters of re-quantified image')

%% Section 3
% Display the evolution of the metrics

% Display evolution of the cost function
figure, plot(1:length(J_rgb),J_rgb), hold on
title('Cost Function','FontSize',16)
xlabel('Iteration','FontSize',14)
ylabel('Cost','FontSize',14)
grid on, hold off

% Display the results of the trace metrics
disp(['La medida de tr(St\Sw) es: ', num2str(trace1_rgb)])
disp(['La medida de tr(Sw\Sb) es: ', num2str(trace2_rgb)])

%% Section 4
% Evaluate how many bits we need to store the original and re-quantified
    images

numero_bits = 8 * (numel(imageclp));
```

```
disp(['We need ', num2str(numero_bits), ' bits to store the Lena image'])
    ;

K_quant = 16;
numero_bits_codificada = log2(K_quant) * (numel(imageclp));
disp(['We need ', num2str(numero_bits_codificada), ...
    ' bits to store the quantified image']);

%% Section 5
% Import a different image and apply Lena's centroids

imageclp2 = imread('images/PacoLucia.jpg');

original_red_p = reshape(imageclp2(:,:,1),1,[]);
original_green_p = reshape(imageclp2(:,:,2),1,[]);
original_blue_p = reshape(imageclp2(:,:,3),1,[]);
image_rgbp = [original_red_p ; original_green_p ; original_blue_p ];

vector_image_paco = zeros([1, size(image_rgbp')]);

for i = 1:length(vector_image_paco)
    norms = sum(abs(repmat(...
        double(image_rgbp(:,i)), 1, K) - double(Centroides_definitiu))
    .^2,1);
    [Minim_value, index] = min(norms);

    % Assign the RGB value of the closest centroid
    vector_image_paco(1,i,:) = Centroides_definitiu(:,index);
end

requantified_paco = uint8(reshape(vector_image_paco, size(imageclp2)));

% Plot result of classifying another image with the previous centroids
figure
% subplot(1,2,1)
imshow(imageclp2)
title('Paco de Lucia''s original image','FontSize',16);

figure
% subplot(1,2,2)
imshow(requantified_paco)
title(['Paco de Lucia''s Quantified image with K=', num2str(K)],'FontSize
    ',16);
```

../src/main_2_2.m

4.3. main_2_3

```
%% Exercise 2.3 script of the K-Means classifier
% Classify a new database

close all
clear

heatmap = 1;
plot_clusters = 1;

%% Parse Iris database

file_path = 'db/bezdekIris.data';
[ DB, db_labels ] = CLP_Parse_DB( file_path );

d = size(DB, 1);

%% Classify DB with K-means

K = 3;
th = 0.0005;

[ Centroids, Labels, n, J, tr_1, tr_2, Sw, Sb ] = CLP_Kmeans(DB, K, d, th
    );

%% Plot results
% There will be 2 figures, one for petals and one for sepals. This
    separation is
% necessary, as the DB is 4-dimensional

close all

if plot_clusters

    % Plot DB with color labeling
    c = rand(K,3); % Use the same colours for the two figures

    figure
%     subplot(2,2,1)
    hold on
    for i=1:K
        scatter(DB(1,Labels==i), DB(2,Labels==i), ...
            'MarkerEdgeColor', c(i,:)/sum(c(i,:)),...
            'MarkerFaceColor', c(i,:)/sum(c(i,:)))
        scatter(Centroids(1,i,:), Centroids(2,i,:), ...
            'x', 'MarkerEdgeColor', 1 - c(i,:)/sum(c(i,:)))
    end

    title('Sepal length vs Sepal width')
    xlabel('Sepal length (cm)')
    ylabel('Sepal width (cm)')
    hold off
```

```
    figure
%     subplot(2,2,2)
    hold on
    for i=1:K
        scatter(DB(3,Labels==i), DB(4,Labels==i), ...
            'MarkerEdgeColor', c(i,:)/sum(c(i,:)),...
            'MarkerFaceColor', c(i,:)/sum(c(i,:)))
        scatter(Centroids(3,i,:), Centroids(4,i,:), ...
            'x', 'MarkerEdgeColor', 1 - c(i,:)/sum(c(i,:)))
    end

    title('Petal length vs Petal width')
    xlabel('Petal length (cm)')
    ylabel('Petal width (cm)')
    hold off
end

% Display accuracy of the classification as heatmaps
if heatmap
    % Colormap of database labels
    figure
%     subplot(2,2,3)
    colormap('hot')
    imagesc(db_labels * db_labels')
    colorbar
    title('Database labels heatmap')

    % Colormap of classifier labels
    figure
%     subplot(2,2,4)
    colormap('hot')
    imagesc(Labels * Labels')
    colorbar
    title('K-Means labels heatmap')

    % If the classifier was perfect, the 9 squares in the K-Means would
    look
    % like the DB squares (although the order of the colors can be
    different)
    %
    % If there are any lines in the classifier heatmap, those lines
    indicate
    % misclassified vectors
end
```

../src/main_2_3.m

4.4. CLP_Generate

```
function [ DB, N_new ] = CLP_Generate( L, N, d, priori_prob )
%CLP_Generate Generate a synthetic gaussian database
%
%   [ DB, Nnew ] = CLP_Generate( L, N, d, probabilities )
%
%   This function generates L clusters of gaussian-distributed d-
    dimensional
%   vectors.
%
%   The L clusters contains a number of samples <= N. The total number
%   of samples (N_new) may be less than N. This could happen due to the
    rounding
%   process in which each cluster is assigned a number of vectors,
    following the
%   a priori probabilities vector (priori_prob).
%
%   INPUTS:
%
    ============================================================================

%   - L: Number of clusters to be generated
%   - N: Total number of samples to be generated
%   - d: Number of dimensions in the vectors
%   - priori_prob: Vector of a priori probabilities for the clusters
%
%   Notice that L == length(priori_prob)
%
%   OUTPUTS:
%
    ============================================================================

%   - DB: Matrix containing the generated database (first d rows contain
    the
%         vector components, last row contains the labels)
%   - N_new: Actual total number of generated vectors (N_new <= N)
%
%   TODO: Insert license notice

%% Check the input parameters for errors
assert(length(priori_prob) == L,strcat('Input parameter size mismatch.',
    ...
    'Please, make sure the vector priori_prob has L parameters'))

%% Initialize parameters
% Compute the number of samples in each cluster
values = round(N*priori_prob);

% Compute the total number of samples in the generated clusters
N_new = sum(values);

% Preallocate mean and variance matrices
```

```
matriu_mitjes = zeros(d,L);
matriu_sigma = zeros(d,d,L);

% The last row contains labeling information
DB = zeros(d +1,N_new);

% Initialize the range of the clusters' mean
a = -10;
b = 10;

%% Compute cluster vectors
index = 1;
for i=1:L
    % Compute means of the current cluster
    matriu_mitjes(:,i) = (b-a).*rand(d,1) + a;
    % Compute variances of the current cluster
    matriu_sigma(:,:,i) = diag(rand(d,1));
    % Get random values for the current cluster
    DB(1:d, index:index+values(i)-1) = ...
        mvnrnd(matriu_mitjes(:,i), matriu_sigma(:,:,i),values(i))';
    % Save "label" of current cluster
    DB(d+1, index:index+values(i)-1) = i;
    % Increase the index (it is used to access the columns in the DB that
    % correspond to a single cluster
    index = index+values(i);
end

% Shuffle database columns
DB=DB(:,randperm(length(DB)));
end
```

../src/CLP_Generate.m

4.5. CLP_Kmeans

```
function [Centroids, Labels, n, J, tr1, tr2, Sw, Sb] = CLP_Kmeans(DB, K,
    d, th)
%CLP_Kmeans Classify matrix with a K-Means algorithm
%
%   [ Centroids, Labels, n, J, tr_1, tr_2, Sw, Sb ] = CLP_Kmeans(DB, K, d
    )
%
%   Detailed explanation goes here

%% Initialize centroids and labels matrices

Centroids = datasample(DB, K, 2, 'Replace', false);

%% Classify database
% threshold = 0.0005;

Labels = zeros(length(DB), 1);
n = 1;

J = zeros(50,1);


% Iterate while the cost function variates enough
condition = n <= 2;
while condition == 1

    J(n) = 0; % Cost function

    % Classify database
    for i = 1:length(DB)
        norms = sum(abs(repmat(...
            double(DB(:,i)), 1, K) - double(Centroids(:,:,end))).^2,1);
        [Minimum_value, Labels(i)] = min(norms);
        J(n) = Minimum_value + J(n);
    end

    % Update centroids
    for i = 1:K
        Centroids(:, i, n) = mean(double(DB(:, Labels==i)), 2);
    end

    if n > 1
        diff = J(n-1) - J(n);
        condition = (diff) > th;
    end

    n = n+1;
end

% Take actual values of J (eliminate the rests of preallocated data)
J = J(1:n-1);
```

```
%% Compute error metrics
% Within-cluster scatter matrix
Sw = zeros(d,d);
ni = zeros(1,K);

for i = 1:length(DB)
    Sw = Sw + (double(DB(:,i))-double(Centroids(:,Labels(i),end)))*...
        (double(DB(:,i))-double(Centroids(:,Labels(i),end)))';
    ni(Labels(i)) = ni(Labels(i)) + 1; % Add one sample to detected class
end

% Between-cluster scatter matrix
Sb = zeros(d,d);

for j = 1:K
    m = (1/length(DB))*ni*double(Centroids(:,:,end))';
    Sb = Sb + ni(j)*(double(Centroids(:,j,end))-m')*(double(Centroids(:,j
    ,end))-m')';
end

% Total scatter matrix
St = Sb+Sw;

% Trace metrics
tr1 = trace(St\Sw);
tr2 = trace(Sw\Sb);

end
```

../src/CLP_Kmeans.m