**Lab 11. Clustering**

# Objective

During this lab, students will reinforce concepts related to non-supervised learning and specifically the k-means algorithm. Students could program their own k-means algorithm or, alternatively, they could fill up the gaps that are in the code presented below.

At the end of the lab session, students will submit a pdf document explaining what snippet of code does, in a couple of lines, using the own student words. The snippets are the ones that the students have to introduce in the gaps, and they are placed at the end of this section, in the "Code" sub-section. The pdf will be named as *“StudentName\_clustering.pdf*”, and it will be upload using the task that is located in the agora.

# Recommended steps

* It is advised to **copy** first each function and the main script –that are at the end of this handout- into a different file**, giving to each file the same name as the function has**.
* Later, a good way to proceed is to *complete* and run the main script, MainScriptToRunKmeansISI, until the section 3.1 inclusive. This way the matrix "Dataset" will be available in the Matlab workspace and it will be possible to use it when programming the kmeans algorithm.
* After that, it is recommended to start programming or completing the kmeans algorithm. To do that, a good approach is to run each single cell (the code lines that are between two %%) seeing the results in the Matlab workspace.
* After finishing the k-means algorithm it will be possible to run the main script, "MainScriptToRunKmeansISI", to test the algorithm and to visualize the results from different *k* choices.

Using the main script, the student will carry out the following steps:

1. He or she will load three different sets of images that are characterized because they are from a dominant color, red, green or mixed. The images can be found in a zip file named "Color Images", in the agora.
2. The student will be able to visualize the loaded images.
3. For each image we are going to use its average RGB value as descriptor, so the next step will be to obtain the dataset to be classified computing the Red, Green a Blue average values for each image and storing this 3-elements vector in a matrix.
4. Using the previously programmed k-means algorithm, the student will try different values of K, the number of clusters, to see what happens with the dataset. It will be possible to visualize in a 3-D plot the position and color (class) of the dataset points.
5. Finally, after several test, it also will be possible to visualize the images classified as belonging to each specific class.

The student can copy-paste the code, if he or she want to, checking after that there are no errors in the function produced by the previous operation.

# Code

The gaps in yellow in the different functions can be filled out with one of the following snippets of code. They are not in order. Each line goes into a different gap but it could be more than one gap where the same snippet fits.

1. Size [7]
2. IndexMinimum [11]
3. Dataset(:,IndexRandomInitializationSamples(i)); [8]
4. mean(Dataset(:,IndicesClosestToK),2); [12]
5. fEuclideanDistVectToMatrix(Dataset(:,i),MuCentroidsMatrix); [9]
6. [~,IndexMinimum]=min(DistancesVector); [10]
7. fReadDirNamAndIma13 [1 to 3]
8. fEuclideanDistVectToVect(PreviousCentroids, CurrentCentroids); [13]
9. NewMuCentroidsMatrix; [14]
10. fCellImageToRGBVectors [4 to 6]

# 1. Script MainScriptToRunKmeansISI to execute the kmeans algorithm

%% Script MainScriptToRunKmeansISI for testing the kmeans clustering algorithm % ISI Lab. Universidad de Leon

% EAlegre April2013

%% 1) Reading images from disk

%% 1.1) Read the 3 images dataset

% A. "Read" images

[CImaNamRed,ThePathRed]= COMPLETE THIS PART! **[1]**... ('jpg','Select the folder with the Red images');

% CImaNamRed <2x35 cell>

% B. "Green" images

[CImaNamGreen,ThePathGreen]= COMPLETE THIS PART! **[2]** ('jpg','Select the folder with the Green images');

% CImaNamGreen <2x35 cell>

% C. "Mixed" images, with Red and Green colors

[CImaNamMixed,ThePathMixed]= COMPLETE THIS PART!. **[3]** ('jpg','Select the folder with the Mixed images');

% CImaNamMixed <2x10 cell>

%% 1.2) Visualize the images stored in the cell array

% A. Visualize the RED images fVisualizeImagesInCell(CImaNamRed);

% B. Visualize the GREEN images fVisualizeImagesInCell(CImaNamGreen);

% C. Visualize the MIXED images fVisualizeImagesInCell(CImaNamMixed);

% To visualize any image, change the number of image that you want to

% visualize, in the cell containing the colour images and uncomment these % lines

% NumIma=5;

% figure, imshow(CImaNamMixed{1,NumIma}), title(numb2str(NumIma))

%% 2) Extract the feature vectors from the images

% Computes the average value for each colour plane in every image stored in

% the cell array. The result is a 3-elements vector with the R, G, and B % average colour

RGBMeansMatrixRed= COMPLETE THIS PART! **[4]** (CImaNamRed);

RGBMeansMatrixGreen= COMPLETE THIS PART! **[5]** (CImaNamGreen);

RGBMeansMatrixMixed= COMPLETE THIS PART! **[6]** (CImaNamMixed);

%% 3) Classify the red and green images, without knowing the labels, **in 2**

**% clusters**, using 2-means (kmeans with k=2)

% 3.1) Create the dataset

Dataset=[RGBMeansMatrixRed,RGBMeansMatrixGreen];

K=2;

% 3.2) Classify the vectors in the dataset using kmeans

VisualizeIterationsEvol=1; % Change to 0 if you do not want to check how % the centroids evolve

[ClosestCentroidToEachSample,Centroids ] =...

fKmeansISI(Dataset, K, VisualizeIterationsEvol);

% 3.3) Visualize the point's distribution in the space fPlotDataPoints(Dataset, Centroids,ClosestCentroidToEachSample,K,...

strcat('Red and Green, K=',num2str(K)))

%% 4) Classify the red and green images, without knowing the labels**, in 5**

**% clusters,** using 5-means (kmeans with k=5)

% 4.1) Create the dataset

Dataset=[RGBMeansMatrixRed,RGBMeansMatrixGreen];

K=5;

% 4.2) Classify the vectors in the dataset using kmeans

VisualizeIterationsEvol=1; % Change to 0 if you do not want to check how % the centroids evolve

[ ClosestCentroidToEachSample,Centroids ] =...

fKmeansISI(Dataset, K, VisualizeIterationsEvol);

% 4.3) Visualize the point's distribution in the space fPlotDataPoints(Dataset, Centroids,ClosestCentroidToEachSample,K,...

strcat('Red and Green, K=',num2str(K)))

%% 5) Classify the red, green and mixed images, without knowing the labels,

% **in 3 clusters,** using 3-means (kmeans with K=3)

% 5.1) Create the dataset

Dataset=[RGBMeansMatrixRed,RGBMeansMatrixGreen,RGBMeansMatrixMixed];

K=3;

% 5.2) Classify the vectors in the dataset using kmeans

VisualizeIterationsEvol=1;

[ClosestCentroidToEachSample,Centroids ] =...

fKmeansISI(Dataset, K, VisualizeIterationsEvol);

% 5.3) Visualize the point's distribution in the space

fPlotDataPoints(Dataset, Centroids,ClosestCentroidToEachSample,K,...

strcat('Red, Green and Mixed, K=',num2str(K)))

%% 6) Classify the red, green and mixed images, without knowing the labels,

% **in 6 clusters,** using 6-means (kmeans with K=6)

% 6.1) Create the dataset

Dataset=[RGBMeansMatrixRed,RGBMeansMatrixGreen,RGBMeansMatrixMixed];

K=6;

% 6.2) Classify the vectors in the dataset using kmeans

VisualizeIterationsEvol=1;

[ClosestCentroidToEachSample,Centroids ] =...

fKmeansISI(Dataset, K, VisualizeIterationsEvol);

% 6.3) Visualize the point's distribution in the space

fPlotDataPoints(Dataset, Centroids,ClosestCentroidToEachSample,K,... strcat('Red, Green and Mixed, K=',num2str(K)))

%% 7) Visualize image in Clusters

% 7.1) Compose the Cell Arrays containing the images

CImaNamAllImages=[CImaNamRed,CImaNamGreen,CImaNamMixed];

% 7.2) Index of the images in each cluster NumSamples=length(ClosestCentroidToEachSample); VIndexCluster=zeros(K,NumSamples); for i=1:K, % K=6 because the previous settings VIndexCluster(i,:)=ClosestCentroidToEachSample==i; end

VIndexCluster=logical(VIndexCluster);

% 7.3) Visualize all the images in each subcluster for i=1:K,

fVisualizeImagesInCell(CImaNamAllImages(:,VIndexCluster(i,:))); end

# 2. Programming the kmeans algorithm

Program your own kmeans algorithm version or fill the gaps in the one presented below, completing the sections highlighted in yellow:

0. Errors. Dimension of the number of samples

1.2. Introduction of random centroids in a matrix (centroid initialization)

2.1 Compute the distances between each sample and each centroid

1. Average of the values in previous clusters
2. Distance between centroids in an iteration and the previous ones.

4. Updating the old centroids

function [ClosestCentroidToEachSample,NewMuCentroidsMatrix] =...

fKmeansISI(Dataset,K,VisualizeIterationsEvol) % [ClosestCentroidToEachSample,NewMuCentroidsMatrix] =...

% fKmeansISI(Dataset, K, VisualizeIterationsEvol)

%

% This function uses the kmeans algorithm to return the cluster "K" to

% which each sample belongs to in the ClosestCentroidToEachSample vector. %

% INPUT:

% - Dataset: Matrix containing the Feature vectors of the samples to % be clustered.

% - K: Number of clusters

% - VisualizeIterationsEvol: Flag. If it is true, values as the numbers % of iterations employed, the distance between vectors and the old % and new centroids are visualized.

% OUTPUT:

% - ClosestCentroidToEachSample: vector containing the label -is a

% number- of the closest centroid to every specific sample.

% - NewMuCentroidsMatrix: It is a matrix of size "number of

% characteristics in the feature vector" x "number of Ks", containing % the values of the "number of K" centroids computed in the % algorithm.

%

% EAlegre April2013 %

%% Algorithm

% k=number of clusters

% Training set, {x(1), x(2), ..., x(m)} with a number "m" of samples

% 1. Randomly initialize K cluster centroids mu1, mu2,..., muk,

% belonging to Rn

%

% Repeat{

%

% 2. Assign closest centroid to each sample

% for i=1 to m,

% Centroids(i)= index (from 1 to k) of cluster centroid closest to

% x(i)

%

% 3. Recompute centroids

% for k=1 to K

% muk=average (mean) of points assigned to cluster k

% } % Until there are not changes in the centroids

%% 0. Errors

% Check if the number of clusters is smaller than the number of samples

DimensionsDataset= COMPLETE THIS PART! **[7]** (Dataset);

NumberOfSamplesM=max(DimensionsDataset);

SizeFeatureVector=min(DimensionsDataset);

%%

if K>NumberOfSamplesM,

sprintf('Error. Number of cluster bigger than number of samples') return

end

%% 1. Randomly initialize K cluster centroids

% 1.1. Generate K pseudo-random numbers % Preallocation for speed

IndexRandomInitializationSamples=zeros(1,K);

MuCentroidsMatrix=zeros(SizeFeatureVector,K);

NewMuCentroidsMatrix=zeros(SizeFeatureVector,K);

ClosestCentroidToEachSample=zeros(1,NumberOfSamplesM);

for i=1:K,

RandomNumbers=rand(1,K,'single'); IndexRandomInitializationSamples(i)=...

floor(NumberOfSamplesM\*RandomNumbers(i)); end

% The first centroids are the previously initialized

% 1.2. Introduce the centroids in a matrix

for i=1:K,

MuCentroidsMatrix(:,i)= COMPLETE THIS PART! **[8]** end

%% REPEAT THE FOLLOWING UNTIL THE CENTROIDS BE THE SAME ONES

DistanceBetweenOldAndNews=ones(1,K);

NumberOfIterationsNeeded=0;

while all(DistanceBetweenOldAndNews), % While distance is not 0

% Just to know how many iterations were needed

NumberOfIterationsNeeded=NumberOfIterationsNeeded+1;

%% 2. Assign centroids to samples

% 2.1 Compute the distances between each sample and each centroid

% and assigns the closest centroid to each sample

% (if the Statistic Toolbox is available in your system, an easiest way

% is to use the "pdist" function)

for i=1:NumberOfSamplesM,

% Compute the distance of each sample to each Centroid

DistancesVector= COMPLETE THIS PART! **[9]**

% Obtain the index of the minimum Centroid

COMPLETE THIS PART! **[10]**

% Assign the sample to the closest Centroid

ClosestCentroidToEachSample(i)= COMPLETE THIS PART! **[11]** end

%% 3. Obtain new centroids by averaging the values on each previous cluster % for i=1:K,

IndicesClosestToK= ClosestCentroidToEachSample==i;

NewMuCentroidsMatrix(:,i)= COMPLETE THIS PART! **[12]**;

end

%% 4. Check if the centroids are the same as in the previous iteration

% If they are the same, the distance between the centroids will be 0 %

% Convert the centroid matrix in a single vector

PreviousCentroids=reshape(MuCentroidsMatrix,1,[]);

CurrentCentroids=reshape(NewMuCentroidsMatrix,1,[]);

DistanceBetweenOldAndNews= COMPLETE THIS PART! **[13]**

%% Update the old centroids with the new ones

if VisualizeIterationsEvol,

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% If VisualizeIterationsEvol is true it will be possible to see how the

% centroids and the distances evolve

%

NumberOfIterationsNeeded

DistancesVector

MuCentroidsMatrix NewMuCentroidsMatrix pause

sprintf('Press any key to continue iterating') end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

MuCentroidsMatrix= COMPLETE THIS PART! **[14]**;

end % while end % function

# 3. Rest of functions

## 3.1 Function fReadDirNamAndIma13

function [CImaNam,ThePath]=fReadDirNamAndIma13... (extension,message,defaultpath,varargin)

%% [CImaNam,ThePath]=fReadDirNamAndIma13...

% (extension,message, defaultpath))

%

% This function saves the images and their names in the CellArray CImaNam.

% The user is asked to choose a directory and all the images and their

% names with the extension provided as parameter are stored in the cell % array "CImaNam" and returned.

%

%

% INPUT:

% - extension: -char- that is the file extension of the files such as

% 'jpg', 'bmp', 'tif', etc. It has to be a Matlab supported format.

% - message: a string in case you could pass a message to be showed

% - defaultpath: the directory where de windows will be opened % % OUTPUT:

% - CImaNam: A cell array containing the images and theirs names

% - ThePath: The path chosen for the user

%

% EAlegre April2013

%% 0. Checking the input arguments

if nargin == 0, extension = 'jpg'; message = 'Where are the files?'; defaultpath=pwd; % Is the current directory

elseif nargin == 1,

message = 'Where are the files?'; defaultpath=pwd; % Is the current directory

elseif nargin == 2,

defaultpath=pwd; % Is the current directory end

%% 1. Getting directory name where the images are stored

% Display standard dialog box for selecting a directory ........

%% try

ThePath = uigetdir... (defaultpath,message); catch MessError beep; sprintf('Error selecting directory') sprintf(MessError.message) end

%% 2. Saving in "CNames" the name of every image with "extension" extension,

% Put contents of the directory selected in struct

SNamImages=dir(ThePath);

% Checking if the extension has a dot NSizeExtension=size(extension,2); if NSizeExtension==3, extension=strcat('.',extension); elseif or(... or(NSizeExtension<3,NSizeExtension>4),... and(NSizeExtension==4,~strcmp(extension(1),'.'))) disp('Error de longitud de la extension. Introduzca 3 caracteres sin punto.') return end

%%

NNumFiles=size(SNamImages,1);

% Preallocation for speed

CNames=cell(1,NNumFiles);

j=0; for i=1:NNumFiles,

if size(SNamImages(i).name,2)>4,

if strcmpi(SNamImages(i).name(end-3:end),extension), j=j+1;

CNames{j}=SNamImages(i).name; end end end %

if isempty(SNamImages)

disp('There are not files with that extension') return; end

% Removing empty cells in the last positions NNames=size(CNames,2); while isempty(CNames{NNames}), CNames=CNames(1,1:NNames-1);

NNames=NNames-1; end

%% 3. Reading the images which names are in CNames

% Loading the images into the cell ........

% Returns the number of images read

% Displaying the progress bar ... h = waitbar(0,'Reading images ...');

numimag=size(CNames,2); % preallocation for speed

CImaNam=cell(numimag,2);

%%

for i=1:numimag

% obtains the name of each image thename=CNames{i}; % Reading every image

image=imread(fullfile(ThePath,thename));

% Saving image and its name into cell

CImaNam{i,1}=image;

CImaNam{i,2}=thename;

waitbar(i/numimag) end close(h) CImaNam=CImaNam';

**3.2 Function fVisualizeImagesInCell** function fVisualizeImagesInCell(CImaNam,NumColumsPlot,NumRowsPlot,varargin)

% fVisualizeImagesInCell(CImaNam,NumColumsPlot,NumRowsPlot)

% Visualiza the images contained in a cell array showing them in a subplot.

% The user can introduce the number of colums and the number of rows of the

% plot. If he or she do not introduce those values, the plot will be a

% matrix of images with 4 colums and as many rows as necessary to visualize

% all the images in the cell array %

% INPUT:

% CImaNam: cell array contaning the images to visualize in its first row

% NewFigure: if it is true, create

% NumColumsPlot: Number of colums of the visualization matrix

% NumRowsPlot: Number of rows of the visualization matrix

%

% EAlegre April2013

%% 0. Checking the input arguments if nargin == 0,

sprintf ('ERROR: You must introduce a cell array containing images') return elseif nargin == 1,

NumOfImages=length(CImaNam);

NumColumsPlot=4;

NumRowsPlot=ceil(NumOfImages/NumColumsPlot);

elseif nargin == 2,

NumOfImages=length(CImaNam);

NumRowsPlot=ceil(NumOfImages/NumColumsPlot); end

%% Creating the figure and the subplot

% Figure h=figure; set(h,'Name','Images to visualize') % Set the name of the figure

% Show images for i=1:NumOfImages, subplot(NumColumsPlot,NumRowsPlot,i); imshow(CImaNam{1,i}); title(num2str(i)); end % for end

**3.3 Function fCellImageToRGBVectors** function RGBMeansMatrix=fCellImageToRGBVectors(CImaNamColor)

% RGBMeansMatrix=fCellImageToRGBVectors(CImaNamColor)

% This function expects a cell array containing in its first row the images

% to be processed. It takes each image and returns a 3-dimensions vector

% for each one. The vector's elements are the average value of the pixel % level for each color plane, R (Red), G (Green) and B (Blue) % INPUT:

% CImaNamColor: cell array contanining in its first row the images %

% OUTPUT:

% RGBMeansMatrix: A Matrix containing the RGB average color for each

% image in the input cell array

%

% EAlegre April2013

% Number of images in the cell array to be processed

NumImages=length(CImaNamColor);

% Preallocation for speed

RGBMeansMatrix=zeros(3,NumImages);

% Obtains the RGB average vector for each image and stores it in the

% RGBMeansMatrix

for i=1:NumImages,

RGBMeansMatrix(1,i)=mean(mean(im2double(CImaNamColor{1,i}(:,:,1))));

RGBMeansMatrix(2,i)=mean(mean(im2double(CImaNamColor{1,i}(:,:,2))));

RGBMeansMatrix(3,i)=mean(mean(im2double(CImaNamColor{1,i}(:,:,3))));

end % for end

**3.4 Function fPlotDataPoints** function fPlotDataPoints(X,Centroids,idx,K,TitleOfFigure) % fPlotDataPoints(X,Centroids, idx, K, TitleOfFigure)

% Plots data points in X, coloring them so that those with

% the same index assignments in idx have the same color % % INPUT:

% - X: Matrix containing 3-dimensional vectors (vectors to visualize)

% - Centroids: Values of the cluster's centroids

% - idx: index indicating at which cluster belongs to each sample

% - K: Number of clusters

% - TitleOfFigure: Text that will be plotted as title in the figure %

% EAlegre April2013

%% Create palette palette = hsv(K + 1); colors = palette(idx,:);

%% Plot the data figure,

scatter3(X(1,:), X(2,:), X(3,:),30,colors); hold on, scatter3(Centroids(1,:), Centroids(2,:), Centroids(3,:),250,hsv(K),'s','fill'); title (TitleOfFigure) end

## 3.5 Function fEuclideanDistVectToMatrix

function DistancesVector= fEuclideanDistVectToMatrix(Vector, Matrix) % Returns a row vector containing the Euclidean distance between the Vector % and each element of the matrix "Matrix"

% Both must have the same number of rows, corresponding with the features % of each sample. % % INPUT:

% - Vector: Column vector whose distance to the elements of x is going to % be computed.

% - Matrix: Matrix containing the dataset.

% It has as many rows as features per sample and as many % columns as samples. %

% OUTPUT:

% -DistancesVector: a row vector with the distances between Vector and

% each element in Matrix %

% EAlegre April2013

%% 1. Number of colums of the Matrix (containing the dataset)

NumberOfColums = size(Matrix,2);

%% 2. Replicating the rows of the Vector as many times as colums are in Matrix % This way, it is easiest to operate

Vector = repmat(Vector, 1, NumberOfColums);

%% 3. Computing the Euclidean distance

DistancesVector = sqrt(sum((Vector-Matrix).^2, 1));

end

**3.6 Function fEuclideanDistVectToVect** function DistanceValue= fEuclideanDistVectToVect(FirstVector, SecondVector) % Returns the value correponging with the Euclidean distance between two % vectors.

% Both vectors must have the same number of elements % % INPUT:

% - FirstVector: The first vector.

% - SecondVector: The second one %

% OUTPUT:

% -DistancesValue: the distance between both of them

%

% EAlegre April2013

%% 1. Check if both inputs are vectors

if ~and(isvector(FirstVector),isvector(SecondVector)) % The "if" returs 1 when both are vectors

% As the result of the condition is the NOT AND,... % The function finishes if any of them is not a vector

sprintf('ERROR in fEuclideanDistVectToVect: Both inputs must be a vector') return

end

%% 2. Checking the shape and number of elements in each vector

% Dimensions of each vector

DimFirstVec=size(FirstVector);

DimSecondVec=size(SecondVector);

% Check if it is a column vector and if it not, transpose it

if DimFirstVec(1)==1 % is a row vector FirstVector=FirstVector'; % transpose it end if DimSecondVec(1)==1 % is a row vector SecondVector=SecondVector'; % transpose it end

% Check if both vectors have the same length

if ~length(FirstVector)==length(SecondVector), % if they have not same length sprintf...

('ERROR in fEuclideanDistVectToVect: Vectors have different lengths') return end

%% 3. Computing the Euclidean distance

DistanceValue = sqrt(sum((FirstVector-SecondVector).^2, 1));

end

**% ============================================================**

%%

%% 3.3 Function fCellImageToRGBVectors

function RGBMeansMatrix=fCellImageToRGBVectors(CImaNamColor)

% RGBMeansMatrix=fCellImageToRGBVectors(CImaNamColor)

% This function expects a cell array containing in its first row the images

% to be processed. It takes each image and returns a 3-dimensions vector

% for each one. The vector's elements are the average value of the pixel

% level for each color plane, R (Red), G (Green) and B (Blue)

% INPUT:

% CImaNamColor: cell array contanining in its first row the images

%

% OUTPUT:

% RGBMeansMatrix: A Matrix containing the RGB average color for each

% image in the input cell array

%

% EAlegre April2013

% Number of images in the cell array to be processed

NumImages=length(CImaNamColor);

% Preallocation for speed

RGBMeansMatrix=zeros(3,NumImages);

% Obtains the RGB average vector for each image and stores it in the

% RGBMeansMatrix

for i=1:NumImages,

RGBMeansMatrix(1,i)=mean(mean(im2double(CImaNamColor{1,i}(:,:,1))));

RGBMeansMatrix(2,i)=mean(mean(im2double(CImaNamColor{1,i}(:,:,2))));

RGBMeansMatrix(3,i)=mean(mean(im2double(CImaNamColor{1,i}(:,:,3))));

end % for

end

**% ============================================================**

**% ============================================================**

% 3.5 Function fEuclideanDistVectToMatrix

function DistancesVector= fEuclideanDistVectToMatrix(Vector, Matrix)

% Returns a row vector containing the Euclidean distance between the Vector

% and each element of the matrix "Matrix"

% Both must have the same number of rows, corresponding with the features

% of each sample. %

% INPUT:

% - Vector: Column vector whose distance to the elements of x is going to be computed.

% - Matrix: Matrix containing the dataset.

% It has as many rows as features per sample and as many columns as samples.

%

%

%

%

%

%

% OUTPUT:

% -DistancesVector: a row vector with the distances between Vector and each element in Matrix

% % % %

NumberOfColums = size(Matrix,2);

%% 2. Replicating the rows of the Vector as many times as colums are in Matrix % This way, it is easiest to operate

Vector = repmat(Vector, 1, NumberOfColums);

%% 3. Computing the Euclidean distance

DistancesVector = sqrt(sum((Vector-Matrix).^2, 1));

end

**% ============================================================**

**% ============================================================**

%%

%% 3.6 Function fEuclideanDistVectToVect

function DistanceValue= fEuclideanDistVectToVect(FirstVector, SecondVector)

% Returns the value correponging with the Euclidean distance between two

% vectors.

% Both vectors must have the same number of elements

%

% INPUT:

% - FirstVector: The first vector.

% - SecondVector: The second one.

%

% OUTPUT:

% -DistancesValue: the distance between both of them

%

% EAlegre April2013

%% 1. Check if both inputs are vectors

if ~and(isvector(FirstVector),isvector(SecondVector))

% The "if" returs 1 when both are vectors

% As the result of the condition is the NOT AND,...

% The function finishes if any of them is not a vector

sprintf('ERROR in fEuclideanDistVectToVect: Both inputs must be a vector')

return

end

%% 2. Checking the shape and number of elements in each vector

% Dimensions of each vector

DimFirstVec=size(FirstVector);

DimSecondVec=size(SecondVector);

% Check if it is a column vector and if it not, transpose it

if DimFirstVec(1)==1 % is a row vector

FirstVector=FirstVector'; % transpose it

end

if DimSecondVec(1)==1 % is a row vector

SecondVector=SecondVector'; % transpose it

end

% Check if both vectors have the same length

if ~length(FirstVector)==length(SecondVector), % if they have not same length

sprintf...

('ERROR in fEuclideanDistVectToVect: Vectors have different lengths')

return

end

%% 3. Computing the Euclidean distance

DistanceValue = sqrt(sum((FirstVector-SecondVector).^2, 1));

end

**% ============================================================**

**% ============================================================**

function [ClosestCentroidToEachSample,NewMuCentroidsMatrix] =...

fKmeansISI(Dataset,K,VisualizeIterationsEvol)

% [ClosestCentroidToEachSample,NewMuCentroidsMatrix] =...

% fKmeansISI(Dataset, K, VisualizeIterationsEvol)

%

% This function uses the kmeans algorithm to return the cluster "K" to

% which each sample belongs to in the ClosestCentroidToEachSample vector.

%

% INPUT:

% - Dataset: Matrix containing the Feature vectors of the samples to

% be clustered.

% - K: Number of clusters

% - VisualizeIterationsEvol: Flag. If it is true, values as the numbers

% of iterations employed, the distance between vectors and the old

% and new centroids are visualized.

% OUTPUT:

% - ClosestCentroidToEachSample: vector containing the label -is a

% number- of the closest centroid to every specific sample.

% - NewMuCentroidsMatrix: It is a matrix of size "number of

% characteristics in the feature vector" x "number of Ks", containing

% the values of the "number of K" centroids computed in the

% algorithm.

%

% EAlegre April2013

%

%% Algorithm

% k=number of clusters

% Training set, {x(1), x(2), ..., x(m)} with a number "m" of samples

% 1. Randomly initialize K cluster centroids mu1, mu2,..., muk,

% belonging to Rn

%

% Repeat{

%

% 2. Assign closest centroid to each sample

% for i=1 to m,

% Centroids(i)= index (from 1 to k) of cluster centroid closest to

% x(i)

%

% 3. Recompute centroids

% for k=1 to K

% muk=average (mean) of points assigned to cluster k

% } % Until there are not changes in the centroids

%% 0. Errors

% Check if the number of clusters is smaller than the number of samples

DimensionsDataset= size(Dataset);

NumberOfSamplesM=max(DimensionsDataset);

SizeFeatureVector=min(DimensionsDataset);

%%

if K>NumberOfSamplesM

sprintf('Error. Number of cluster bigger than number of samples')

return

end

%% 1. Randomly initialize K cluster centroids

% 1.1. Generate K pseudo-random numbers

% Preallocation for speed

IndexRandomInitializationSamples=zeros(1,K);

MuCentroidsMatrix=zeros(SizeFeatureVector,K);

NewMuCentroidsMatrix=zeros(SizeFeatureVector,K);

ClosestCentroidToEachSample=zeros(1,NumberOfSamplesM);

for i=1:K

RandomNumbers=rand(1,K,'single');

IndexRandomInitializationSamples(i)=...

floor(NumberOfSamplesM\*RandomNumbers(i));

end

% The first centroids are the previously initialized

% 1.2. Introduce the centroids in a matrix

MuCentroidsMatrix = Dataset(:,IndexRandomInitializationSamples);

%% REPEAT THE FOLLOWING UNTIL THE CENTROIDS BE THE SAME ONES

DistanceBetweenOldAndNews=ones(1,K);

NumberOfIterationsNeeded=0;

while all(DistanceBetweenOldAndNews), % While distance is not 0

% Just to know how many iterations were needed

NumberOfIterationsNeeded=NumberOfIterationsNeeded+1;

%% 2. Assign centroids to samples

% 2.1 Compute the distances between each sample and each centroid

% and assigns the closest centroid to each sample

% (if the Statistic Toolbox is available in your system, an easiest way

% is to use the "pdist" function)

for i=1:NumberOfSamplesM,

% Compute the distance of each sample to each Centroid

DistancesVector= fEuclideanDistVectToMatrix(Dataset(:,i), MuCentroidsMatrix);

% Obtain the index of the minimum Centroid

index = find(DistancesVector == min(DistancesVector));

% Assign the sample to the closest Centroid

ClosestCentroidToEachSample(i)= index;

end

%% 3. Obtain new centroids by averaging the values on each previous cluster

%

for i=1:K

IndicesClosestToK= ClosestCentroidToEachSample==i;

NewMuCentroidsMatrix(:,i)= mean(Dataset(:,IndicesClosestToK),2);

end

%% 4. Check if the centroids are the same as in the previous iteration

% If they are the same, the distance between the centroids will be 0

%

% Convert the centroid matrix in a single vector

PreviousCentroids=reshape(MuCentroidsMatrix,1,[]);

CurrentCentroids=reshape(NewMuCentroidsMatrix,1,[]);

DistanceBetweenOldAndNews= fEuclideanDistVectToVect(PreviousCentroids, CurrentCentroids);

%% Update the old centroids with the new ones

if VisualizeIterationsEvol,

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% If VisualizeIterationsEvol is true it will be possible to see how the

% centroids and the distances evolve

%

NumberOfIterationsNeeded

DistancesVector

MuCentroidsMatrix

NewMuCentroidsMatrix

% pause

% sprintf('Press any key to continue iterating')

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

MuCentroidsMatrix = NewMuCentroidsMatrix;

end % while

end % function

**% ============================================================**

**% ============================================================**

%%

%% 3.4 Function fPlotDataPoints

function fPlotDataPoints(X,Centroids,idx,K,TitleOfFigure)

% fPlotDataPoints(X,Centroids, idx, K, TitleOfFigure)

% Plots data points in X, coloring them so that those with

% the same index assignments in idx have the same color

%

% INPUT:

% - X: Matrix containing 3-dimensional vectors (vectors to visualize)

% - Centroids: Values of the cluster's centroids

% - idx: index indicating at which cluster belongs to each sample

% - K: Number of clusters

% - TitleOfFigure: Text that will be plotted as title in the figure

%

% EAlegre April2013

%% Create palette

palette = hsv(K + 1);

colors = palette(idx,:);

%% Plot the data

figure,

scatter3(X(1,:), X(2,:), X(3,:),30,colors);

hold on,

scatter3(Centroids(1,:), Centroids(2,:), Centroids(3,:),250,hsv(K),'s','fill');

title (TitleOfFigure)

end

**% ============================================================**

**% ============================================================**

% ALL THE REST Functions. Kmeans Lab

%% 3.1. fReadDirNamAndIma13

function [CImaNam,ThePath]=fReadDirNamAndIma13...

(extension,message,defaultpath,varargin)

%% [CImaNam,ThePath]=fReadDirNamAndIma13...

% (extension,message, defaultpath))

%

% This function saves the images and their names in the CellArray CImaNam.

% The user is asked to choose a directory and all the images and their

% names with the extension provided as parameter are stored in the cell

% array "CImaNam" and returned.

%

%

% INPUT:

% - extension: -char- that is the file extension of the files such as

% 'jpg', 'bmp', 'tif', etc. It has to be a Matlab supported format.

% - message: a string in case you could pass a message to be showed

% - defaultpath: the directory where de windows will be opened

%

% OUTPUT:

% - CImaNam: A cell array containing the images and theirs names

% - ThePath: The path chosen for the user

%

% EAlegre April2013

%% 0. Checking the input arguments

if nargin == 0

extension = 'jpg';

message = 'Where are the files?';

defaultpath=pwd; % Is the current directory

elseif nargin == 1

message = 'Where are the files?';

defaultpath=pwd; % Is the current directory

elseif nargin == 2

defaultpath=pwd; % Is the current directory

end

%% 1. Getting directory name where the images are stored

% Display standard dialog box for selecting a directory ........

%%

try

ThePath = uigetdir...

(defaultpath,message);

catch MessError

beep;

sprintf('Error selecting directory')

sprintf(MessError.message)

end

%% 2. Saving in "CNames" the name of every image with "extension" extension,

% Put contents of the directory selected in struct

SNamImages=dir(ThePath);

% Checking if the extension has a dot

NSizeExtension=size(extension,2);

if NSizeExtension==3,

extension=strcat('.',extension);

elseif or(...

or(NSizeExtension<3,NSizeExtension>4),...

and(NSizeExtension==4,~strcmp(extension(1),'.')))

disp('Error de longitud de la extension. Introduzca 3 caracteres sin punto.')

return

end

%%

NNumFiles=size(SNamImages,1);

% Preallocation for speed

CNames=cell(1,NNumFiles);

j=0;

for i=1:NNumFiles,

if size(SNamImages(i).name,2)>4,

if strcmpi(SNamImages(i).name(end-3:end),extension),

j=j+1;

CNames{j}=SNamImages(i).name;

end

end

end

%

if isempty(SNamImages)

disp('There are not files with that extension')

return;

end

% Removing empty cells in the last positions

NNames=size(CNames,2);

while isempty(CNames{NNames}),

CNames=CNames(1,1:NNames-1);

NNames=NNames-1;

end

%% 3. Reading the images which names are in CNames

% Loading the images into the cell ........

% Returns the number of images read

% Displaying the progress bar ...

h = waitbar(0,'Reading images ...');

numimag=size(CNames,2);

% preallocation for speed

CImaNam=cell(numimag,2);

%%

for i=1:numimag

% obtains the name of each image

thename=CNames{i};

% Reading every image

image=imread(fullfile(ThePath,thename));

% Saving image and its name into cell

CImaNam{i,1}=image;

CImaNam{i,2}=thename;

waitbar(i/numimag)

end

close(h)

CImaNam=CImaNam';

end

**% ============================================================**

**% ============================================================**

%% 3.2. fVisualizeImagesInCell

function fVisualizeImagesInCell(CImaNam,NumColumsPlot,NumRowsPlot,varargin)

% fVisualizeImagesInCell(CImaNam,NumColumsPlot,NumRowsPlot)

% Visualiza the images contained in a cell array showing them in a subplot.

% The user can introduce the number of colums and the number of rows of the

% plot. If he or she do not introduce those values, the plot will be a

% matrix of images with 4 colums and as many rows as necessary to visualize

% all the images in the cell array

%

% INPUT:

% CImaNam: cell array contaning the images to visualize in its first row

% NewFigure: if it is true, create

% NumColumsPlot: Number of colums of the visualization matrix

% NumRowsPlot: Number of rows of the visualization matrix

%

% EAlegre April2013

%% 0. Checking the input arguments

if nargin == 0,

sprintf ('ERROR: You must introduce a cell array containing images')

return

elseif nargin == 1,

NumOfImages=length(CImaNam);

NumColumsPlot=4;

NumRowsPlot=ceil(NumOfImages/NumColumsPlot);

elseif nargin == 2,

NumOfImages=length(CImaNam);

NumRowsPlot=ceil(NumOfImages/NumColumsPlot);

end

%% Creating the figure and the subplot

% Figure

h=figure;

set(h,'Name','Images to visualize') % Set the name of the figure

% Show images

for i=1:NumOfImages,

subplot(NumColumsPlot,NumRowsPlot,i);

imshow(CImaNam{1,i}); title(num2str(i));

end % for

end

**% ============================================================**

**% ============================================================**

%% Script MainScriptToRunKmeansISI for testing the kmeans clustering algorithm

% ISI Lab. Universidad de Leon

% EAlegre April2013

%% 1) Reading images from disk

%% 1.1) Read the 3 images dataset

% A. "Read" images

[CImaNamRed,ThePathRed]= fReadDirNamAndIma13...

('jpg','Select the folder with the Red images');

% CImaNamRed <2x35 cell>

% B. "Green" images

[CImaNamGreen,ThePathGreen]= fReadDirNamAndIma13...

('jpg','Select the folder with the Green images');

% CImaNamGreen <2x35 cell>

% C. "Mixed" images, with Red and Green colors

[CImaNamMixed,ThePathMixed]= fReadDirNamAndIma13...

('jpg','Select the folder with the Mixed images');

% CImaNamMixed <2x10 cell>

%% 1.2) Visualize the images stored in the cell array

% A. Visualize the RED images

fVisualizeImagesInCell(CImaNamRed);

% B. Visualize the GREEN images

fVisualizeImagesInCell(CImaNamGreen);

% C. Visualize the MIXED images

fVisualizeImagesInCell(CImaNamMixed);

% To visualize any image, change the number of image that you want to

% visualize, in the cell containing the colour images and uncomment these

% lines

% NumIma=5;

% figure, imshow(CImaNamMixed{1,NumIma}), title(numb2str(NumIma))

%% 2) Extract the feature vectors from the images

% Computes the average value for each colour plane in every image stored in

% the cell array. The result is a 3-elements vector with the R, G, and B

% average colour

RGBMeansMatrixRed= fCellImageToRGBVectors(CImaNamRed);

RGBMeansMatrixGreen= fCellImageToRGBVectors(CImaNamGreen);

RGBMeansMatrixMixed= fCellImageToRGBVectors(CImaNamMixed);

%% 3) Classify the red and green images, without knowing the labels, in 2

% clusters, using 2-means (kmeans with k=2)

% 3.1) Create the dataset

Dataset=[RGBMeansMatrixRed,RGBMeansMatrixGreen];

K=2;

% 3.2) Classify the vectors in the dataset using kmeans

VisualizeIterationsEvol=1; % Change to 0 if you do not want to check how

% the centroids evolve

[ClosestCentroidToEachSample,Centroids ] =...

fKmeansISI(Dataset, K, VisualizeIterationsEvol);

% 3.3) Visualize the point's distribution in the space

fPlotDataPoints(Dataset, Centroids,ClosestCentroidToEachSample,K,...

strcat('Red and Green, K=',num2str(K)))

pause

%% 4) Classify the red and green images, without knowing the labels, in 5

% clusters, using 5-means (kmeans with k=5)

% 4.1) Create the dataset

Dataset=[RGBMeansMatrixRed,RGBMeansMatrixGreen];

K=5;

% 4.2) Classify the vectors in the dataset using kmeans

VisualizeIterationsEvol=1; % Change to 0 if you do not want to check how

% the centroids evolve

[ ClosestCentroidToEachSample,Centroids ] =...

fKmeansISI(Dataset, K, VisualizeIterationsEvol);

% 4.3) Visualize the point's distribution in the space

fPlotDataPoints(Dataset, Centroids,ClosestCentroidToEachSample,K,...

strcat('Red and Green, K=',num2str(K)))

pause

%% 5) Classify the red, green and mixed images, without knowing the labels,

% in 3 clusters, using 3-means (kmeans with K=3)

% 5.1) Create the dataset

Dataset=[RGBMeansMatrixRed,RGBMeansMatrixGreen,RGBMeansMatrixMixed];

K=3;

% 5.2) Classify the vectors in the dataset using kmeans

VisualizeIterationsEvol=1;

[ClosestCentroidToEachSample,Centroids ] =...

fKmeansISI(Dataset, K, VisualizeIterationsEvol);

% 5.3) Visualize the point's distribution in the space

fPlotDataPoints(Dataset, Centroids,ClosestCentroidToEachSample,K,...

strcat('Red, Green and Mixed, K=',num2str(K)))

pause

%% 6) Classify the red, green and mixed images, without knowing the labels,

% in 6 clusters, using 6-means (kmeans with K=6)

% 6.1) Create the dataset

Dataset=[RGBMeansMatrixRed,RGBMeansMatrixGreen,RGBMeansMatrixMixed];

K=6;

% 6.2) Classify the vectors in the dataset using kmeans

VisualizeIterationsEvol=1;

[ClosestCentroidToEachSample,Centroids ] =...

fKmeansISI(Dataset, K, VisualizeIterationsEvol);

% 6.3) Visualize the point's distribution in the space

fPlotDataPoints(Dataset, Centroids,ClosestCentroidToEachSample,K,...

strcat('Red, Green and Mixed, K=',num2str(K)))

pause

%% 7) Visualize image in Clusters

% 7.1) Compose the Cell Arrays containing the images

CImaNamAllImages=[CImaNamRed,CImaNamGreen,CImaNamMixed];

% 7.2) Index of the images in each cluster

NumSamples=length(ClosestCentroidToEachSample);

VIndexCluster=zeros(K,NumSamples);

for i=1:K, % K=6 because the previous settings

VIndexCluster(i,:)=ClosestCentroidToEachSample==i;

end

VIndexCluster=logical(VIndexCluster);

% 7.3) Visualize all the images in each subcluster

for i=1:K,

fVisualizeImagesInCell(CImaNamAllImages(:,VIndexCluster(i,:)));

end

**% ============================================================**

**Lab 9. Clustering**

MainScriptToRunKmeansISI:

**Gap[1], [2], [3]:**

* **[CImaNamRed,ThePathRed] =  fReadDirNamAndIma13('jpg','Select the folder with the Mixed images','C:\Users\Irene\Desktop\Universidad\Segundo Curso\segundo semestre\ISI\Codigo\_practica9\_Color Images\Color Images\RedImages');**
* **[CImaNamGreen,ThePathGreen]= fReadDirNamAndIma13('jpg','Select the folder with the Mixed images','C:\Users\Irene\Desktop\Universidad\Segundo Curso\segundo semestre\ISI\Codigo\_practica9\_Color Images\Color Images\GreenImages');**
* **[CImaNamMixed,ThePathMixed]= fReadDirNamAndIma13('jpg','Select the folder with the Mixed images','C:\Users\Irene\Desktop\Universidad\Segundo Curso\segundo semestre\ISI\Codigo\_practica9\_Color Images\Color Images\MixedImages');**

Con esta función lee las imágenes y las añade en un cellarray. La función **fReadDirNamAndIma13,**  la cual recibe 3 parámetros: la extensión, un mensaje y el directorio donde se encuentra.

Por ello, la extensión en todas ellas será “jpg”, ya que se tratan dataset de imágenes.

El mensaje será “Selecciona la carpeta”.

Y el directorio será la ruta absoluta de cada carpeta.

**Gap[4], [5], [6]:**

**RGBMeansMatrixRed= fCellImageToRGBVectors(CImaNamRed);**

**RGBMeansMatrixGreen= fCellImageToRGBVectors(CImaNamGreen);**

**RGBMeansMatrixMixed= fCellImageToRGBVectors(CImaNamMixed);**

Para extraer los vectores de características de las imágenes, se creará un cell mediante la función **fCellImageToRGBVectors(CImaNamColour),**  donde CImaNamColour es un cell-array con las imágenes en la primera fila. Está funcion nos devuelve la media de color RGB de cada imagen.

fKmeansISI:

**GAP[7]:**

**DimensionsDataset= size(Dataset);**

Para guardar la dimensión de nuestro dataset usamos la función “size()” haciendo que reciba nuestro dataset.

**GAP[8]:**

**MuCentroidsMatrix(:,i)= Dataset(:,IndexRandomInitializationSamples(i));**

Creamos una matriz para guardar los primeros centroides, ya que después vamos a ir recalculándolos. Y para ello vamos a extraer los centroides del dataset, que se seleccionan de manera random y almacenandolo en IndexRandomInitializationSamples(i).

**GAP[9]:**

**DistancesVector= fEuclideanDistVectToMatrix(Dataset(:,i),MuCentroidsMatrix);**

Calculamos la distancia euclídea de cada una de las muestras a cada matriz de centroides, que habíamos rellenado en el GAP[8]. Por ello debemos usar **fEuclideanDistVectToMatrix** y no **fEuclideanDistVectToVect.**

**GAP[10]:**

**[~,IndexMinimum]=min(DistancesVector);**

Obtenemos el índice del centroide mínimo. Como tenemos una matriz con las distancias, buscamos el mínimo y mediante **[~,IndexMinimum],”~”** guarda el valor de ese mínimos y **“IndexMinimum ”** el indice.

**GAP[11]:**

**ClosestCentroidToEachSample(i)=IndexMinimum;**

Asignamos  la muestra, la mínima distancia calculada en el GAP[10],  al centroide más cercano.

**GAP[12]:**

**NewMuCentroidsMatrix(:,i)= mean(Dataset(:,IndicesClosestToK),2);**

Volvemos a calcular los centroides haciendo las medias y los guardamos en una nueva matriz.

**GAP[13]:**

**DistanceBetweenOldAndNews= fEuclideanDistVectToVect(PreviousCentroids, CurrentCentroids);**

Como en el GAP[12] hemos creado una nueva matriz de centroides, debemos usar **fEuclideanDistVectToVect,** para la distancia entre los viejos centroides y los nuevos.

**GAP[14]:**

**MuCentroidsMatrix= NewMuCentroidsMatrix;**

Actualizamos los nuevos centroides.