Machine Learning Programming
Assignment 2: K-Means Algorithm

Professors: Baptiste Busch, Aude Billard
Assistants: Laila El Hamamsy, Matthieu Dujany
and Victor Faraut
Contacts:
baptiste.busch@epfl.ch, aude.billard@epfl.ch
laila.elhamamsy@epfl.ch, matthieu.dujany@epfl.ch, victor.faraut@epfl.ch

Winter Semester 2018

Introduction

In this practical, you will code the K-Means algorithm in Matlab. You will then use the code for clustering on 2D datasets before applying it on higher dimensional datasets.

Submission Instructions

Deadline: October 30, 2018 @ 6pm. Assignments must be turned in by the deadline. 1pt will be removed for each day late. A day late starts one hour after the deadline.
Procedure: From the course Moodle webpage, the student should download and extract the .zip file named `TP2-KMeans-Assignment.zip` which contains the following files:

Part 1 - Algorithm	Part 2 - Metrics, Applications
`my_distance.m`	`my_metrics.m`
`kmeans_init.m`	`kmeans_eval.m`
`my_distX2Mu.m`	`my_f1measure.m`
`my_kmeans.m`	`test_image_metrics.m`
`compress_image.m`	`test_kmeans_digits.m`
`reconstruct_image.m`	
`test_kmeans_2d.m`	
`test_image_compression.m`	

Assignment Instructions

The assignment consists of implementing the blue colored MATLAB functions from scratch. These functions can be tested with the `test_*.m` scripts. These testing scripts depend on `ML_toolbox`, which is already installed on the Virtual Machines. In case you want to work on your own computer you will have to download it from: `https://github.com/epfl-lasa/ML_toolbox` and change the path in each scripts by modifying the following line.

```
>> addpath(genpath('path_to_ML_toolbox'))
```

Once you have tested your functions, you can submit them as a .zip file with the name: `TP2-KMEANS-Assignment-SCIPER.zip` on the submission link in the Moodle webpage. Your submission archive should contain ONLY the following:

```
🟥 TP2-KMeans-Assignment-SCIPER
 └── 🟦 Part1
 └── 🟦 Part2
```

DO NOT upload `ML_toolbox`, the `check_utils` directory or the `Datasets` directory.

1 Part 1: K-Means Algorithm

Clustering is a process of partitioning a set of data (or objects) in a set of meaningful subclasses, called clusters. The k-means algorithm is a widely used clustering technique that seeks to minimize the average squared distance between points in the same cluster (Figure 1). In other words, given a dataset $\mathbf{X} = \{\mathbf{x}^1, \dots, \mathbf{x}^M\}$ where $\mathbf{x}^i \in \mathbb{R}^N$ with M samples of N dimensions, we can describe it with K clusters, each represented by a centroid $\mu^k \in \mathbb{R}^N$, by minimizing the total squared distance between each point and its closest centroid with the following cost function,

$$J(\mu_1, \dots, \mu_k) = \sum_{k=1}^{K} \sum_{\mathbf{x}^i \in c_k} |\mathbf{x}^i - \mu^{\mathbf{k}}|^2 \tag{1}$$

where $c_k \in \{1, \dots, K\}$ is the cluster label. This is an NP-hard problem, however, the k-means algorithm provides a local search solution for (1) through an iterative procedure proposed in the 1980's by Stuart P. Lloyd [1].

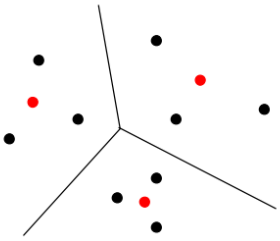

Figure 1: Illustration of k-means clustering. We have a dataset $\mathbf{X}$ with $N = 2$ and $M = 9$, by setting $K = 3$ we would like to estimate the red points corresponding to the $\mu = \{\mu_1, \mu_2, \mu_3\}$ and the decision boundaries describing the data partition (depicted by the black lines between clusters).

K-Means Algorithm

For a deeper description of K-Means and its applications, refer to the Clustering slides from the Applied Machine Learning Course. In this assignment, we will follow the steps of the k-Means algorithm shown on slide 25 of the class. Given a fixed number of K clusters we will go through the following steps:

1. Randomly **initialize** the centroids $\mu = \{\mu^1, \dots, \mu^K\}$
2. Calculate **distances** from each data point $\mathbf{x}^i$ to each centroid μ^k.
3. **Assignment Step:** For each cluster $c_k \in \{1, \dots, K\}$, set the cluster C_k to be the set of points in $\mathbf{X}$ that are closer to μ^k than any other cluster centroid.
4. **Update Step:** For each cluster $c_k \in \{1, \dots, K\}$, set μ^k to be the centroid of all points assigned to C_k : $\mu^k = \frac{1}{|C_k|} \sum_{\mathbf{x} \in C_k} \mathbf{x}$
5. Go back to step 2 and repeat until the cluster centroids μ^k stabilize (i.e. error between current and previous estimates of μ^k are below a threshold `thres=1e-4`) or the maximum number of iterations (`MaxIter`) is reached.

Interesting Properties of K-Means

- There are always K clusters.
- The clusters do not overlap.
- Each member of a cluster is closer to its cluster than to any other cluster.

1.1 Distances in Clustering

Many clustering algorithms (specifically k-means) rely on a metric or "distance" function between the data points. While one typically uses the Euclidean distance as a metric in $\mathbb{R}^N$, it actually comes from a more general class of metrics, i.e. the Minkowski metric:

$$L_p(\mathbf{x}, \mathbf{x}') = \left(\textstyle\sum_{i=1}^{N} |x_i - x_i'|^p\right)^{1/p},$$

otherwise known as the L_P norm. From this norm, we can recover **three** important distances used in the machine learning applications. For example, when $p = 1$ we recover the L_1 norm, also known as the Manhattan distance,

$$d_{L_1}(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||_1 = \sum_{i=1}^{N} |x_i - x_i'|, \tag{2}$$

this distance gives the shortest path between $\mathbf{x}$ and $\mathbf{x}'$ on a grid aligned to the data's coordinate system, also called the grid or rectilinear distance. When $p = 2$ we get the L_2 norm otherwise known as the Euclidean distance, the most common metric in literature for $\mathbb{R}^N$:

$$d_{L_2}(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||_2 = \sqrt{\sum_{i=1}^{N} |x_i - x_i'|^2}. \tag{3}$$

The Euclidean distance measures the distance between two points as the length of a straight line between them. Finally, $p = \infty$ yields the L_∞ norm

$$d_{L_\infty}(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||_\infty = \max_i |x_i - x_i'|. \tag{4}$$

it measures the maximum distance between dimensions.

TASK 1: Implement my_distance.m function (3pts)
The next task is to implement the `my_distance.m` function which includes Eq. 2, 3 and 4. The function should compute either of these equations depending on the `type` variable, which should be a string and should accept the following values `type = {'L1', 'L2', 'LInf'}`.

```
function [d] =  my_distance(x_1, x_2, type)
...
end
```

Test Implementation

We will begin by loading our 2D test dataset (Figure 2) which is generated by a GMM (Gaussian Mixture Model) with $K = 4$ components as seen in Figure 3. By running the **second** code block, the encrypted `test_mydistance.p` function will evaluate your `my_distance.m` function. If you get the following messages on your MATLAB command window, you can move on to the next task.

```
--- Checking my_distance.m on random samples from X ---
[Test 1] L1 Norm Computation: Correct.
[Test 2] L2 Norm Computation: Correct.
[Test 3] Linf Norm Computation: Correct.
```

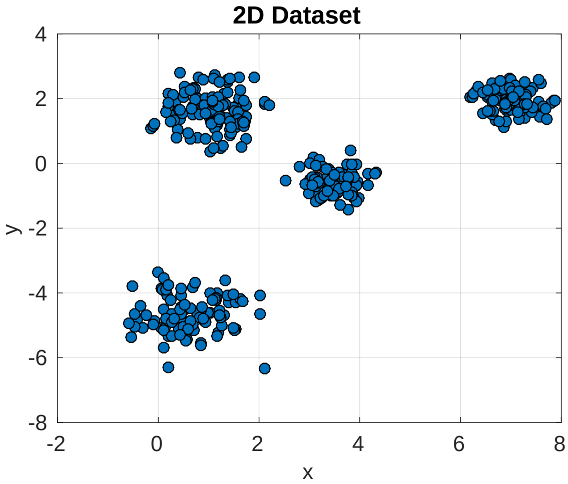

Figure 2: 2D k-Means Testing Dataset. Expected clusters $K = 4$.

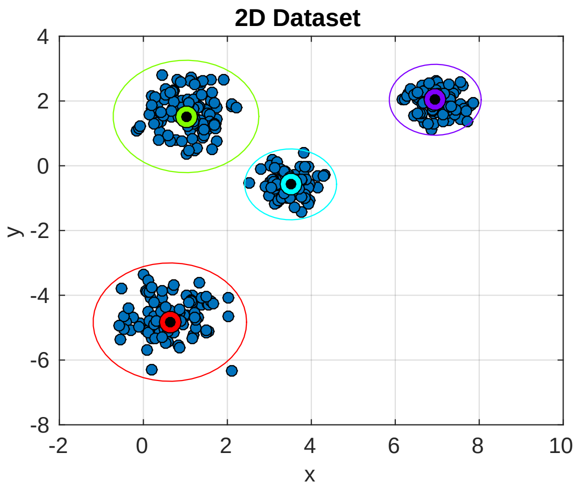

Figure 3: GMM parameters (μ, $\mathbf{\Sigma}$) used to generated the testing datapoints. The goal of k-means is to recover the centroids (μ); i.e. the colored points.

1.2 K-Means algorithm

Step 1. Centroid μ^k Initialization

In the k-means algorithm, the initialization of the centroids μ^k is one of the most important steps, if the centroids are initialized far away from the expected cluster centers, the algorithm might not be able to reach the expected solution. Hence, several methods have been proposed to initialize the centroids. In this assignment, we should implement **two** initialization methods:

- **random**: Select K data points from $\mathbf{X}$ at random.
- **uniform**: Select K data points uniformly at random from the range of $\mathbf{X}$; i.e. if we have a 2D dataset we select points at random between $[\min(\mathbf{X}_1), \min(\mathbf{X}_2)] \rightarrow [\max(\mathbf{X}_1), \max(\mathbf{X}_2)]$

TASK 2: Implement kmeans_init.m function (4pts)
Now we will implement each of these initialization steps in the `kmeans_init.m` function where `init` = {‘`random`’,‘`uniform`’}.

```
function [Mu] =  kmeans_init(X, K, init)
% init = 'random':  Select K datapoints at random from X
% init = 'uniform': Select K datapoints uniformly at random from range(X)
end
```

`Implementation Hint:` `Useful functions randsample(), datasample(), min(), max(), range().`

Test Implementation

If your implementation is correct, by running the **third** code block, we should visualize a plot similar to Figure 4, the way to check for correctness is if your centroids are points from the dataset. By modifying `init='uniform'`, you should see a plot similar to Figure 5, the way to check for correctness is if your centroids are random points in the 2D space and lie within the bounds of your datapoints. By running the code block multiple times, you should get different centroid locations each time. Further, By running the encrypted `test_kmeansinit.p` function will evaluate your `kmeans_init.m` function. If you get the following messages on your MATLAB command window, you can move on to the next task.

```
--- Checking kmeans_init.m on X ---
[Test 1 - random] Centroids seem random: Correct.
[Test 2 - random] Centroids are samples from X: Correct.
[Test 3 - uniform] Centroids seem random: Correct.
[Test 4 - uniform] Centroids are within range(X): Correct.
```

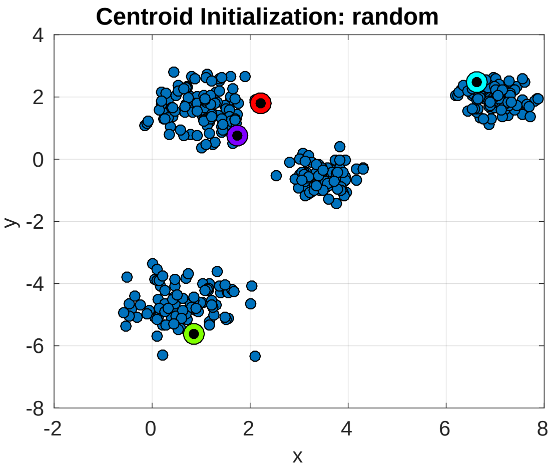

Figure 4: Centroids initialized through the **random** method.

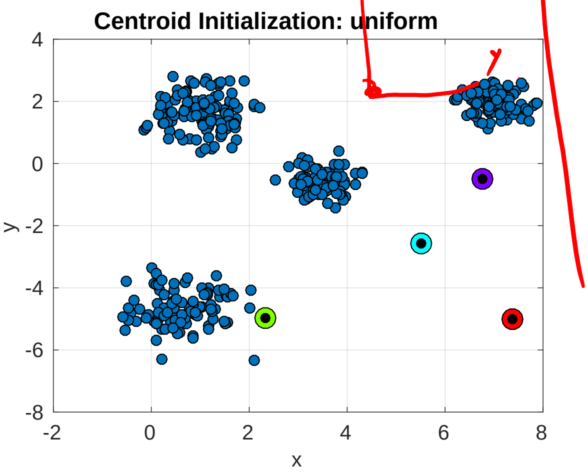

Figure 5: Centroids initialized through the **uniform** method.

Step 2. Calculate Distances between X and centroids μ^k

Once the centroids are initialized, with whichever chosen method, we can start the iterative estimation part of K-Means. The first step in the loop is to compute the distances between $\mu = \{\mu^1, \dots, \mu^K\}$ and X.

TASK 3: Implement my_distX2Mu.m function (2pts)

The next task is to implement the `my_distX2Mu.m` function. It should compute a $\mathbf{d} \in \mathbb{R}^{K \times N}$ matrix of distance from all points in $\mathbf{X}$ to the k centroids $\mu = \{\mu^1, \dots, \mu^K\}$ using your `my_distance.m` function.

```
function [d] =  my_distX2Mu(X, Mu, type)
% Compute distances
d = ...;
end
```

Implementation Hint: Populate **d** with my_distance().

Test Implementation

By running the **fourth** code block, the hidden `test_mydistX2Mu.p` function will evaluate your `my_distX2Mu.m` function. If you get the following messages on your MATLAB command window, you can move on to the next task.

```
--- Checking my_distX2Mu.m with X and Mu---
[Test 1] Distances from X to Mu with L1 Norm Correct.
[Test 2] Distances from X to Mu with L2 Norm Correct.
[Test 3] Distances from X to Mu with LInf Norm: Correct.
```

Step 3. Assignment Step: Centroid Responsibility

After calculating the distances from each point of $\mathbf{X} = \{\mathbf{x}^1, \dots, \mathbf{x}^M\}$ to each centroid $\mu = \{\mu^1, \dots, \mu^K\}$, we should assign points to their closest cluster centroids. Algorithmically, we do this by using the concept of responsibility; i.e. the cluster centroid μ^k closest to a data point $\mathbf{x}^i$ is responsible for that data point, the closest centroid can be found as follows,

$$k_i = \underset{k}{\operatorname{argmin}}\{d(\mathbf{x}^i, \mu^k)\}, \tag{5}$$

where $k_i \in \{1, \dots, k\}$ is the 'id' or 'label' of the closest cluster. The responsibility of each k-th cluster for data point $\mathbf{x}^i$ can then be computed as follows,

$$r_i^k = \begin{cases} 1 & \text{if} \quad k_i = k \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

If a tie happens (i.e. two centroids are equidistant to a data point), one assigns the data point to the centroid with the smallest winning cluster size.

Step 4. Update Step: Recompute centroids

We must then adjust the centroids of C_k to be the means of all data points assigned to them, this is called the maximization step and is computed as follows:

$$\mu^k = \frac{\sum\limits_i r_i^k \mathbf{x}^i}{\sum\limits_i r_i^k} \tag{7}$$

We then go back to **step 2** and **repeat** the process until the clusters are stable (i.e. there is no change between $\mu^{(t)}$ and $\mu^{(t-1)}$ where t is the iteration) or the maximum `MaxIter` is reached.

TASK 4: Implement my_kmeans.m function (5pts)
Now we will use our `kmeans_init.m` and `my_distX2Mu.m` functions and implement the three remaining steps in an iterative loop within the `my_kmeans.m` function, for a maximum of `MaxIter` iterations. `plot_iter` is a plotting boolean which if set to 1 plots the initial centroids, the first and the final iterations of k-means, for a 2D dataset. This is given to you in the function template. You should only fill in the required k-means steps.

```matlab
function [labels, Mu, Mu_init, iter] =  my_kmeans(X, K, init, type, MaxIter, ...
    plot_iter)
% Step 1. Initialize Mu_0
Mu_init = ...
while true
    % Step 2. Distances from X to Mu
    ...
    % Step 3. Assignment Step: Mu Responsability
    % Equation 5 and 6

    % Step 4. Update Step: Recompute Mu
    % Equation 7

    % Check for stopping conditions (Mu stabilization or MaxIter)
    ...
end
```

`Implementation Hint:  Useful functions:  my_distX2Mu(), sort(), min() and sum().`

Test Implementation

By running the **fifth** code block, you can directly test **my_kmeans.m** function, visualize the initialization (Figure 6), the first and last iterations (Figure 7 and 8) and the recovered decision boundaries (Figure 9). You should get similar plots as these. You can also try different distance types and you should see plots like Figure 10 if you chose the L_1 norm or Figure 11 for the L_∞, notice that for this simple example the clustering results are the same but the boundaries are different, this is due to the change in L_p.

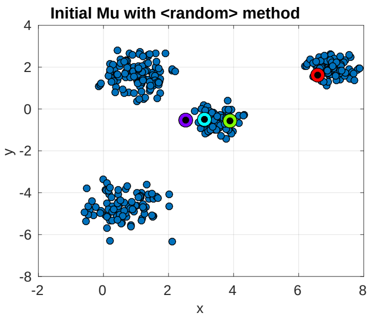

Figure 6: Centroids initialized through the **random** method.

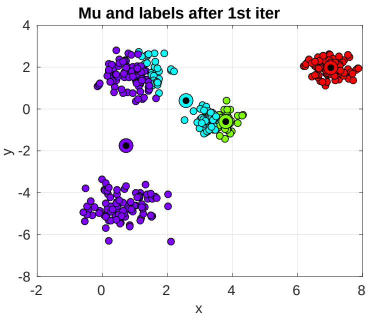

Figure 7: Updated Centroids after 1 iteration of k-means.

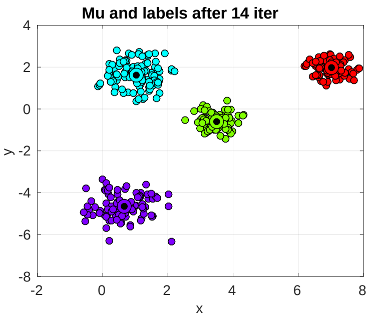

Figure 8: Final Centroids after converging.

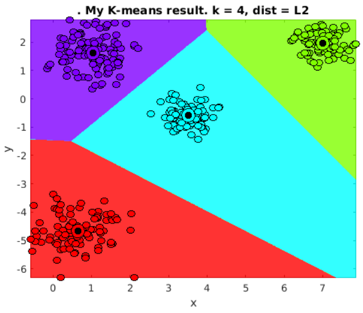

Figure 9: Decision Boundary from $K = 4$ and L_2 norm.

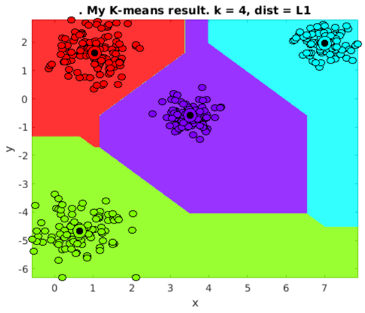

Figure 10: Decision Boundary from $K = 4$ and L_1 norm.

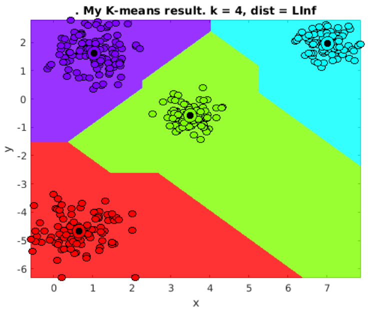

Figure 11: Decision Boundary from $K = 4$ and L_∞ norm..

Further, by running the encrypted `test_mykmeans.p` function with different values of K, you can evaluate your the K-means algorithm starting with `Mu_init`. If you get the following messages on your MATLAB command window for different K values, you can move on to the next task.

```
--- Checking my_kmeans.m on X with K = 5, type = L2, MaxIter = 100 ---
[Test 1 - init=random] Mu output dimensionality is Correct.
[Test 2 - init=random] All K clusters have points assigned to them: Correct.
[Test 3 - init=random] Stopping iter, Mu and labels: Correct
[Test 4 - init=uniform] Mu output dimensionality is Correct.
[Test 5 - init=uniform] All K clusters have points assigned to them: Correct.
[Test 6 - init=uniform] Stopping iter, Mu and labels: Correct
```

NOTE: If you happen to get the following error:

```
[Test 5 - init=uniform] Only K=6 clusters have points assigned to them, you ...
    have 1 EMPTY clusters!
```

you should resolve this by re-initializing `Mu_init`. If this error is not addressed in the submission, **1pt** will be deducted.

1.3 Application: Image compression

The core principle of k-means is to cluster points based on their features. To test your implementation of the algorithm, you have applied the clusterisation on a 2D dataset but this generalizes to higher dimensionnal data. We will use this concept to compress an image similarly as we did in the PCA assignment. Neighbors pixels that are close in terms of color value can be replaced by a single color, in this case, the centroid of their cluster.

Step 1. Image compression

You will apply the k-means algorithm to the image shown in Fig 12. As an effect, it will clusters the image based on pixel color values. For this to work properly you will have first to reshape the image to a $3 \times (width * height)$ matrix. Be careful to not break the order of the pixels when reshaping otherwize you will not end up with the correct number of colors in the reconstructed image. You will probably need to perform this operation in two successive steps.

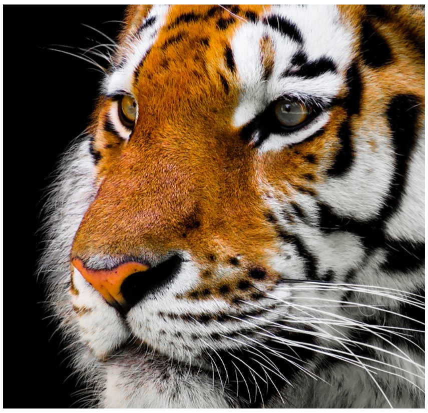

Figure 12: The original image on wich to apply clustering

TASK 5: Implement compress_image.m function (1pts)

Implement the fonction that compresses the image by applying your implementation of the k-means algorithm to cluster pixel values by color. Do not forget to reshape the image prior to this.

```
function [labels, centroids] =  compress_image(img, K, init, type, MaxIter)
...
end
```

Implementation hint: Functions to consider using: **reshape()**, **tranpose()**.

Test implementation

Run the script test_image_compression.m. If your implentation of the compression is correct you should see the following output:

```
    [Test 1] Dimensions of compression are correct
    [Test 2] Compression is correct
```

In case of errors, the problem most certainly comes from the reshaping step.

Step 2. Image reconstruction

The output of the clustering provides an array of labels and k centroids. Each values in the label array corresponds to the label of the centroid assigned to the pixel. Therefore, it is a $1 \times (width * height)$ array and the centroids matrix is $3 \times k$. The next step is to reconstruct the image from the label array and the centroid matrix. To do so, you have to replace each pixel by its associated centroid value. As a result, the reconstructed image should only contain k colors as in Fig. 13. If your reconstructed images seems to present more colors, it means that you have broken the pixel ordering during the reshaping phase.

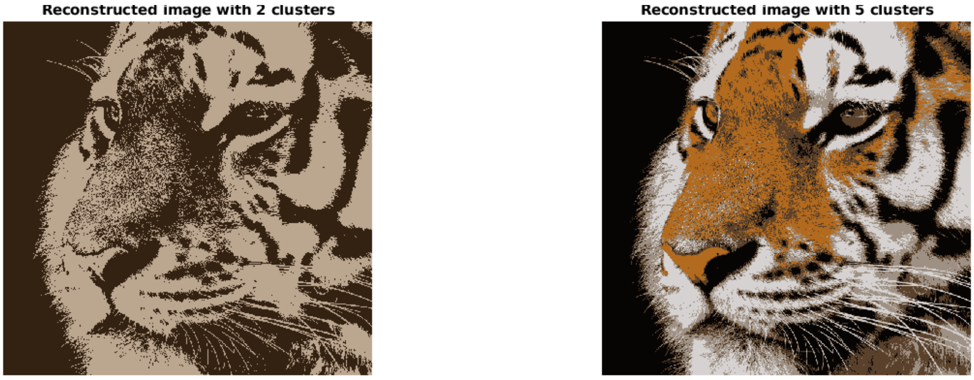
Reconstructed image with 2 clusters
Reconstructed image with 5 clusters

Figure 13: The image clustered with 2 (left) and 5 (right) colors

TASK 6: Implement reconstruct_image.m function (1pts)

Implement the fonction that reconstruct the image from the labels and centroids arrays. You will need to assign the corresponding centroid value to each pixel.

```
function [rimg] = reconstruct_image(labels,centroids,imgSize)
...
end
```

Test implementation

If your implentation of the reconstruction is correct you should see the following output:

```
[Test 3] Dimensions of reconstruction are correct
[Test 4] Reconstruction is correct
```

You can also check that the displayed images at the end of the script seem coherent in terms of number of colors.

2 Part 2: Metrics, and Semi-Supervised Clustering

In this part of the assignment, you will define some metrics that will help you choosing the optimal number of clusters. You will first apply it to the image compression task and then on a higher dimensionnal dataset.

2.1 Evaluation Metrics for Clustering

How do we determine the best clustering? For this, we can use a set of metrics which are used to evaluate clustering performance based on Equation 1. Following we provide the equations of the **three** clustering metrics that you will implement for k-means. For more explanations on the metrics, refer to the slides on clustering evaluation metrics from the Advanced Machine Learning course.

- **RSS:** The **R**esidual **S**um of **S**quares is in fact the cost function that k-means is trying to minimize (Equation 1.), hence

$$RSS = \sum_{k=1}^{K} \sum_{\mathbf{x}^i \in c_k} |\mathbf{x}^i - \mu^{\mathbf{k}}|^2 \tag{8}$$

- **AIC:** The AIC metric is a maximum-likelihood measure that penalizes for model complexity $AIC = -2\ln\mathcal{L} + 2B$ where B is the number of parameters and $\mathcal{L}$ the likelihood of the model. Even though the K-means algorithm does not provide a likelihood estimate of the model it can be formulated as a metric based on RSS as follows:

$$AIC_{RSS} = RSS + 2B \tag{9}$$

  where $B = (K * N)$ for K clusters and N dimensions.

- **BIC:** The BIC metric goes even further and penalizes for number of datapoints as well with the following equation $BIC = -2\ln\mathcal{L} + \ln(M)B$. As in AIC_{RSS}, we can formulate a BIC value based on the RSS as follows:

$$BIC_{RSS} = RSS + \ln(M)B \tag{10}$$

  where B is as before and M is the total number of datapoints.

TASK 7: Implement my_metrics.m function (3pts)

```
function [RSS, AIC, BIC] =  my_metrics(X , labels, Mu)
% Compute RSS (Equation 8)
RSS = ...;

% Compute AIC (Equation 9);
AIC = ...;

% Compute AIC (Equation 10)
BIC = ...;
```

Test Implementation

By running the test_image_metrics.m file, in the **second** code block the encrypted `test_mymetrics.p` function will evaluate your `my_metrics.m` function after clustering your data with `my_kmeans.m`. If you get the following messages on your MATLAB command window, you can move on to the next task.

```
--- Checking my_metrics.m ---
[Test 1] RSS computation is Correct.
[Test 2] AIC computation is Correct.
[Test 3] BIC computation is Correct.
```

Choosing Optimal K

To choose the optimal K for your dataset, one can run K-means by increasing monotonically the number of clusters and plotting the results. Since the output of K-means is not deterministic, we should run K-means for `repeats` times, generally 10, and take the **mean** or *best* of those runs as the result, in this implementation we will take the mean.

As the image contains a large number of samples, this step is computationnaly expensive and takes some time to complete. To reduce your waiting time, you will only **repeat** the clustering 3 times. For the RSS metric, the ‘elbow’ or ‘plateau’ method can be reliable for certain datasets. This method finds the optimal K at the elbow of the curve;i.e. when the values stop decreasing rapidly For the AIC and BIC metrics, the optimal K is found as the minimum value in the curve in Figure 14). This proves that for the image compression task, the optimal number of clusters is arround 7 with the L_2 norm.

TASK 8: Implement kmeans_eval.m function (3pts)

```
function [RSS_curve, AIC_curve, BIC_curve] =  kmeans_eval(X, K_range, ...
    repeats, init, type, MaxIter)

% Populate Curves
for i=1:length(K_range)
    ...
    RSS_curve(i) = ...;
    AIC_curve(i) = ...;
    BIC_curve(i) = ...;
end
```

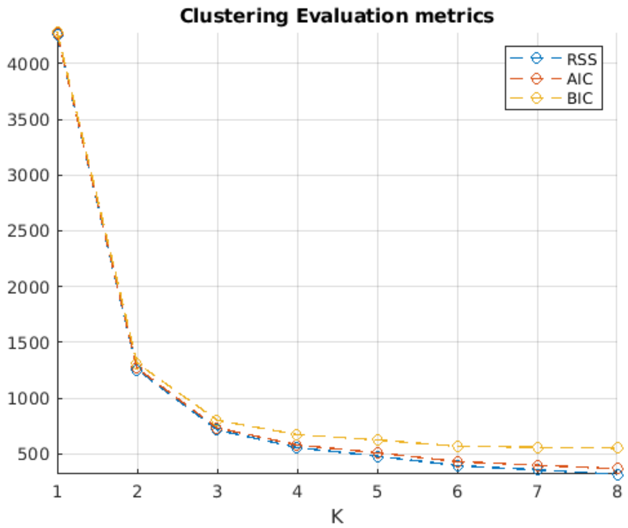

Figure 14: Clustering Evaluation metrics for the image compression. Optimal $K = 7$ with L_2 norm, `init='random'` and `repeats=3`.

Test Implementation

By running the encrypted `test_kmeanseval.p` function will evaluate your `kmeans_eval.m`. If you get the following messages on your MATLAB command window, you can move on to the next task.

```
--- Checking kmeans_eval.m on X ---
[Test 1] RSS Curve computation is Correct.
[Test 2] AIC Curve computation is Correct.
[Test 3] BIC Curve computation is Correct.
```

2.2 High-Dimensional Dataset and Semi-Supervised Clustering

In this part we will use K-means to cluster a real, high dimensional dataset. We will use the Digits dataset[1] which has $K = 10$ clusters and is in a 64-dimensional space. Further, we will find the F1-measure on the clustered data to evaluate how well it is clustered. In this part, we will use the script, `test_kmeans_digits.m`, and implement the function, `my_f1measure.m`.

2.3 Loading and visualizing the dataset

The digits dataset is loaded in the first block of the script, `test_kmeans_digits.m`. We can change the number of classes to be loaded from the data by varying `true_K` $\in [1, 10]$. A subset of the individual samples of the handwritten data can be visualized by running the second block of code in the script, as shown in Figure 15. It can also be visualized as a scatter plot as shown in Figure 16, where the first eight dimensions are displayed.

2.4 Cluster the original data

First we begin by clustering the original data. By using the functions created previously (`kmeans_eval.m`) and running the **second** code block we should visualized in Figure 17.

[1] The Pen-Based Recognition of Handwritten Digits Data Set can be found in the UCI Machine Learning Repository: `http://archive.ics.uci.edu/ml/datasets/Pen-Based+Recognition+of+Handwritten+Digits`

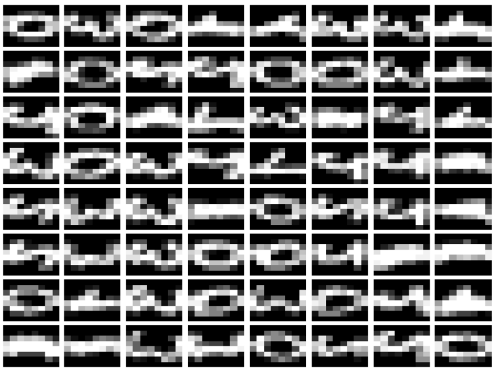

Figure 15: A sample visualization of a subset of the digits dataset as a 8 × 8 images.

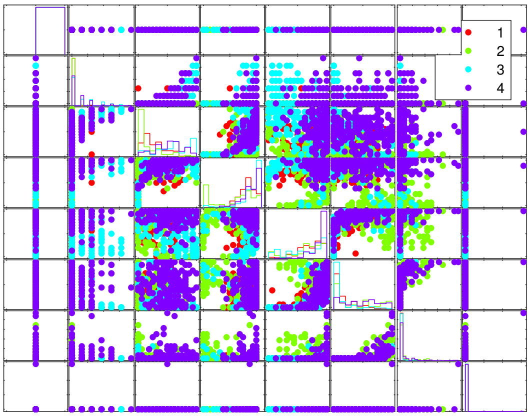

Figure 16: Visualizing the digits dataset as data points in the first eight dimensions, with $K = 4$ classes.

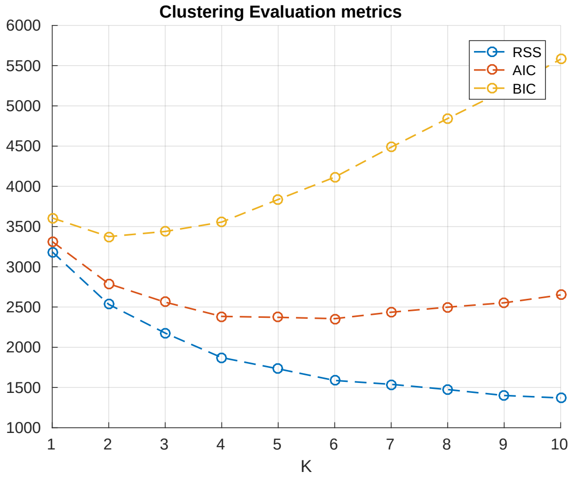

Figure 17: The variation of the evaluation metrics with K for clustering the digits data.

As we can see, it is hard to identify the optimal K for this dataset. Moreover, due to its high dimensionality, the result of a single k-means run cannot be visualized. For this reason, we will try in the next task to project our original dataset to a lower dimensional space.

2.5 Cluster the projected data

One of the advantages of PCA is not only that it projects the dataset to a lower dimensional space, but also, since it projects the dataset onto a new basis (guided by maximum variance) it can make the dataset more 'linearly separable' and hence easier to cluster! Here we will use the functions you implemented in the previous assignment, so make sure they are in your MATLAB search path. By running the **third** code block of `test_kmeans_digits.m`, we will apply PCA to the digits dataset and project it to a 4-D sub-space as shown in Figure 18. Then we evaluate K-means on this lower-dimensional space and should obtain the plot in Figure 19. As can be seen, the clustering metrics on the lower-dimensional space seem to yield a more decisive optimal 'K'.

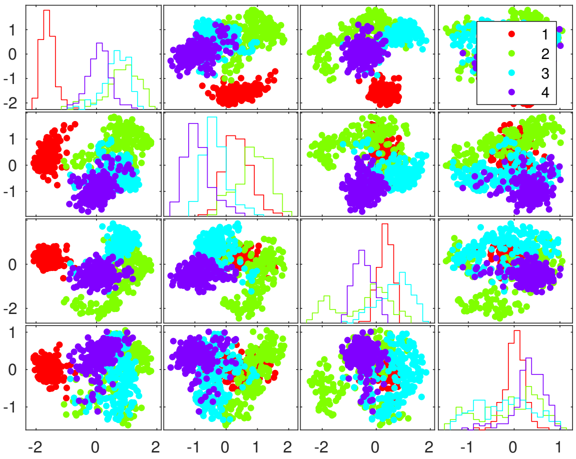

Figure 18: Visualization of the digits dataset projected to lower (4) dimensional space (from 64 original dimensions) using PCA.

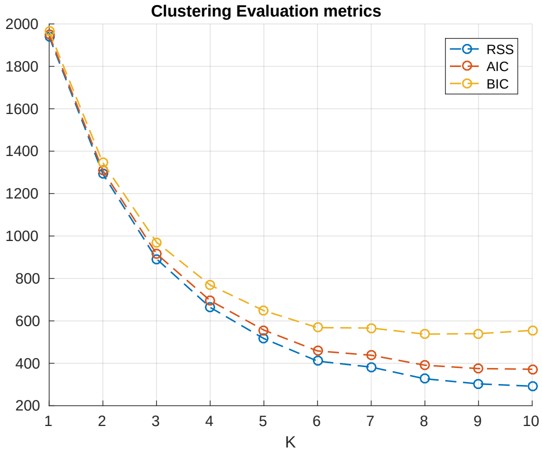

Figure 19: The variation of the evaluation metrics with K for clustering the projected digits data.

2.6 Semi-Supervised Clustering with F1-measure

When a subset of labels are available for the dataset we are clustering, we can apply an external metric to evaluate the results of our clustering algorithm. The external metric we will use in this practical is the F1-measure. For more details on this, refer to the slides on evaluation of clustering methods from the Applied Machine Learning Course.

The F1-measure can be estimated by using the following equations which are to be implemented in the function, `my_f1measure.m`.

$$F_1(C, K) = \sum_{c_i \in C} \frac{|c_i|}{M} \max_k \{F_1(c_i, k)\} \tag{11}$$

$$F_1(c_i, K) = \frac{2R(c_i, k)P(c_i, k)}{R(c_i, k) + P(c_i, k)} \tag{12}$$

$$R(c_i, k) = \frac{n_{ik}}{|c_i|} \tag{13}$$

$$P(c_i, k) = \frac{n_{ik}}{|k|} \tag{14}$$

TASK 9: Implement my_f1measure.m function (3pts)

```
function [F1_overall, P, R, F1] =  my_f1measure(cluster_labels, class_labels)
% Auxiliary variables
nClasses = ...
nClusters = ...
% Initializing variables
P = zeros(nClusters, nClasses); R = zeros(nClusters, nClasses);
F1 = zeros(nClusters, nClasses);
%For all clusters and classes combination
% Implement the precision equation here
P = ... % implement equation 14
% Implement the recall equation here
R = ... % implement equation 13
% Implement the F1 measure for each cluster here
F1 = ... % implement equation 12
% Implement the F1 measure for all the clusters here
F1_overall = ... % implement equation 11
```

Test Implementation

On running the 4th code block in `test_kmeans_digits.m`, the implementation for finding the F1-measure can be verified with the following output.

```
--- Checking my_f1measure.m ---
[Test 1] Precision Computation is Correct.
[Test 2] Recall Computation is Correct.
[Test 3] Per cluster F1 Computation is Correct.
[Test 4] Overall F1 Computation is Correct.
```

By running the **final** code-block, the F1-measure will evaluated for various values of K on both the original and projected datasets, with the encrypted `f1measure_eval.p` function. This function will use your implementation of `my_kmeans.m` and `my_f1measure.m`. The expected results should be similar to Figure 20 and Figure 21.

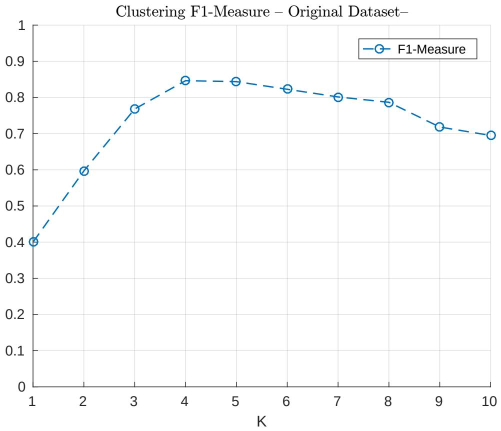

Figure 20: Mean F1-measure values for a range of K on the original dataset.

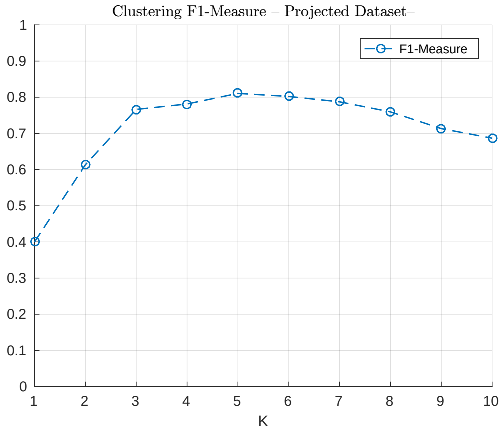

Figure 21: Mean F1-measure values for a range of K on the projected dataset.

References

[1] S. Lloyd. Least squares quantization in pcm. *IEEE Trans. Inf. Theor.*, 28(2):129–137, September 2006.