Lab 08

Basic Image Processing
Fall 2018

K-means Algorithm

- *K-means* algorithm (MacQueen'67): a heuristic method
  - Each cluster is represented by the centre of the cluster and the algorithm converges to stable centroids of clusters.
  - K-means algorithm is the simplest partitioning method for clustering analysis and widely used in data mining applications.
  - Each sample will belong to the cluster with the nearest mean.
- The objective is to minimize the within-cluster sum of squares:

$$\underset{S}{\operatorname{argmin}} \sum_{i=1}^{K} \sum_{x \in S_i} d^2(x, \mu_i), \qquad d^2(x, \mu_i) = \|x - \mu_i\|^2 = \sum_{n=1}^{N} (x_n - \mu_{in})^2$$

  - where $x \in \mathbb{R}^N$ are the data samples, μ_i is the mean (prototype) of the points in the cluster S_i *($i = 1 \dots K$).*

K-means Algorithm

- Given the cluster number *K*, the *K-means* algorithm is carried out in three steps after initialization:
  1) **Initialisation**: set the K cluster seed points (randomly)
  2) **Assignment step**: Assign each object to the cluster of the nearest seed point measured with a specific distance metric
  3) **Update step**: Compute new seed points as the centroids of the clusters of the current partition (the centroid is the centre, i.e., ***mean point***, of the cluster)

$$\mu_i^{(t+1)} = \frac{1}{\left|S_i^{(t)}\right|} \sum_{x_j \in S_i^{(t)}} x_j$$

  4) **Go back** to Step 1), stop when no more new assignment (i.e., membership in each cluster no longer changes)

Spaces & dimensions (in pure math)

It is important to understand all the spaces and their dimensionalities in this task.

Consider an S space: $S \subset \mathbb{R}^n$

Every element of S is a vector with n coordinates:

$$\mathbf{x}_a = [x_a^1, x_a^2, \ldots, x_a^n] \in S$$

The S_i subsets of S are the clusters: $S_i \subset S$

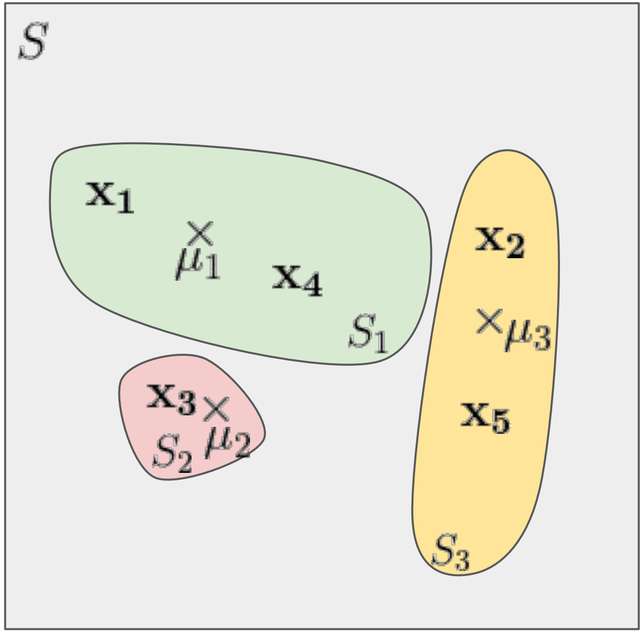
$$\bigcup_{i=1}^{k} S_i = S \quad \text{and} \quad S_i \cap S_j = \emptyset \quad \forall i \neq j$$

and where $i = 1, 2, \ldots, \boxed{k}$

These are the ‘k’-s in the term ‘k-means’. They tell you the number of clusters.

Also, there are μ vectors representing the mean values aka the centroids of every cluster:

$$\mu_i = [\mu_i^1, \mu_i^2, \ldots, \mu_i^n] \in S \qquad i = 1, 2, \ldots, \boxed{k}$$



Spaces & dimensions (in MATLAB)

Let us translate the terms of the previous slide into MATLAB.

Consider an `S` <u>space</u>, represented by a *matrix*.

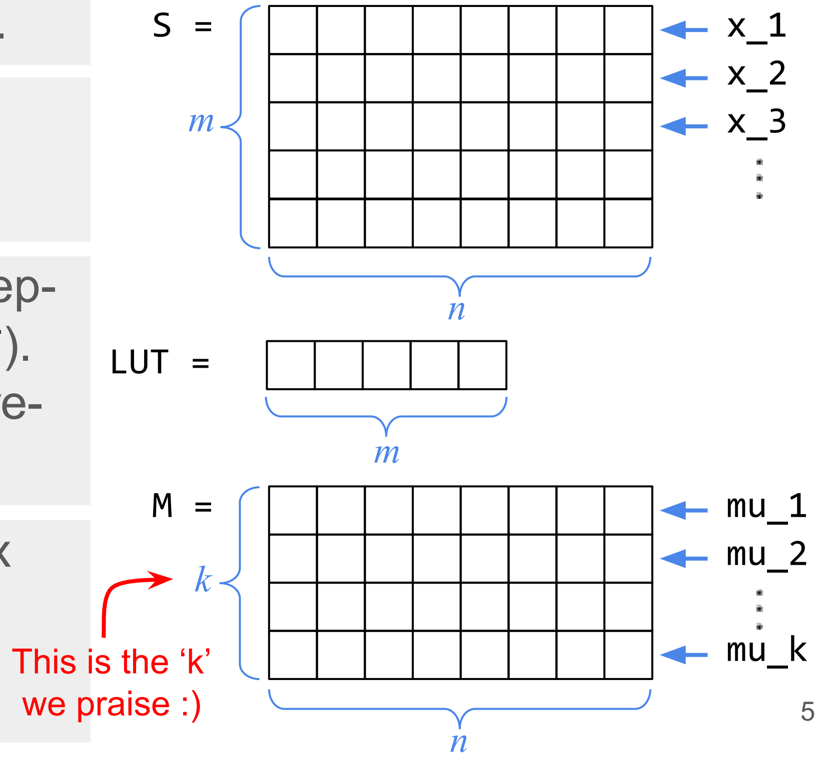
Every *row* of `S` is a <u>vector</u> with n items:

```
S(1,:) = x_1 = [x_11 x_12 … x_1n]
```

The `S_i` subsets of `S` are the <u>clusters</u>. Their representations are stored in a look-up-table (`LUT`). The index represents the index, the value represents the cluster # of a row vector `x_a` of `S`.

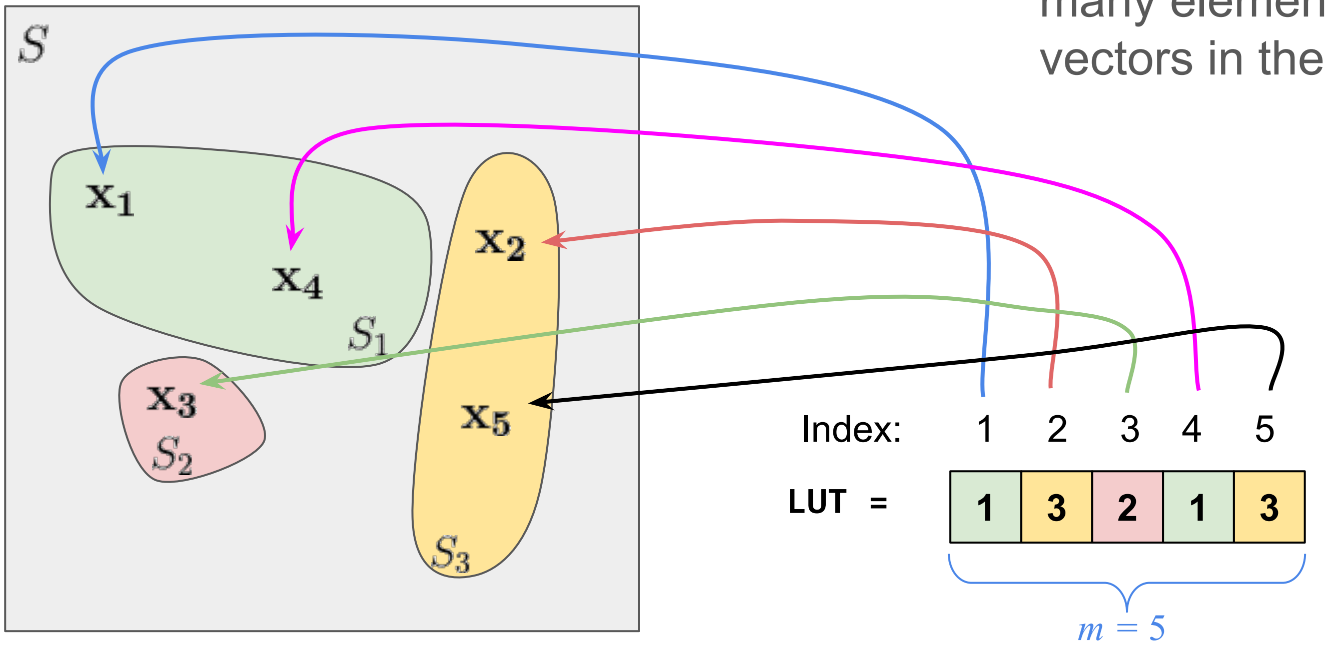
Also, the μ mean vectors are stored in a matrix similar to `S`, denoted by `M`. Its elements are

```
M(j,:) = mu_j = [mu_j1 … mu_jn]
```



So what is stored in the LUT?

The LUT does the vector-cluster mapping.



The LUT is a vector. It has as many elements as the number of vectors in the space S.

Every element of the LUT has an *index* and a *value*.

The value at position j tells us which cluster does vector $\mathbf{x}_j$ belong to.

Now please

download the ‘Lab 08’ code package

from the

submission system

Exercise 1

Implement the function `step1_initialization` in which:

- The function has 2 inputs and 2 outputs:

  Inputs:

  - `S` set of points to be clustered
  - `k` number of clusters

  Outputs:

  - `LUT` the assignment vector
  - `M` the matrix of centroids

The function should initialize `LUT` and `M` as described on the next slide!

After implementation, test your function with the **script** `test1_initialization`!

Step 1: Initialization

- initialize the `LUT` (as an `1 × m` vector, filled with zeros),
- initialize the `M` matrix (as a `k × n` matrix, filled with zeros),
- choose `k`-many vectors as the initial cluster center points and store them in `M`.

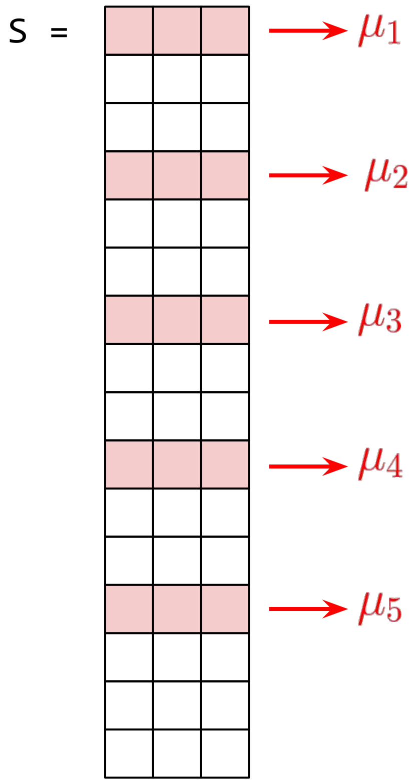
Now (to be able to reproduce the results)
<u>we are NOT using random initialization</u>
but an equidistant distribution!

$\lfloor \cdot \rfloor$ means **`floor()`**

We choose every $\left\lfloor \frac{m}{k} \right\rfloor$-th element as an initial center.

E.g: $m = 16$, $k = 5$ $\Rightarrow \left\lfloor \frac{16}{5} \right\rfloor = 3$

The first and then every 3rd element of `S` will be chosen to be an element of `M`.



Exercise 2

Implement the function `step2_assignment` in which:

- The function has 4 inputs and 1 output:

  Inputs:

  - `S` set of points to be clustered
  - `k` number of clusters
  - `LUT` the assignment vector
  - `M` the matrix of centroids

  Outputs:

  - `LUT` the **updated** assignment vector

The function should update `LUT` as described on the next slide!

After implementation, test your function with the **script** `test2_assignment`!

Step 2: Assignment

In this step:

For every `x_i` vector in `S` (i=1..m)

    For every `mu_j` vector in `M` (j=1..k)

        Calculate the distance between `x_i` and `mu_j`:

$$d_{ij} = d^2(\mathbf{x}_i, \mu_j) = \|\mathbf{x}_i - \mu_j\|^2 = \sum_{p=1}^{n}(x_i^p - \mu_j^p)^2$$

    From the calculated `d_ij` distances choose the smallest one, and store the index of the minimum in the `LUT` at position `i` :

$$\mathrm{LUT}_i = \arg\min_{j=1..k} d_{ij}$$

Exercise 3

Implement the function `step3_update` in which:

- The function has 4 inputs and 1 output:

  Inputs:

  - `S` set of points to be clustered
  - `k` number of clusters
  - `LUT` the assignment vector
  - `M` the matrix of centroids

  Outputs:

  - `M` the **updated** matrix of centroids

The function should update `M` as described on the next slide!

After implementation, test your function with the **script** `test3_update`!

Step 3: Update

In this step:

For every `mu_j` vector in `M` `(j=1..k)`

Select every `x` vector of `S` that is assigned to the `j`-th cluster:

MATLAB hint: You can index a vector logically! If the LUT is a vector and you write `LUT == 1` then this expression will return a logical vector: 1 if the element == 1, 0 otherwise.

If `A = [1 2 3 1 1 2 1]` then `A == 1` returns `[1 0 0 1 1 0 1]`

The other trick is that if you index a vector or matrix with a logical vector, the result will be the set of those elements that has the same indices where the logical vector contained 1-s.

If `B = [1 2 3 4 5 6 7` then `B(:,[1 0 0 1 1 0 1])` returns `[1 4 5 7 .`
`2 0 2 0 1 0 7]` `2 0 1 7]`

Update `mu_j`: the new value is the mean of the vectors of this cluster:

$$\mu_j^{(t+1)} = \frac{1}{|S_j|} \sum_{\mathbf{x}_i \in S_j} \mathbf{x}_i$$

Exercise 4

Implement the function `mykmeans` in which:

- The function has 2 inputs and 2 outputs:

  Inputs:

  - `S` set of points to be clustered
  - `k` number of clusters

  Outputs:

  - `LUT` the **final** assignment vector
  - `M` the **final** matrix of centroids

The function should realize the iterative procedure described on the next slide. Please print the number of iterations after the execution of the iterative procedure.

After implementation, test your function with the **script** `test4_mykmeans`!

Pseudo-code of the k-means algorithm

function `mykmeans`(S, k)

  Initialization step

  while `not` *converged* `and` *number of iterations is less than 100*

    Assignment step

    Update step

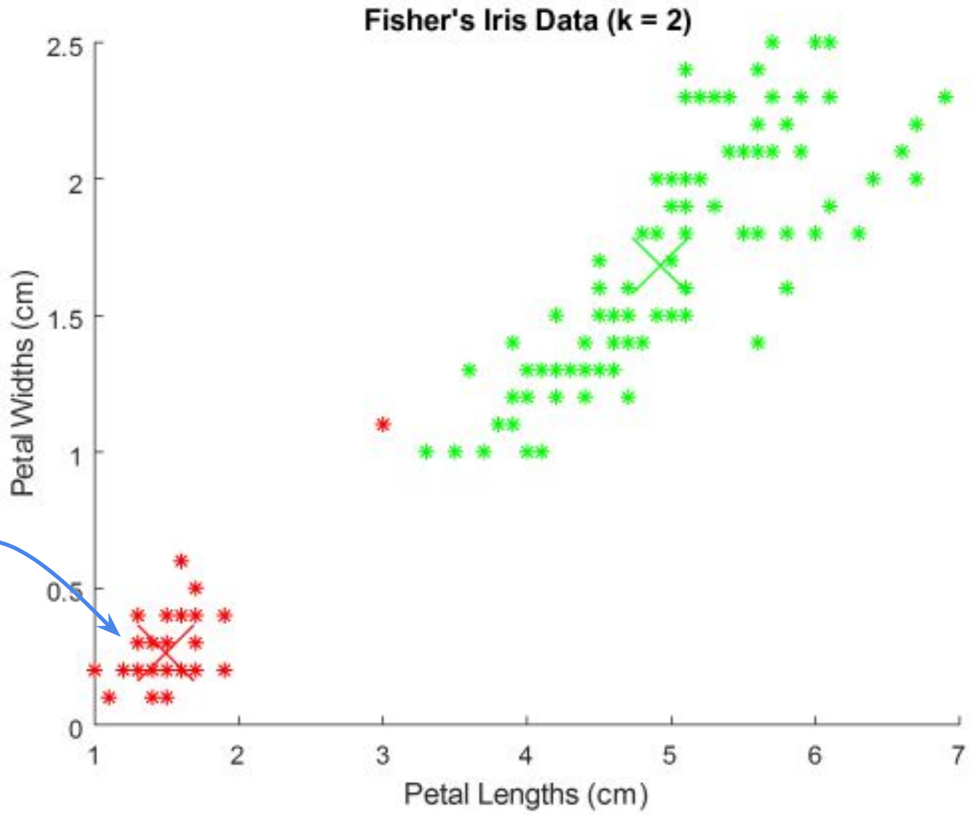
The algorithm *converged* if in the update step the sum of the distances between the old and new cluster center points is less than a threshold:

$$\sum_{j=1}^{k} \left| \left\| \mu_j^{(t+1)} - \mu_j^{(t)} \right\|^2 \right| < \varepsilon \qquad \varepsilon = 0.02$$

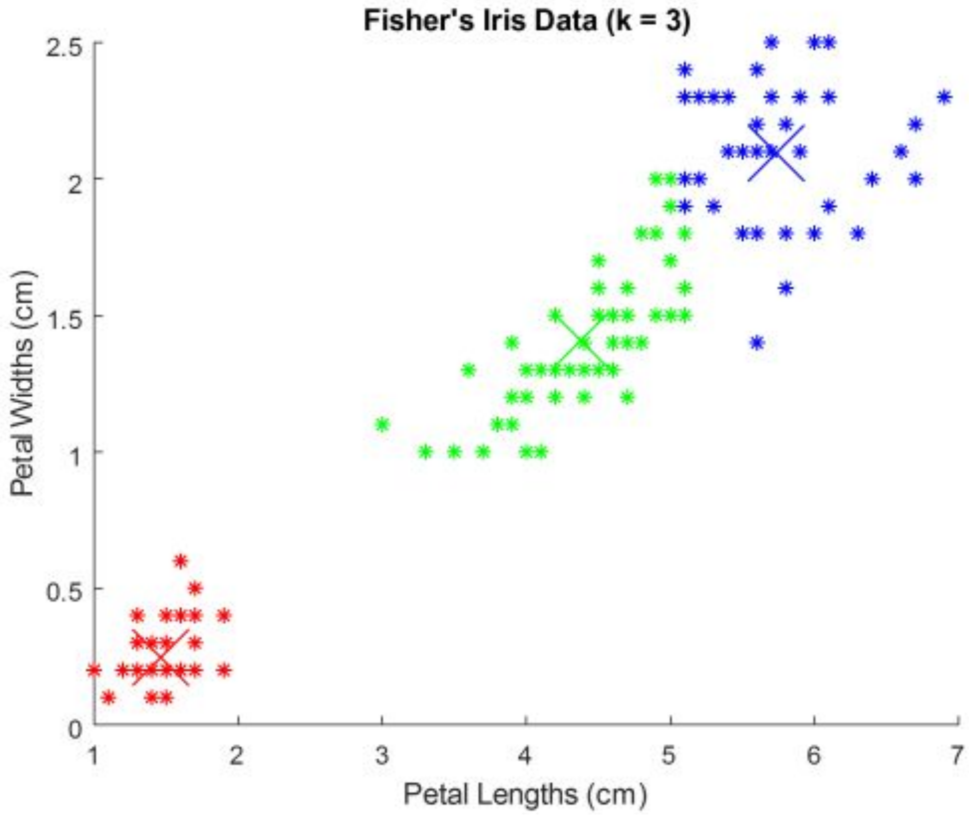
Exercise 4 -- what you should get

Color represents the cluster for which the data point belongs to.

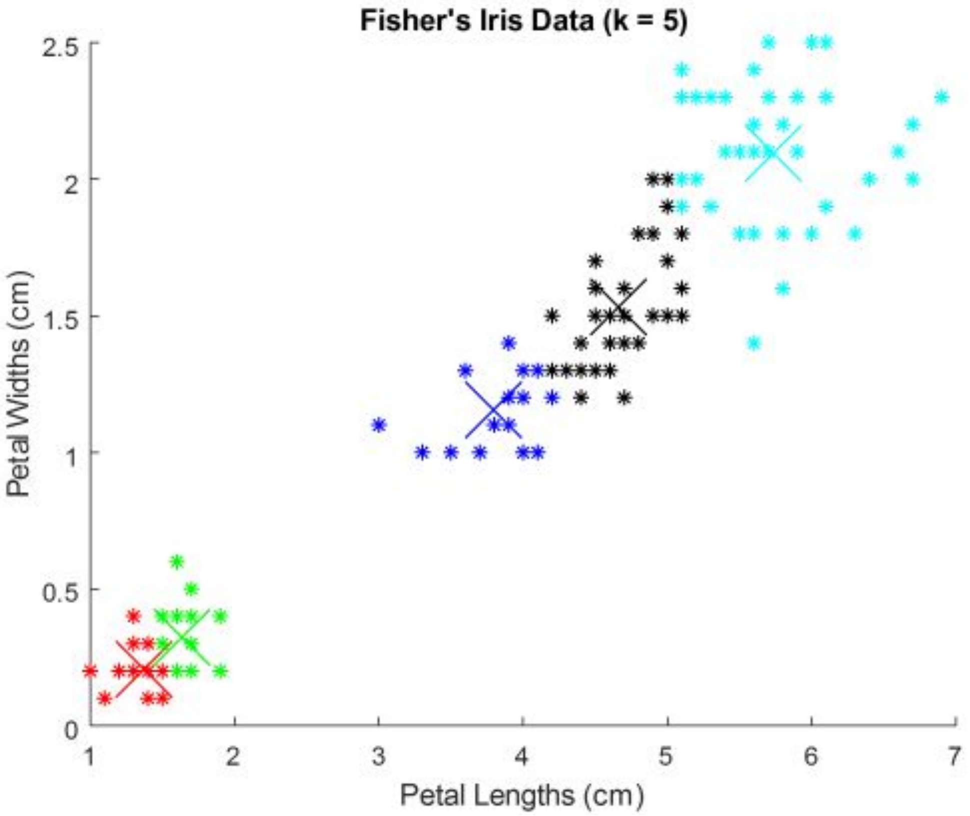
A big × marks the centroid of the cluster.



Exercise 4 -- what you should get



Exercise 4 -- what you should get



THE END