MATH10017 Assessed Coursework 4 (2024/25)

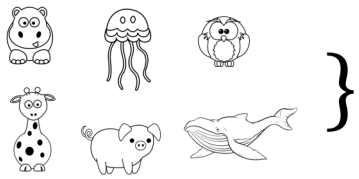
Matteo Fasiolo

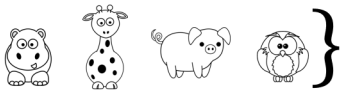
Context

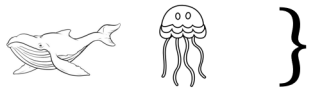
Clustering is a machine learning task that **groups similar objects together**. Given a dataset with n observations, $D := \{d_i\}_{i=1}^n$, we would like to divide D into k disjoint subsets or clusters:

$$D = \cup_{i\in\{1\ldots k\}} D_i \text{ and } D_i \cap_{i\neq j} D_j = \emptyset,$$

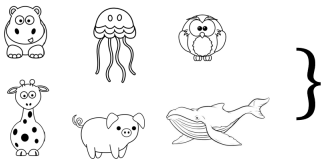
such that observations in each subset D_i **are similar** to each other. The subsets found by clustering are **not** unique but, depending on **how the similarity is defined**, you can get different clustering results. Example 1:

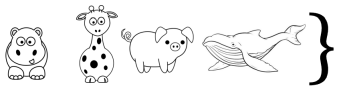
$D :=$ 

$D_1 =$ 

$D_2 =$ 

Example 2:

$D :=$ 

$D_1 =$ 

$D_2 =$ 

The criteria used to determine the two different clustering of the same data should be quite clear here!

In mathematics, similarities between objects are usually defined by a metric or distance function. A standard choice of distance is the L_1 or "Taxicab" distance. If two objects can be expressed as two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ in a d-dimensional space, the L_1 distance between them is

$$\text{dist}(\boldsymbol{a},\boldsymbol{b}) := \sum_{i=1}^{d} |a_i - b_i|.$$

1 The Iris data set

Ronald Fisher created the Iris dataset, where he measured the length, width of the sepals and petals of 150 iris flowers. In this dataset, each flower is a 4-dimensional vector.

```
head(iris) # load iris dataset in R by typing "iris".

##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
```

```
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

We will test a clustering algorithm defined below on this data.

2 A clustering algorithm

Given a dataset and a distance function, how to do clustering? There are many clustering algorithms, K-means being a very simple and popular choice. Clustering algorithms are widely used, and many machine learning libraries (such as sklearn or PyTorch) provide off the shelf implementations. Often, the idea is to compute the similarity between the observations and the "centers" of each subset or cluster, and then to assign observations to the closest cluster center. After the assignments are made, the cluster centers are updated. This is repeated until the cluster centers are not changing any more or the maximum number of iterations has been reached.

In this coursework we consider a clustering algorithm comprising two phases. At each iteration of Phase 1, each data point shifted. The shifting operation is repeated until the shifted data points have converged or the maximum number of iterations has been reached. In Phase 2, the shifted data points are merged into few cluster centers, and the corresponding original data points are attributed to the corresponding cluster. The details are provided in the following.

Suppose that the $(n \times d)$ matrix X contains your data, that is n observations in d-dimensional space. Let $X_{i:}$ be the i-th row of X, let δ be a small positive number and let h be a positive number (not necessarily small). In Phase 1, we will shift the rows of X using the following algorithm:

1. Set $\tilde{X} \leftarrow X$.
2. Set $\tilde{X}^{\text{old}} \leftarrow \tilde{X}$.
3. For each row $\tilde{X}_{i:}$ of $\tilde{X}$:
   (a) Compute the L_1 distance between $\tilde{X}_{i:}$ and each row of $\tilde{X}^{\text{old}}$. The vector of distances is called z, with elements $z_j = \text{dist}(\tilde{X}_{i:}, \tilde{X}^{\text{old}}_{j:})$.
   (b) Compute the weights vector w, with elements
   $$w_j = \exp(-z_j^2/h^2);$$
   for $j = 1, \ldots, n$.
   (c) Shift $\tilde{X}_{i:}$ by doing
   $$\tilde{X}_{i:} \leftarrow \frac{\sum_{j=1}^n w_j \tilde{X}^{\text{old}}_{j:}}{\sum_{l=1}^n w_l}.$$
4. If the maximum number of iterations has been reached or if $\text{dist}(\tilde{X}_{i:}, \tilde{X}^{\text{old}}_{i:}) < \delta$ for all $i = 1, \ldots, n$ (which means that the points are not moving much anymore), we terminate the algorithm. Otherwise, we go back to step 2.

The output of Phase 1 is the matrix of shifted data points $\tilde{X}$. Depending on the value of h, many of the rows of $\tilde{X}$ might be quite similar. That is, the algorithm implemented in Phase 1 shifted the data points toward a small number of locations.

In Phase 2, we use the shifted points to cluster the data. Let ε be a small positive number, let C be the set containing the positions of the cluster centers and let cl be the vector of cluster assignments, such that $cl_i = j$ means that the i-th observations belong to the j-th cluster. Then in Phase 2 we:

1. Initialise C as the empty set, that is $C \leftarrow \emptyset$. Initialise all the n elements of cl to zero.
2. For each row $\tilde{X}_{i:}$ of $\tilde{X}$:
   (a) For each element C_j of C (skip this loop if C is the empty set):
   - If $\text{dist}(\tilde{X}_{i:}, C_j) < \varepsilon$ then we assume that $\tilde{X}_{i:}$ belongs to the j-th cluster. Hence, we set $cl_i \leftarrow j$, we exit this for loop and we move to the next step.

   (b) If $cl_i = 0$ (that is, we have not found a cluster center close to $\tilde{X}_{i:}$), we add $\tilde{X}_{i:}$ to C (that is $C \leftarrow \{C, \tilde{X}_{i:}\}$) and we set cl_i to the number of elements of C (i.e. if C is an R list, we do `cl[i] <- length(C)`).

The output of Phase 2 is the set C of cluster centers and the vector cl of cluster assignments.

3 Your Task

Part I

1. Create a function `dist_vect` that takes as inputs two vectors x and y of arbitrary dimension d and returns the distance

$$\text{dist}(x,y) = \sum_{i=1}^{d} |x_i - y_i|.$$

2. Create a function `dist_mat` that takes as inputs a matrix X of dimension (n $\times d$) and a d-dimensional vector y, and returns an n-dimensional vector with elements

$$v = \{\text{dist}(X_{1:},y), \text{dist}(X_{2:},y), \ldots, \text{dist}(X_{n:},y)\}.$$

where $X_{i:}$ is the i-th row of X. That is, `dist_mat` computes the distances between y and each row of X.

3. Unless you have already done it in the previous point (if you did, just set `dist_mat_fast <- dist_mat`), produce a vectorised version of `dist_mat` (called `dist_mat_fast`). The new function does not have to contain any loop or call to `apply`, `lapply`, etc. Hint: the following code might help:

```
A <- matrix(0, 2, 5)
A

##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0    0    0    0
## [2,]    0    0    0    0    0

A - 1:2

##      [,1] [,2] [,3] [,4] [,5]
## [1,]   -1   -1   -1   -1   -1
## [2,]   -2   -2   -2   -2   -2
```

4. Test whether your functions work on the matrix and vector generated by this code:

```
set.seed(416)
X <- matrix(rnorm(10, 3), 10, 3)
y <- rnorm(3)
```

and make sure that your code prints out the output of

```
dist_mat(X, y)
dist_mat_fast(X, y)
```

Part II

1. Run the following code to generate some simulated data:

```
set.seed(123)
n <- 150
X <- rbind(
  cbind(rnorm(n / 3, mean = 0, sd = 0.3), rnorm(n / 3, mean = 0, sd = 0.3)),
  cbind(rnorm(n / 3, mean = 1, sd = 0.3), rnorm(n / 3, mean = 1, sd = 0.3)),
  cbind(rnorm(n / 3, mean = 2, sd = 0.3), rnorm(n / 3, mean = 0, sd = 0.3))
)
```

2. Write a function called `w_fun`, that takes as inputs a vector z and a positive scalar h, and returns a vector w with elements

$$w_i = \exp(-z_i^2/h^2), \quad \text{for} \quad i = 1,\ldots,n.$$

3. Write a function called `w_mean`, that takes as input a d-dimensional vector y, an $(n \times d)$ matrix X and a scalar h, and returns the vector

$$\tilde{y} = \frac{\sum_{i=1}^{n} w_i X_{i:}}{\sum_{l=1}^{n} w_l},$$

where

$$w_i = \exp\left(-\frac{\text{dist}(y, X_{i:})^2}{h^2}\right).$$

You should reuse the code from the previous points, as well as from Part I.

4. Write a function called `w_mean_vett`, that takes as input a $(n \times d)$ matrix Y, an $(n \times d)$ matrix X and a scalar h, and returns an $(n \times d)$ matrix $\tilde{Y}$, where each row is computed by

$$\tilde{Y}_{i:} = \texttt{w\_mean}(Y_i, X, h).$$

Test your functions using the following code:

```
par(mfrow = c(2, 2))
kk <- 1
for(ii in c(0.1, 1, 2, 10)){
  plot(X)
  mns <- w_mean_vett(y = X, x = X, h = ii)
  points(mns, col = 2, pch = 2)
}
```

which should produce four plots, showing the effect of h.

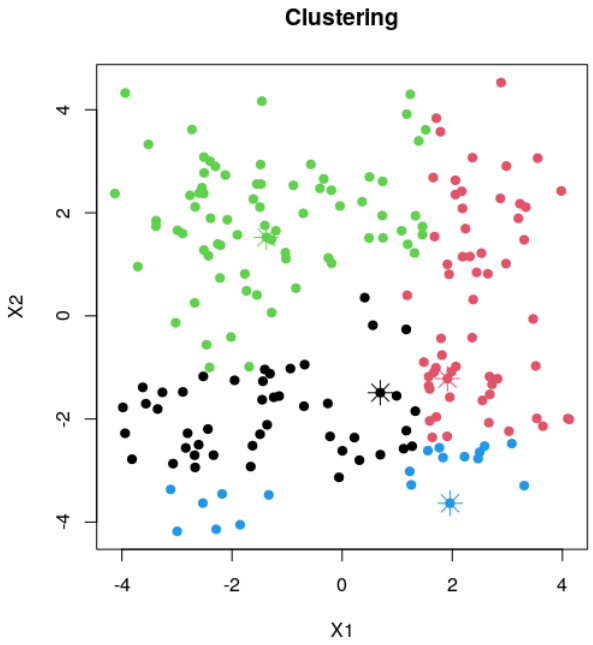
Part III

Here you should:

1. Create the function `phase_1` that takes as inputs `X`, `h`, `maxit` and `delta`, and implements the steps described in Phase 1 above. The functions should return a matrix $\tilde{X}$ of shifted points, and it should use the functions you developed in Part I and II.

2. Create the function `phase_2` that takes as inputs `Xt` (the output $\tilde{X}$ of `phase_1`) and `eps` (the tolerance ε described in Phase 2 above). It should returns a list with two elements: the list of cluster centers, C, and the vector of cluster assignments, cl.

3. Create the function `plot_clustering` that takes as inputs `X`, `C` and `cl`, and plots the data using a scatterplot such as this one:



but not exactly this one, as this is based on different data! The stars are the cluster centers, the dots the data points and the colours show the cluster assignment. The details of the plots are up to you (you can choose your favourite colours, symbols, etc).

4. Run the following code:

```
Xt <- phase_1(X = X, h = 0.5, maxit = 300, delta = 1e-3)

clustering <- phase_2(Xt = Xt, eps = 1e-2)

plot_clustering(X = X, Xt = clustering$C, cl = clustering$cl)
```

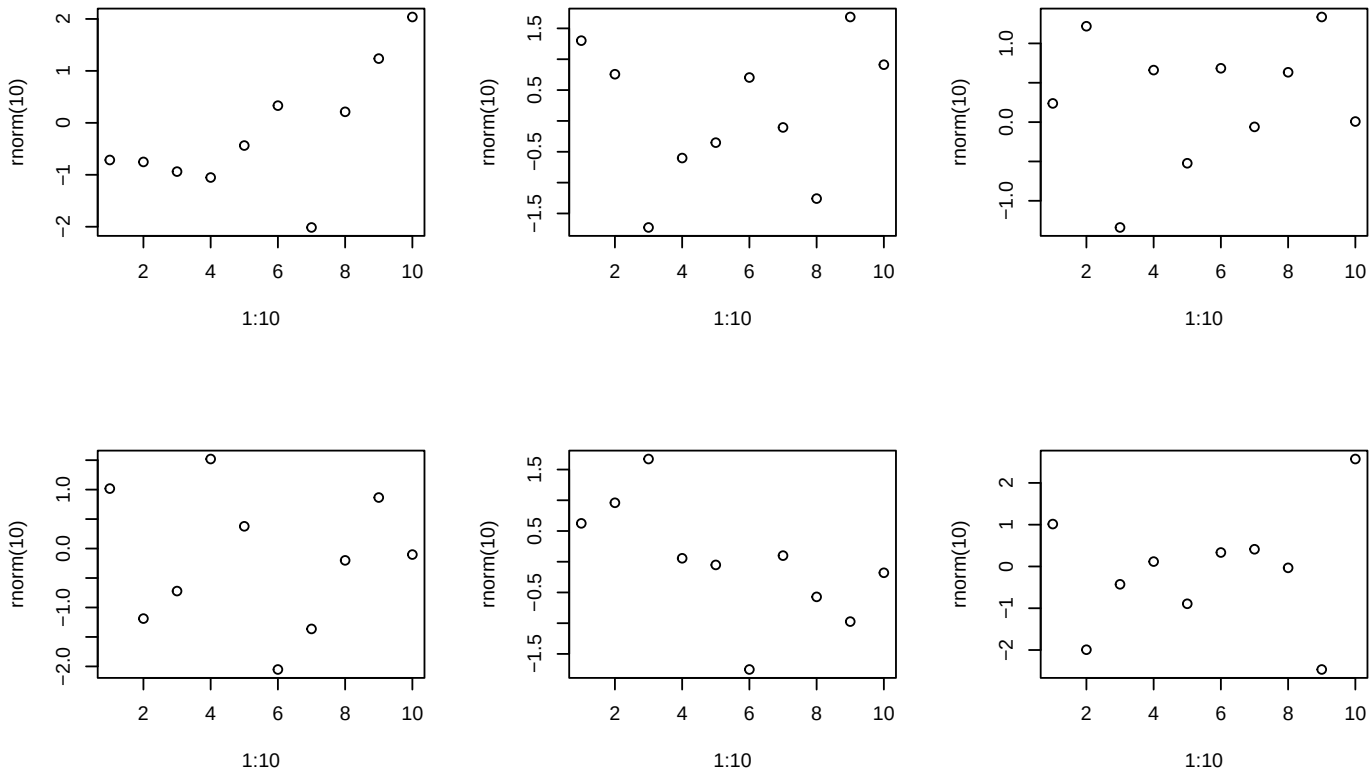
which should perform the clustering and plot the results.

Part IV

Now, let us apply the clustering algorithm to the iris dataset. You should:

1. Load the iris dataset by typing `data("iris")` and inspect the dataset. There are 5 variables in this dataset: Sepal.Length, Sepal.Width, Petal.Length, Petal.Width, Species. The first four variables are the properties of flowers. The fifth variable indicates the types of flowers.
2. Create a list of 6 new datasets from the iris dataset, by picking every pairs of variables from the first four variables.
3. Find a way to apply the clustering algorithm you previously wrote to the entire list of iris data sets and perform clustering analysis. You should set the clustering parameters to `h = 1`, `eps = 1e-2`, `maxit = 300` and `delta = 1e-3`.
4. Visualize the assignments obtained from the clustering algorithm, you can use something like this to get 6 figures in one plots:

```
par(mfrow = c(2, 3)) # <- this creates a 2D array of plots
for(ii in 1:6){
 plot(1:10, rnorm(10))
}
```



Your solution should include code for visualising the clustering on each of the 6 data sets.

Marking criteria

Please submit only an single R script names xyz123.R (where xyz123 is your username). You do not need to submit any data file. You do not need to submit any image that your code generates.

The marking scheme below is indicative and is intended only as a guide to the relative weighting of the different parts of the project. So the final mark is roughly determined by:

1. Part I: 20 points
2. Part II: 25 points
3. Part III: 25 points
4. Part IV: 10 points
5. Vectorization and functional programming (FP): 10 points. Your code uses vectorization and does not use unnecessary loops. Your code uses some FP features (do not overdo it, your code does not need to be completely written in FP!).
6. Coding Style: 10 points. Appropriate comments, variable names and code formatting.

NOTE Hard-coded solutions will be penalised. To see what is a hard-coded solution, imagine that I am asking you to write a function that sums the elements of a vector and to apply it to the specific vector

```
n <- 7
x <- rnorm(n)
```

This is an hard coded solution:

```
sum_hard <- function(v){
  out <- 0
  for(ii in 1:7){
   out <- out + v[ii]
  }
  return(out)
}
sum_hard(v = x)

## [1] 2.730368
```

while this is an OK (i.e. general) solution

```
sum_ok <- function(v){
  out <- 0
  for(ii in 1:length(v)){
   out <- out + v[ii]
  }
  return(out)
}
sum_ok(v = x)

## [1] 2.730368
```

Plagiarism policy

While you can discuss the general strategy of your code with your classmates, you must code independently. Code sharing and/or co-developing will be treated as academic collusion. This is a serious offence, see here.

You should be able to write the code for this project only using the functions and libraries we have used to far in the course. See also the hints above. The use of code found on the internet is discouraged. But, if you use do use code found on the web or elsewhere, it should be limited to a very small section of your code and you will need to disclose its the source in your comments. Otherwise, it will be regarded as plagiarism.