MATH10017 Assessed Coursework 4 (2023/24)

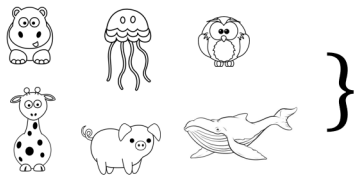
Matteo Fasiolo

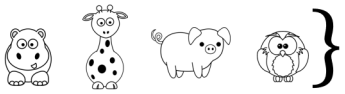
Context

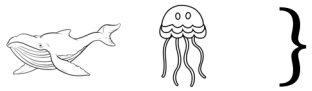
Clustering is a machine learning task that **groups similar objects together**. Given a dataset with n observations, $D := \{d_i\}_{i=1}^n$, we would like to divide D into k disjoint subsets or clusters:

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

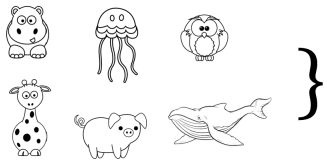
such that observations in each subset D_i **are similar** to each other. The subsets found by clustering are **not** unique but, depending on **how the similarity is defined**, you can get different clustering results. Example 1:

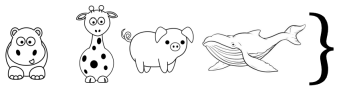
$D :=$ 

$D_1 =$ 

$D_2 =$ 

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. The main 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, it updates 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.

Suppose that the $(n \times d)$ matrix X contains your data set, that is n observations in d-dimensional space. Let $X_{i:}$ be the i-th row of X. We will divide X into k clusters by using the following algorithm:

1. Randomly select k observations from your dataset (i.e., k rows from X), and use them as cluster centers $c_1, \ldots, c_K$.
2. For each row of $X_{i:}$ of X:
   (a) Find the cluster center that is closest to $X_{i:}$ in terms of L_1 distance.
   (b) Assign $X_{i:}$ to that cluster (e.g., the 2nd cluster if $X_{i:}$ is closest to c_2).
3. Compute the L_1 distance between each row $X_{i:}$ and the corresponding cluster center, and sum the distances you just computed over i. The resulting scalar is the total distance or cost of the current clustering.
4. For $j = 1, \ldots, k$ and for $i = 1, \ldots, n$ (i.e., two nested for loops)
   (a) Consider using $X_{i:}$ as the j-th cluster center (that is, setting $c_j \leftarrow X_{i:}$).
   (b) Go through steps 2 and 3 (above) again to check what would be the total cost under this alternative clustering.
   (c) If, following this move, the total cost is lower than under the current clustering, store the move "setting $X_{i:}$ as the j-th cluster" in a list of good moves.
5. If the list of good moves is empty (i.e., no cost reduction is possible) or if the maximum number of iterations has been reached, stop the algorithm, otherwise go to the next step.
6. Look at the proposed moves from Step 4, find the one that leads to the lowest total cost and update you clustering based on that move. E.g., if using "setting $X_{7:}$ as the 3-rd cluster center c_3" leads to the lowest cost then update c_3 accordingly (that is, set $c_3 \leftarrow X_{7:}$).
7. Go back to step 2.

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 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 (in which case, 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: you could get inspiration from this code:

```
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(414)
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(4184)
n <- 200
X <- matrix(rnorm(n*2, sample(c(-2, 2), n*2, replace = TRUE), 1), n, 2)
```

2. Assume that we are using $k = 4$ clusters centers, and initialise their locations at the first 4 data points by doing:

```
k <- 4
id_C <- 1:k
C <- X[id_C, ]
```

So now each row of C contains the position of one of the initial cluster centers.

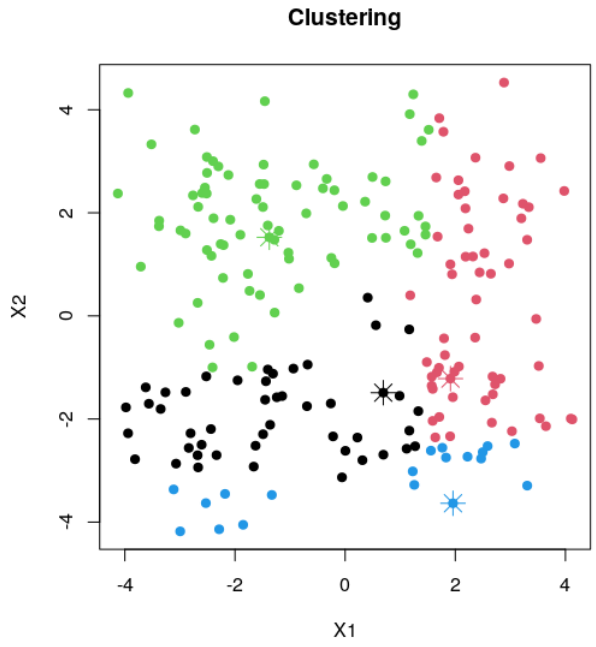
3. Create the function `find_nearest` that takes as inputs `X` and `C` and returns a list (named `cl`) of cluster assignments, specifying which data points belong to which cluster. In particular, the i-th element of `cl` is a vector whose elements are the indices of the data points for which the i-th cluster center is the closest in terms of the L_1 distance. For example, if `X[3, ]` is closest to `C[2, ]` then the index 3 should appear (only) in the vector `cl[[2]]`.

4. Create the function `dist_tot` that takes as inputs `X`, `C` and `cl` and returns the total distance of the data points from the corresponding cluster centers. That is, the function should return the scalar:

$$\text{dtot} = \sum_{j=1}^{k} \sum_{i \in cl_j} dist(X_{i:}, C_{j:}),$$

where cl_j is the j-th element of the list `cl` and $C_{j:}$ is the j-th row of C.

5. 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 don't need to use the same colours, symbols, etc).

6. Run the following code:

```
cl <- find_nearest(X, C)
dist_tot(X, C, cl)
```

and add to your R script the code below to save the plot produced by your function as a .png file:

```
png("./points.png", width = 500, height = 500)
#create the plot
plot_clustering(X, C, cl)
#close the file
dev.off()
```

Part III

To fill this part you should be able to use some of the functions that you have implemented above.

1. Implement the clustering algorithm described in Section 2 above. It should be implemented as a function called `clu_algo`, with arguments `X` (the matrix of data points), `k` (the number of clusters) and `max_iter` (the maximal number of iterations). The function should return a list with elements `cl` (the assignment list, see above), `C` (matrix where each row is a cluster center) and `cost` (the total distance of the data points from their cluster center).

2. The cluster centers should be initialised randomly from `k` rows of `X` (Hint: see `?sample`).

3. Run the algorithm on the data `X` generated above, using `k = 4` cluster and `max_iter = 50`. Make sure that your R script prints out the total cost of the clustering at convergence.

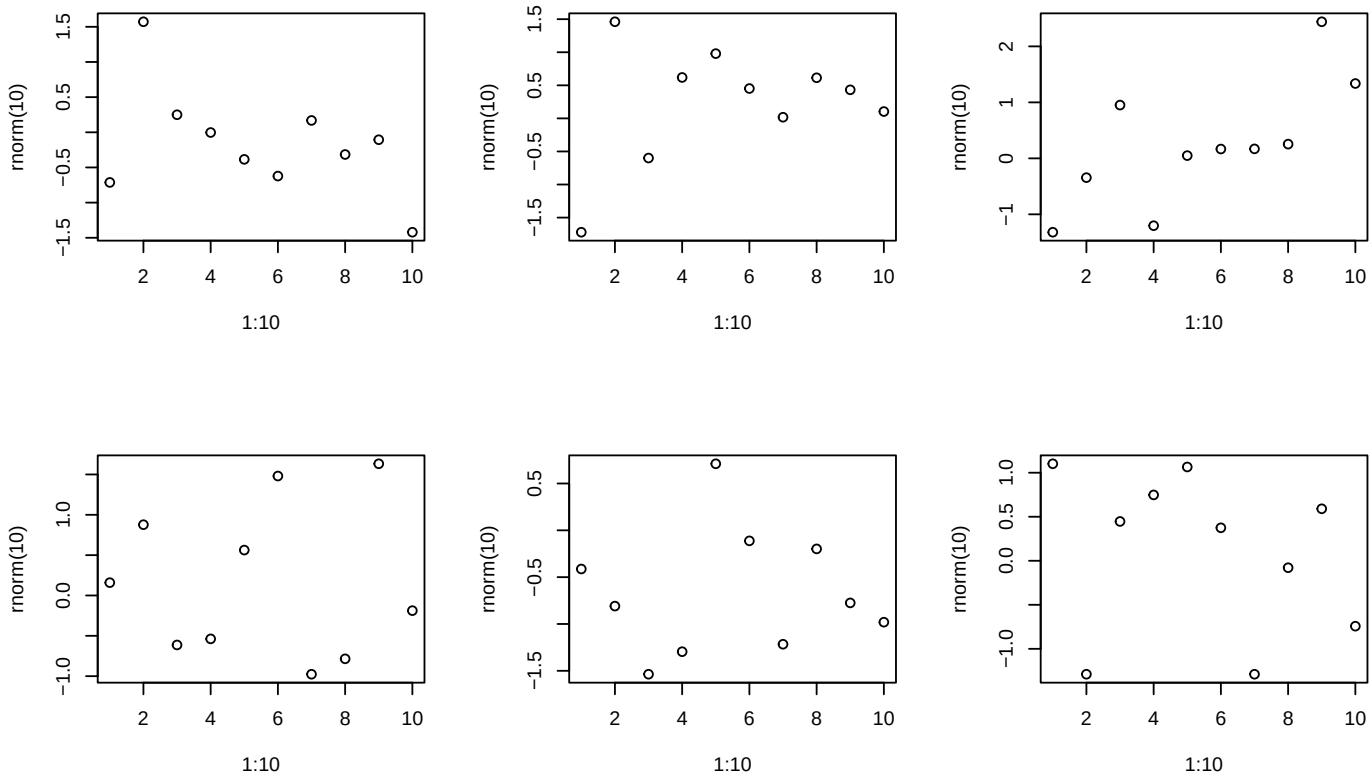
4. Plot the output of the algorithm using the `plot_clustering` function you have created above and check that the selected clustering is reasonable. Export the plot as a .png file by modifying the code above (you'll need to change `"./points.png"` to (say) `"./clust.png"`).

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 use `k = 2` clusters.
4. Visualize the assignments obtained from the clustering algorithm, and save your plots (6 in total) in a single png files called `"iris.png"`. 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))
}
```



Additonal Hints

In your solution you might find useful some of the following functions:

- `which` returns the indices of a logical vector that are TRUE:

```
z <- c(FALSE, TRUE, FALSE)
which(z)

## [1] 2
```

```r
which(c(1, 3, 4, 0) == 4)

## [1] 3
```

- `which.min` finds the index of the smallest element of a vector:

```r
x <- c(1.1, 3.2, -1, 0)
which.min(x)

## [1] 3
```

- `rep` repeats the first argument an arbitrary number of times:

```r
rep(x, 2)

## [1]  1.1  3.2 -1.0  0.0  1.1  3.2 -1.0  0.0

rep(x, each = 2)

## [1]  1.1  1.1  3.2  3.2 -1.0 -1.0  0.0  0.0

rep(3, 5)

## [1] 3 3 3 3 3
```

- `unlist` reduces a list to a vector:

```r
y <- lapply(1:3, FUN = function(a) a^2)
y

## [[1]]
## [1] 1
##
## [[2]]
## [1] 4
##
## [[3]]
## [1] 9

unlist(y)

## [1] 1 4 9
```

Marking criteria

Please submit a zip file containing your R script (or an Rmd file if you use R markdown). Ideally, you should submit only one file to facilitate marking. You do not need to submit any data file. You do not need to submit any images your code generate.

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: 30 points
3. Part III: 20 points

4. Part IV: 10 points
5. Vectorization and 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] -1.627086
```

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] -1.627086
```

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.