ADS2 - Clustering and Machine Learning

Rob Young (based on Dmytro Shytikov and Melanie Stefan)

Semester 2, 2023/24

Work through this guide alone or in groups. Facilitators are here to help, and are interested in what you're doing - do not be shy to ask them questions, or discuss your results with them.

The time it takes to complete this practical can vary between individuals - this is OK. Do not worry if you do not finish within the session.

Learning Objectives

- Practice the use of k-means and hierarchical clustering.

Creating a seating plan

Dr. Fischer has organised a regional scientific meeting on C. elegans. 20 researchers from all over the region have signed up for the conference dinner, but they don't know each other. What table should they sit at?

Dr. Fischer thought it would be nice to group dinner guests by age and by how long they have had to travel (in hours) to get to the conference. That way, people at the same table would likely have something in common, and find it easier to start a conversation.

The data is provided in the file `guests.csv`. Dr. Fisher has already normalised it, so that both variables fall between 0 and 1.

Import and plot the data to get a sense of what it looks like!

k-means clustering

k=4

Let's assume there are four tables available, but that it is not necessary for each table to have the same number of people at it.

Since we know the number of clusters, this seems like a great opportunity to try k-means clustering (Here, with k=4). Of course there are functions you can use to do this, but in this practical, we will go through it "by hand", so that you get a sense of the algorithm. If you feel adventurous, go ahead and implement it on your own. If you need a little bit more guidance, follow the steps outlaid below.

In this practical, we will only do 1 round of clustering (i.e. with one set of initial conditions). Check the problem set if you want to do more!

Steps for k-means clustering:

- Adapting the data frame: It is probably useful to add extra columns to the guests data frame: One to keep track of what) cluster every data point is in, we can call that "cluster". And then four more to store the distance of each data point from each of the four centroids.

First, I read 4 in the data as a single data frame and create a colour palette for my four clusters. Of course, you might like to define the colours differently. I created some extra columns to record the distance of each data point to the four centroids (which I have not yet defined).

```
data = read.csv("guests.csv", stringsAsFactors = T)

library(RColorBrewer)
colours <- brewer.pal(4, "Dark2")

# Now make a list of clusters
data$cluster <- NA
data$distance_1 <- NA
data$distance_2 <- NA
data$distance_3 <- NA
data$distance_4 <- NA
data$colour <- "black"
```

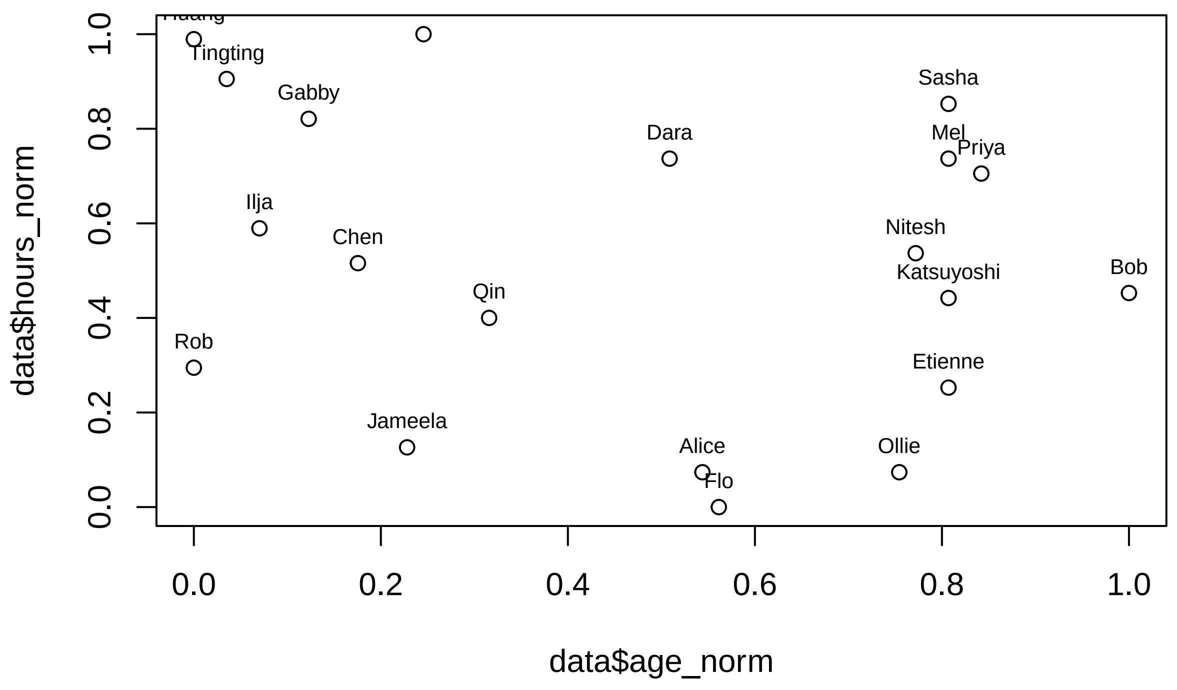
- Assigning and storing the centroids: I found it most useful to use a different data frame for the centroids. Populate it at first with four randomly chosen data points from the guests data frame (the `sample()` command may be useful here).

```
# take four initial samples
initial_centroids <- sample(data$names, 4)
```

- Now if you want to plot the initial situation (data points and initial centroids), go ahead and do that. Remember that if you have a plot p in ggplot2 where you have plotted a data set, you can add additional points from another dataset by just adding to the plot. For instance, once I had defined my plot p, I added points from the "centroids" data set like this:

```
p <- p + geom_point(data = centroids, aes(x = age_norm, y = hours_norm),
    shape = 3, size = 2, colour = "red")
```

```
# Plot the data
plot(data$hours_norm ~ data$age_norm)
text(data$age_norm, data$hours_norm, labels = data$name, cex = 0.7,
    pos = 3)
```

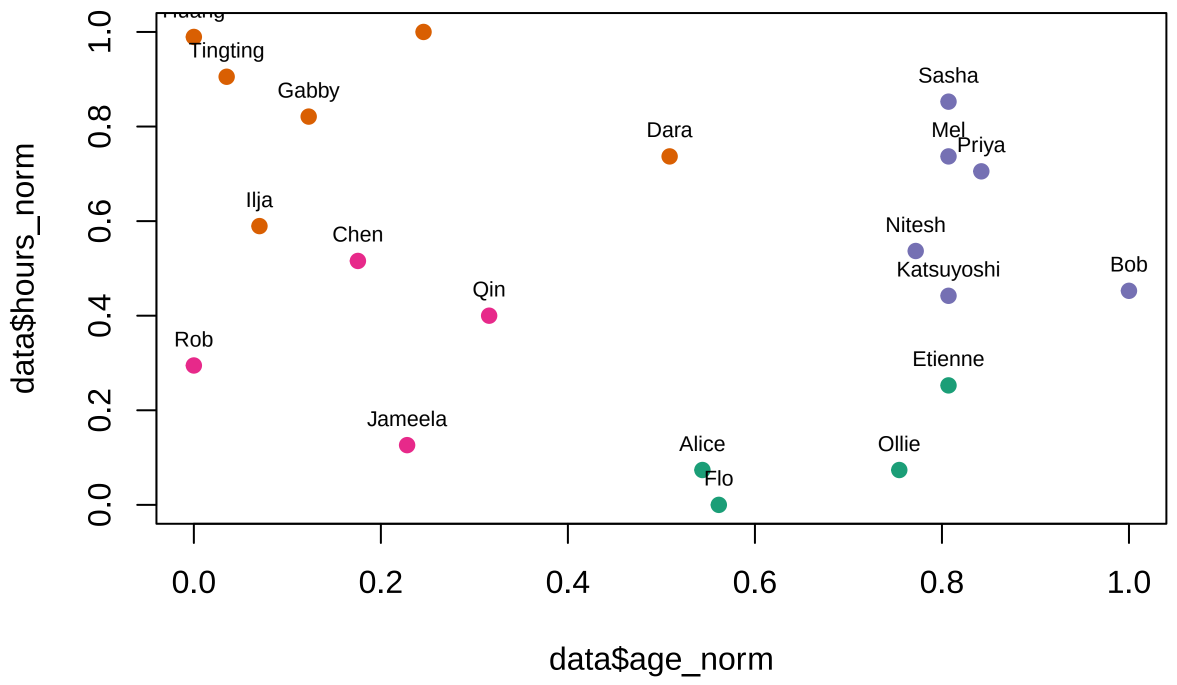


- OK, now you have to go through several rounds of the following:
  - computing the distance between each data point and each centroid.
  - assigning each data point to the cluster of its nearest centroid. There are various ways to do this. The `which()` function is quite useful to search for the minimum value across the four distance columns, but you may have a better way of doing it.
  - for each cluster, re-compute a new centroid (mean of all x values, mean of all y values), store that in the centroid dataframe.

This is my first cluster assignment, using the four random data points selected above.

```r
# for each row, calculate the distance of each pair to all
# four centroids use the dist function not the absolute
# distances then select the minimum and assign the point to
# that cluster
for (a in 1:nrow(data)) {
    data_age <- data[a, 2]
    data_hour <- data[a, 3]

    for (b in 1:length(initial_centroids)) {
        centroid_age <- subset(data$age_norm, data$names == initial_centroids[b])
        centroid_hour <- subset(data$hours_norm, data$names ==
            initial_centroids[b])
        distance <- dist(matrix(c(data_age, centroid_age, data_hour,
            centroid_hour), ncol = 2))
        data[a, (b + 4)] <- distance
    }
    cluster_name <- which(data[a, 5:8] == min(data[a, 5:8]))
    data[a, 4] <- cluster_name
}
# add the colours based on the cluster
for (a in 1:4) {
    data$colour[data$cluster == a] <- colours[a]
}
```

```r
plot(data$hours_norm ~ data$age_norm, col = data$colour, pch = 19)
text(data$age_norm, data$hours_norm, labels = data$name, cex = 0.7,
    pos = 3)
```

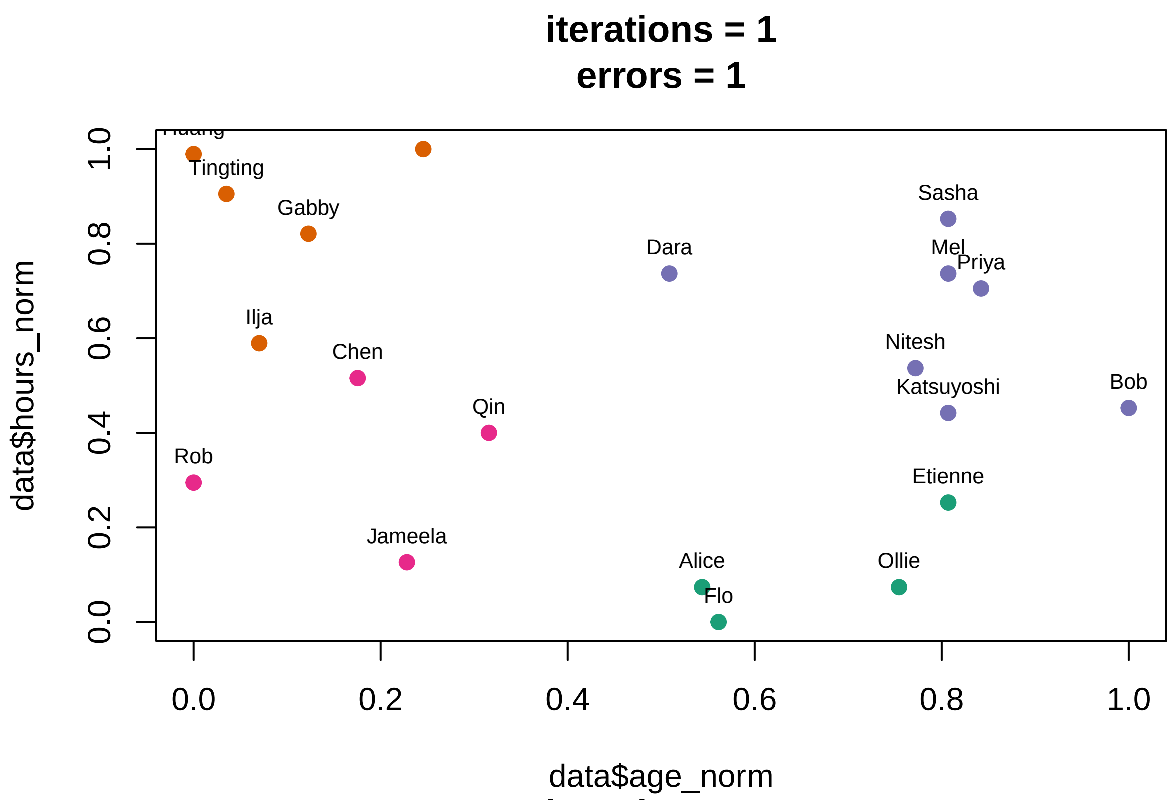


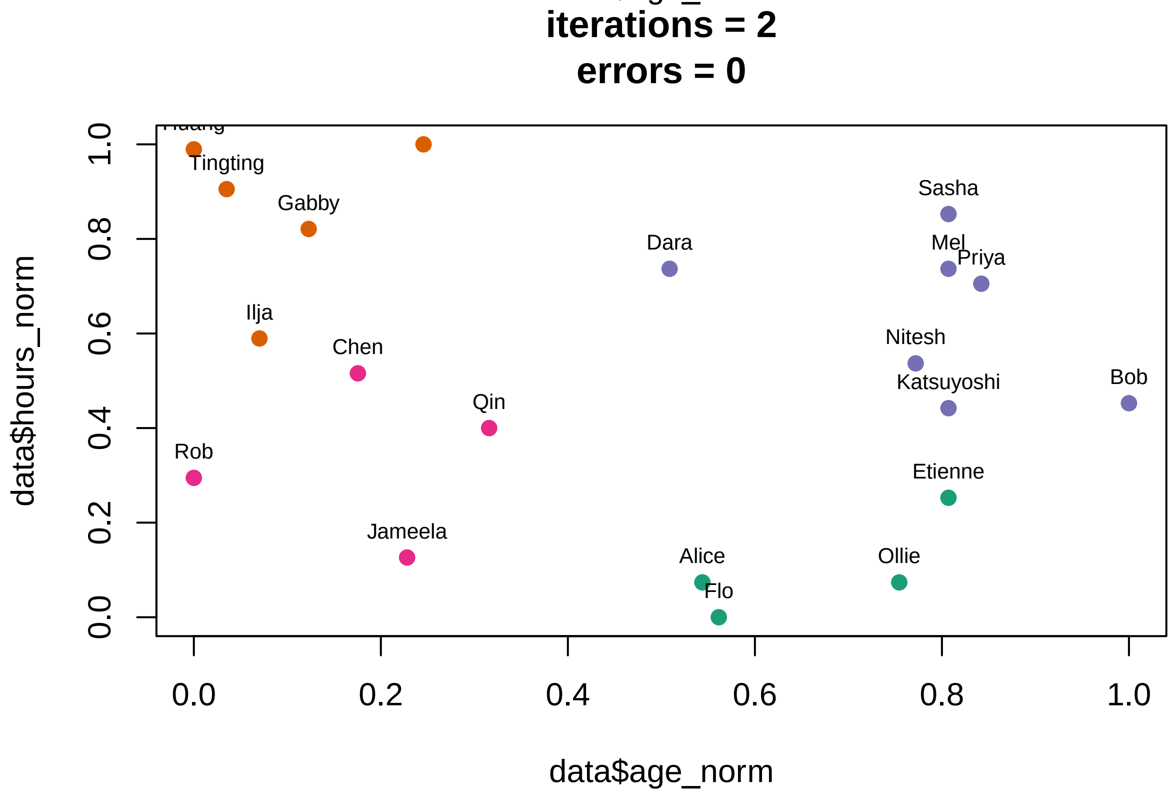
- To see what's going on, I would recommend plotting the data set every time you re-assign clusters and every time you re-compute centroids, so that you can visually follow the progress of the k-means algorithm. You can save a plot from within a for-loop using the "png" command, like this:

```r
png("filename.png", width = 15, height = 10, units = "cm", pointsize = 12,
    res = 300)
print(p)
dev.off()
```

I now want to calculate new centroid locations, and reassign each data point to their new centroid. Rather than selecting a specific number of iterations of these calculations, I need to repeat this until the datapoint-cluster assignments remain constant. To do this, I keep a record of the previous cluster assignments in each loop and then, once I've repeated the process, I calculate the absolute difference between cluster numbers for each row in the dataset. If this number is zero, then we can conclude that the results have converged.

```r
# calculate new clusters for each iterations keep the old
# clusters in old_cluster

# set the number of errors to 1 to initialise the loop
errors <- 1
# keep a record of the number of iterations of the loop
iterations <- 0

while (errors > 0) {

    # store the last round of cluster assignments
    data$old_cluster <- data$cluster
    # reinitialise everything
    data$distance_1 <- NA
    data$distance_2 <- NA
    data$distance_3 <- NA
    data$distance_4 <- NA
    data$cluster <- NA
    data$colour <- NA

    for (a in 1:nrow(data)) {
        data_age <- data[a, 2]
        data_hour <- data[a, 3]

        for (b in 1:4) {

            # calculate the new mean position for the
            # cluster
            cluster_data <- subset(data, data$old_cluster ==
                b)
            centroid_age <- mean(cluster_data$age_norm)
            centroid_hour <- mean(cluster_data$hours_norm)

            # recalculate the distances
            distance <- dist(matrix(c(data_age, centroid_age,
                data_hour, centroid_hour), ncol = 2))
            data[a, (b + 4)] <- distance
        }
        # cluster reassignment
        cluster_name <- which(data[a, 5:8] == min(data[a, 5:8]))
        data[a, 4] <- cluster_name
    }
    # calculate the error rates note that this is not the
    # number of datapoints which change cluster, but it
    # will be zero if nothing changes
    errors <- sum(abs(data$cluster - data$old_cluster))
    iterations <- iterations + 1

    # add the colours based on the cluster
    for (a in 1:4) {
        data$colour[data$cluster == a] <- colours[a]
    }
    plot(data$hours_norm ~ data$age_norm, col = data$colour,
        pch = 19, main = paste("iterations = ", iterations, "\nerrors = ",
            errors, sep = ""))
    text(data$age_norm, data$hours_norm, labels = data$name,
        cex = 0.7, pos = 3)
}
```





Plotting all stages will help you understand the algorithm. It will also help you understand what the problem is if something goes wrong. * How many iterations do you need? Remember this is an algorithm that converges quickly. It is likely that you won't need more than 10 iterations. But do check that there is no more "movement" at the end.

Hierarchical clustering

The caterer for the conference dinner calls Dr. Fischer. There is a problem with tables, and they don't know what kinds of tables they will be able to use yet. They will be able to set all 20 guests, but it is not clear whether it will be many smaller tables or fewer big tables.

Oh no! What to do?

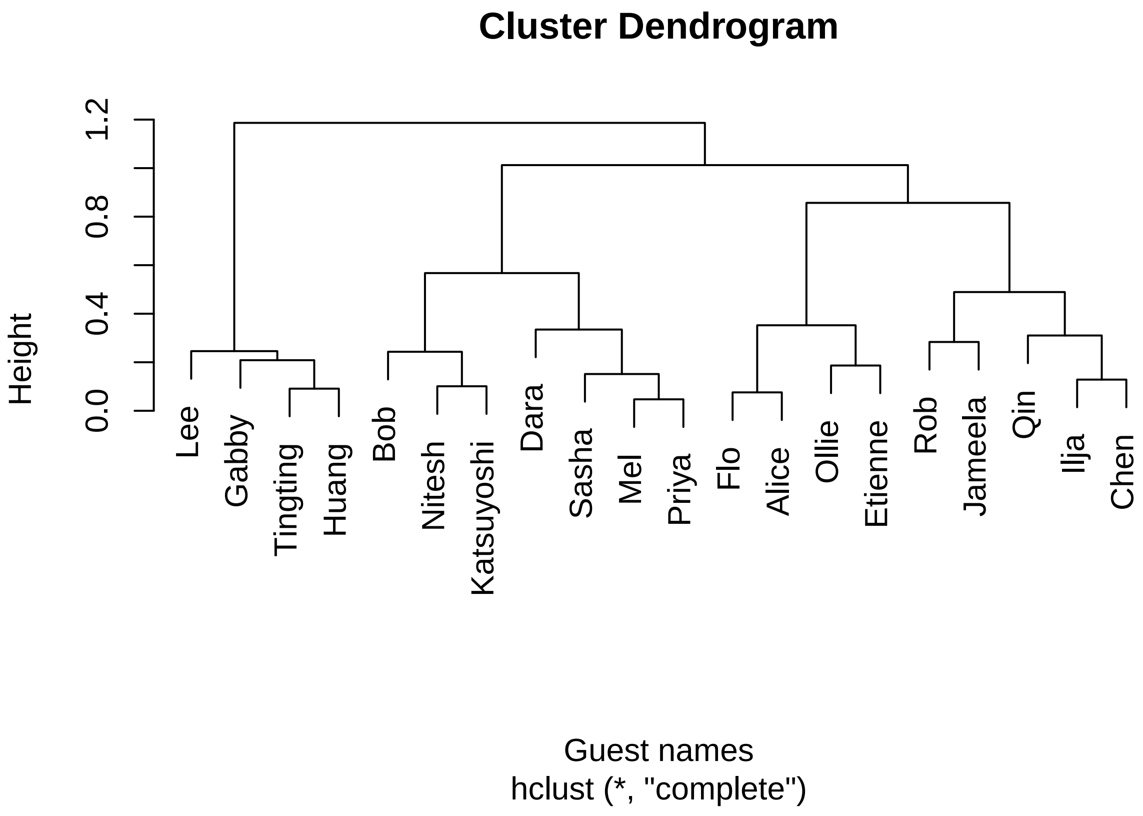
Well, fortunately you know the answer: Hierarchical clustering!

In this example, we are going to rely on one of the inbuilt R functions to perform the hierarchical clustering: hclust. This algorithm requires a distance matrix as the input which reflects the distance between every two pairs of data points.

- First, look through the hclust help page in R to confirm for yourself what the expected input is and the possible outputs you could expect.
- Now, using just the `age_norm` and `hours_norm` variables, perform the hierarchical clustering and store the information in a new R object.
- Plot the output of your clustering into a dendrogram of the type we saw in the lecture. Can you extract the correct names to label each branch?

In this code snippet, I first create the hclust object from clustering the `age_norm` and `hours_norm` variables.

```r
# cluster the data
h_cluster <- hclust(dist(data[, 2:3]))
# plot the data
plot(h_cluster, xlab = "Guest names", labels = data$names)
```



- Lets see if we can help the caterer out a bit more. He calls again to state that they have the options of 2, 3 or 9 tables. These tables can be extended with sliding panels underneath and can therefore accommodate any number of individuals up to 16. Using your dendrogram, how would you divide the 20 guests into 2, 3 and 9 tables?

I was a bit lazy and only did this by eye. I suggested for 2 tables, you would have one small table (Jameela, Nitesh, Mel and Ilja) and one large table for everybody else - although that's probably unlikely to be a popular conference dinner format. For 3 tables, I would keep that one small table but split the larger table into two: one with Hugo, Alice, Rob, Sasha, Etienne, Bob and Priya and the other with the remaining members. You could create 9 tables by cutting the dendrogram further down with most tables having 2 individuals and one tbale with three (Huang, Alice, Rob).

The point I was trying to get here is that you do not *ever* have to make a specific choice about cluster sizes, you can use the tree created by hierarchical clustering to decide on how you want to split the data - or indeed, if you want to split it at all. Maybe you just want to look at the relationships between the data without making permanent distinctions.

- How are you getting on? What differences do you see between hierarchical and k-means clustering?