Lab 23 - Clustering

Chad Murchison, Guy Twa, Eddie-Williams Owiredu

2023-11-06

Today's Lab

Today's lab will introduce you to two unsupervised clustering methods, k-means and hierarchical clustering. We'll learn how to cluster data sets using both methods, and compare the resulting clusterings. We'll also see how intial parameters can have important outcomes on clustering results.

Loading Packages and Data

No packages to load today, we're working in base `R` ! We'll be working with some randomly generated data, so be sure to set your seed beforehand.

```
set.seed(123456)
```

Our first data set `X1` will have two variables, with 50 observations. To create two groups, we'll shift the values for the first 25 observations.

```
X1 <- matrix(rnorm(50*2), ncol = 2)
X1[1:25 ,1] <- X1[1:25 ,1] + 3
X1[1:25 ,2] <- X1[1:25 ,2] - 4
```

Our second data `X2` set will be a bit more complicated with four distinct clusters of different sizes, at a 20:60:10:10 split.

```
X2 <- matrix(rnorm(100*2), ncol = 2)
X2[1:20 ,1] <- X2[1:20 ,1] + 3
X2[1:20 ,2] <- X2[1:20 ,2] - 4
X2[71:90 ,1] <- X2[81:90 ,1] - 1
X2[71:90 ,2] <- X2[81:90 ,2] + 1
X2[91:100 ,1] <- X2[91:100 ,1] - 6
X2[91:100 ,2] <- X2[91:100 ,2] + 5
```

K-means clustering

The `kmeans()` function from base `R` is a simple and easy to use implementation of a few k-means algorithms. It has just a few parameters

- `x` = input data
- `centers` = the number of clusters. (can also be predefined cluster centers)
- `iter.max` = the maximum number of clustering iterations

- `nstart` = if `centers` is a number, how many random starting points should be chosen
- `algorithm` = name of a particular clustering algorithm (this is beyond our scope for now)

Let's cluster our first data set.

```
X1_km_2 <- kmeans(X1,
                  centers = 2,  #K, the number of centers
                  nstart = 20   #The number of random sets, defaults to 1 but should alwa
ys be made bigger
                  )
```

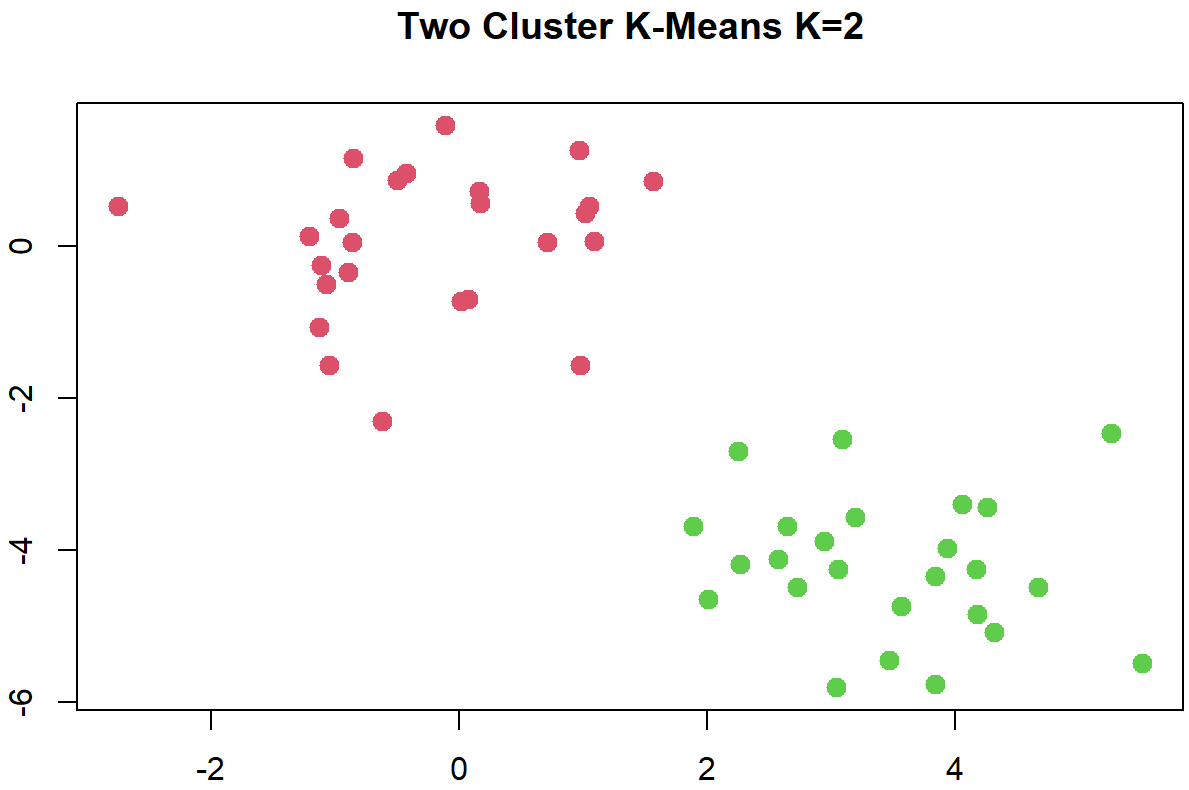
We can check the cluster assignments. These should reflect the 50:50 split

```
X1_km_2$cluster
```

```
##  [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1
## [39] 1 1 1 1 1 1 1 1 1 1 1 1
```

We can also plot the data by our two variables and color points by their clustering.

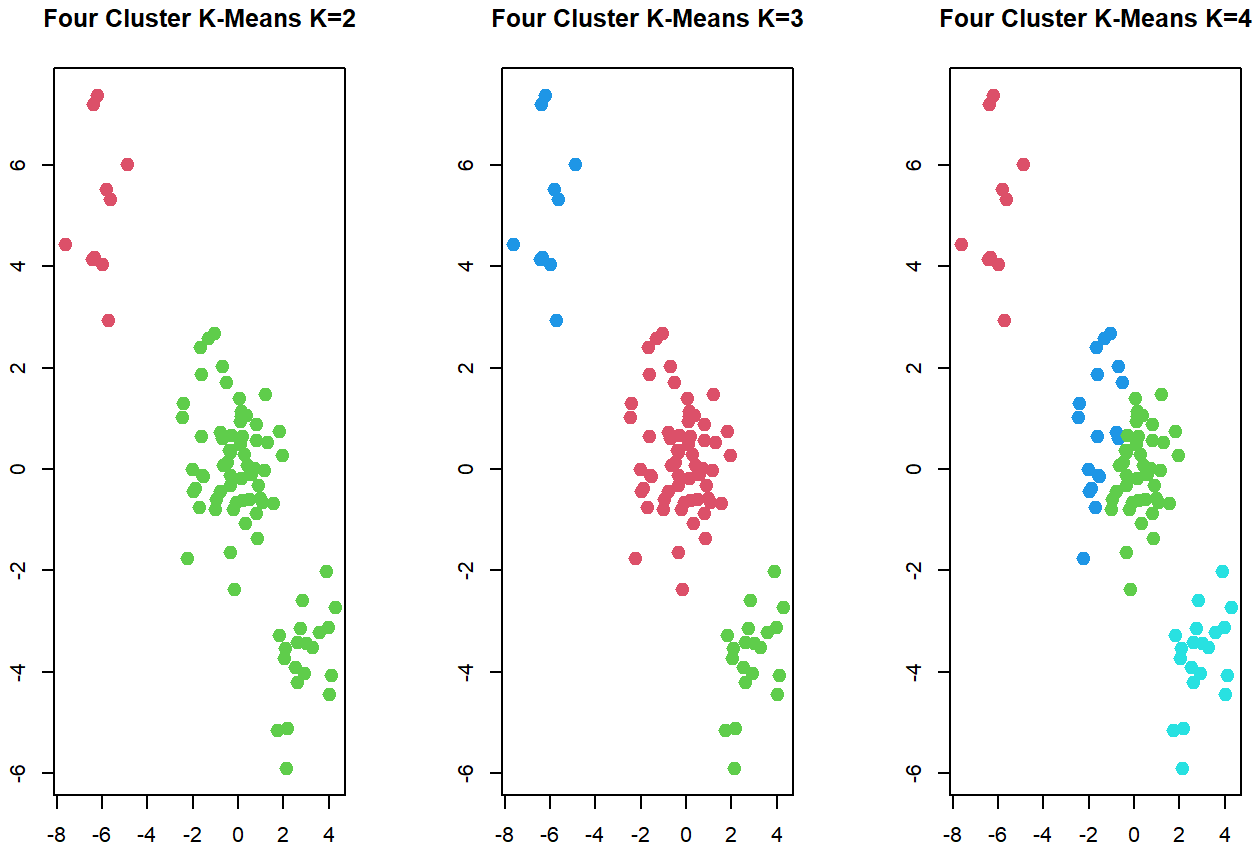
```
plot(X1, col = (X1_km_2$cluster + 1),
     main="Two Cluster K-Means K=2",
     xlab = "", ylab = "", pch = 20, cex = 2)
```



These points pretty nicely fall into our distinct clusters based on our shifts in the data.

Let's see how a similar process compares with our four cluster design. We'll start with the same set of 2 centers then try 3 and 4.

```
X2_km_2 <- kmeans(X2, center = 2, nstart = 20)
X2_km_3 <- kmeans(X2, center = 3, nstart = 20)
X2_km_4 <- kmeans(X2, center = 4, nstart = 20)


#Comparing all three side-by-side
par(mfrow = c(1,3))
plot(X2, col = (X2_km_2$cluster + 1),
     main="Four Cluster K-Means K=2",
     xlab = "", ylab = "", pch = 20, cex = 2)
plot(X2, col = (X2_km_3$cluster + 1),
     main="Four Cluster K-Means K=3",
     xlab = "", ylab = "", pch = 20, cex = 2)
plot(X2, col = (X2_km_4$cluster + 1),
     main="Four Cluster K-Means K=4",
     xlab = "", ylab = "", pch = 20, cex = 2)
```




```
par(mfrow = c(1,1))
```

We can see quite effectively how it heavily emphasizes distance starting with the 10 very removed data points.

It does start to lose resolution when some centers are relatively close, most notably with the final set of 10 points that are only shift -1,1

```
table(X2_km_4$cluster)
```

```
##
##  1  2  3  4
## 10 45 25 20
```

Let's take a look at the output for one of these clusterings.

```
X2_km_3
```

```
## K-means clustering with 3 clusters of sizes 70, 20, 10
##
## Cluster means:
##         [,1]       [,2]
## 1 -0.3564318  0.3941392
## 2  2.9055116 -3.7283429
## 3 -6.1136552  5.1191960
##
## Clustering vector:
##   [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3
##
## Within cluster sum of squares by cluster:
## [1] 165.59199  29.16000  23.24488
##  (between_SS / total_SS =  83.4 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

The important output is going to be the cluster centers (means) and the number of nodes in each cluster. As shown in the plot we can extract the vector of cluster assignments as other slots from the "available components"

In terms of the loss function we're minimizing with k-means, it's the total within cluster sum-of-squares (tot.withinss). This measures how tightly our observations are clustered around the mean. We access this and the within-cluster sum-of-squares for each cluster from our clustering object.

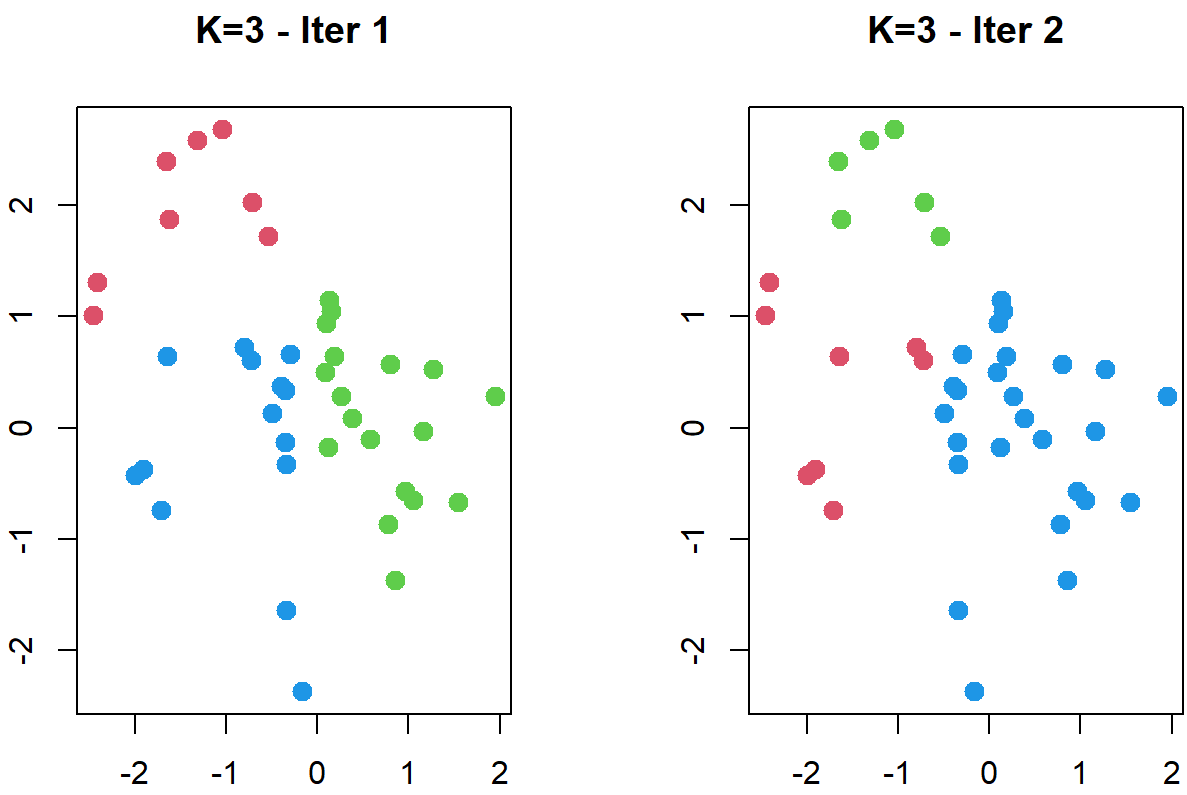
```
X2_km_3$tot.withinss; X2_km_3$withinss
```

```
## [1] 217.9969
```

```
## [1] 165.59199  29.16000  23.24488
```

Finally, we can see what happens when we set our starting random assignment sets too small, even on dummy data like this.

```
set.seed(1111)
X2_km_4_iter1 <- kmeans(X2[41:80,], center = 3, nstart = 1)
X2_km_4_iter2 <- kmeans(X2[41:80,], center = 3, nstart = 1)
par(mfrow = c(1,2))
plot(X2[41:80,], col = (X2_km_4_iter1$cluster + 1),
     main="K=3 - Iter 1",
     xlab = "", ylab = "", pch = 20, cex = 2)
plot(X2[41:80,], col = (X2_km_4_iter2$cluster + 1),
     main="K=3 - Iter 2",
     xlab = "", ylab = "", pch = 20, cex = 2)
```




```
par(mfrow = c(1,1))
```

In this case we get fairly different mixtures since there's so much overlap

Hierarchical Clustering

For hierarchical clustering, we'll use the same four center mixed data for hierarchical clustering.

The first step in clustering is to build our distance matrix using the `dist()` function. This matrix has pairwise distances for all observations. There are a number of different distance metrics you can choose from. The most common is “euclidean”, while “manhattan” is probably the 2nd most common. It can be tough to choose a particular distance metric, and it’ll often depend on the nature of your data for what works best.

```
X2_distance <- dist(X2, method = "euclidean")
```

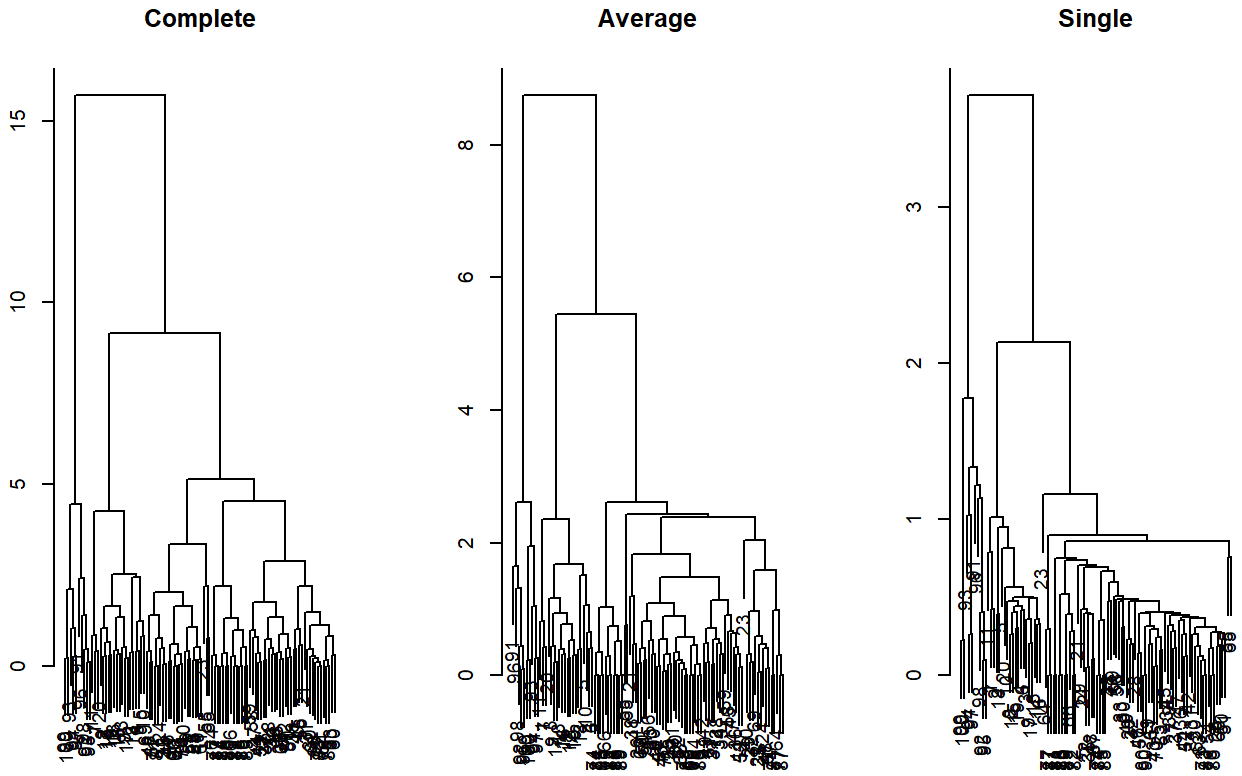
Next we’ll build the hierarchical tree using the `hclust()` function. It takes a distance matrix as input and the most important parameter is `method` , which determines the linkage method used. There are a number of other linkage methods that all come with their own quirks, we’ll try three of them here.

- Complete = merge the two clusters whose distance itself has the smallest diameter (so merge on center)
- Single - merge the two clusters whose two **closest members** have the smallest distance (merge on edges by individuals)
- Average - essentially a mixture of the two, uses dot-products in vector space

```
X2_hclust_comp <- hclust(X2_distance, method = "complete")
X2_hclust_single = hclust(X2_distance, method = "single")
X2_hclust_avg = hclust(X2_distance, method = "average")
```

The best way to compare these is by plotting and looking at the resulting trees.

```
par(mfrow = c(1,3))
plot(X2_hclust_comp, main = "Complete",
     xlab = "", ylab = "", sub = "", cex = 0.9)
plot(X2_hclust_avg, main = "Average",
     xlab = "", ylab = "", sub = "", cex = 0.9)
plot(X2_hclust_single, main = "Single",
     xlab = "", ylab = "", sub = "", cex = 0.9)
```




```
par(mfrow = c(1,1))
```

We can also determine cluster labels using the cutree() function.

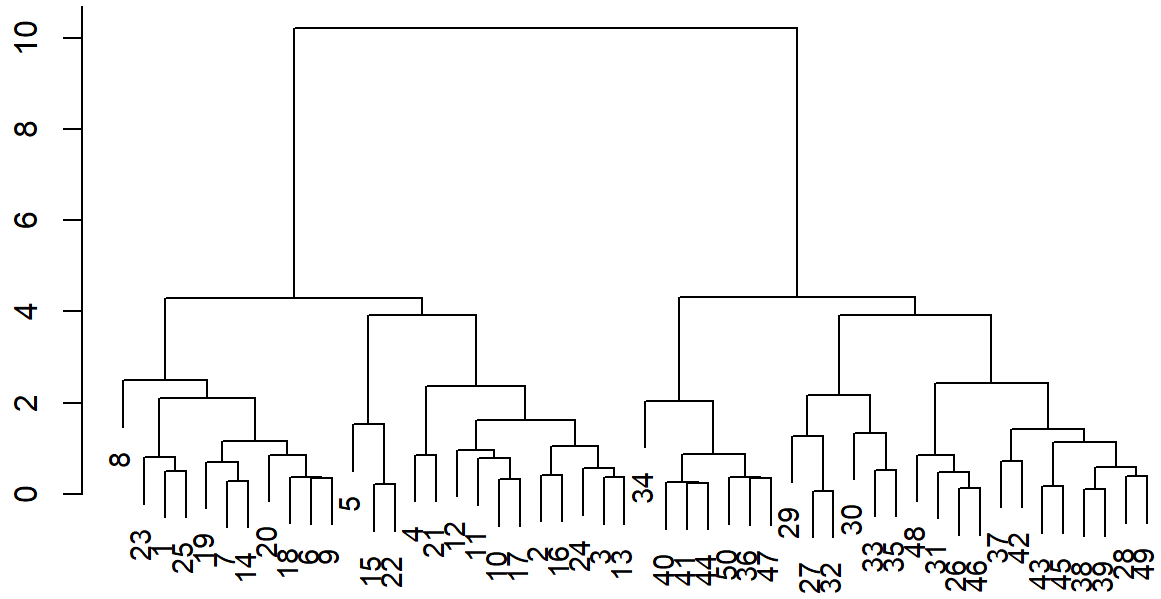
```
cut_points <- data.frame(complete = cutree(X2_hclust_comp, 4),
                         average = cutree(X2_hclust_avg, 4),
                         single = cutree(X2_hclust_single, 4),
                         true = c(rep(1, 20), rep(2, 60), rep(3, 10), rep(4, 10)))

invisible(
lapply(seq_len(ncol(cut_points)-1), function(.col){
  .dat <- data.frame(pred = cut_points[,.col], true = cut_points$true)
  writeLines(paste0("\n\n", colnames(cut_points)[.col]))
  print(table(.dat$pred, .dat$true))
  return(NULL)})
  )
```

```
##
##
## complete
##
##      1  2  3  4
##   1 20  0  0  0
##   2  0 38  8  0
##   3  0 22  2  0
##   4  0  0  0 10
##
##
## average
##
##      1  2  3  4
##   1 20  0  0  0
##   2  0 60 10  0
##   3  0  0  0  5
##   4  0  0  0  5
##
##
## single
##
##      1  2  3  4
##   1 20  0  0  0
##   2  0 60 10  0
##   3  0  0  0  8
##   4  0  0  0  2
```

We see that complete and average do fairly well but single falls apart pretty quickly. Much like with k-means, the clustering has difficulty with the highly mixed center clusters

To more easily represent a smaller dendrogram, let's look at the original dataset

```
X1_distance <- dist(X1, method = "euclidean")
X1_hclust <- hclust(X1_distance, method = "complete")
plot(X1_hclust, main = "X1 Complete",
     xlab = "", ylab = "", sub = "", cex = 0.9)
```




```
table(cutree(X1_hclust, 2), c(rep(1,25), rep(2,25)))
```

```
##
##      1  2
##   1 25  0
##   2  0 25
```