PRACTICAL 8

Aim:

The aim of this practical is to illustrate how both hierarchical and *k*-means clustering models can be fit in R. We will also discuss working with simulated data.

Before you start

Load the `tidyverse`.

The clustering functions required for both hierarchical clustering and *k*-means clustering are contained in the `stats()` package, which is always automatically loaded when R / RStudio starts, so there is no need to install or load any additional packages.

Instructions:

This document contains discussions of some basic concepts, and instructions which you should execute in RStudio. **Text in bold** indicate actions that are to be taken within RStudio. `Text highlighted in grey` shows the code you should enter in RStudio.

Due date:

For Tutorial 8, it will be assumed that you have mastered the basic concepts discussed here.

Simulated data

1. Clustering is an unsupervised learning technique. We therefore don't have a target variable, and we also don't have a "true" grouping to learn from. In practice we also will not know the "true" number of clusters in a dataset!

   In this practical, we will therefore work with simulated (i.e. artificially generated) data. We will specifically create a dataset of 50 observations in 2 distinct clusters. We will also use only 2 attributes, so that we can easily visualise the data.

   Working with simulated data means that we know the true clusters, and we can therefore evaluate different clustering solutions to see how well they do, compared to the truth.

2.1 **Start by simulating 100 observations from a normal distribution, and store this in an object called `mydata`.** The first 50 observations should be used as the values for the first attribute (*X*) and the next 50 observations should be used as the values for the second attribute (*Y*). The details of the generation process – using the function `rnorm()` – is not important (and is beyond the scope of this course). You can simply type in the code below. **You should name the first attribute *X* and the second attribute *Y*, and make sure that your `mydata` object is a data frame. Also take note of the fact that you should set a seed before generating the data, to ensure that your results are reproducible.**

```
set.seed(2)
mydata <- matrix(rnorm(50*2), ncol = 2)
colnames(mydata) <- c("X","Y")
mydata <- data.frame(mydata)
```

2.2 **View the `mydata` object created in Step 2.1 above.**

```
mydata
```

2.3 **Now plot the `mydata` object.**

```
ggplot(data = mydata, mapping = aes(x = X, y = Y)) +
geom_point()
```

3. Note from the plot created in Step 2.3 that there are not yet any obvious clusters in the data. This is because all the data points were created from the same distribution.

3.1 **You should now create distinct clusters by moving the *X* values for the first 25 observations 3 units to the right, and the corresponding *Y* values 4 units down.**

```
mydata[1:25, 1] <- mydata[1:25, 1] + 3
mydata[1:25, 2] <- mydata[1:25, 2] – 4
```

3.2 **Plot the `mydata` object again.**

```
ggplot(data = mydata, mapping = aes(x = X, y = Y)) +
geom_point()
```

Note how there are now 2 fairly distinct clusters.

3.3 Due to the way the data was simulated, we know that the first 25 observations are in one cluster and the next 25 in another cluster.

Add an attribute `clus` to the `mydata` object, containing the cluster designation.

You can simply call the clusters `C1` and `C2`. Remember that `clus` should be a factor.

```
clus <- as.factor(c(rep(1,25), rep(2, 25)))
mydata <- cbind(mydata, clus)
```

3.4 **Plot the `mydata` object once again, but this time indicating the cluster membership by plotting points from different clusters in different shapes and colours.**

```
ggplot(data = mydata, mapping = aes(x = X, y = Y, colour =
clus, shape = clus)) + geom_point()
```

---

k-means clustering

4. Since we simulated the data, we know that the true value of $k = 2$.

4.1 **Perform *k*-means clustering with $k = 2$. Save the model in an object named `twokm`.**

```
twokm <- kmeans(x = mydata[,-3], centers = 2, nstart = 20)
```

The `centers` argument specifies the number of clusters you would like to create. The `nstart` argument specifies the number of starting assignments that should be used. Recall that the *k*-means algorithm starts by creating *k* initial cluster centers (usually randomly). Specifying a `nstart` value greater than 1 allows the use of multiple random starting assignments. Only the best results are reported, based on the minimum total within-cluster sum of squares. It is advisable to always run *k*-means clustering with a large value of `nstart`, such as `nstart` = 20.

4.2 **View the contents of the `twokm` object created in Step 4.1**

```
twokm
```

Note that this provides the size of each cluster (i.e. the number of observations in each), the final cluster means, the cluster assignments and also the within cluster sum of squares for each cluster.

4.3 **Create a new data frame named `twodat`, which contains the original data but also a column named `predclus` which contains the assigned clusters from the `twokm` model.**

```
twodat <- data.frame(cbind(mydata, predclus =
as.factor(twokm$cluster)))
```

4.4 **Compare the actual clusters and the clusters assigned by the `twokm` model by cross-tabulating these values, and also by plotting the data in a scatterplot.** You can use colour to indicate the cluster assignments, and shapes to indicate the actual clusters.

```
table(twodat$clus, twodat$predclus)
ggplot(data = twodat, mapping = aes(x = X, y = Y, colour = predclus, shape = clus)) + geom_point()
```

Note that the *k*-means clustering algorithm with $k = 2$ is successful in identifying the correct clusters.

5. Although we know that there are 2 "true" clusters in our simulated dataset, remember that when performing clustering we usually do not know the true number of clusters. We will therefore see what happens when we perform *k*-means clustering on our data with $k = 3$.

5.1 **Change the seed value to 4, and then perform *k*-means clustering with $k = 3$. Save the model in an object named `threekm`.**

```
set.seed(4)
threekm <- kmeans(x = mydata[,-3], centers = 3, nstart = 20)
threekm
```

5.2 **Investigate the cluster assignments by following steps similar to those in 4.3 and 4.4 above.**

```
threedat <- data.frame(cbind(mydata, predclus = as.factor(threekm$cluster)))
table(threedat$clus, threedat$predclus)
ggplot(data = threedat, mapping = aes(x = X, y = Y, colour = predclus, shape = clus)) + geom_point()
```

Note that while one of the clusters is mostly correctly identified, the other cluster is split into two.

Hierarchical clustering

We will also investigate the performance of a hierarchical clustering algorithm (with different linkage functions) on the same simulated dataset.

6. The first linkage function we will consider, is complete linkage. Recall that this considers the distance between any two clusters to be the maximum distance from any point in one cluster to any point in the other.

6.1 **Perform hierarchical clustering on the `mydata` dataset, using complete linkage. Save the model in an object called `hiercom`.**

Note that the `hclust()` function requires a distance function as input. Make sure that you don't include the actual cluster assignments (which we added as a 3rd column to the dataset).

```
hiercom <- hclust(dist(mydata[,-3]), method = "complete")
```

You can look at the help file for the `dist()` function to see what this function does. Since we did not specify a method within the `dist()` function, the distance measure used by default is Euclidean distance. Other distance measures such as maximum distance, Manhattan distance or Minkowski distance can also be explicitly specified.

6.2 A hierarchical clustering model is best viewed in a dendrogram.

Plot a dendrogram for the `hiercom` object created in Step 6.1

```
plot(hiercom, main = "Complete Linkage", xlab = "", sub = "")
```

6.3 Recall that an advantage of hierarchical clustering is that a single dendrogram can be used to obtain any number of clusters. In practice, a visual inspection of a dendrogram can often suggest a sensible number of clusters to use. The `cutree()` function can be used to cut a tree object that results from an `hclust` object. Specifying the argument `k = 2` for instance, will return the cluster assignments if 2 clusters are to be formed.

Obtain the cluster assignments if 2 clusters are to be created from the `hiercom` model. Store these cluster assignments in an object named `compclus`.

```
compclus <- cutree(hiercom, k = 2)
compclus
```

6.4 **Investigate the cluster assignments by following steps similar to those in 4.3 and 4.4 above.**

```
compdat <- data.frame(cbind(mydata, predclus = as.factor(compclus)))
table(compdat$clus, compdat$predclus)
ggplot(data = compdat, mapping = aes(x = X, y = Y, colour = predclus, shape = clus)) + geom_point()
```

7. Repeat Step 6 above, but using average linkage. Save the model in an object named `hieravg`.

```
hieravg <- hclust(dist(mydata[,-3]), method = "average")
plot(hieravg, main = "Average Linkage", xlab = "", sub = "")
avgclus <- cutree(hieravg, k = 2)
avgclus
avgdat <- data.frame(cbind(mydata, predclus = as.factor(avgclus)))
table(avgdat$clus, avgdat$predclus)
ggplot(data = avgdat, mapping = aes(x = X, y = Y, colour = predclus, shape = clus)) + geom_point()
```

8. **Repeat Step 6 above, but using single linkage. Save the model in an object named `hiersin`.**

```
hiersin <- hclust(dist(mydata[,-3]), method = "single")
plot(hiersin, main = "Single Linkage", xlab = "", sub = "")
sinclus <- cutree(hiersin, k = 2)
sinclus
sindat <- data.frame(cbind(mydata, predclus = as.factor(sinclus)))
table(sindat$clus, sindat$predclus)
ggplot(data = sindat, mapping = aes(x = X, y = Y, colour = predclus, shape = clus)) + geom_point()
```

Discussion and comparison of results

For the simulated dataset we considered, the *k*-means algorithm correctly assigned the observations to their two clusters (as long as it was specified that 2 clusters should be used). Hierarchical clustering with complete linkage was also able to separate the observations into their correct clusters, with average linkage performing relatively well too. However, for this dataset, a hierarchical clustering algorithm with single linkage failed to correctly identify the two clusters.

Further note that in this example we did not standardise the data first, since the data was on the same scale when generated. In practice, careful consideration should be given to the standardisation of attributes before clustering.

Reference:

James, G., Witten, D., Hastie, T. & Tibshirani, R. (2013). *An Introduction to Statistical Learning with Applications in R*. Springer, New York. Available online at: https://www.statlearning.com