An Introduction to Machine Learning

Sudhakaran Prabakaran, Matt Wayland and Chris Penfold

2017-09-10

Contents

Chapter 1

About the course

1.1 Overview

Machine learning gives computers the ability to learn without being explicitly programmed. It encompasses a broad range of approaches to data analysis with applicability across the biological sciences. Lectures will introduce commonly used algorithms and provide insight into their theoretical underpinnings. In the practicals students will apply these algorithms to real biological data-sets using the R language and environment.

During this course you will learn about:

- Some of the core mathematical concepts underpinning machine learning algorithms: matrices and linear algebra; Bayes' theorem.
- Classification (supervised learning): partitioning data into training and test sets; feature selection; logistic regression; support vector machines; artificial neural networks; decision trees; nearest neighbours, cross-validation.
- Exploratory data analysis (unsupervised learning): dimensionality reduction, anomaly detection, clustering.

After this course you should be able to:

- Understand the concepts of machine learning.
- Understand the strengths and limitations of the various machine learning algorithms presented in this course.
- Select appropriate machine learning methods for your data.
- Perform machine learning in R.

1.2 Registration

Bioinformatics Training: An Introduction to Machine Learning

1.3 Prerequisites

- Some familiarity with R would be helpful.
- For an introduction to R see An Introduction to Solving Biological Problems with R course.

1.4 Github

bioinformatics-training/intro-machine-learning

1.5 License

GPL-3

1.6 Contact

If you have any **comments**, **questions** or **suggestions** about the material, please contact the authors: Sudhakaran Prabakaran, Matt Wayland and Chris Penfold.

1.7 Colophon

This book was produced using the **bookdown** package (Xie, 2017), which was built on top of R Markdown and **knitr** (Xie, 2015).

Chapter 2

Introduction

You can label chapter and section titles using `{#label}` after them, e.g., we can reference Chapter 2. If you do not manually label them, there will be automatic labels anyway, e.g., Chapter **??**.

Figures and tables with captions will be placed in `figure` and `table` environments, respectively.

```
par(mar = c(4, 4, .1, .1))
plot(pressure, type = 'b', pch = 19)
```

Reference a figure by its code chunk label with the `fig:` prefix, e.g., see Figure 2.1. Similarly, you can reference tables generated from `knitr::kable()`, e.g., see Table 2.1.

```
knitr::kable(
  head(iris, 20), caption = 'Here is a nice table!',
  booktabs = TRUE
)
```

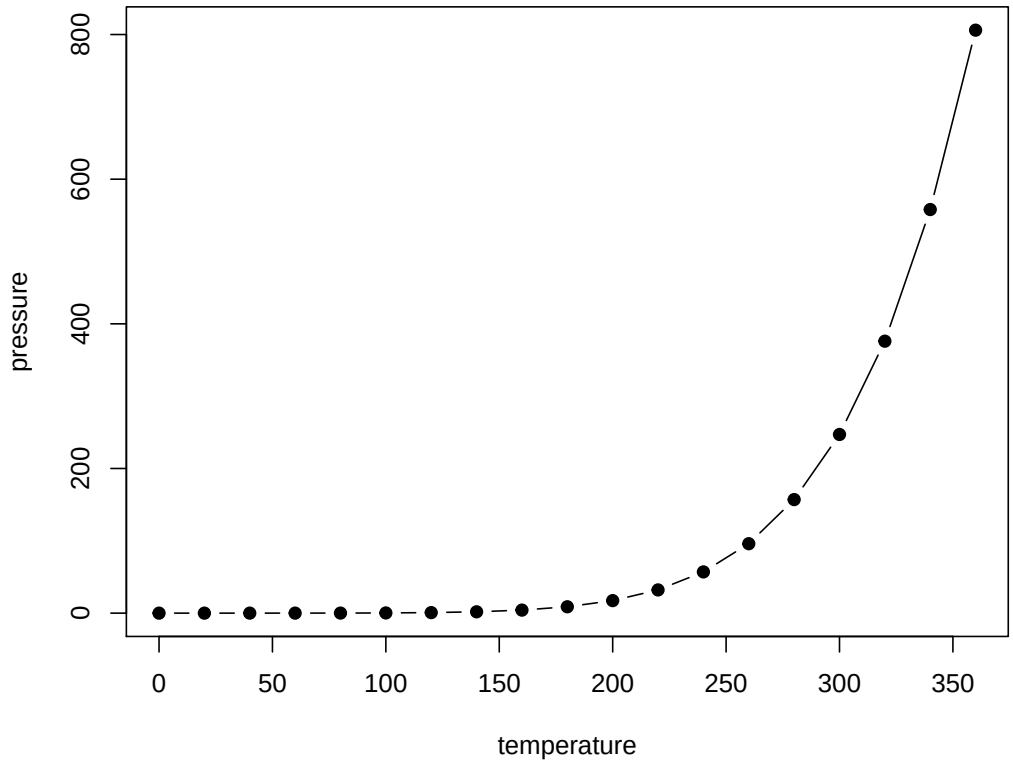

Figure 2.1: Here is a nice figure!

Table 2.1: Here is a nice table!

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa
5.1	3.8	1.5	0.3	setosa

Chapter 3

Linear models and matrix algebra

3.1 Exercises

Solutions to exercises can be found in appendix B

Chapter 4

Linear and non linear logistic regression

4.1 Exercises

Solutions to exercises can be found in appendix C.

Chapter 5

Nearest neighbours

Parallel processing with doMC registerDoMC() getDoParWorkers()

5.1 Example one

5.2 Example two

5.3 Exercises

Solutions to exercises can be found in appendix D.

Chapter 6

Decision trees and random forests

6.1 Exercises

Solutions to exercises can be found in appendix E.

Chapter 7

Support vector machines

7.1 Exercises

Solutions to exercises can be found in appendix F

Chapter 8

Artificial neural networks

8.1 Exercises

Solutions to exercises can be found in appendix G.

Chapter 9

Dimensionality reduction

9.1 Linear Dimensionality Reduction

9.1.1 Principle Component Analysis

9.1.2 Horeshoe effect

9.2 Nonlinear Dimensionality Reduction

9.2.1 t-SNE

9.2.2 Gaussian Process Latent Variable Models

9.2.3 GPLVMs with informative priors

9.3 Exercises

Solutions to exercises can be found in appendix H.

Chapter 10

Clustering

10.1 Introduction

What is clustering - add figure showing idea of minimizing intra-cluster variation and maximizing inter-cluster variation.

Hierarchic (produce dendrogram) vs partitioning methods

- Hierarchic agglomerative
- k-means
- DBSCAN

10.2 Distance metrics

dist function cor as.dist(1-cor(x))

Minkowski distance:

$$distance\,(x, y, p) = \left(\sum_{i=1}^{n} abs(x_i - y_i)^p \right)^{1/p} \tag{10.1}$$

Graphical explanation of euclidean, manhattan and max (Chebyshev?)

10.2.1 Image segmentation

10.3 Hierarchic agglomerative

Get to see clusters for all number of clusters k

10.3.1 Linkage algorithms

Single linkage - nearest neighbours linkage Complete linkage - furthest neighbours linkage Average linkage - UPGMA (Unweighted Pair Group Method with Arithmetic Mean)

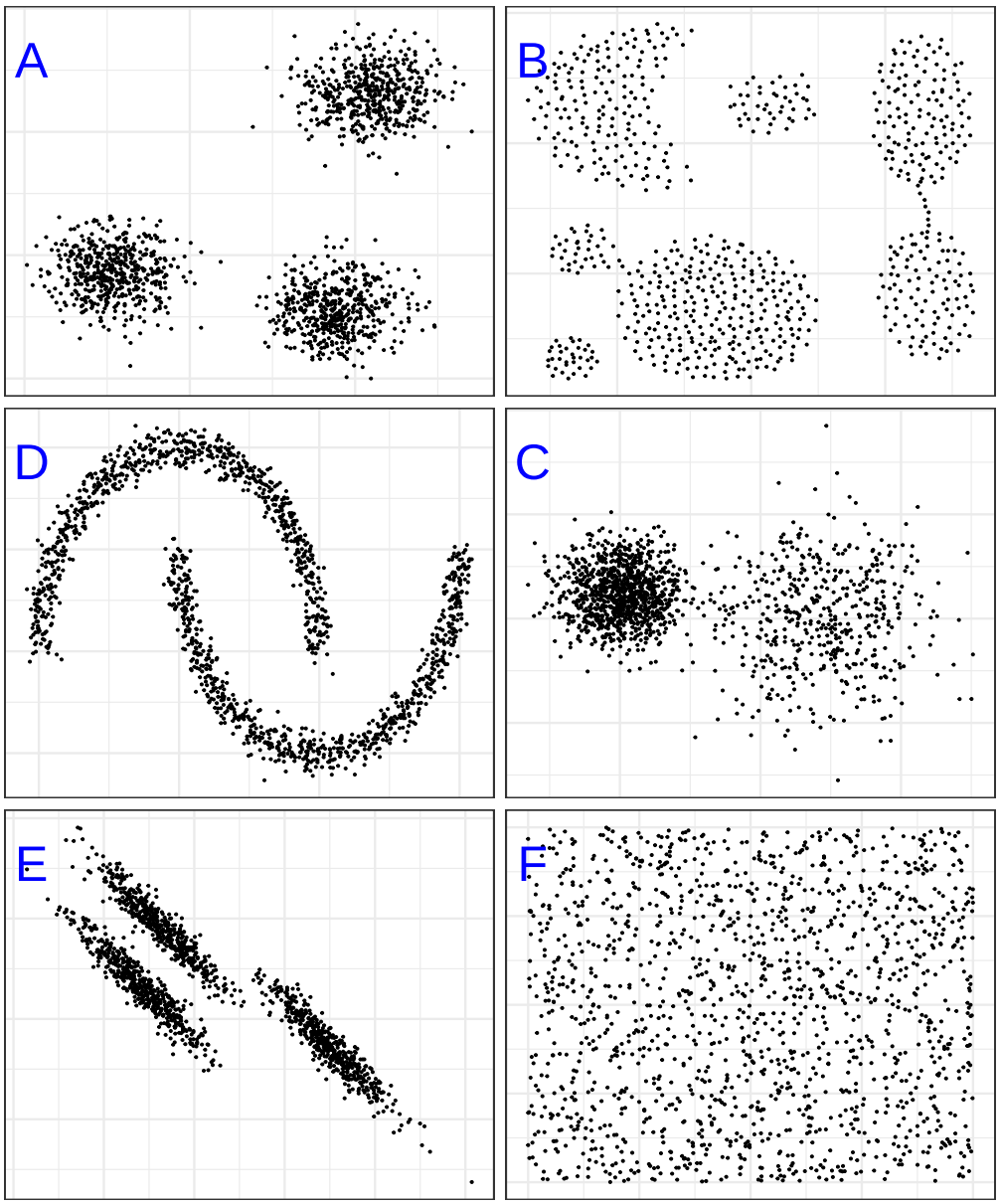

Figure 10.1: Example clusters. **A**, *blobs*; **B**, *aggregation* [@Gionis2007]; **C**, *noisy moons*; **D**, *different density*; **E**, *anisotropic distributions*; **F**, *no structure*.

Table 10.1: Example distance matrix

	A	B	C	D
B	2			
C	6	5		
D	10	10	5	
E	9	8	3	4

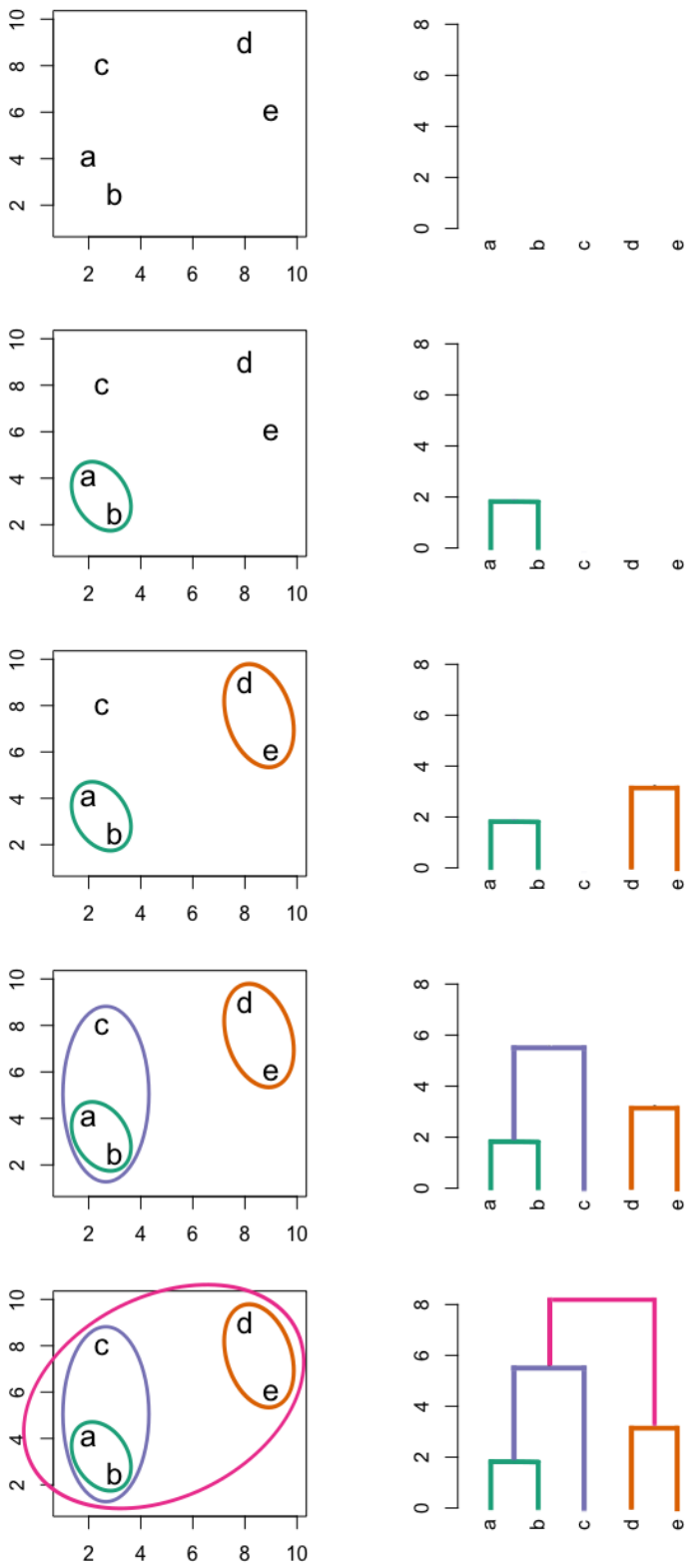

Figure 10.2: Building a dendrogram using hierarchic agglomerative clustering.

Table 10.2: Merge distances for objects in the example distance matrix using three different linkage methods.

Groups	Single	Complete	Average
A,B,C,D,E	0	0	0
(A,B),C,D,E	2	2	2
(A,B),(C,E),D	3	3	3
(A,B)(C,D,E)	4	5	4.5
(A,B,C,D,E)	5	10	8

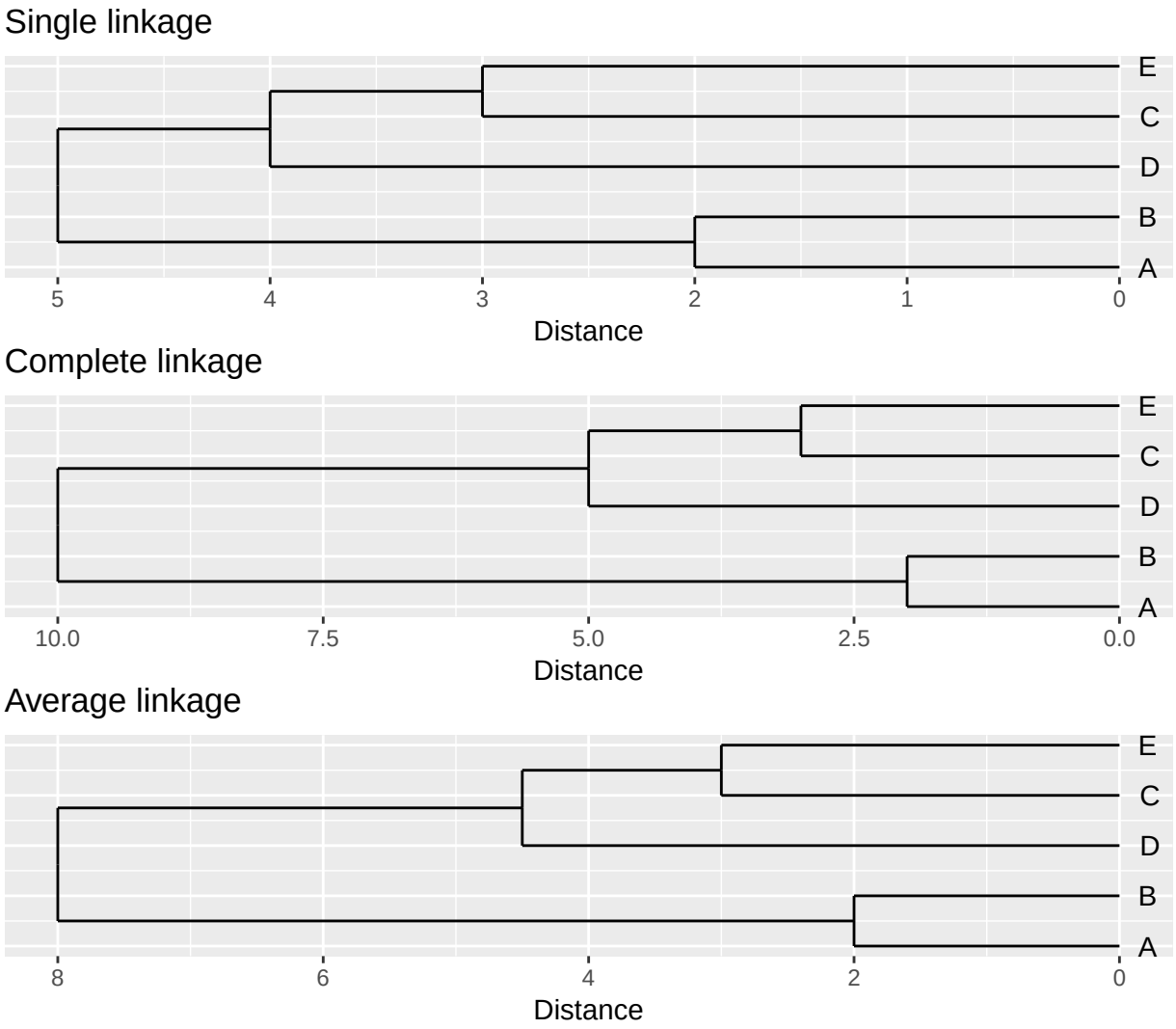

Figure 10.3: Dendrograms for the example distance matrix using three different linkage methods.

10.3.2 Example: clustering synthetic data sets

10.3.2.1 Step-by-step instructions

1. Load required packages.

```
library(RColorBrewer)
library(dendextend)
```

```
##
## ---------------------
## Welcome to dendextend version 1.5.2
## Type citation('dendextend') for how to cite the package.
##
## Type browseVignettes(package = 'dendextend') for the package vignette.
## The github page is: https://github.com/talgalili/dendextend/
##
## Suggestions and bug-reports can be submitted at: https://github.com/talgalili/dendextend/issues
## Or contact: <tal.galili@gmail.com>
##
##  To suppress this message use:  suppressPackageStartupMessages(library(dendextend))
## ---------------------
```

```
##
## Attaching package: 'dendextend'
```

```
## The following object is masked from 'package:ggdendro':
##
##     theme_dendro
```

```
## The following object is masked from 'package:stats':
##
##     cutree
```

```
library(ggplot2)
library(GGally)
```

2. Retrieve a palette of eight colours.

```
cluster_colours <- brewer.pal(8,"Dark2")
```

3. Read in data for **blobs** example.

```
blobs <- read.csv("data/example_clusters/blobs.csv", header=F)
```

4. Create distance matrix using Euclidean distance metric.

```
d <- dist(blobs[,1:2])
```

5. Perform hierarchical clustering using the **average** agglomeration method and convert the result to an object of class **dendrogram**. A **dendrogram** object can be edited using the advanced features of the **dendextend** package.

```
dend <- as.dendrogram(hclust(d, method="average"))
```

6. Cut the tree into three clusters

```
clusters <- cutree(dend,3,order_clusters_as_data=F)
```

7. The vector **clusters** contains the cluster membership (in this case *1*, *2* or *3*) of each observation (data point) in the order they appear on the dendrogram. We can use this vector to colour the branches of

the dendrogram by cluster.

```
dend <- color_branches(dend, clusters=clusters, col=cluster_colours[1:3])
```

8. We can use the **labels** function to annotate the leaves of the dendrogram. However, it is not possible to create legible labels for the 1,500 leaves in our example dendrogram, so we will set the label for each leaf to an empty string.

```
labels(dend) <- rep("", length(blobs[,1]))
```

9. If we want to plot the dendrogram using **ggplot**, we must convert it to an object of class **ggdend**.

```
ggd <- as.ggdend(dend)
```

10. The **nodes** attribute of **ggd** is a data.frame of parameters related to the plotting of dendogram nodes. The **nodes** data.frame contains some NAs which will generate warning messages when **ggd** is processed by **ggplot**. Since we are not interested in annotating dendrogram nodes, the easiest option here is to delete all of the rows of **nodes**.

```
ggd$nodes <- ggd$nodes[!(1:length(ggd$nodes[,1])),]
```

11. We can use the cluster membership of each observation contained in the vector **clusters** to assign colours to the data points of a scatterplot. However, first we need to reorder the vector so that the cluster memberships are in the same order that the observations appear in the data.frame of observations. Fortunately the names of the elements of the vector are the indices of the observations in the data.frame and so reordering can be accomplished in one line.

```
clusters <- clusters[order(as.numeric(names(clusters)))]
```

12. We are now ready to plot a dendrogram and scatterplot. We will use the **ggmatrix** function from the **GGally** package to place the plots side-by-side.

```
plotList <- list(ggplot(ggd),
                 ggplot(blobs, aes(V1,V2)) +
                   geom_point(col=cluster_colours[clusters], size=0.2)
                 )

pm <- ggmatrix(
  plotList, nrow=1, ncol=2, showXAxisPlotLabels = F, showYAxisPlotLabels = F,
  xAxisLabels=c("dendrogram", "scatter plot")
) + theme_bw()

pm
```

10.3.2.2 Clustering of other synthetic data sets

```
aggregation <- read.table("data/example_clusters/aggregation.txt")
noisy_moons <- read.csv("data/example_clusters/noisy_moons.csv", header=F)
diff_density <- read.csv("data/example_clusters/different_density.csv", header=F)
aniso <- read.csv("data/example_clusters/aniso.csv", header=F)
no_structure <- read.csv("data/example_clusters/no_structure.csv", header=F)

hclust_plots <- function(data_set, n){
  d <- dist(data_set[,1:2])
  dend <- as.dendrogram(hclust(d, method="average"))
  clusters <- cutree(dend,n,order_clusters_as_data=F)
  dend <- color_branches(dend, clusters=clusters, col=cluster_colours[1:n])
```

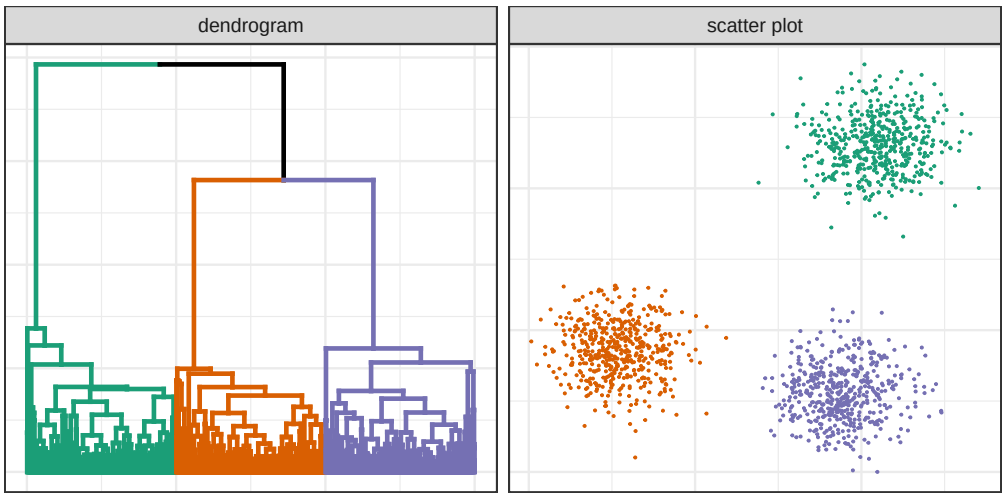

Figure 10.4: Hierarchical clustering of the blobs data set.

```
  clusters <- clusters[order(as.numeric(names(clusters)))]
  labels(dend) <- rep("", length(data_set[,1]))
  ggd <- as.ggdend(dend)
  ggd$nodes <- ggd$nodes[!(1:length(ggd$nodes[,1])),]
  plotPair <- list(ggplot(ggd),
    ggplot(data_set, aes(V1,V2)) +
      geom_point(col=cluster_colours[clusters], size=0.2))
  return(plotPair)
}

plotList <- c(
  hclust_plots(aggregation, 7),
  hclust_plots(noisy_moons, 2),
  hclust_plots(diff_density, 2),
  hclust_plots(aniso, 3),
  hclust_plots(no_structure, 3)
)

pm <- ggmatrix(
  plotList, nrow=5, ncol=2, showXAxisPlotLabels = F, showYAxisPlotLabels = F,
  xAxisLabels=c("dendrogram", "scatter plot"),
  yAxisLabels=c("aggregation", "noisy moons", "different density", "anisotropic", "no structure")
) + theme_bw()

pm
```

10.3.3 Example: gene expression profiling of human tissues

10.3.3.1 Basics

Load required libraries

```
library(RColorBrewer)
library(dendextend)
```

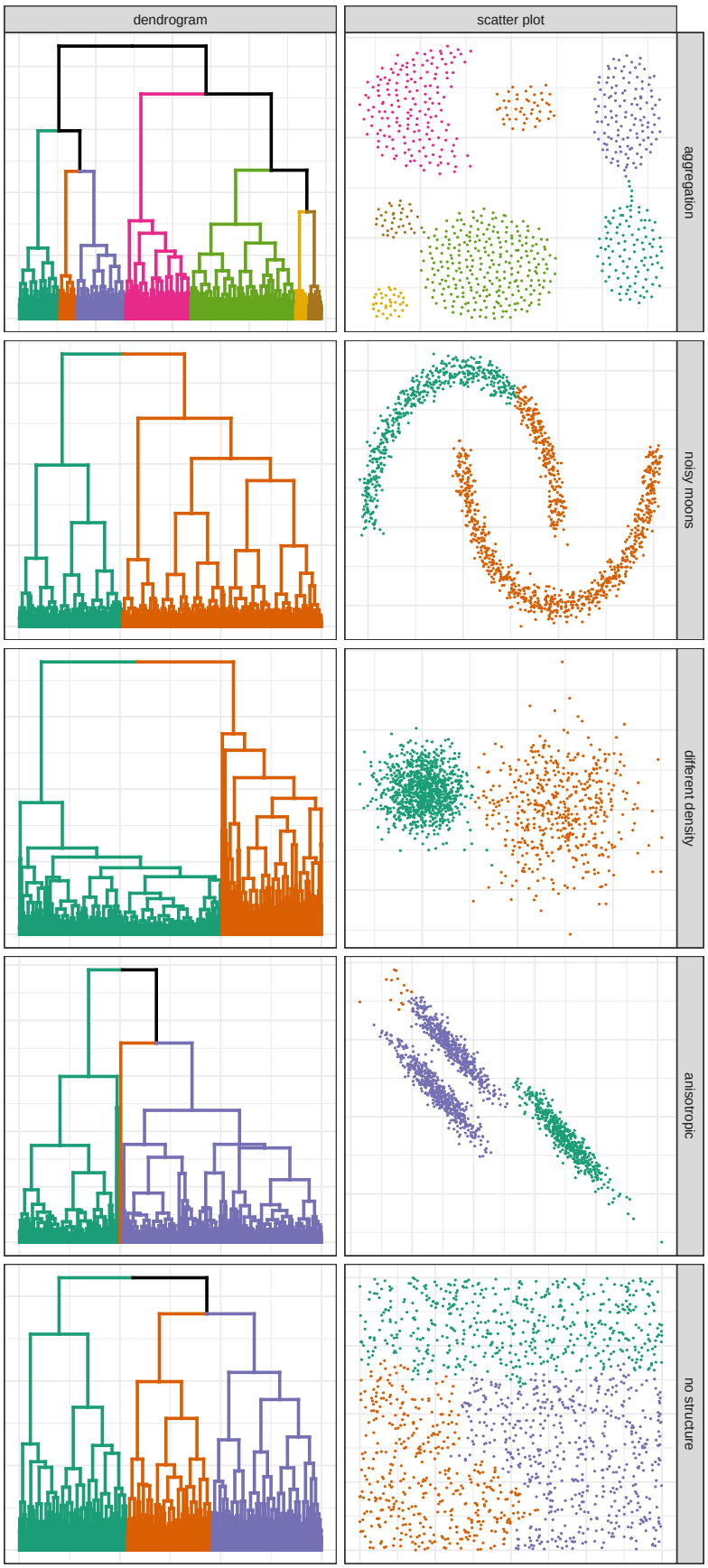

Figure 10.5: Hierarchical clustering of synthetic data-sets.

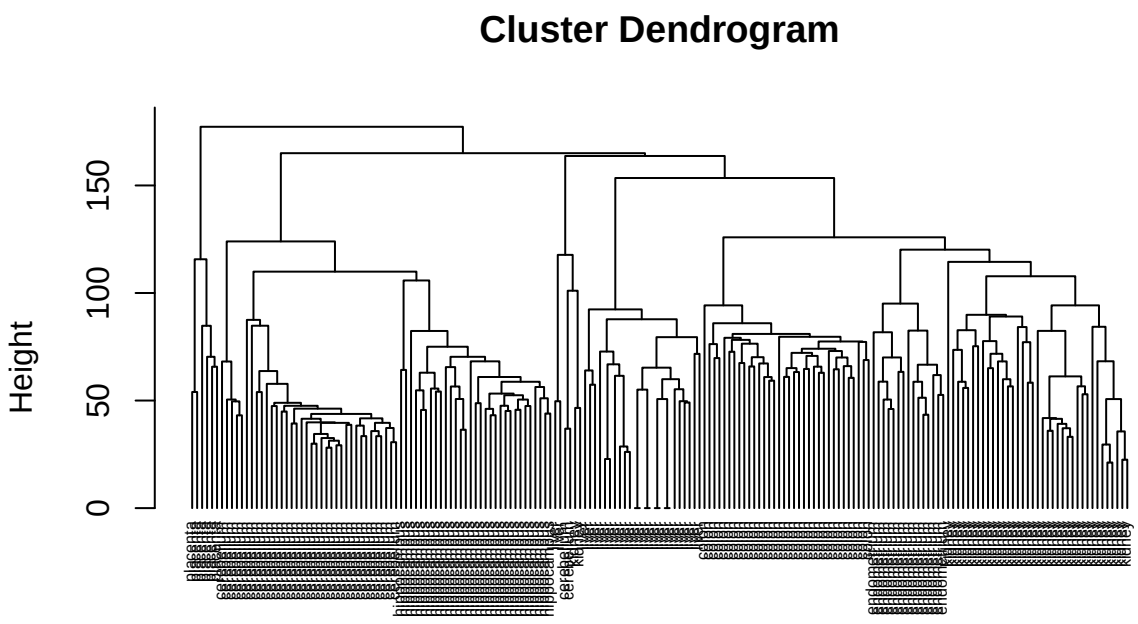

Figure 10.6: Clustering of tissue samples based on gene expression profiles.

Load data

```
load("data/tissues_gene_expression/tissuesGeneExpression.rda")
```

Inspect data

```
table(tissue)
```

```
## tissue
##  cerebellum       colon endometrium hippocampus      kidney       liver
##          38          34          15          31          39          26
##    placenta
##           6
```

```
dim(e)
```

```
## [1] 22215   189
```

Compute distance between each sample

```
d <- dist(t(e))
```

perform hierarchical clustering

```
hc <- hclust(d, method="average")
plot(hc, labels=tissue, cex=0.5, hang=-1, xlab="", sub="")
```

10.3.3.2 Colour labels

use dendextend library to plot dendrogram with colour labels

```
tissue_type <- unique(tissue)
dend <- as.dendrogram(hc)
dend_colours <- brewer.pal(length(unique(tissue)),"Dark2")
```

```
names(dend_colours) <- tissue_type
labels(dend) <- tissue[order.dendrogram(dend)]
labels_colors(dend) <- dend_colours[tissue][order.dendrogram(dend)]
labels_cex(dend) = 0.5
plot(dend, horiz=T)
```

10.3.3.3 Defining clusters by cutting tree

Define clusters by cutting tree at a specific height

```
plot(dend, horiz=T)
abline(v=125, lwd=2, lty=2, col="blue")
```

```
hclusters <- cutree(dend, h=125)
table(tissue, cluster=hclusters)
```

```
##              cluster
## tissue         1  2  3  4  5  6
##   cerebellum   0 36  0  0  2  0
##   colon        0  0 34  0  0  0
##   endometrium 15  0  0  0  0  0
##   hippocampus  0 31  0  0  0  0
##   kidney      37  0  0  0  2  0
##   liver        0  0  0 24  2  0
##   placenta     0  0  0  0  0  6
```

Select a specific number of clusters.

```
plot(dend, horiz=T)
abline(v = heights_per_k.dendrogram(dend)["8"], lwd = 2, lty = 2, col = "blue")
```

```
hclusters <- cutree(dend, k=8)
table(tissue, cluster=hclusters)
```

```
##              cluster
## tissue         1  2  3  4  5  6  7  8
##   cerebellum   0 31  0  0  2  0  5  0
##   colon        0  0 34  0  0  0  0  0
##   endometrium  0  0  0  0  0 15  0  0
##   hippocampus  0 31  0  0  0  0  0  0
##   kidney      37  0  0  0  2  0  0  0
##   liver        0  0  0 24  2  0  0  0
##   placenta     0  0  0  0  0  0  0  6
```

10.3.3.4 Heatmap

Base R provides a **heatmap** function, but we will use the more advanced **heatmap.2** from the **gplots** package.

```
library(gplots)
```

```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
```

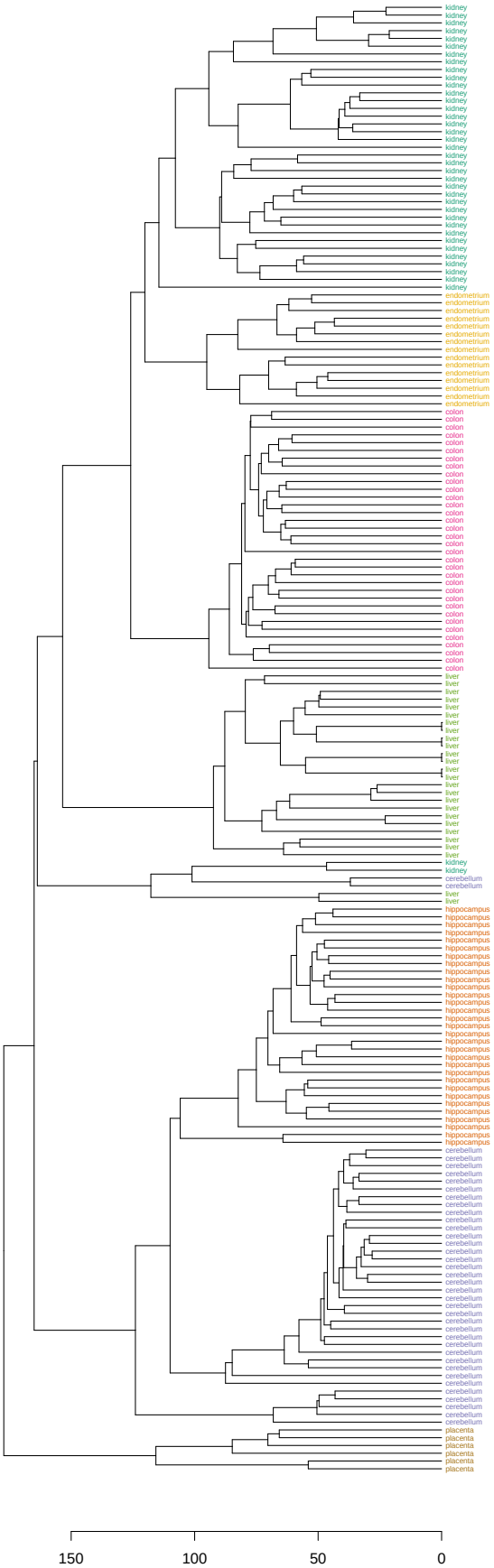

Figure 10.7: Clustering of tissue samples based on gene expression profiles with labels coloured by tissue type.

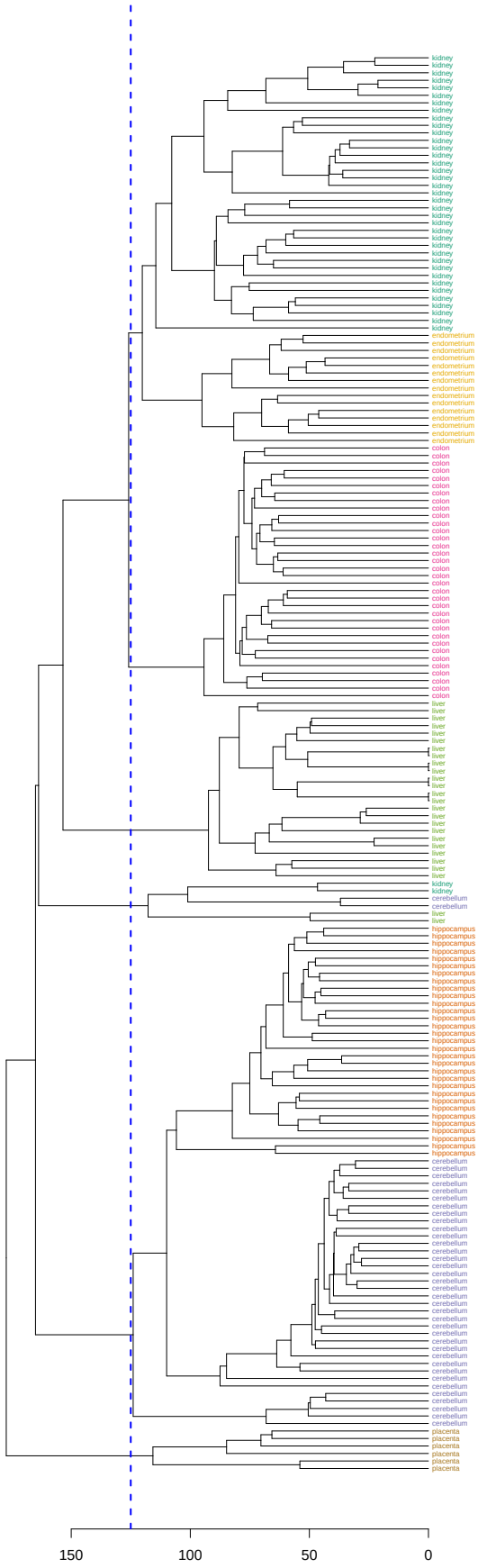

Figure 10.8: Clusters found by cutting tree at a height of 125

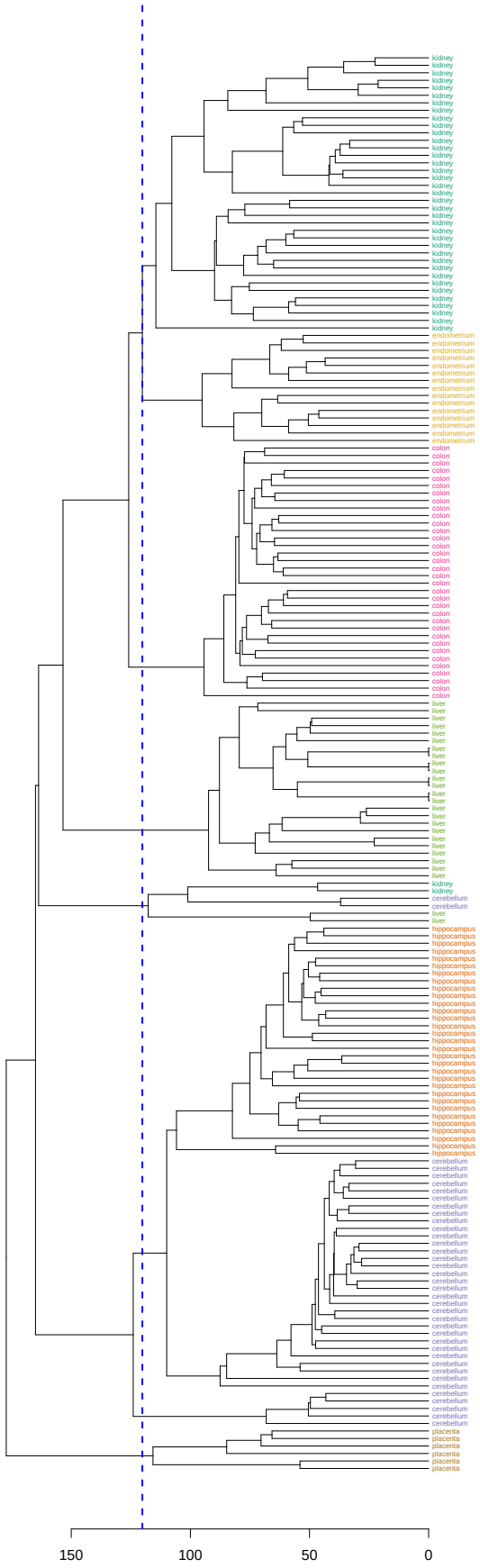

Figure 10.9: Selection of eight clusters from the dendogram

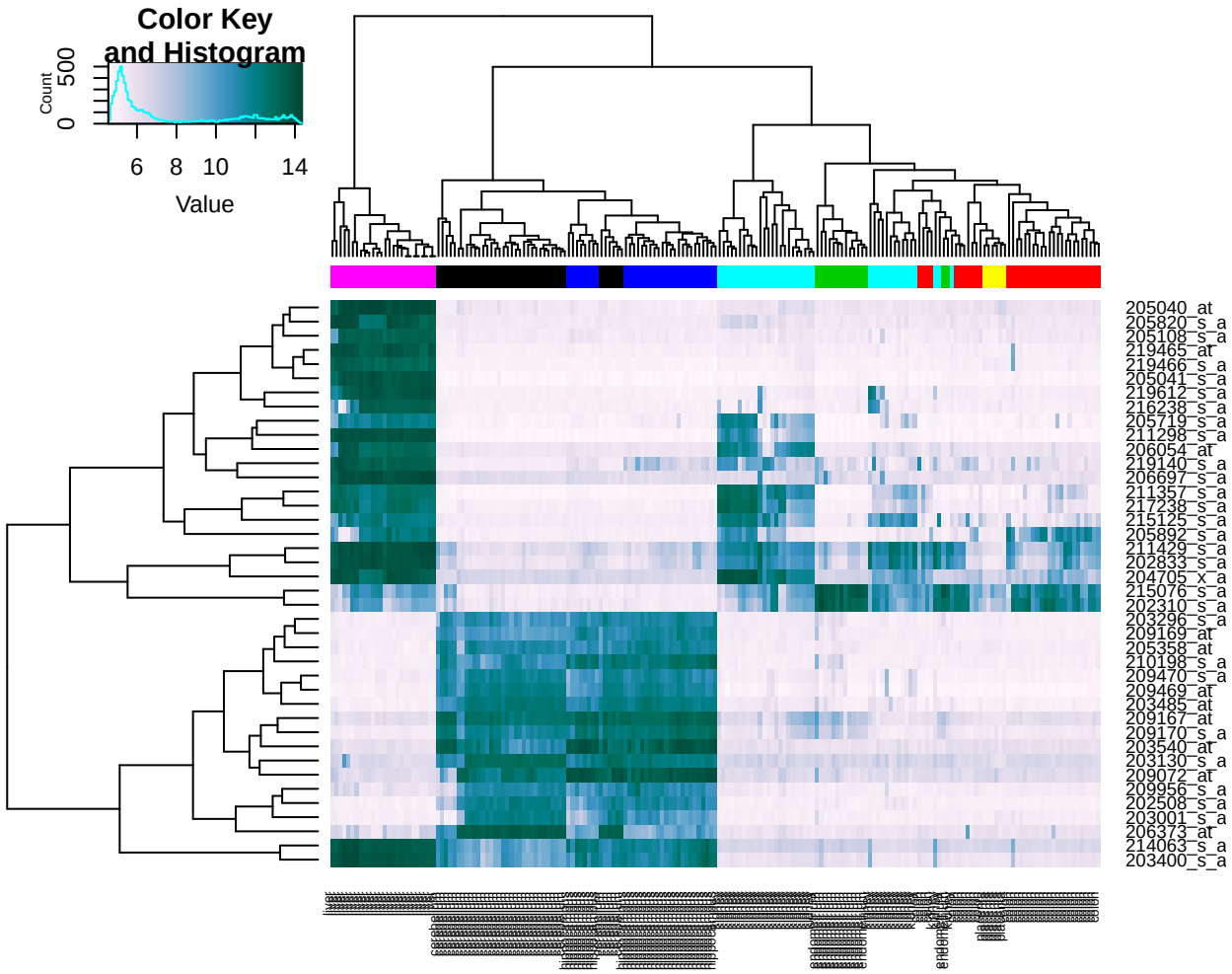

Figure 10.10: Heatmap of the expression of the 40 genes with the highest variance.

```
##     lowess
```

Define a colour palette (also known as a lookup table).

```
heatmap_colours <- colorRampPalette(brewer.pal(9, "PuBuGn"))(100)
```

Calculate the variance of each gene.

```
geneVariance <- apply(e,1,var)
```

Find the row numbers of the 40 genes with the highest variance.

```
idxTop40 <- order(-geneVariance)[1:40]
```

Define colours for tissues.

```
tissueColours <- palette(brewer.pal(8, "Dark2"))[as.numeric(as.factor(tissue))]
```

Plot heatmap.

```
heatmap.2(e[idxTop40,], labCol=tissue, trace="none",
          ColSideColors=tissueColours, col=heatmap_colours)
```

10.4 K-means

10.4.1 Algorithm

Pseudocode

The default setting of the **kmeans** function is to perform a maximum of 10 iterations and if the algorithm fails to converge a warning is issued. The maximum number of iterations is set with the argument **iter.max**.

10.4.2 Choosing initial cluster centres

```
library(RColorBrewer)
point_shapes <- c(15,17,19)
point_colours <- brewer.pal(3,"Dark2")
point_size = 1.5
center_point_size = 8

blobs <- as.data.frame(read.csv("data/example_clusters/blobs.csv", header=F))

good_centres <- as.data.frame(matrix(c(2,8,7,3,12,7), ncol=2, byrow=T))
bad_centres <- as.data.frame(matrix(c(13,13,8,12,2,2), ncol=2, byrow=T))

good_result <- kmeans(blobs[,1:2], centers=good_centres)
bad_result <- kmeans(blobs[,1:2], centers=bad_centres)

plotList <- list(
ggplot(blobs, aes(V1,V2)) +
  geom_point(col=point_colours[good_result$cluster], shape=point_shapes[good_result$cluster],
             size=point_size) +
  geom_point(data=good_centres, aes(V1,V2), shape=3, col="black", size=center_point_size) +
  theme_bw(),
ggplot(blobs, aes(V1,V2)) +
  geom_point(col=point_colours[bad_result$cluster], shape=point_shapes[bad_result$cluster],
             size=point_size) +
  geom_point(data=bad_centres, aes(V1,V2), shape=3, col="black", size=center_point_size) +
  theme_bw()
)

pm <- ggmatrix(
  plotList, nrow=1, ncol=2, showXAxisPlotLabels = T, showYAxisPlotLabels = T,
  xAxisLabels=c("A", "B")
) + theme_bw()

pm
```

Convergence to a local minimum can be avoided by starting the algorithm multiple times, with different random centres. The **nstart** argument to the **k-means** function can be used to specify the number of random sets and optimal solution will be selected automatically.

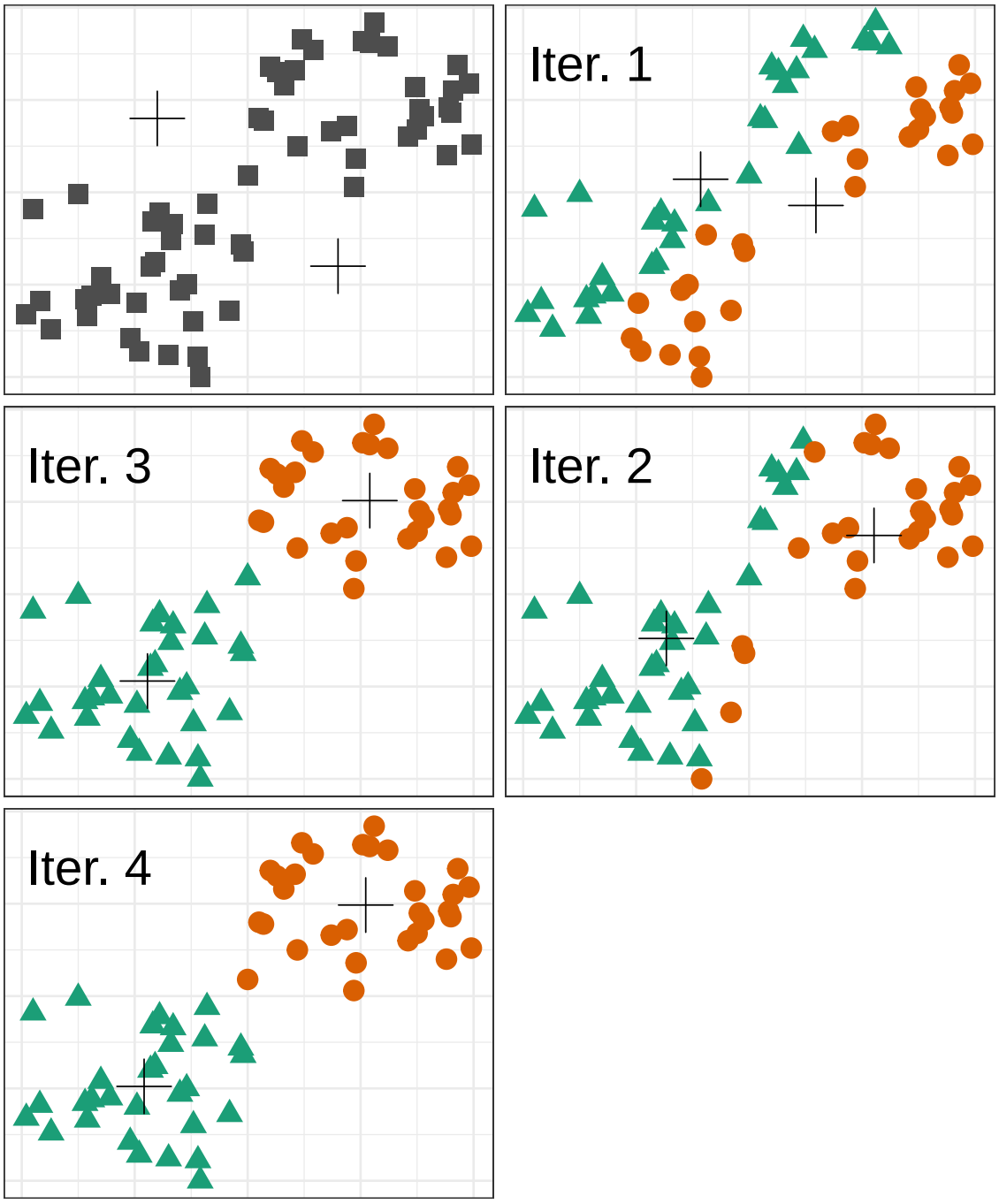

Figure 10.11: Iterations of the k-means algorithm

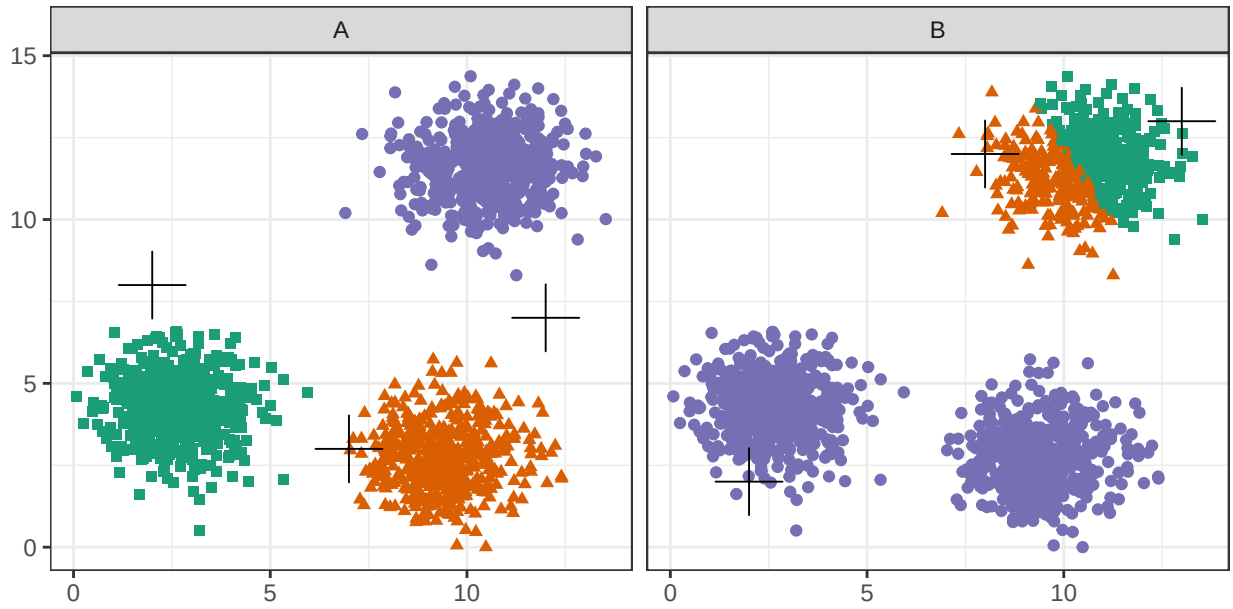

Figure 10.12: Initial centres determine clusters. The starting centres are shown as crosses. **A**, real clusters found; **B**, convergence to a local minimum.

10.4.3 Choosing k

```
point_colours <- brewer.pal(9,"Set1")
k <- 1:9
res <- lapply(k, function(i){kmeans(blobs[,1:2], i, nstart=50)})

plotList <- lapply(k, function(i){
  ggplot(blobs, aes(V1, V2)) +
    geom_point(col=point_colours[res[[i]]$cluster], size=1) +
    geom_point(data=as.data.frame(res[[i]]$centers), aes(V1,V2), shape=3, col="black", size=5) +
    annotate("text", x=2, y=13, label=paste("k=", i, sep=""), size=8, col="black") +
    theme_bw()
}
)

pm <- ggmatrix(
  plotList, nrow=3, ncol=3, showXAxisPlotLabels = T, showYAxisPlotLabels = T
) + theme_bw()

pm
```

```
tot_withinss <- sapply(k, function(i){res[[i]]$tot.withinss})
qplot(k, tot_withinss, geom=c("point", "line"),
      ylab="Total within-cluster sum of squares") + theme_bw()
```

N.B. we have set `nstart=50` to run the algorithm 50 times, starting from different, random sets of centroids.

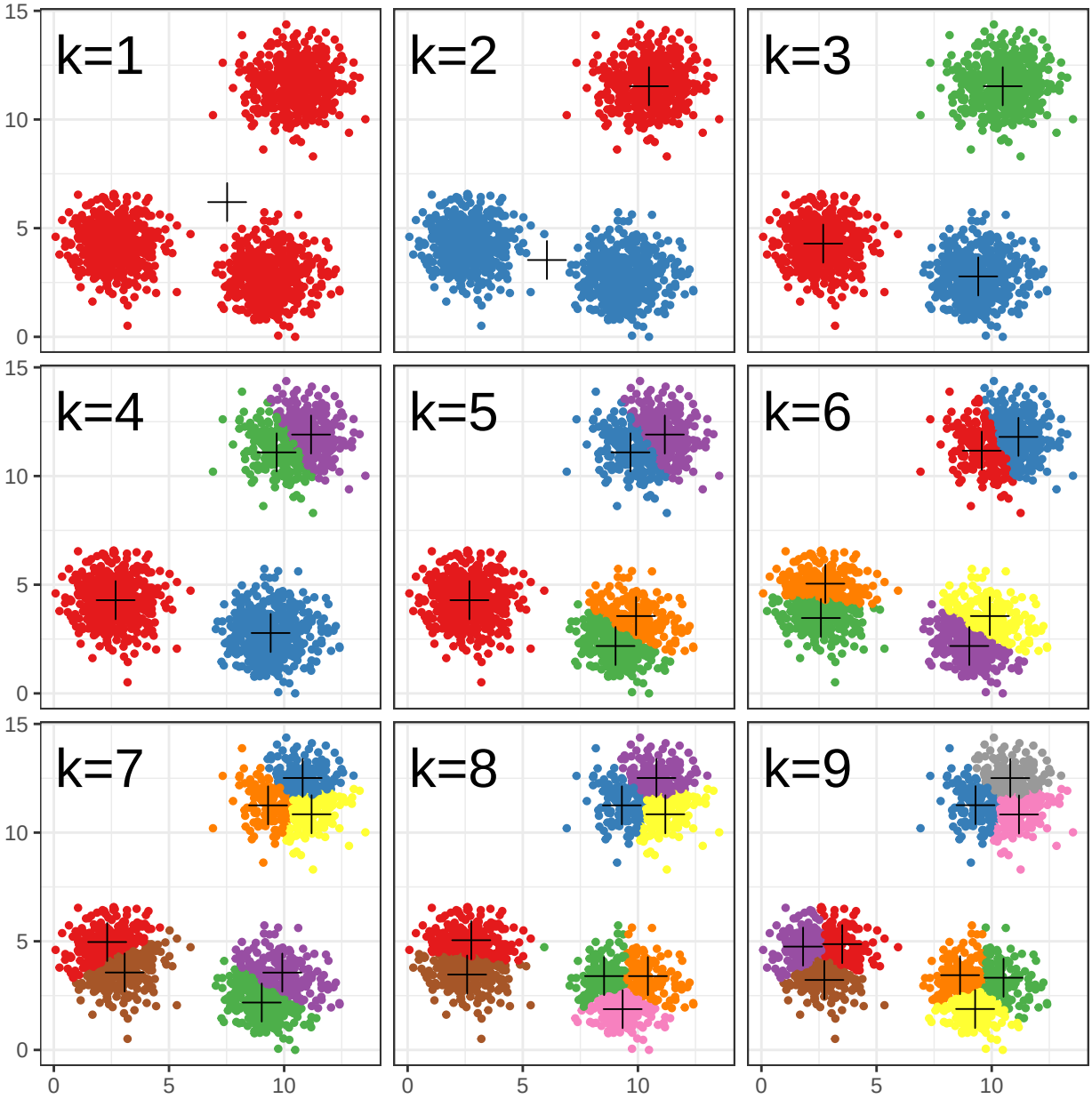

Figure 10.13: K-means clustering of the blobs data set using a range of values of k from 1-9. Cluster centres indicated with a cross.

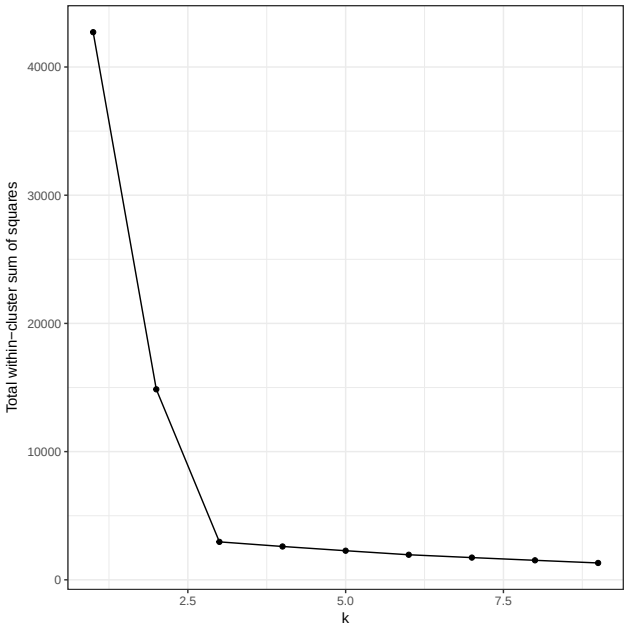

Figure 10.14: Variance within the clusters. Total within-cluster sum of squares plotted against k.

10.4.4 Example: clustering synthetic data sets

Let's see how k-means performs on the other toy data sets. First we will define some variables and functions we will use in the analysis of all data sets.

```
k=1:9
point_shapes <- c(15,17,19,5,6,0,1)
point_colours <- brewer.pal(7,"Dark2")
point_size = 1.5
center_point_size = 8

plot_tot_withinss <- function(kmeans_output){
  tot_withinss <- sapply(k, function(i){kmeans_output[[i]]$tot.withinss})
  qplot(k, tot_withinss, geom=c("point", "line"),
        ylab="Total within-cluster sum of squares") + theme_bw()
}

plot_clusters <- function(data_set, kmeans_output, num_clusters){
    ggplot(data_set, aes(V1,V2)) +
    geom_point(col=point_colours[kmeans_output[[num_clusters]]$cluster],
               shape=point_shapes[kmeans_output[[num_clusters]]$cluster],
               size=point_size) +
    geom_point(data=as.data.frame(kmeans_output[[num_clusters]]$centers), aes(V1,V2),
               shape=3,col="black",size=center_point_size) +
    theme_bw()
}
```

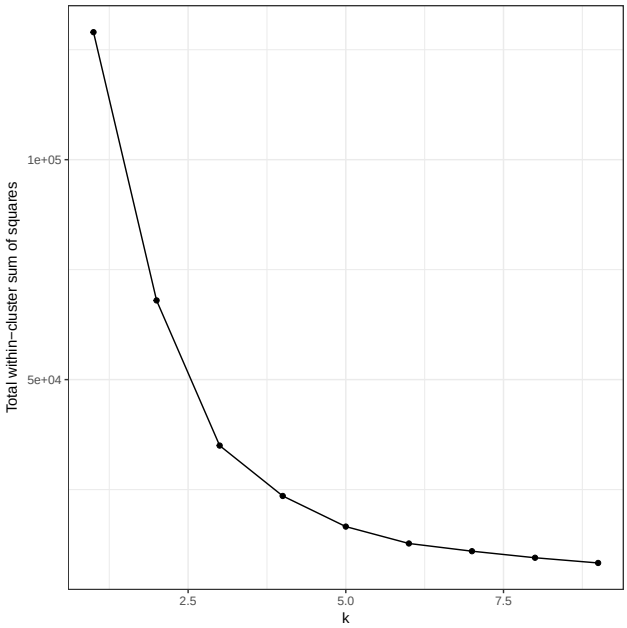

Figure 10.15: K-means clustering of the aggregation data set: variance within clusters.

10.4.4.1 Aggregation

```
aggregation <- as.data.frame(read.table("data/example_clusters/aggregation.txt"))
res <- lapply(k, function(i){kmeans(aggregation[,1:2], i, nstart=50)})
plot_tot_withinss(res)
```

```
plotList <- list(
  plot_clusters(aggregation, res, 3),
  plot_clusters(aggregation, res, 7)
)
pm <- ggmatrix(
  plotList, nrow=1, ncol=2, showXAxisPlotLabels = T, showYAxisPlotLabels = T,
  xAxisLabels=c("k=3", "k=7")
) + theme_bw()
pm
```

10.4.4.2 Noisy moons

```
noisy_moons <- read.csv("data/example_clusters/noisy_moons.csv", header=F)
res <- lapply(k, function(i){kmeans(noisy_moons[,1:2], i, nstart=50)})
plot_tot_withinss(res)
```

```
plot_clusters(noisy_moons, res, 2)
```

10.4.4.3 Different density

```
diff_density <- as.data.frame(read.csv("data/example_clusters/different_density.csv", header=F))
res <- lapply(k, function(i){kmeans(diff_density[,1:2], i, nstart=50)})
```

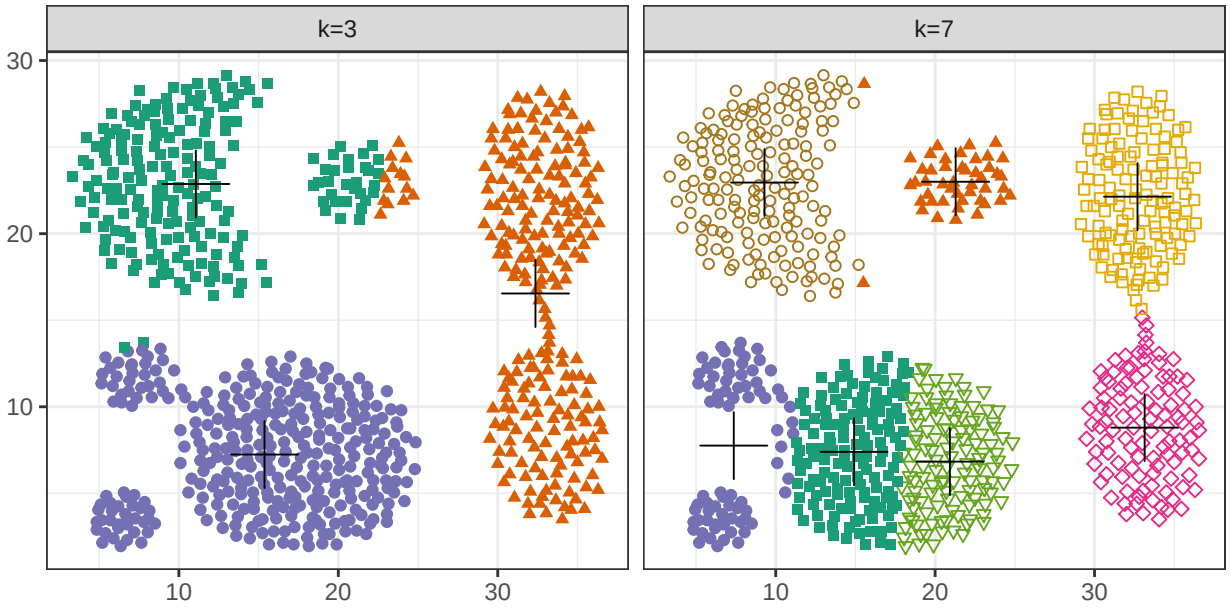

Figure 10.16: K-means clustering of the aggregation data set: scatterplots of clusters for k=3 and k=7. Cluster centres indicated with a cross.

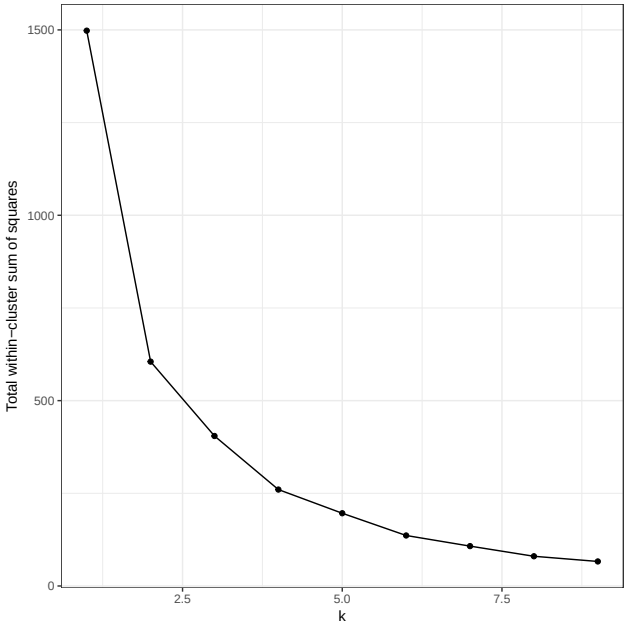

Figure 10.17: K-means clustering of the noisy moons data set: variance within clusters.

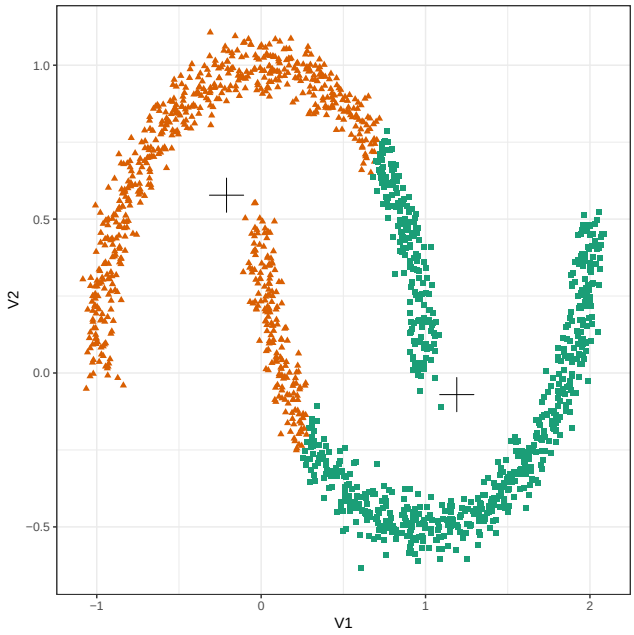

Figure 10.18: K-means clustering of the noisy moons data set: scatterplot of clusters for k=2. Cluster centres indicated with a cross.

```
## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations
```

Failure to converge, so increase number of iterations.

```
res <- lapply(k, function(i){kmeans(diff_density[,1:2], i, iter.max=20, nstart=50)})
plot_tot_withinss(res)
```

```
plot_clusters(diff_density, res, 2)
```

10.4.4.4 Anisotropic distributions

```
aniso <- as.data.frame(read.csv("data/example_clusters/aniso.csv", header=F))
res <- lapply(k, function(i){kmeans(aniso[,1:2], i, nstart=50)})
plot_tot_withinss(res)
```

```
plotList <- list(
  plot_clusters(aniso, res, 2),
  plot_clusters(aniso, res, 3)
)
pm <- ggmatrix(
  plotList, nrow=1, ncol=2, showXAxisPlotLabels = T,
  showYAxisPlotLabels = T, xAxisLabels=c("k=2", "k=3")
) + theme_bw()
pm
```

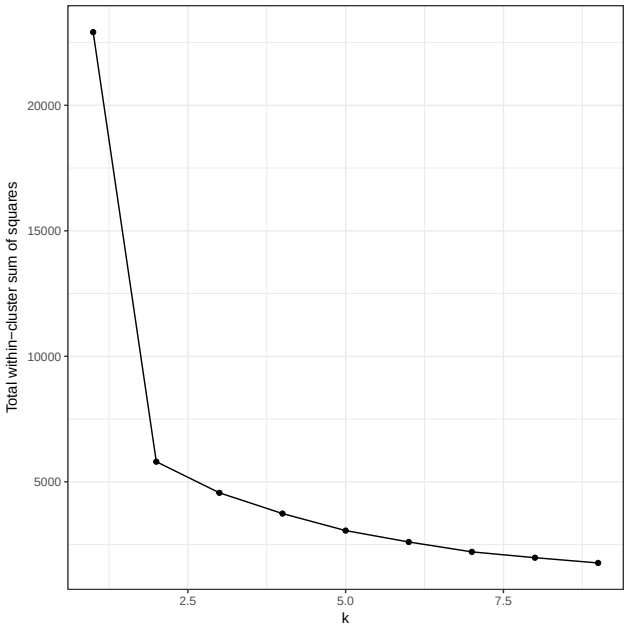

Figure 10.19: K-means clustering of the different density distributions data set: variance within clusters.

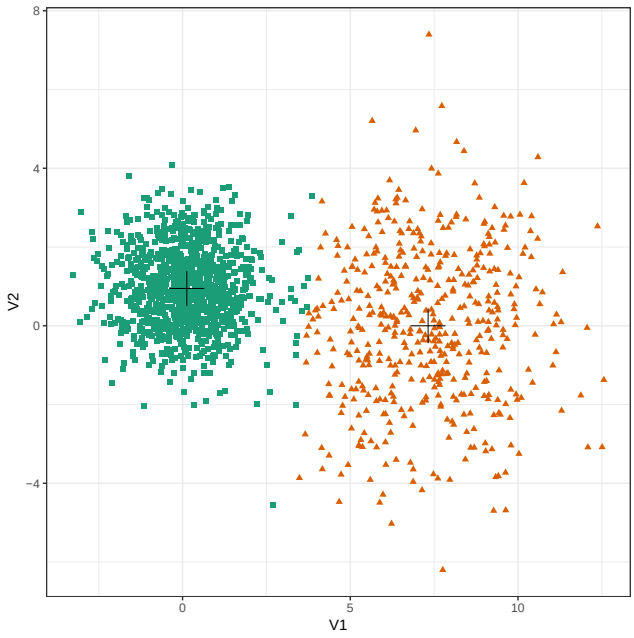

Figure 10.20: K-means clustering of the different density distributions data set: scatterplots of clusters for k=2 and k=3. Cluster centres indicated with a cross.

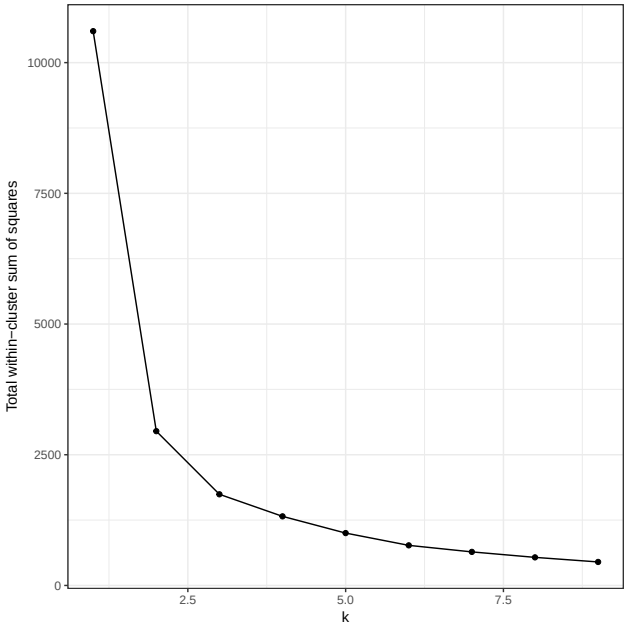

Figure 10.21: K-means clustering of the anisotropic distributions data set: variance within clusters.

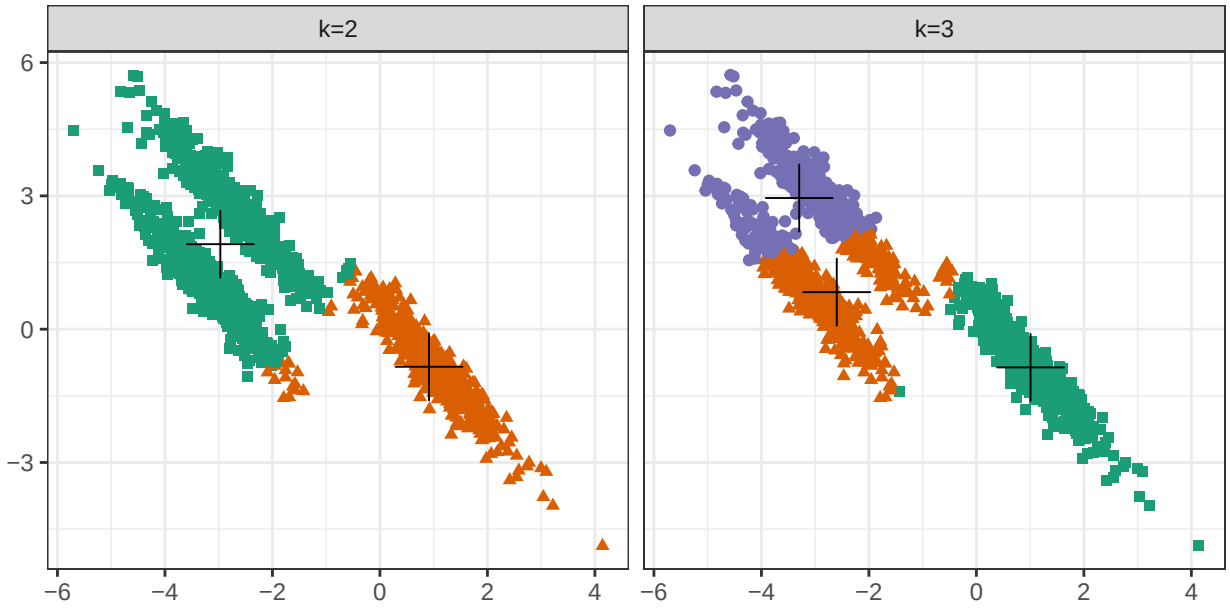

Figure 10.22: K-means clustering of the anisotropic distributions data set: scatterplots of clusters for k=2 and k=3. Cluster centres indicated with a cross.

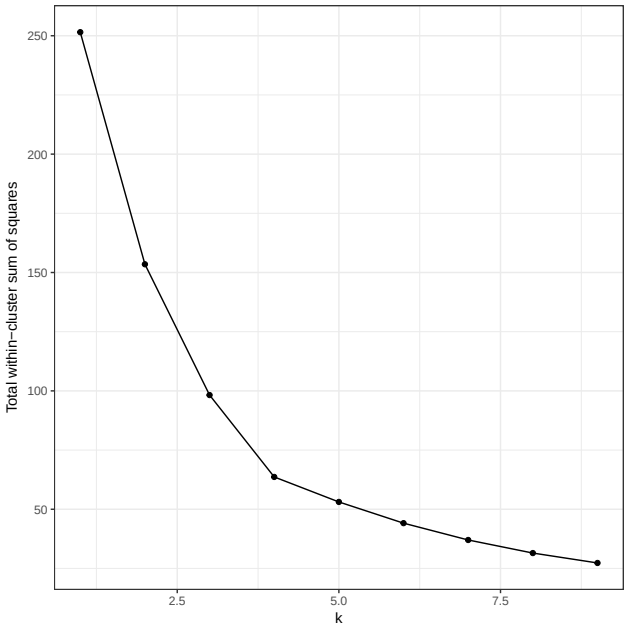

Figure 10.23: K-means clustering of the data set with no structure: variance within clusters.

10.4.4.5 No structure

```
no_structure <- as.data.frame(read.csv("data/example_clusters/no_structure.csv", header=F))
res <- lapply(k, function(i){kmeans(no_structure[,1:2], i, nstart=50)})
plot_tot_withinss(res)
```

```
plot_clusters(no_structure, res, 4)
```

10.4.5 Example: gene expression profiling of human tissues

Let's return to the data on gene expression of human tissues. Load data

```
load("data/tissues_gene_expression/tissuesGeneExpression.rda")
```

As we saw earlier, the data set contains expression levels for over 22,000 transcripts in seven tissues.

```
table(tissue)
```

```
## tissue
##  cerebellum       colon endometrium hippocampus      kidney       liver
##          38          34          15          31          39          26
##    placenta
##           6
```

```
dim(e)
```

```
## [1] 22215   189
```

First we will examine the total intra-cluster variance with different values of k. Our data-set is fairly large, so clustering it for several values or k and with multiple random starting centres is computationally quite intensive. Fortunately the task readily lends itself to parallelization; we can assign the analysis of each 'k' to a different processing core. As we have seen in the previous chapters on supervised learning, caret has

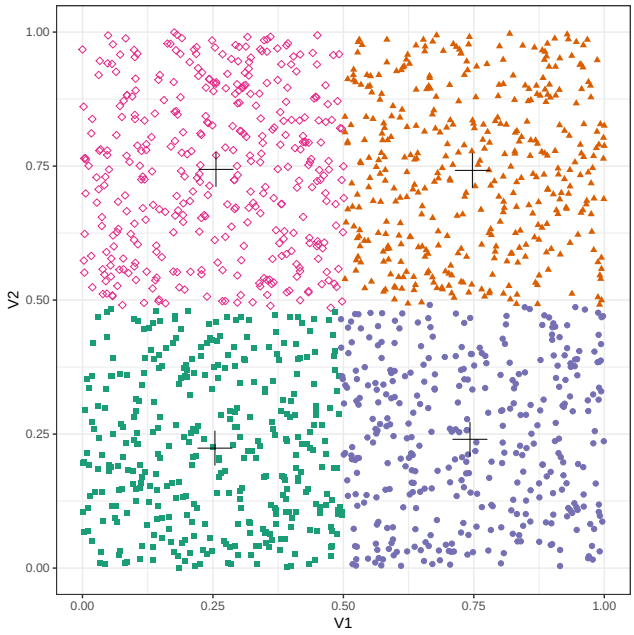

Figure 10.24: K-means clustering of the data set with no structure: scatterplot of clusters for k=4. Cluster centres indicated with a cross.

parallel processing built in and we simply have to load a package for multicore processing, such as doMC, and then register the number of cores we would like to use. Running **kmeans** in parallel is slightly more involved, but still very easy. We will start by loading doMC and registering all available cores:

```
library(doMC)
```

```
## Loading required package: foreach
```

```
## Loading required package: iterators
```

```
## Loading required package: parallel
```

```
registerDoMC()
```

To find out how many cores we have registered we can use:

```
getDoParWorkers()
```

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

Instead of using the **lapply** function to vectorize our code, we will instead use the parallel equivalent, **foreach**. Like **lapply**, **foreach** returns a list by default. For this example we have set a seed, rather than generate a random number, for the sake of reproducibility. Ordinarily we would omit `set.seed(42)` and `.options.multicore=list(set.seed=FALSE)`.

```
k<-1:15
set.seed(42)
res_k_15 <- foreach(
  i=k,
  .options.multicore=list(set.seed=FALSE)) %dopar% kmeans(t(e), i, nstart=10)
plot_tot_withinss(res_k_15)
```

There is no obvious elbow, but the rate of decrease in the total-within sum of squares appears to slow after k=5. Since we know that there are seven tissues in the data set we will try k=7.

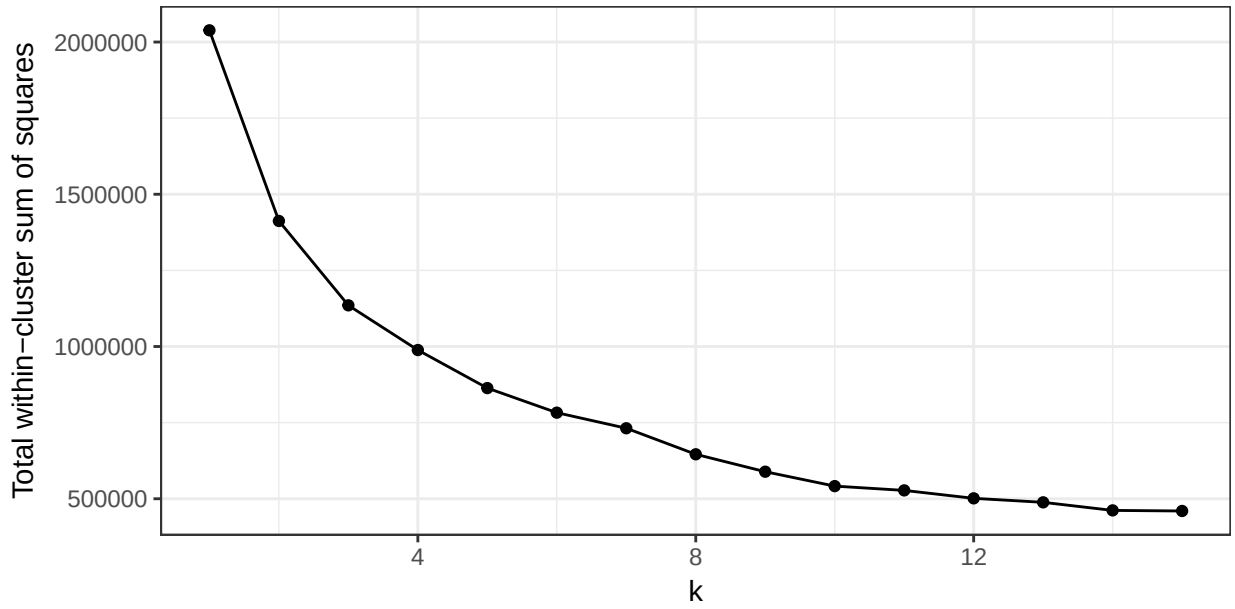

Figure 10.25: K-means clustering of human tissue gene expression: variance within clusters.

```
table(tissue, res_k_15[[7]]$cluster)
```

```
##
## tissue         1  2  3  4  5  6  7
##   cerebellum   0  0  0  0  0  5 33
##   colon        0  0 34  0  0  0  0
##   endometrium 15  0  0  0  0  0  0
##   hippocampus  0  0  0  0 31  0  0
##   kidney      37  2  0  0  0  0  0
##   liver        0 26  0  0  0  0  0
##   placenta     0  0  0  6  0  0  0
```

The analysis has found a distinct cluster for each tissue and therefore performed slightly better than the earlier hierarchical clustering analysis, which placed endometrium and kidney observations in the same cluster.

To visualize the result in a 2D scatter plot we first need to apply dimensionality reduction. We will use principal component analysis (PCA), which was described in chapter 9.

```
pca <- prcomp(t(e))
ggplot(data=as.data.frame(pca$x), aes(PC1,PC2)) +
  geom_point(col=brewer.pal(7,"Dark2")[res_k_15[[7]]$cluster],
             shape=c(49:55)[res_k_15[[7]]$cluster], size=5) +
  theme_bw()
```

10.5 DBSCAN

Density-based spatial clustering of applications with noise

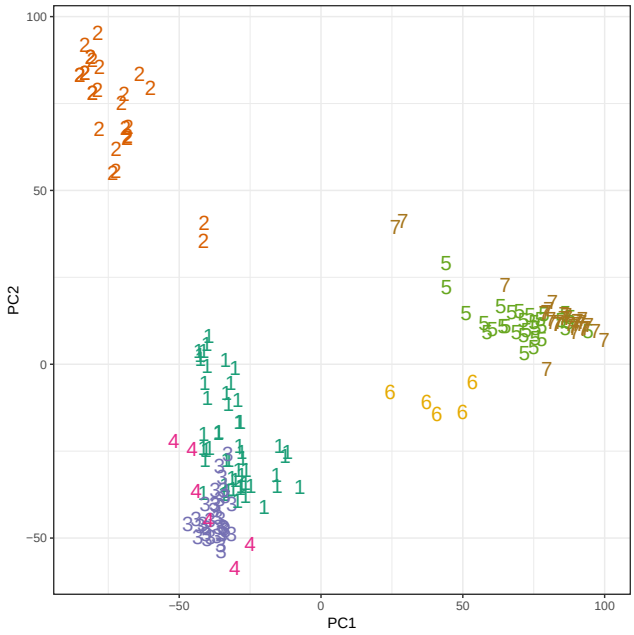

Figure 10.26: K-means clustering of human gene expression (k=7): scatterplot of first two principal components.

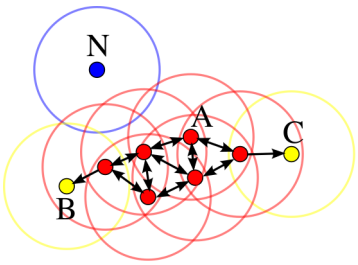

Figure 10.27: Illustration of the DBSCAN algorithm.

10.5.1 Algorithm

Abstract DBSCAN algorithm in pseudocode (Schubert et al., 2017)

```
1 Compute neighbours of each point and identify core points   // Identify core points
2 Join neighbouring core points into clusters                 // Assign core points
3 foreach non-core point do
      Add to a neighbouring core point if possible            // Assign border points
      Otherwise, add to noise                                 // Assign noise points
```

The method requires two parameters; MinPts that is the minimum number of samples in any cluster; Eps that is the maximum distance of the sample to at least one other sample within the same cluster.

This algorithm works on a parametric approach. The two parameters involved in this algorithm are: * e (eps) is the radius of our neighborhoods around a data point p. * minPts is the minimum number of data points we want in a neighborhood to define a cluster.

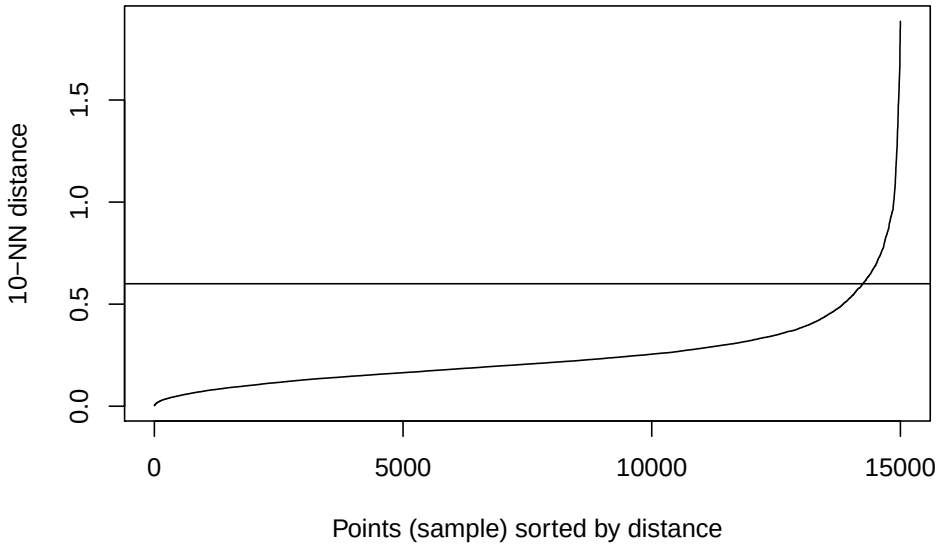

Figure 10.28: 10-nearest neighbour distances for the blobs data set

10.5.2 Implementation in R

DBSCAN is implemented in two R packages: dbscan and fpc. We will use the package dbscan, because it is significantly faster and can handle larger data sets than fpc. The function has the same name in both packages and so if for any reason both packages have been loaded into our current workspace, there is a danger of calling the wrong implementation. To avoid this we can specify the package name when calling the function, e.g.:

```
dbscan::dbscan
```

We load the dbscan package in the usual way:

```
library(dbscan)
```

10.5.3 Choosing parameters

The algorithm only needs parameteres **eps** and **minPts**.

Read in data and use **kNNdist** function from dbscan package to plot the distances of the 10-nearest neighbours for each observation (figure 10.28).

```
blobs <- read.csv("data/example_clusters/blobs.csv", header=F)
kNNdistplot(blobs[,1:2], k=10)
abline(h=0.6)
```

```
res <- dbscan::dbscan(blobs[,1:2], eps=0.6, minPts = 10)
table(res$cluster)
```

```
##
##   0   1   2   3
##  43 484 486 487
```

```
ggplot(blobs, aes(V1,V2)) +
  geom_point(col=brewer.pal(8,"Dark2")[c(8,1:7)][res$cluster+1],
             shape=c(4,15,17,19)[res$cluster+1],
```

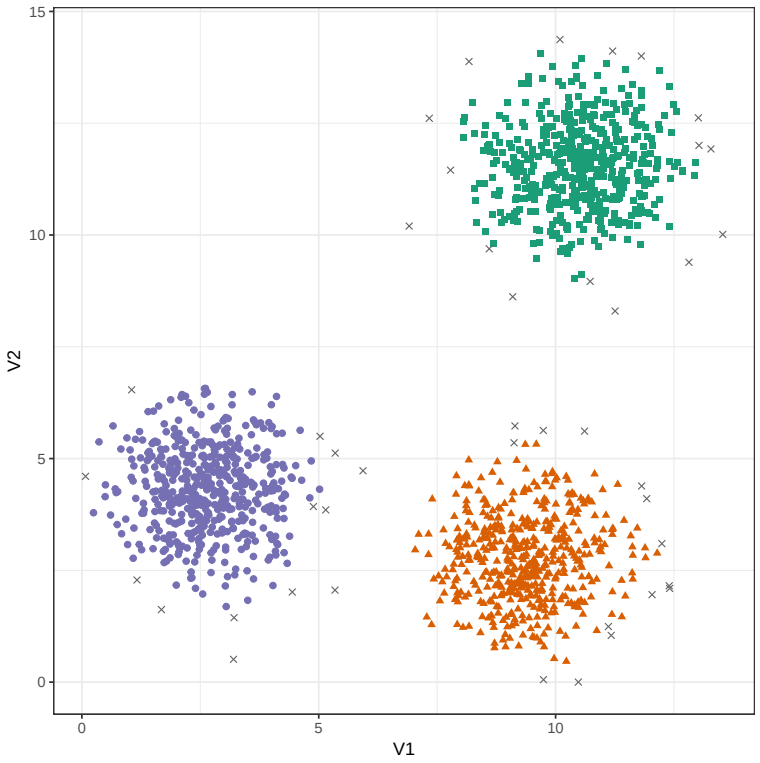

Figure 10.29: DBSCAN clustering (eps=0.6, minPts=10) of the blobs data set. Outlier observations are shown as grey crosses.

```
              size=1.5) +
  theme_bw()
```

10.5.4 Example: clustering synthetic data sets

```
point_shapes <- c(4,15,17,19,5,6,0,1)
point_colours <- brewer.pal(8,"Dark2")[c(8,1:7)]
point_size = 1.5
center_point_size = 8

plot_dbscan_clusters <- function(data_set, dbscan_output){
  ggplot(data_set, aes(V1,V2)) +
    geom_point(col=point_colours[dbscan_output$cluster+1],
               shape=point_shapes[dbscan_output$cluster+1],
               size=point_size) +
    theme_bw()
}
```

10.5.4.1 Aggregation

```
aggregation <- read.table("data/example_clusters/aggregation.txt")
kNNdistplot(aggregation[,1:2], k=10)
```

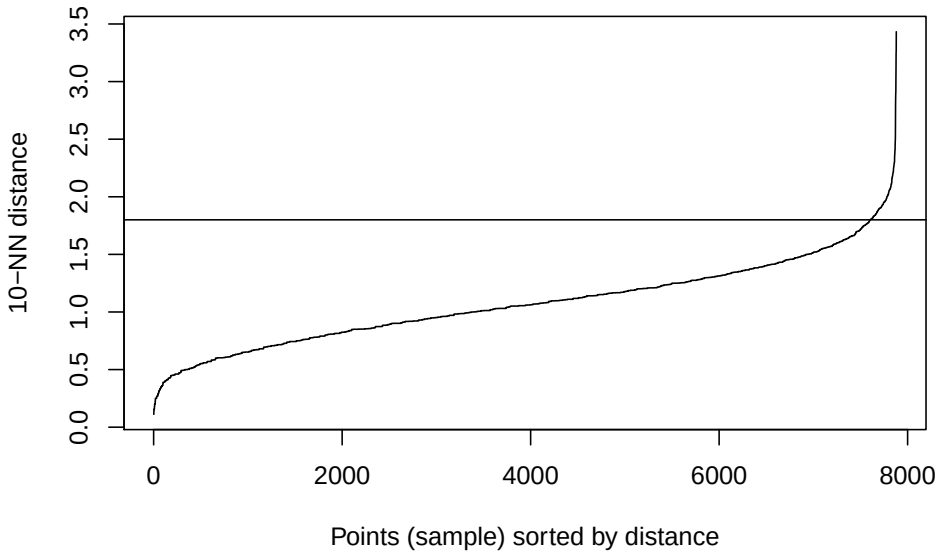

Figure 10.30: 10-nearest neighbour distances for the aggregation data set

```
abline(h=1.8)
```

```
res <- dbscan::dbscan(aggregation[,1:2], eps=1.8, minPts = 10)
table(res$cluster)
```

```
##
##   0   1   2   3   4   5   6
##   2 168 307 105 127  45  34
```

```
plot_dbscan_clusters(aggregation, res)
```

10.5.4.2 Noisy moons

```
noisy_moons <- read.csv("data/example_clusters/noisy_moons.csv", header=F)
kNNdistplot(noisy_moons[,1:2], k=10)
abline(h=0.075)
```

```
res <- dbscan::dbscan(noisy_moons[,1:2], eps=0.075, minPts = 10)
table(res$cluster)
```

```
##
##   0   1   2
##   8 748 744
```

```
plot_dbscan_clusters(noisy_moons, res)
```

10.5.4.3 Different density

```
diff_density <- read.csv("data/example_clusters/different_density.csv", header=F)
kNNdistplot(diff_density[,1:2], k=10)
abline(h=0.9)
abline(h=0.6, lty=2)
```

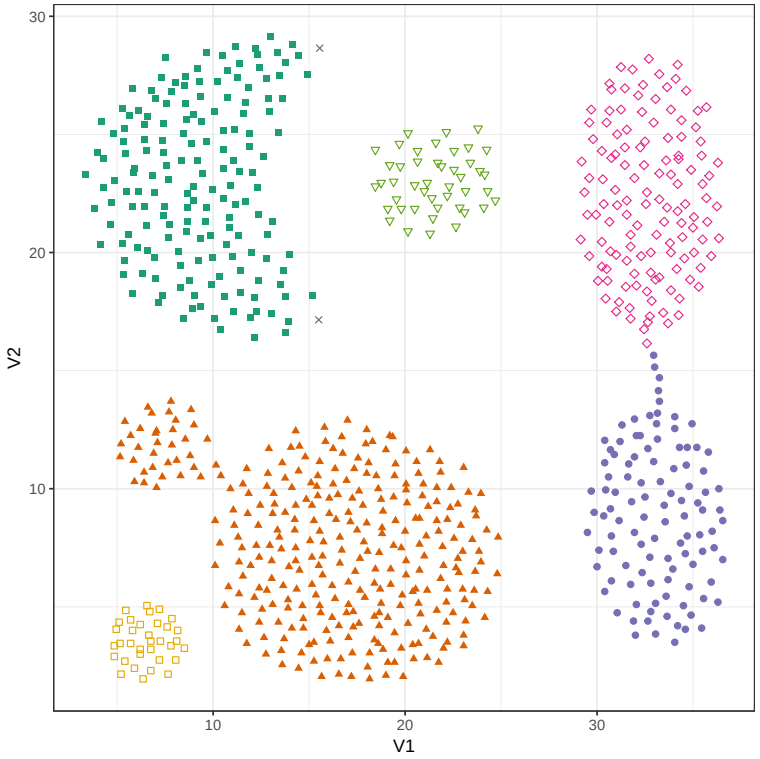

Figure 10.31: DBSCAN clustering (eps=1.8, minPts=10) of the aggregation data set. Outlier observations are shown as grey crosses.

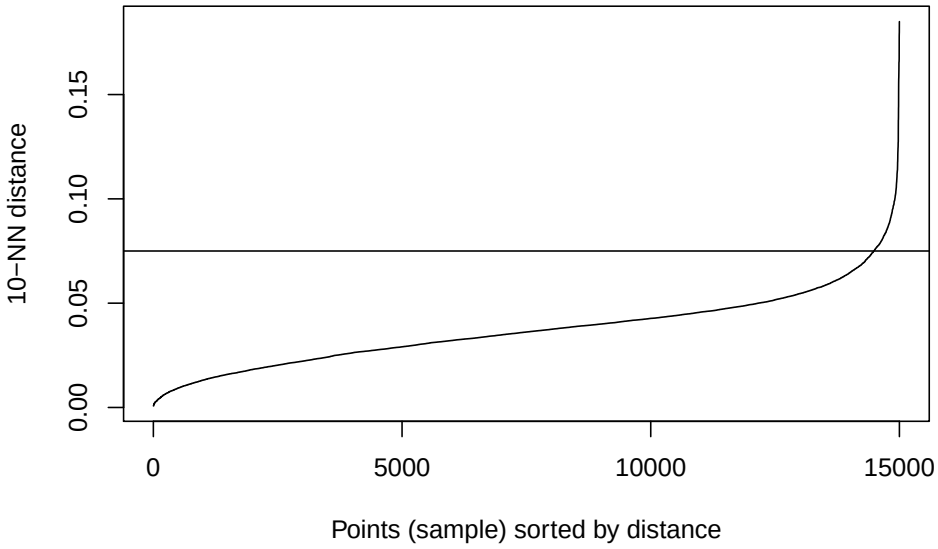

Figure 10.32: 10-nearest neighbour distances for the noisy moons data set

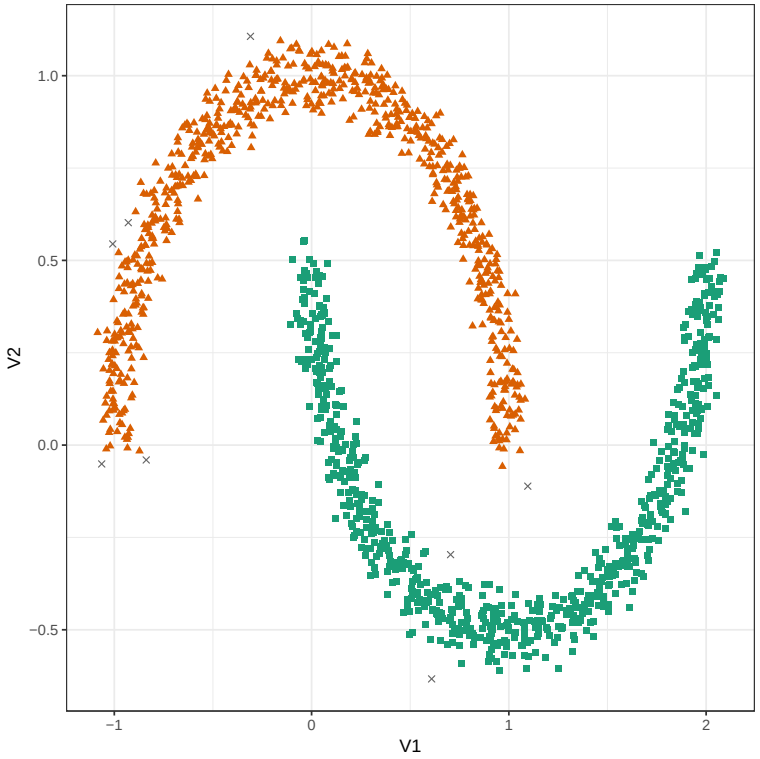

Figure 10.33: DBSCAN clustering (eps=0.075, minPts=10) of the noisy moons data set. Outlier observations are shown as grey crosses.

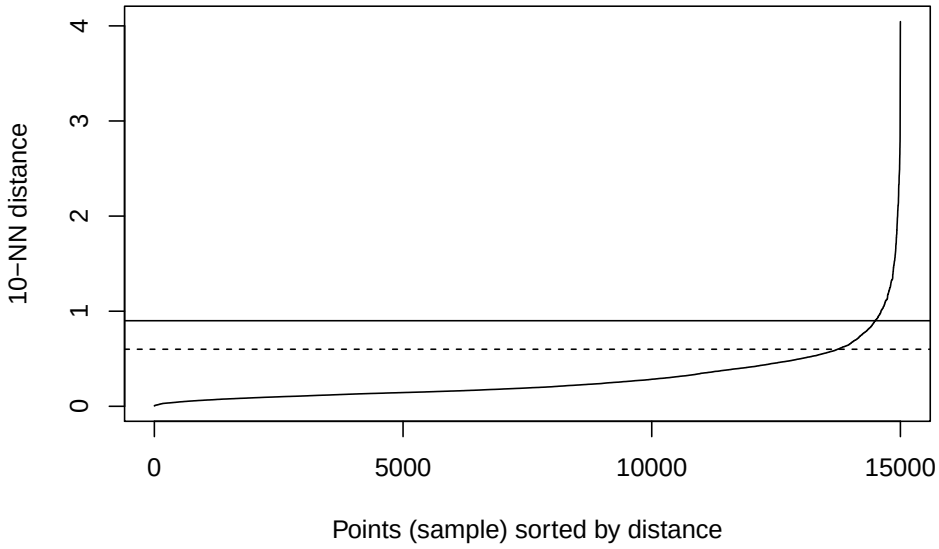

Figure 10.34: 10-nearest neighbour distances for the different density distributions data set

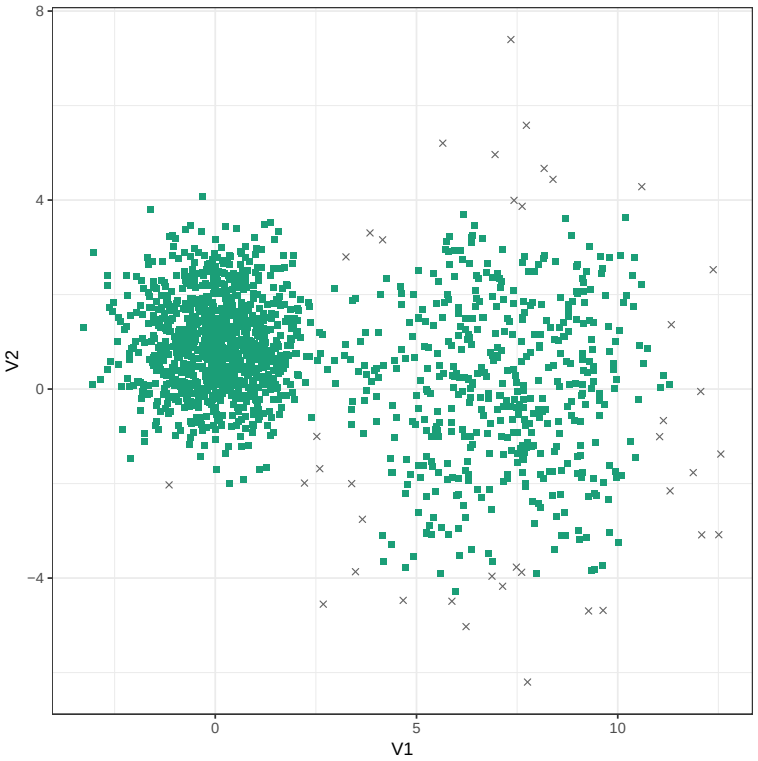

Figure 10.35: DBSCAN clustering of the different density distribution data set with eps=0.9 and minPts=10. Outlier observations are shown as grey crosses.

```
res <- dbscan::dbscan(diff_density[,1:2], eps=0.9, minPts = 10)
table(res$cluster)
```

```
##
##    0    1
##   40 1460
```

```
plot_dbscan_clusters(diff_density, res)
```

```
res <- dbscan::dbscan(diff_density[,1:2], eps=0.6, minPts = 10)
table(res$cluster)
```

```
##
##   0   1   2
## 109 399 992
```

```
plot_dbscan_clusters(diff_density, res)
```

10.5.4.4 Anisotropic distributions

```
aniso <- read.csv("data/example_clusters/aniso.csv", header=F)
kNNdistplot(aniso[,1:2], k=10)
abline(h=0.35)
```

```
res <- dbscan::dbscan(aniso[,1:2], eps=0.35, minPts = 10)
table(res$cluster)
```

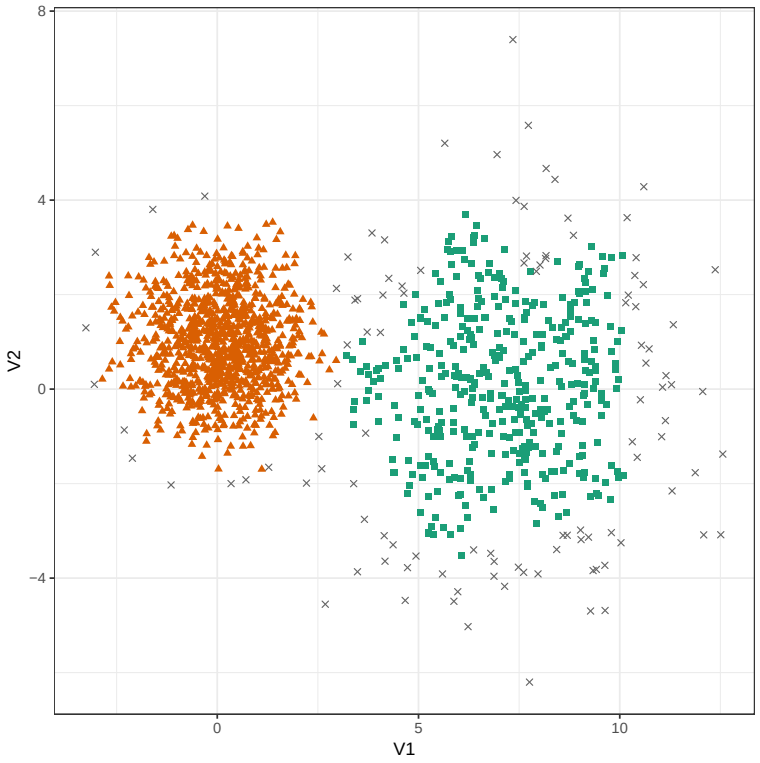

Figure 10.36: DBSCAN clustering of the different density distribution data set with eps=0.6 and minPts=10. Outlier observations are shown as grey crosses.

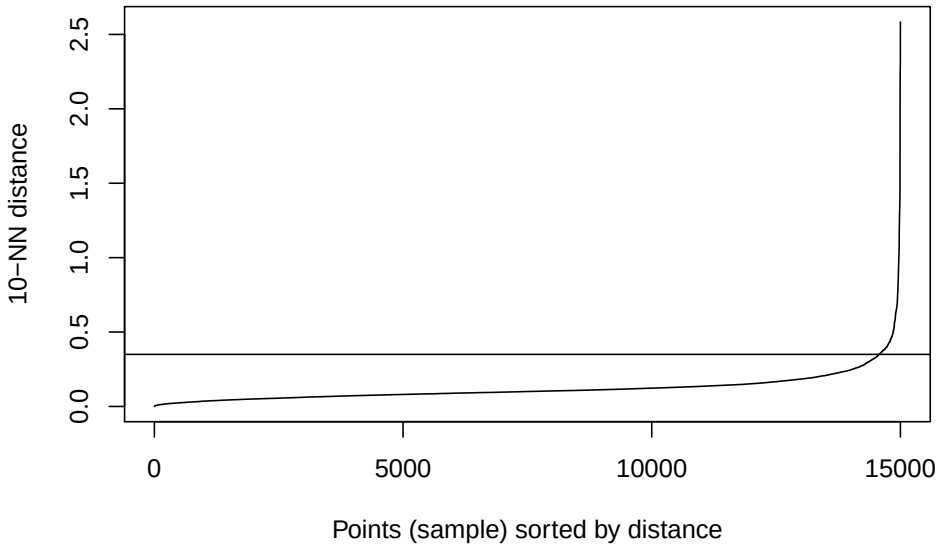

Figure 10.37: 10-nearest neighbour distances for the anisotropic distributions data set

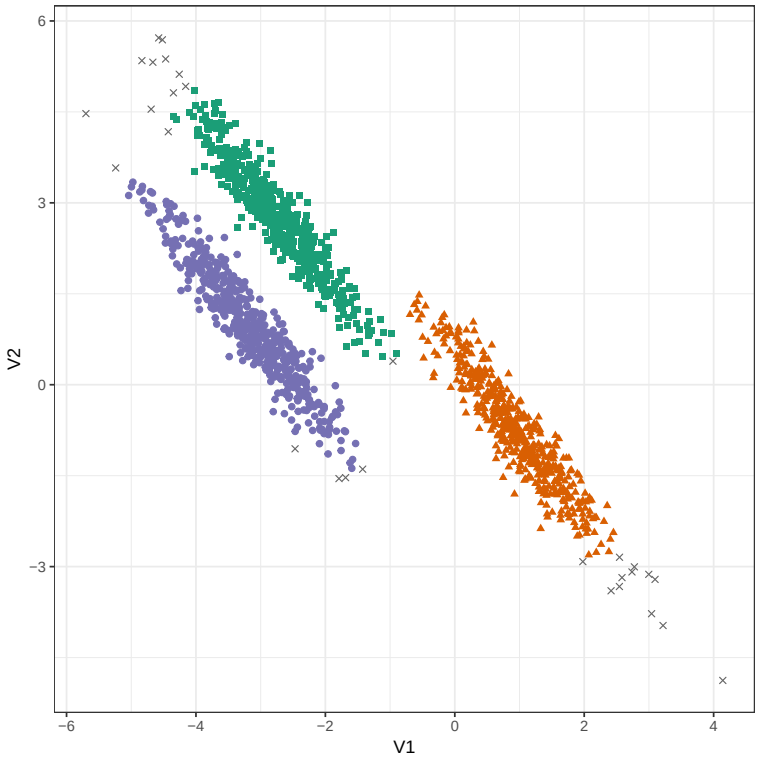

Figure 10.38: DBSCAN clustering (eps=0.3, minPts=10) of the anisotropic distributions data set. Outlier observations are shown as grey crosses.

```
##
##   0   1   2   3
##  29 489 488 494
```

```
plot_dbscan_clusters(aniso, res)
```

10.5.4.5 No structure

```
no_structure <- read.csv("data/example_clusters/no_structure.csv", header=F)
kNNdistplot(no_structure[,1:2], k=10)
abline(h=0.057)
```

```
res <- dbscan::dbscan(no_structure[,1:2], eps=0.57, minPts = 10)
table(res$cluster)
```

```
##
##    1
## 1500
```

10.5.5 Example: gene expression profiling of human tissues

Returning again to the data on gene expression of human tissues.

```
load("data/tissues_gene_expression/tissuesGeneExpression.rda")
```

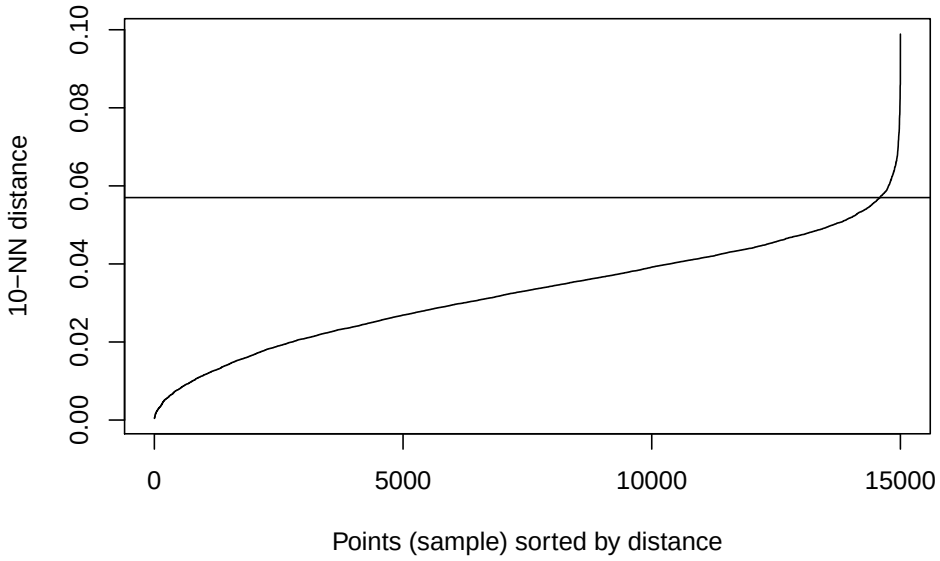

Figure 10.39: 10-nearest neighbour distances for the data set with no structure.

```
table(tissue)
```

```
## tissue
##  cerebellum       colon endometrium hippocampus      kidney       liver
##          38          34          15          31          39          26
##    placenta
##           6
```

We'll try k=5 (default for dbscan), because there are only six observations for placenta.

```
kNNdistplot(t(e), k=5)
abline(h=85)
```

```
set.seed(42)
res <- dbscan::dbscan(t(e), eps=85, minPts=5)
table(res$cluster)
```

```
##
##  0  1  2  3  4  5  6
## 12 37 62 34 24 15  5
```

```
table(tissue, res$cluster)
```

```
##
## tissue         0  1  2  3  4  5  6
##   cerebellum   2  0 31  0  0  0  5
##   colon        0  0  0 34  0  0  0
##   endometrium  0  0  0  0  0 15  0
##   hippocampus  0  0 31  0  0  0  0
##   kidney       2 37  0  0  0  0  0
##   liver        2  0  0  0 24  0  0
##   placenta     6  0  0  0  0  0  0
```

```
pca <- prcomp(t(e))
ggplot(data=as.data.frame(pca$x), aes(PC1,PC2)) +
  geom_point(col=brewer.pal(8,"Dark2")[c(8,1:7)][res$cluster+1],
```

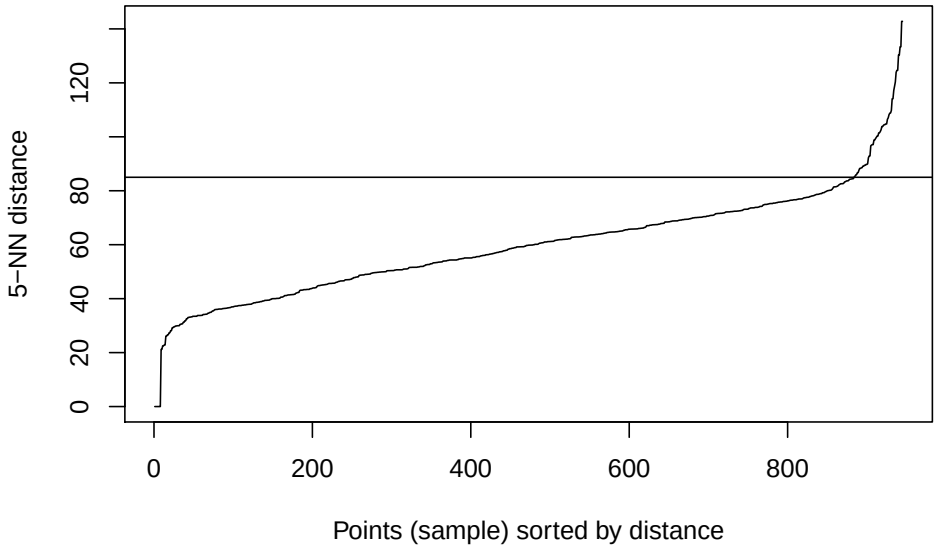

Figure 10.40: Five-nearest neighbour distances for the gene expression profiling of human tissues data set.

```
            shape=c(48:55)[res$cluster+1], size=5) +
  theme_bw()
```

10.6 Summary

10.6.1 Applications

10.6.2 Strengths

10.6.3 Limitations

10.7 Evaluating cluster quality

10.7.1 Silhouette method

Silhouette

$$s(i) = \frac{b(i) - a(i)}{max\left(a(i), b(i)\right)} \tag{10.2}$$

Method can be applied to clusters generated using any algorithm.

10.7.2 Example - k-means clustering of blobs data set

Load library required for calculating silhouette coefficients and plotting silhouettes.

```
library(cluster)
```

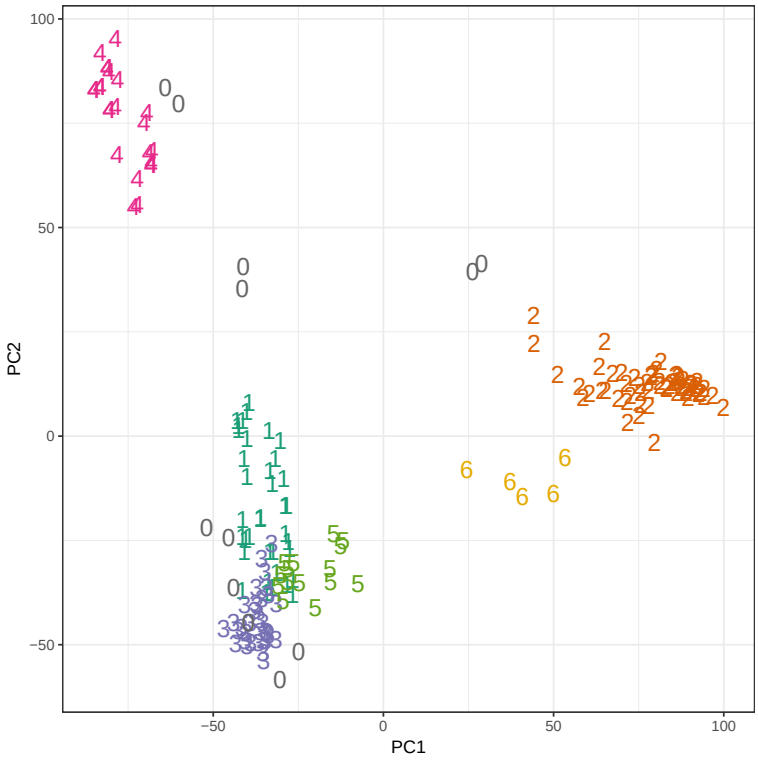

Figure 10.41: Clustering of human tissue gene expression: scatterplot of first two principal components.

We are going to take another look at k-means clustering of the blobs data-set (figure 10.13). Specifically we are going to see if silhouette analysis supports our original choice of k=3 as the optimum number of clusters (figure 10.14).

Silhouette analysis requires a minimum of two clusters, so we'll try values of k from 2 to 9.

```
k <- 2:9
```

Create a palette of colours for plotting.

```
kColours <- brewer.pal(9,"Set1")
```

Perform k-means clustering for each value of k from 2 to 9.

```
res <- lapply(k, function(i){kmeans(blobs[,1:2], i, nstart=50)})
```

Calculate the Euclidean distance matrix

```
d <- dist(blobs[,1:2])
```

Silhouette plot for k=2

```
s2 <- silhouette(res[[2-1]]$cluster, d)
plot(s2, border=NA, col=kColours[sort(res[[2-1]]$cluster)], main="")
```

Silhouette plot for k=9

```
s9 <- silhouette(res[[9-1]]$cluster, d)
plot(s9, border=NA, col=kColours[sort(res[[9-1]]$cluster)], main="")
```

Let's take a look at the silhouette plot for k=3.

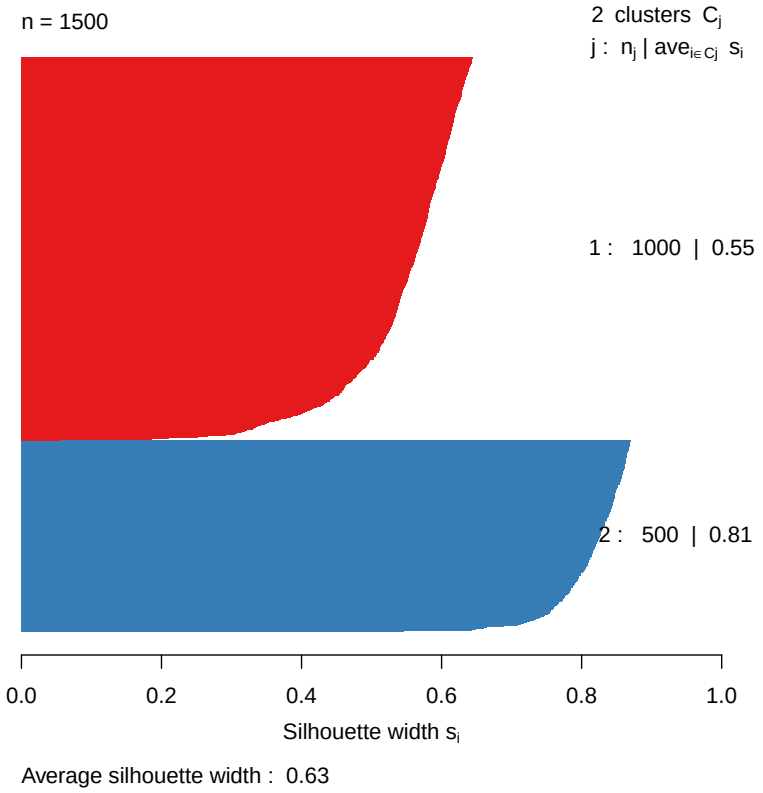

Figure 10.42: Silhouette plot for k-means clustering of the blobs data set with k=2.

```
s3 <- silhouette(res[[3-1]]$cluster, d)
plot(s3, border=NA, col=kColours[sort(res[[3-1]]$cluster)], main="")
```

So far the silhouette plots have shown that k=3 appears to be the optimum number of clusters, but we should investigate the silhouette coefficients at other values of k. Rather than produce a silhouette plot for each value of k, we can get a useful summary by making a barplot of average silhouette coefficients.

First we will calculate the silhouette coefficient for every observation (we need to index our list of **kmeans** outputs by `i-1`, because we are counting from k=2).

```
s <- lapply(k, function(i){silhouette(res[[i-1]]$cluster, d)})
```

We can then calculate the mean silhouette coefficient for each value of k from 2 to 9.

```
avgS <- sapply(s, function(x){mean(x[,3])})
```

Now we have the data we need to produce a barplot.

```
dat <- as.data.frame(cbind(k, avgS))
ggplot(data=dat, aes(x=k, y=avgS)) +
         geom_bar(stat="identity", fill="steelblue") +
  geom_text(aes(label=round(avgS,2)), vjust=1.6, color="white", size=3.5)+
  labs(y="Average silhouette coefficient") +
  scale_x_continuous(breaks=2:9) +
  theme_bw()
```

The bar plot (figure 10.45) confirms that the optimum number of clusters is three.

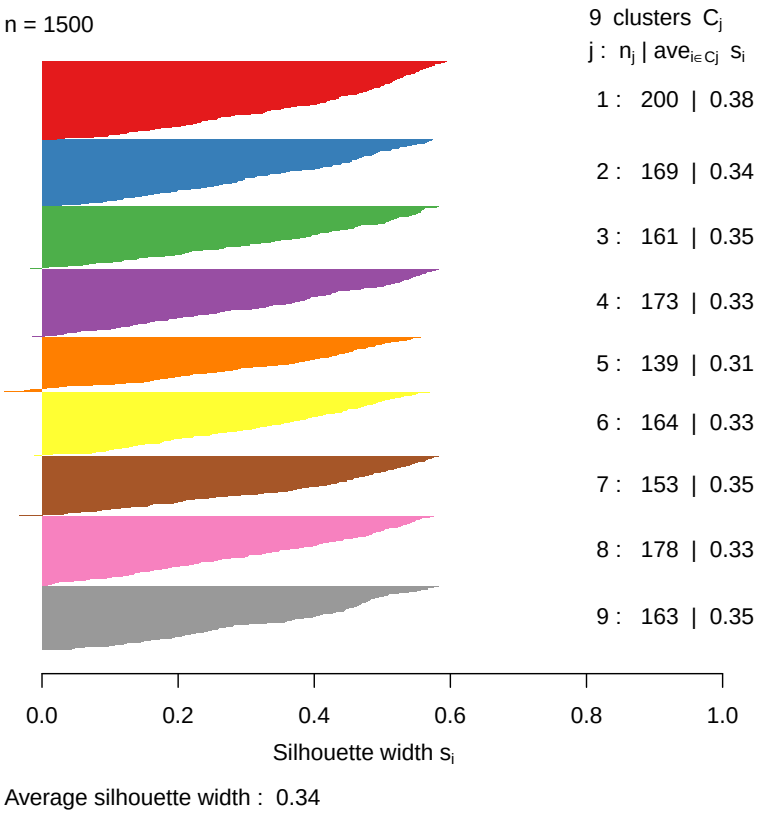

Figure 10.43: Silhouette plot for k-means clustering of the blobs data set with k=9.

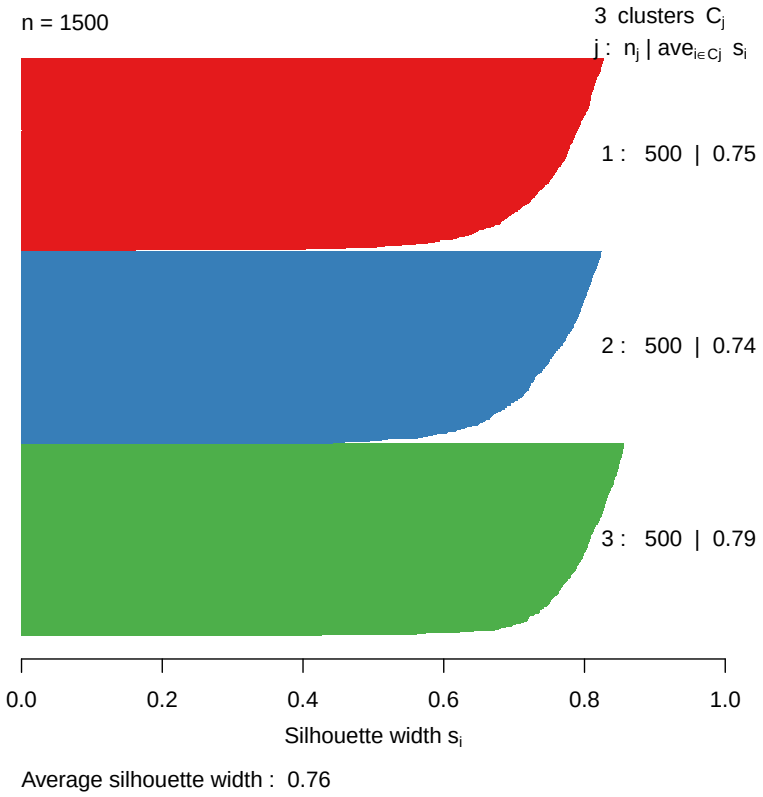

Figure 10.44: Silhouette plot for k-means clustering of the blobs data set with k=3.

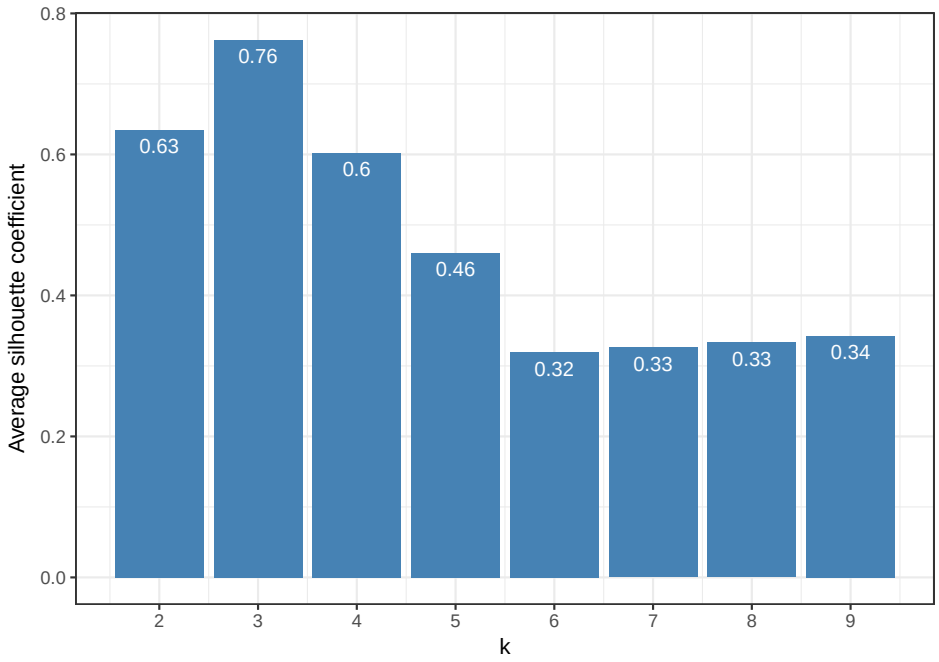

Figure 10.45: Barplot of the average silhouette coefficients resulting from k-means clustering of the blobs data-set using values of k from 1-9.

10.8 Exercises

Exercise solutions: I

Solutions to exercises can be found in appendix I.

Appendix A

Resources

A.1 Python

scikit-learn

A.2 Machine learning data set repositories

A.2.1 MLDATA

mldata.org

This repository manages the following types of objects:

- Data Sets - Raw data as a collection of similarily structured objects.
- Material and Methods - Descriptions of the computational pipeline.
- Learning Tasks - Learning tasks defined on raw data.
- Challenges - Collections of tasks which have a particular theme.

A.2.2 UCI Machine Learning Repository

Machine learning database at the University of California, Irvine, School of Information and Computer Sciences (Lichman, 2013).

Appendix B

Solutions ch. 3 - Linear models and matrix algebra

Solutions to exercises of chapter 3.

B.1 Exercise 1

B.2 Exercise 2

Appendix C

Solutions ch. 4 - Linear and non-linear logistic regression

Solutions to exercises of chapter 4.

C.1 Exercise 1

C.2 Exercise 2

Appendix D

Solutions ch. 5 - Nearest neighbours

Solutions to exercises of chapter 5.

D.1 Exercise 1

D.2 Exercise 2

Appendix E

Solutions ch. 6 - Decision trees and random forests

Solutions to exercises of chapter 6.

E.1 Exercise 1

E.2 Exercise 2

Appendix F

Solutions ch. 7 - Support vector machines

Solutions to exercises of chapter 7.

F.1 Exercise 1

F.2 Exercise 2

Appendix G

Solutions ch. 8 - Artificial neural networks

Solutions to exercises of chapter 8.

G.1 Exercise 1

G.2 Exercise 2

Appendix H

Solutions ch. 9 - Dimensionality reduction

Solutions to exercises of chapter 9.

H.1 Exercise 1

H.2 Exercise 2

Appendix I

Solutions ch. 10 - Clustering

Solutions to exercises of chapter 10.

I.1 Exercise 1

I.2 Exercise 2

Bibliography

Lichman, M. (2013). UCI machine learning repository.

Schubert, E., Sander, J., Ester, M., Kriegel, H. P., and Xu, X. (2017). Dbscan revisited, revisited: Why and how you should (still) use dbscan. *ACM Trans. Database Syst.*, 42(3):19:1–19:21.

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.

Xie, Y. (2017). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.4.