Chapter 7

Cluster Analysis and Trees

A problem which often arises in Bioinformatics is to find genes which have similar expression patterns. Genes that express similarly under certain conditions may have similar functions. In general, cluster analysis (i.e. *unsupervised learning*) consists of several methods for discovering a subset of data points (such as genes) which form a group under some observable similarity criteria (such as gene expression). These methods are based on a *distance function* and an algorithm to join data points into clusters (or groups) based on their relative distances to each other. *Hierarchical clustering* analysis is intuitively appealing and often applied in bioinformatics as a starting point. single-linkage, and other heirarchical methods, are appealing in that several clusters of genes can be discovered without specifying the number of clusters beforehand. Other methods, such as *k-means clustering*, do require that the number of k clusters be specified. Each heirarchical clustering method produces a tree which represents similar genes as closely connected leaves in clades (subtrees) and dissimilar ones as members in distant clades within the tree.

Another measure to investigate similarity or dependency of pairs of gene expressions is the *correlation coefficient*. It's also often useful to search a data set for directions of large variance within the variable space. That is, since gene expression data sets tend to be large, it's often important to discover important "directions" in the data that may capture or explain most of the observed variance. A frequently used method to find such directions is *principal components analysis (PCA)*. We will explore PCA's basic properties and how it can be applied in combination with cluster analysis.

When it is difficult to formulate distributional assumptions of a statistic, it's very useful to construct a *confidence interval* around the statistic. In this chapter, we will illustrate how the *bootstrap* method can be applied to construct 95% confidence intervals. Lastly, we will explore several examples to clarify the applications of cluster analysis and principal components analysis.

7.1 Distance

The concept of distance plays a crucial role in all types of cluster analysis. For real numbers a and b, a possible *distance* function $d(a,b)$ is defined as the absolute value of their difference:

$$d(a,b) = |a-b| = \sqrt{(a-b)^2}.$$

The properties of a distance function should be in line with our intuition. That is, if $a = b$, then $d(a,a) = 0$ and if $a \neq b$, then $d(a,b) > 0$. Hence, the distance measure should be *definitive* in the sense that $d(a,b) = 0$ if and only if $a = b$. Since the square is *symmetric*, it also follows that:

$$d(a,b) = |a-b| = \sqrt{(a-b)^2} = \sqrt{(b-a)^2} = |b-a| = d(b,a).$$

In other words, the distance between a and b equals that between b and a: $d(a,b) = d(b,a)$. Furthermore, it holds that for all points c between a and b, $d(a,b) = d(a,c) + d(c,b)$. In addition. for all points c not between a and b, it follows that $d(a,b) < d(a,c) + d(c,b)$. The latter two notions can be summarized by the so-called triangle inequality. That is, for all real c it holds that:

$$d(a,b) \leq d(a,c) + d(c,b).$$

In other words, the distance going directly from a to b is shorter than going from a to c to b. Lastly, the distance between two points a and b should increase as these move further apart.

Example 1. Let $a = 1$ and $b = 3$. Then the distance $d(a,b) = d(1,3) = 2$. The number $c = 2$ is between a and b, so that $d(1,3) = 2 = 1+1 = d(1,2) + d(2,3)$ and the triangle inequality becomes an equality.

When analyzing gene expression values for several patients, it's important to define a distance between vectors of gene expressions, such as the distance between $\boldsymbol{a} = (a_1, \cdots, a_n)$ and $\boldsymbol{b} = (b_1, \cdots, b_n)$. In this chapter, we will focus mainly on the Euclidian distance, which is defined as the root of the sum of the squared differences:

$$d(\boldsymbol{a}, \boldsymbol{b}) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}.$$

The Euclidian distance measure satisfies the above properties of definiteness, symmetry, and triangle inequality. Although many other, but often highly similar, distance functions are available, we will mainly concentrate on the Euclidian distance because it is applied most frequently in bioinformatics.

Example 2. Suppose that $\boldsymbol{a} = (a_1, a_2) = (1, 1)$ and $\boldsymbol{b} = (b_1, b_2) = (4, 5)$. Then

$$d(\boldsymbol{a}, \boldsymbol{b}) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2} = \sqrt{(1-4)^2 + (1-5)^2} = \sqrt{9+16} = 5.$$

Since the differences are squared, it's immediate that the distance from $\boldsymbol{a}$ to $\boldsymbol{b}$ equals that from $\boldsymbol{b}$ to $\boldsymbol{a}$: $d(\boldsymbol{a}, \boldsymbol{b}) = d(\boldsymbol{b}, \boldsymbol{a})$. For $\boldsymbol{c} = (c_1, c_2) = (2, 2)$, we have that $d(\boldsymbol{a}, \boldsymbol{c}) = \sqrt{2}$, $d(\boldsymbol{b}, \boldsymbol{c}) = \sqrt{2^2 + 3^2} = \sqrt{13}$. Hence,

$$d(\boldsymbol{a}, \boldsymbol{b}) = 5 < \sqrt{2} + \sqrt{13} = d(\boldsymbol{a}, \boldsymbol{c}) + d(\boldsymbol{b}, \boldsymbol{c}),$$

so that the triangle inequality is strict. This is in line with our intuitive idea that the road directly from $\boldsymbol{a}$ to $\boldsymbol{b}$ is shorter than from $\boldsymbol{a}$ to $\boldsymbol{b}$ via $\boldsymbol{c}$.

Example 3. To compute the Euclidian distance between two vectors we can do the following:

```
> a <- c(1,1); b <- c(4,5)
> sqrt(sum((a-b)^2))
[1] 5
```

Example 4: Distances between Cyclin gene expressions. We can use the built-in function `dist()` to calculate the Euclidian distance between two vectors of gene expression values. To select the genes related to the biological term "Cyclin" and then compute the Euclidian distance between their expression values in the Golub data, we can do the following:

```
> library(multtest); data(golub)
> cyclins <- grep("Cyclin",golub.gnames[,2])
> golub.gnames[cyclins,2]
 [1] "CCND2 Cyclin D2"
 [2] "CDK2 Cyclin-dependent kinase 2"
 [3] "CCND3 Cyclin D3"
 [4] "CDKN1A Cyclin-dependent kinase inhibitor 1A (p21, Cip1)"
 [5] "CCNH Cyclin H"
 [6] "Cyclin-dependent kinase 4 (CDK4) gene"
 [7] "Cyclin G2 mRNA"
 [8] "Cyclin A1 mRNA"
 [9] "Cyclin-selective ubiquitin carrier protein mRNA"
[10] "CDK6 Cyclin-dependent kinase 6"
[11] "Cyclin G1 mRNA"
[12] "CCNF Cyclin F"
> dist.cyclin <- dist(golub[cyclins,],method="euclidian")
> distanceMatrix <- as.matrix(dist.cyclin)
> rownames(distanceMatrix) <- colnames(distanceMatrix) <- golub.gnames[cyclins,3]
> distanceMatrix[1:5,1:5]
           D13639_at M68520_at M92287_at U09579_at U11791_at
D13639_at   0.000000  8.821806  11.55349 10.056814  8.669112
M68520_at   8.821806  0.000000  11.70156  5.931260  2.934802
M92287_at  11.553494 11.701562   0.00000 11.991333 11.900558
U09579_at  10.056814  5.931260  11.99133  0.000000  5.698232
U11791_at   8.669112  2.934802  11.90056  5.698232  0.000000
```

Above, we first use the `grep()` function to find the indices of those genes with the phrase "Cyclin" in their names and save the indices in a vector called `index`. The pair-wise euclidian distances between all these genes are saved in a matrix called `distanceMatrix`. The diagonal of `distanceMatrix` contains distances between identical genes, and is therefore always zero. The distance between the first gene (CCND2 Cyclin D2) and the third (CCND3 Cyclin D3) is relatively small, which is in line with the fact the these genes are paralogs and have similar functions. However, note that there are genes with even smaller distances.

Example 5: Finding the ten genes with expression patterns most similar to the MME gene. After selecting one or more genes, it's often useful to find other genes which have the most similar expressions patterns to the selected ones. This can be done with the `genefinder()` function. The gene to match can be specified either with an index or a name consistent with the geneNames of the exprSet. To find genes from the ALL data (Chiaretti et al., 2004) with expression patterns close to the MME expression values of the probe with identifier `1389_at`, we can do the following:

```
> library("genefilter"); library("ALL"); data(ALL)
> closeto1389_at <- genefinder(ALL, "1389_at", 10, method = "euc")
> closeto1389_at[[1]]$indices
 [1]  2653  1096  6634  9255  6639 11402  9849  2274  8518 10736
```

```
> round(closeto1389_at[[1]]$dists,1)
 [1] 12.6 12.8 12.8 12.8 13.0 13.0 13.1 13.2 13.3 13.4
> featureNames(ALL)[closeto1389_at[[1]]$indices]
 [1] "32629_f_at" "1988_at"    "36571_at"   "39168_at"   "36576_at"
 [6] "41295_at"   "39756_g_at" "32254_at"   "38438_at"   "40635_at"
>
```

The function `genefinder()` produces a list from which row numbers and then probe names can be extracted.[1] From the output, we see that the gene expressions of row 2653 with probe identifier `32629_f_at` has the smallest distance (12.6) from those of `1389_at`.

7.2 Two types of Cluster Analysis

Two of the most important types of cluster analysis are *hierarchical* and *k-means.*

7.2.1 Hierarchical clustering

A cluster I is simply a set of data points $I = \{\boldsymbol{x}_i\}$, where $\boldsymbol{x}_i$ is the i-th vector of gene expressions. In *hierarchical clustering*, the gene expression samples are iteratively clustered together, based upon expression similarity, to form a binary tree. The samples that are most similar are clustered together first and then those clusters are iteratively clustered with each other until one large binary tree (called a *dendogram*) emerges. While other distance metrics exist, the Euclidean distance is almost always used to measure the distance between two vectors (samples) of gene expression or other biological observations. However, there are three main linkage criteria that are commonly used to determine the distance between two sets of observations: *single-linkage*, *complete-linkage*, and *average-linkage* or *UPGMA* clustering. Each of the three linkage criteria have certain advantages and disadvantages. However, in the context of gene expression clustering, *UPGMA* is the most commonly used linkage criteria since it is more robust to noise than the other two.

[1]For information on lists, see Chapter 6 of the manual "An Introduction to R".

Single-linkage clustering

In single-linkage cluster analysis, the distance between clusters I and J is defined as the smallest distance over all pairs of points from the two clusters:

$$d(I,J) = \min_{i,j} \{d(\boldsymbol{x}_i, \boldsymbol{x}_j,) : \boldsymbol{x}_i \text{ in } I \quad \text{and} \quad \boldsymbol{x}_j \text{ in } J\}.$$

Hence, the distance between the two clusters is the same as that of the *nearest neighbors*. The algorithm of single-linkage cluster analysis starts with creating as many clusters as data points and each cluster contains just one data point. Next, the nearest two are determined via the shortest distance between any two data points in each cluster, and these nearest two clusters are merged into one. Similarly, then the next two nearest clusters are determined and merged into one. This process continues until all the data points are contained in one big cluster.

Complete-linkage clustering

In complete-linkage cluster analysis, the distance between clusters I and J is defined as the farthest distance over all pairs of points from the two clusters:

$$d(I,J) = \max_{i,j} \{d(\boldsymbol{x}_i, \boldsymbol{x}_j,) : \boldsymbol{x}_i \text{ in } I \quad \text{and} \quad \boldsymbol{x}_j \text{ in } J\}.$$

Hence, the distance between the two clusters is the same as that of the *farthest neighbors*. The algorithm for building the dendrogram is the same as for single-linkage clustering, except for that the distance between two clusters is the farthest distance between any two data points in each cluster.

Average-linkage or UPGMA clustering

In average-linkage or Unweighted Pair Group Method with Arithmetic Mean (UPGMA) cluster analysis, the distance between clusters I and J is defined as the average (arithmetic mean) distance over all pairs of points from the two clusters:

$$d(I,J) = \frac{1}{|I| \cdot |J|} \sum_{\boldsymbol{x}_i \in I} \sum_{\boldsymbol{x}_j \in J} d(\boldsymbol{x}_i, \boldsymbol{x}_j).$$

The algorithm for building the dendrogram is the same as for single-linkage and complete-linkage clustering, except for that the distance between two clusters is the average distance between all pairings of two data points from the two clusters.

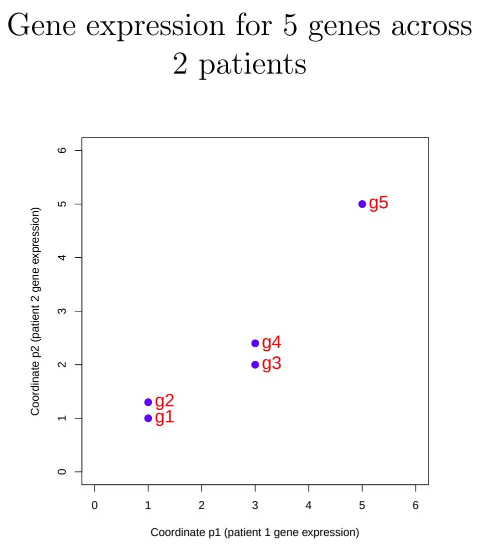

Figure 7.1: Plot of five gene expressions for 2 patients that are to be clustered.

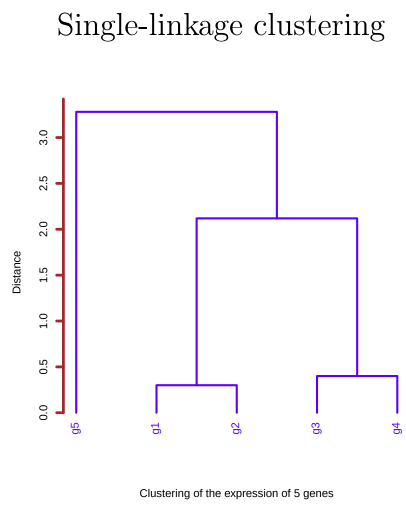

Figure 7.2: Tree of single-linkage cluster analysis.

Example 1: single-linkage with 5 genes. To illustrate single-linkage cluster analysis, suppose we have the following five gene expressions $\boldsymbol{g}_1 = (1,1)$, $\boldsymbol{g}_2 = (1,1.3)$, $\boldsymbol{g}_3 = (3,2)$, $\boldsymbol{g}_4 = (3,2.4)$, and $\boldsymbol{g}_5 = (5,5)$, from two patients. The expressions for each gene can be viewed as coordinates on two perpendicular axis $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$. The code below illustrates the single-linkage clustering process using the `hclust()` function. The results can seen in Figure 7.1:

```
> names <- list(c("g1","g2","g3","g4","g5"),c("Coordinate p1 (patient 1 gene expression)","
   ↪ Coordinate p2 (patient 2 gene expression)"))
> sl.clus.dat <- matrix(c(1,1,1,1.3,3,2,3,2.4,5,5),ncol = 2, byrow = TRUE,dimnames = names)
> plot(sl.clus.dat,
+      pch=19,
+      col="blue",
+      cex=1.4,
+      xlim=c(0,6),
+      ylim=c(0,6))
> text(sl.clus.dat,labels=row.names(sl.clus.dat), pos=4, col="red", cex=1.6)
> print(dist(sl.clus.dat,method="euclidian"),digits=3)
     g1   g2   g3   g4
g2 0.30
g3 2.24 2.12
g4 2.44 2.28 0.40
g5 5.66 5.45 3.61 3.28
```

```
> sl.out<-hclust(dist(sl.clus.dat,method="euclidian"),method="single")
> plot(sl.out,
+      lwd=3,
+      col="blue",
+      col.axis = "brown",
+      ylab="Distance",
+      xlab="Clustering of the expression of 5 genes",
+      hang=-1,
+      main=NA,
+      sub=NA,
+      axes=FALSE)
> axis(side = 2, at = seq(0, 3.5, .5), col = "brown",labels = TRUE, lwd = 4)
```

Above, initially each data point is seen as a separate cluster. Then the nearest two points (genes) from the Euclidian distance matrix are $\boldsymbol{g}_1$ and $\boldsymbol{g}_2$, having $d(\boldsymbol{g}_1, \boldsymbol{g}_2) = 0.10$. These two data points are merged into one cluster $I = \{\boldsymbol{g}_1, \boldsymbol{g}_2\}$. In Figure 7.2 this is illustrated by the horizontal line at height 0.10 in the tree. The other three data points $\boldsymbol{g}_3, \boldsymbol{g}_4, \boldsymbol{g}_5$ are seen as three different clusters. Next, the minimal distance between the clusters can be read from the Euclidian distance matrix. Since the smallest is $d(\boldsymbol{g}_3, \boldsymbol{g}_4) = 0.30$, the new cluster is $J = \{\boldsymbol{g}_3, \boldsymbol{g}_4\}$, corresponding to the horizontal line at height 0.30 in Figure 7.2. Now there are three clusters, I, J, and $K = \{\boldsymbol{x}_5\}$. From the Euclidian distance matrix, we now see that the shortest distance between clusters I and J is 2.19. Hence, the clusters I and J are merged into one with a corresponding horizontal line at the height of 2.19. Finally, the distance between cluster $\{\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_3, \boldsymbol{g}_4\}$ and the data point $\boldsymbol{g}_5$ is $d(\boldsymbol{g}_4, \boldsymbol{g}_5) = 3.36$, and a final horizontal line at this height connects the last data point.

Example 2: Clustering random data. It's important to have some experience with clustered data that does or does not contain "real" clusters. To illustrate this, we perform single-linkage cluster analysis on twenty data points from the same standard normal population:

```
> x<-rnorm(20,0,1)
> singlelinkage.out<-hclust(dist(x,method="euclidian"),method="single")
> plot(singlelinkage.out,
+      lwd=3,
+      col="blue",
+      col.axis = "brown",
+      ylab="Distance",
+      xlab="20 samples with normal random distances",
+      hang=-1,
+      main=NA,
+      sub=NA,
+      axes=FALSE)
> axis(side = 2, at = seq(0, 1.4, .2), col = "brown",labels = TRUE, lwd = 4)
```

From the resulting tree in Figure 7.3 one might get the impression that

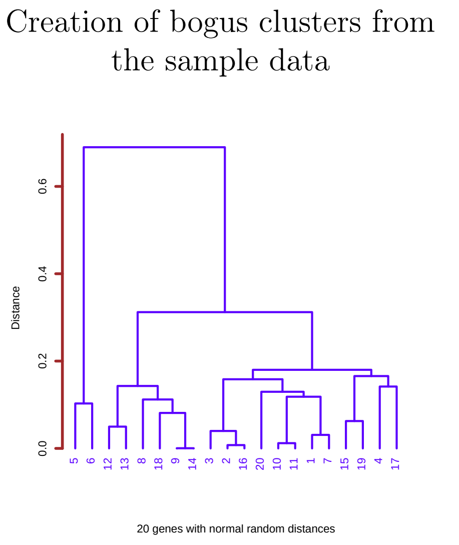

Figure 7.3: Example of a tree from data sampled from one normal distribution.

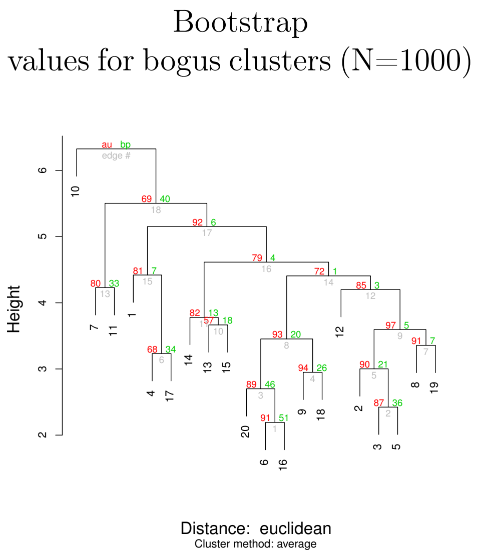

Figure 7.4: The green bootstrap values are significantly less that 100. Therefore, we have no confidence in this tree or the tree in Figure 7.3. Bootstrap was iterated 1000 times.

there are three main separate clusters in the data. However, note that there is no underlying data generation process which produces separate clusters from different populations. This example highlights an important caveat about clustering - clustering will always produce clusters in your data even if all the data is sampled from identical distributions. Therefore, in order to assess the level of uncertainty in the clustering, it is always a good idea to perform bootstrapping. During bootstrapping, many ($N > 1000$) trees are generated while resampling with replacement from the original data. Thus, every resampling is a perturbation of the data. The most common tree is displayed along with bootstrap percentages that indicate the percentage of the time that each branch was observed across all the trees. Below, we use the `pvclust()` function to perform bootstrapping on 20 samples with gene expression that is generated from the same normal distribution as Figure 7.3:

```
> x2 = matrix(rnorm(200,0,1), ncol=20) # 10 samples
> fit <- pvclust(x2, method.hclust="average", method.dist="euclidean", nboot=1000)
Bootstrap (r = 0.5)... Done.
Bootstrap (r = 0.6)... Done.
Bootstrap (r = 0.7)... Done.
Bootstrap (r = 0.8)... Done.
```

Recreation of population clusters from the sample data

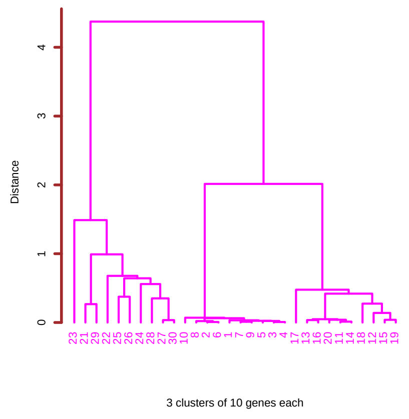

Figure 7.5: Example of a tree from data generated from 3 different normal distributions.

Bootstrap values for the 3 population clusters (N=1000)

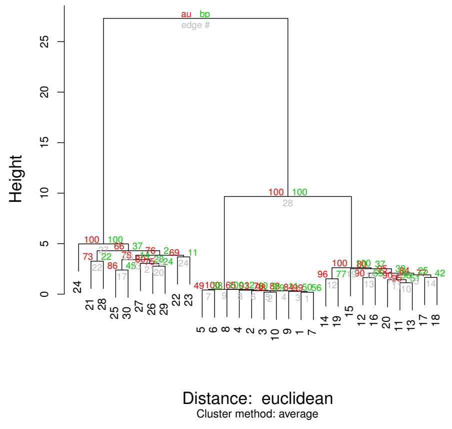

Figure 7.6: The green bootstrap values are all exactly 100 for our 3 main clusters. Therefore, we have a lot of confidence in this tree and the tree in Figure 7.5. Bootstrap was iterated 1000 times.

```
Bootstrap (r = 0.9)... Done.
Bootstrap (r = 1.0)... Done.
Bootstrap (r = 1.1)... Done.
Bootstrap (r = 1.2)... Done.
Bootstrap (r = 1.3)... Done.
Bootstrap (r = 1.4)... Done.
> plot(fit, main="", cex.lab=1.5) # dendogram with p values
```

In the resulting Figure 7.4, we see that the green bootstrap values are significantly less that 100. Therefore, we have no confidence in this tree or the tree in Figure 7.3. The `pvclust()` function also performs multi-scale bootstrap resampling (in red). However, for this analysis the normal bootstrap resampling (in green) is sufficient.

Unlike the previous example, if the data are generated by different normal distributions, then there are different processes that can produce separate (real) clusters that are indeed robust to perturbations of the data. To illustrate this, ten data points were sampled from the $N(0, 0.1)$ population, ten from $N(3, 0.5)$, and ten from $N(10, 1)$:

```
> x <- c(rnorm(10,0,0.1),rnorm(10,3,0.5),rnorm(10,10,1.0))
> sl.out<-hclust(dist(x,method="euclidian"),method="single")
> plot(sl.out,
+      lwd=3,
+      col.axis = "brown",
+      col="magenta",
+      ylab="Distance",
+      xlab="3 clusters of 10 samples each",
+      hang=-1,
+      main=NA,
+      sub=NA,
+      axes=FALSE)
> axis(side = 2, at = seq(0, 5, 1), col = "brown",labels = TRUE, lwd = 4)
```

From the tree in Figure 7.5, we see that there clearly exist three clusters. However, we need to perform bootstrapping on the tree to get rigorous p-values (in the form of percentages) for these discovered clusters:

```
> x2 = matrix(c(rnorm(100,0,0.1),rnorm(100,3,0.5),rnorm(100,10,1.0)), ncol=30)
> fit <- pvclust(x2, method.hclust="average", method.dist="euclidean", nboot=1000)
Bootstrap (r = 0.5)... Done.
Bootstrap (r = 0.6)... Done.
Bootstrap (r = 0.7)... Done.
Bootstrap (r = 0.8)... Done.
Bootstrap (r = 0.9)... Done.
Bootstrap (r = 1.0)... Done.
Bootstrap (r = 1.1)... Done.
Bootstrap (r = 1.2)... Done.
Bootstrap (r = 1.3)... Done.
Bootstrap (r = 1.4)... Done.
> plot(fit, main="", cex.lab=1.5) # dendogram with p values
```

In the resulting Figure 7.6, we see that the green bootstrap values are all exactly 100 for our 3 main clusters. Therefore, we have a lot of confidence in this tree and also the tree in Figure 7.5.

These examples illustrate that results from cluster analysis may very well reveal real population properties, but that some caution is needed and bootstrapping is an effective means to measure uncertainty within the clustering tree.

Example 3: CCND3 and Zyxin gene expression. Recall that the first 27 patients have ALL and the remaining 11 have AML, and that we found earlier that the expression values of the genes CCND3 (Cyclin D3) and Zyxin differ between the ALL and AML patient groups. Figure 7.7 illustrates how gene expression data can be used to properly cluster the patients into the two different patient groups. The code below performs the single-linkage cluster analysis and produces figures 7.7 and 7.8:

```
> data(golub, package="multtest")
> zyxin <- grep("Zyxin",golub.gnames[,2])
> ccnd3 <- grep("CCND3",golub.gnames[,2])
```

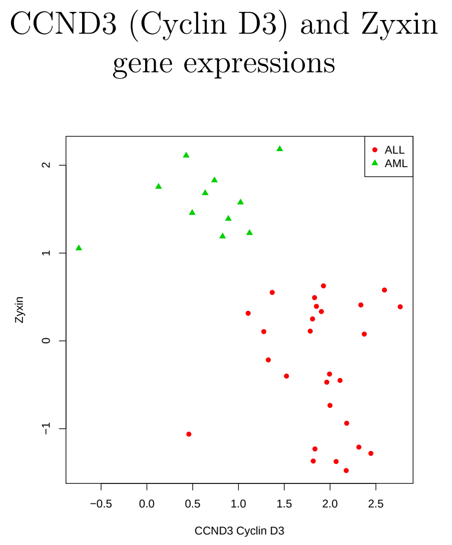

Figure 7.7: Plot of gene CCND3 (Cyclin D3) and Zyxin expressions for ALL and AML patients.

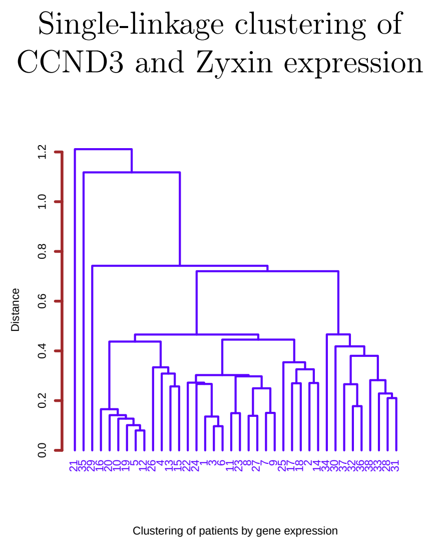

Figure 7.8: Single-linkage cluster diagram from gene CCND3 (Cyclin D3) and Zyxin expressions values. All but two (29, 35) of the AML patients (28-38) cluster together on the right-hand side.

```
> clusdata <- data.frame(golub[ccnd3,],golub[zyxin,])
> colnames(clusdata)<-c("CCND3 Cyclin D3","Zyxin")
> golubFactor <- factor(golub.cl,levels=0:1, labels= c("ALL","AML"))
> plot(clusdata,
+      pch=as.numeric(golubFactor) + 15,
+      col=as.numeric(golubFactor) + 1)
> legend("topright",
+        legend=c("ALL","AML"),
+        pch=16:17,
+        col=c(2,3))
>
> plot(hclust(dist(clusdata,method="euclidian"),method="single"),
+      lwd=3,
+      col="blue",
+      col.axis = "brown",
+      ylab="Distance",
+      xlab="Clustering of patients by gene expression",
+      hang=-1,
+      main=NA,
+      sub=NA,
+      axes=FALSE)
> axis(side = 2, at = seq(0, 1.2, .2), col = "brown",labels = TRUE, lwd = 4)
```

Figure 7.8 gives the resultant tree from the single-linkage cluster analysis. Apart from three misplaced patients, the tree contains two main clusters (or clades) that correctly correspond to the two patient groups.

7.2.2 k-means clustering

k-means cluster analysis is a popular method in bioinfomatics. It is accomplished by minimizing the within cluster sum of squares over k clusters. That is, given the data points $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n$ the method seeks to minimize the function

$$\sum_{k=1}^{k} \sum_{i \in I_k}^{n_k} d^2(\boldsymbol{x}_i, \boldsymbol{a}_k)$$

over all possible points $\boldsymbol{a}_1, \cdots \boldsymbol{a}_k$. This is accomplished by an algorithm (Hartigan & Wong, 1979) which starts by partitioning the data points into k initial clusters, either at random or using some heuristic device. It then computes the cluster means (step 1) and constructs a new partition by associating each point with the closest cluster mean (step 2). The latter yields new clusters of which the means are re-calculated (step 1 again). Then it constructs a new partition by associating each point with the closest, new cluster mean (step 2 again). These two steps are repeated until convergence - which occurs when the data points no longer change clusters. The iterative algorithm is fast in the sense that it often converges in less iterations than the number of points n, but it may not converge to the global minimum. For the optimal points $\boldsymbol{a}_1, \cdots \boldsymbol{a}_k$, it holds that these points are equal to the mean per cluster, that is $\boldsymbol{a}_k = \overline{\boldsymbol{x}}_k$ for each cluster k.

Example 1: k-means cluster analysis for 2 patients and 50 genes. To illustrate k-means cluster analysis, we shall simulate gene expressions from two different normal populations. That is, we will randomly sample 50 gene expressions for two patients from the $N(0, 0.5)$ population and 50 expressions for the two patients from the $N(2, 0.5)$ population. The data points are collected in two 50x2 matrices. Then, 2-means cluster analysis is performed on the combined 100 data points:

```
> data <- rbind(matrix(rnorm(100,0,0.5), ncol = 2), matrix(rnorm(100,2,0.5), ncol = 2))
> cl <- kmeans(data, 2)
K-means clustering with 2 clusters of sizes 50, 50

Cluster means:
        [,1]       [,2]
1 1.87304978 2.01940342
```

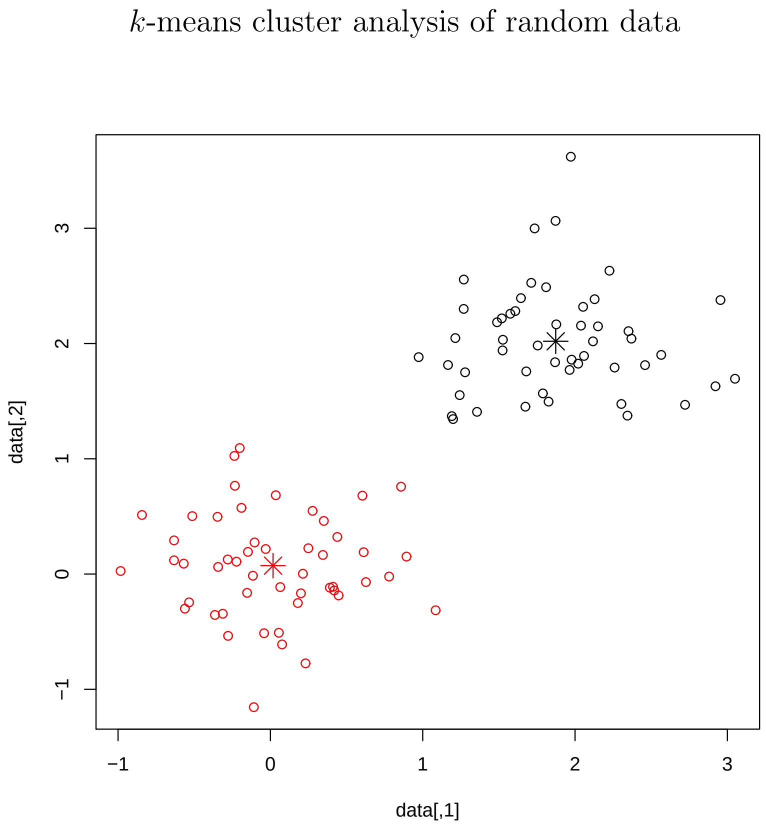

Figure 7.9: k-means clustering of data generated from two different normal distributions.

```
2 0.01720177 0.07320413

Clustering vector:
  [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [38] 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 1 1 1 1 1 1 1 1
 [75] 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

Within cluster sum of squares by cluster:
[1] 22.60733 20.54411

Available components:
[1] "cluster"  "centers"  "withinss" "size"
```

Above, the output of the k-means cluster analysis is assigned to a list called `cl`. Observe that the cluster means are fairly close to the population means $(0, 0)$ and $(2, 2)$. The `Clustering vector` indicates to which cluster each data point (gene) belongs and these correspond exactly to the two populations from which the data are sampled. The variable `cl$cluster` contains the cluster membership and can be used to specify the color of each data point in a plot:

```
> plot(data, col = cl$cluster)
> points(cl$centers, col = 1:2, pch = 8, cex=2)
```

The data points are plotted in red and black circles and the cluster means by a star, see Figure 7.9. The returned sum of the within-cluster sum of squares equals the minimal function value obtained by the algorithm.

In practice, before performing a k-means cluster analysis, single-linkage cluster analysis may reveal the correct number of existing clusters in the data. However, if the number of clusters is not clear, then it becomes questionable whether k-means is appropriate. For cases where the number of clusters is only moderately clear, the k-means algorithm is more likely to get stuck in a solution which is only locally (and not globally) optimal. To cope with the danger of suboptimal solutions, we can simply run the algorithm repeatedly by using the `nstart` option. Another option is to use rational initial starting values for the cluster means based on prior knowledge or hypotheses. In particular, the sample means of potential clusters or the hypothesized population means can be used:

```
> initial <- matrix(c(0,0,2,2), nrow = 2, ncol=2, byrow=TRUE)
> cl <- kmeans(data, initial, nstart = 10)
```

In addition, the so-called *bootstrap* (Efron, 1979) can be used to estimate 95% confidence intervals around cluster means. The idea behind the *bootstrap* is to re-sample with replacement from the dataset many (> 1000) times and then compute the quantiles for the corresponding confidence intervals:

```
> n <- 100; nboot<-1000
> boot.cl <- matrix(0,nrow=nboot,ncol = 4)
> for (i in 1:nboot) {
+   dat.star <- data[sample(1:n,replace=TRUE),]
+   cl <- kmeans(dat.star, initial, nstart = 10)
+   boot.cl[i,] <- c(cl$centers[1,],cl$centers[2,])
+ }
> quantile(boot.cl[,1],c(0.025,0.975))
      2.5%      97.5%
-0.1098886  0.1627979
> quantile(boot.cl[,2],c(0.025,0.975))
       2.5%       97.5%
-0.04830563  0.19721732
> quantile(boot.cl[,3],c(0.025,0.975))
    2.5%    97.5%
1.730495 2.009014
> quantile(boot.cl[,4],c(0.025,0.975))
```

```
    2.5%    97.5%
1.898407 2.162019
```

From the bootstrap confidence intervals above, the null hypothesis that the cluster population means are equal to $(0, 0)$ and $(2, 2)$ is not rejected.

Example 2: CCND3 and Zyxin gene expression. Above, we found that the expression values of the genes CCND3 (Cyclin D3) and Zyxin are closely related to the distinction between ALL and AML. Hence, a 2-means cluster analysis of these gene expression values is appropriate here:

```
> data <- data.frame(golub[ccnd3,],golub[zyxin,])
> colnames(data) <- c("CCND3 (Cyclin D3)","Zyxin")
> cl <- kmeans(data, 2,nstart = 10)
> cl
K-means clustering with 2 clusters of sizes 11, 27

Cluster means:
  CCND3 (Cyclin D3)      Zyxin
1         0.6355909  1.5866682
2         1.8938826 -0.2947926

Clustering vector:
 [1] 2 2 2 2 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

Within cluster sum of squares by cluster:
[1]  4.733248 19.842225
 (between_SS / total_SS =  62.0 %)
```

The two clusters discriminate exactly the ALL patients from the AML patients. This can also be seen from Figure 7.10, where the expression values of CCND3 (Cyclin D3) are depicted on the horizontal axis and those of Zyxin on the vertical, and the ALL patients are in black and the AML patients in red.

The cluster means and their confidence intervals can be estimated using the bootstrap:

```
> mean(data.frame(boot.cl))
        X1         X2         X3         X4 
 0.6381860  1.5707477  1.8945878 -0.2989426 
> quantile(boot.cl[,1],c(0.025,0.975))
     2.5%     97.5%
0.2548907 0.9835898
> quantile(boot.cl[,2],c(0.025,0.975))
    2.5%    97.5%
1.259608 1.800581
> quantile(boot.cl[,3],c(0.025,0.975))
    2.5%    97.5%
1.692813 2.092361
> quantile(boot.cl[,4],c(0.025,0.975))
       2.5%       97.5%
-0.60802142 -0.02420802
```

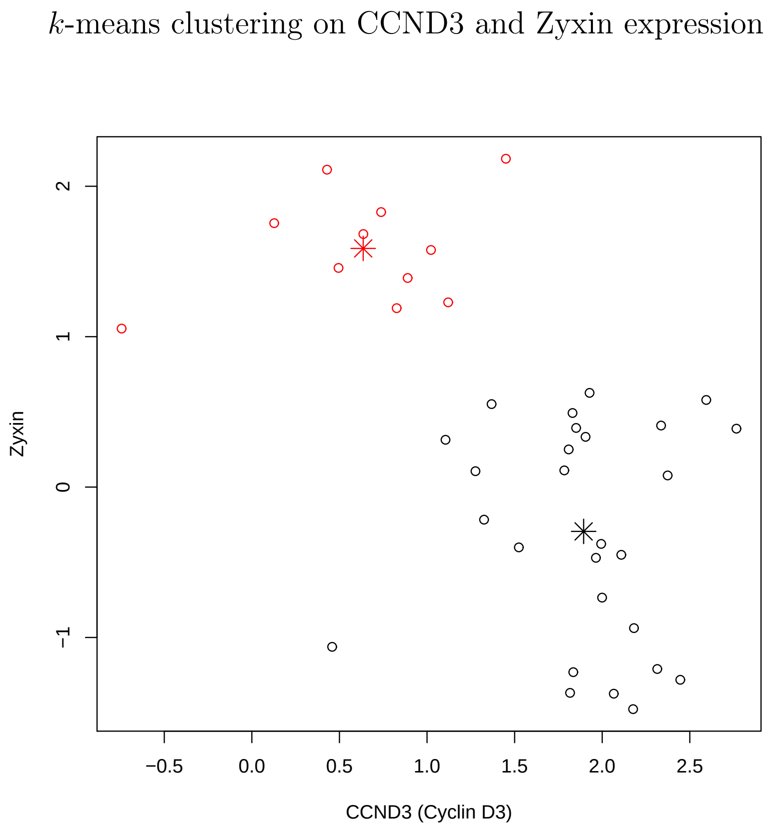

Figure 7.10: Plot of the k-means (stars) in the k-means cluster analysis on CCND3 (Cyclin D3) and Zyxin gene expression discriminating between ALL (black) and AML (red) patients.

The difference between (1) the bootstrap means and (2) the k-means from the original data gives an estimate of the estimation bias. In the example above, we see that the bias is small. The estimation is quite precise because the 95% bootstrap confidence intervals are very small.

7.3 The correlation coefficient

A frequently used coefficient to express the degree of linear relationship between two sets of gene expression values is the *correlation coefficient* ρ. For two sets of gene expressions $\boldsymbol{x} = (x_1, \cdots, x_n)$ and $\boldsymbol{y} = (y_1, \cdots, y_n)$, the correlation coefficient ρ is estimated by:

$$\widehat{\rho} = \frac{\sum_{i=1}^{n}(x_i - \overline{x}_i)(y_j - \overline{y}_j)}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x}_i)^2 \sum_{i=1}^{n}(y_j - \overline{y}_j)^2}}.$$

The value of the correlation coefficient is always between minus one and plus one: $-1 \geq \rho \leq 1$. If the value is close to either of these values, then the variables are linearly related in the sense that the first is a linear transformation of the second. That is, there are constants a and b such that $ax_i + b = y_i$ for all i. The null hypothesis $H_0 : \rho = 0$ can be tested against the alternative $H_0 : \rho \neq 0$ using the function `cor.test()`.

Example 1: Teaching demonstration. To develop an intuition with respect to the correlation coefficient, the function `run.cor.examp(1000)` in the `TeachingDemos` package is quite useful:

```
> library(TeachingDemos)
> run.cor.examp(1000)
```

`run.cor.examp()` launches an interactive plot with 1000 data points on two random variables X and Y. When the correlation is near zero, then the data points are distributed along contours of circles. By moving the slider slowly from the left to the right it can be observed that all points are approximately on a straight line. If the sign of the correlation coefficient is positive, then small/large values of X tend to go together with small/large values of Y.

Example 2: Another teaching demonstration. With the demo function `put.points.demo()`, it's possible to add and delete points to a plot which interactively re-computes the value for the correlation coefficient:

```
> put.points.demo()
```

By first creating a few points that lie together on a circle, the corresponding correlation coefficient will be near zero. Next, by adding an outlier far for this circle of points we observe that the correlation coefficient changes to nearly ± 1 - which illustrates that the correlation coefficient is not robust against outliers.

Example 3: MCM3 gene expression. We will illustrate the correlation coefficient by two sets of expression values of the MCM3 gene in the Golub et al. (1999) data. This gene encodes for highly conserved mini-chromosome maintenance proteins (MCM) which are involved in the initiation of eukaryotic genome replication. Below, we find its row numbers, collect the gene expression value in vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, and compute the value of the correlation coefficient with the function `cor(x,y)`:

```
> mcm3 <- grep("MCM3",golub.gnames[,2])
> mcm3
[1] 2289 2430
> golub.gnames[mcm3,2]
[1] "MCM3 Minichromosome maintenance deficient (S. cerevisiae) 3"
[2] "MCM3 Minichromosome maintenance deficient (S. cerevisiae) 3"
> x <- golub[mcm3[1],]; y <- golub[mcm3[2],]
> cor(x,y)
[1] 0.6376217
```

The value is positive which means that larger values of $\boldsymbol{x}$ occur together with larger values of $\boldsymbol{y}$ and vice versa. This can also be observed by `plot(x,y)`. The null hypothesis $H_0 : \rho = 0$ can be tested against the alternative $H_0 : \rho \neq 0$ using the function `cor.test()`. It also estimates a 95% confidence interval for ρ:

```
> plot(x,y)
> cor.test(x,y)

        Pearson's product-moment correlation

data:  x and y
t = 4.9662, df = 36, p-value = 1.666e-05
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.3993383 0.7952115
sample estimates:
      cor
0.6376217
```

The test is based on the normality assumption and therefore calculates a t-value. Since the corresponding p-value is very small, we reject the null hypothesis of zero correlation. The left bound of the confidence interval falls far to the right-hand-side (RHS) of zero.

Example 4: Confidence interval using the bootstrap. Another method to construct a 95% confidence interval is with the bootstrap. The idea (Efron, 1979) is to obtain a thousand samples from the original sample (with replacement) and to compute the correlation coefficient for each of these. This resampling with replacement yields a thousand correlation coefficients from which the quantiles for the 95% confidence interval can be computed:

```
> nboot <- 1000; boot.cor <- matrix(0,nrow=nboot,ncol = 1)
> data <- matrix(c(x,y),ncol=2,byrow=FALSE)
> for (i in 1:nboot){
+    dat.star <- data[sample(1:nrow(data),replace=TRUE),]
+    boot.cor[i,] <- cor(dat.star)[2,1]}
> mean(boot.cor)
[1] 0.6534167
> quantile(boot.cor[,1],c(0.025,0.975))
     2.5%     97.5%
0.2207915 0.9204865
```

Observe that the 95% confidence interval is larger than that found by `cor.test()`. This discrepancy indicates that the assumption of normality may not be completely valid here. However, since the confidence interval still does not contain zero, we again reject the null hypothesis of zero correlation.

Example 5: Application to the Golub (1999) data. The ALL and AML patients of the Golub et al. (1999) data are indicated by zero and ones in the binary vector `golub.cl`. We can select genes by the correlation of their expression values with this binary vector. Below, we create a sorted list of such correlations using the functions `apply()`, `cor()`, and `order()`:

```
> library(multtest); data(golub)
> corgol<- apply(golub, 1, function(x) cor(x,golub.cl))
> o <- order(corgol)
```

Looking at the top 10 genes with the highest correlations with `golub.gnames[o[3041:3051],2]`, we see various genes referred to by Golub et al. (1999) that indeed seem to have relevant functions that distinguish leukemias. In particular, Interleukin 8 is recently related to inflammatory cytokine production in myeloid cells (Tessarz et al., 2007):

```
> golub.gnames[o[3041:3051],2]
 [1] "Interleukin 8 (IL8) gene"
 [2] "GRN Granulin"
 [3] "GLUTATHIONE S-TRANSFERASE, MICROSOMAL"
 [4] "Leukotriene C4 synthase (LTC4S) gene"
 [5] "CTSD Cathepsin D (lysosomal aspartyl protease)"
 [6] "DF D component of complement (adipsin)"
 [7] "ELA2 Elastatse 2, neutrophil"
 [8] "CD33 CD33 antigen (differentiation antigen)"
 [9] "Zyxin"
[10] "CYSTATIN A"
[11] "CST3 Cystatin C (amyloid angiopathy and cerebral hemorrhage)"
```

7.4 Principal Components Analysis (PCA)

The whole idea of principal components analysis (PCA) is to find the directions in the data along which there is maximal variation. To make the basic ideas behind PCA explicit, it helps to start with a small artificial example. Suppose that for six genes the standardized expression values for two patients (variables) are as given in Table 7.1. The data are collected in a 6 by 2 data matrix $\boldsymbol{Z}$, where the gene expression value z_{21} belongs to the second gene of the first patient:

Table 7.1: Data set for principal components analysis.

	Variable 1	Variable 2
gene 1	1.63	1.22
gene 2	−0.40	0.79
gene 3	0.93	0.97
gene 4	−1.38	−1.08
gene 5	−0.17	−0.96
gene 6	−0.61	−0.93

A direction in the data is defined as a linear combination $\boldsymbol{Zk}$ of the columns of $\boldsymbol{Z}$ by a vector $\boldsymbol{k}$ with linear weights. The i-th element of the linear combination is the weighted sum $\sum_{j=1}^{2} z_{ij}k_j$. The direction of maximal variation is defined as the linear combination with maximal variance. We use the correlation matrix to find this direction of maximal variation. The correlation matrix contains the correlations between each pair of patients (variables). In our case, the correlations between the columns (patients) in Table 7.1 can be placed in a matrix $\boldsymbol{R}$, which has ones on the diagonal and the value 0.8 elsewhere.

To illustrate a direction of variation, let's try the linear combination $\boldsymbol{k} = (2,1)$[2] of the sample correlation matrix $\boldsymbol{R}$. We use matrix multiplication to calculate the resultant vector $\boldsymbol{Rk}$:

$$\boldsymbol{Rk} = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 2.8 \\ 2.6 \end{bmatrix}.$$

[2]For the sake of simple notation we shall not use the transposition operator ^{T} to indicate rows.

Both vectors $\boldsymbol{k}$ and $\boldsymbol{Rk}$ can be plotted in the xy-plane. The vector $(2, 1)$ is plotted by drawing an arrow from (0,0) to the point with $x = 2$ and $y = 1$. This is similarly done for the vector $(2.8, 2.6)$ in Figure 7.11. We see that the two vectors (arrows) do not fall on the same line (are not collinear) and therefore have different directions. The crux of principal components analysis is that a linear combination with the same direction as the weights represents an orthogonal direction of maximum variation - and is called a principal component or eigenvector. Such is the case if $\boldsymbol{Rk}$ differs from $\boldsymbol{k}$ only by a constant of multiplication, that is there exists a constant d such that $\boldsymbol{Rk} = d\boldsymbol{k}$. We shall determine such a constant by finding the weights vector first. To do so, observe from our correlations matrix that the sum of both rows equals 1.8. Taking $\boldsymbol{k} = (1, 1)$ yields:

$$\boldsymbol{Rk} = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1.8 \\ 1.8 \end{bmatrix} = 1.8 \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 1.8\boldsymbol{k}.$$

Therefore, we obtain $d = 1.8$. A similar result follows by observing that the differences per row are equal in absolute value. That is, taking $\boldsymbol{k} = (1, -1)$ yields:

$$\boldsymbol{Rk} = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.2 \\ -0.2 \end{bmatrix} = 0.2 \begin{bmatrix} 1 \\ -1 \end{bmatrix} = 0.2\boldsymbol{k}.$$

A vector $\boldsymbol{k}$ for which $\boldsymbol{Rk} = d\boldsymbol{k}$ holds is called an *eigenvector* corresponding to the *eigenvalue* d. Eigenvectors are often re-scaled by dividing by their Euclidian length. Since the Euclidian length of $(1, 1)$ is $\sqrt{1^2 + 1^2} = \sqrt{2}$, we obtain the new eigenvector $\boldsymbol{k}_1 = (1/\sqrt{2}, 1/\sqrt{2}) \approx (0.71, 0.71)$. Since the length of eigenvector $(1, -1)$ also equals $\sqrt{2}$, the re-scaled second eigenvector equals $\boldsymbol{k}_2 = (1/\sqrt{2}, -1/\sqrt{2}) \approx (0.71, -0.71)$. Now, the first principal component is defined as $\boldsymbol{Zk}_1$ and the second as $\boldsymbol{Zk}_2$. In practical applications, the actual computation of eigenvectors and eigenvalues is performed by well-designed numerical methods (Golub & Van Loan, 1983).

Example 1: Calculating eigenvectors and eigenvalues. It is convenient to store the data of the first two columns of Table 7.1 as a matrix object called `Z`. The correlations matrix can be computed with the built-in function `cor()` and the eigenvectors and eigenvalues with the built-in function `eigen()`, as follows:

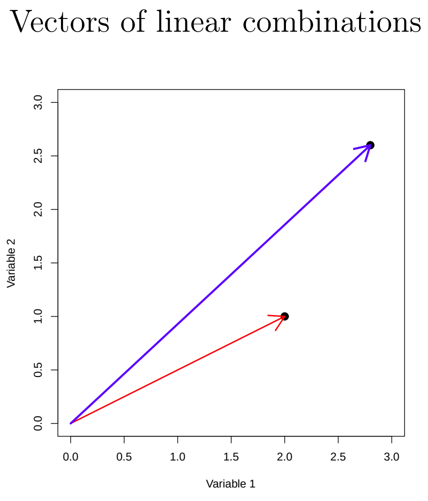

Figure 7.11: The vector [2,1] (red) is transformed to the vector [2.8,2.6] (blue) when the vector [2,1] is applied as a linear combination of the correlation matrix. Since the vectors [2,1] and [2.8,2.6] are not collinear, then the vector [2,1] is not an eigenvector.

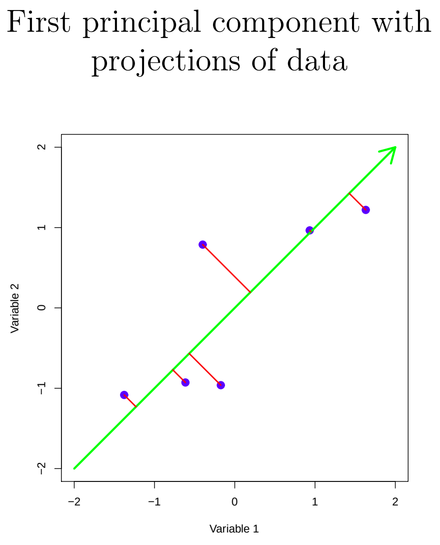

Figure 7.12: The first principal component (green) is the eigenvector with the greatest variance in the sample data. The data points are plotted with their projections (red) on the first principal component.

```
> Z <- matrix(c( 1.63, 1.22, -0.40, 0.79, 0.93, 0.97, -1.38, -1.08, -0.17, -0.96, -0.61,
    ↪ -0.93), nrow=6, byrow=TRUE)
> K <- eigen(cor(Z))
```

The output of the `eigen()` function is stored in an object called `K` which can be printed to the screen in two digits:

```
> print(K,digits=2)
$values
[1] 1.8 0.2

$vectors
     [,1]  [,2]
[1,] 0.71  0.71
[2,] 0.71 -0.71
```

The eigenvalues are assigned to `K$values` and the eigenvectors are the columns of `K$vectors`. To compute the principal components we use the matrix multiplication operator `%*%`. The first principal component is defined as the linear combination of the data with the first eigenvector, `Z %*% K$vec[,1]`. To print the component scores on the first and the second principal component we can do the following:

```
> print(Z %*% K$vec, digits=2)
      [,1]   [,2]
[1,]  2.02  0.290
[2,]  0.28 -0.841
[3,]  1.34 -0.028
[4,] -1.74 -0.212
[5,] -0.80  0.559
[6,] -1.09  0.226
```

Note that the scores are also called the new component coordinates for the data when using the new eigenvactors as the basis. To illustrate the first principal component, the six data points from the `Z` matrix are plotted as small circles in Figure 7.12. For instance, gene 1 has x coordinate 1.63 and y coordinate 1.22 and appears therefore in the upper right corner.

Also, a convenient means to perform principal components analysis is by using the built-in function `princomp()`:

```
> pca <- princomp(Z, center = TRUE, cor=TRUE, scores=TRUE)
> pca$scores
         Comp.1     Comp.2
[1,]  2.0165742 -0.2914752
[2,]  0.2747101  0.8396882
[3,]  1.3424567  0.0224855
[4,] -1.7401295  0.2081581
[5,] -0.8027377 -0.5569849
[6,] -1.0908738 -0.2218717
```

In statistics and the `princomp()` function, the *scores* refer to the *component scores* (or *factor scores*). These component scores are the new coordinates for all the data points under the new basis (orthogonal coordinate system) defined by the eigenvectors. Also, the *loadings* are the vectors of weights by which each standardized original variable is scaled in order to calculate the new component scores.

The eigenvalues represent the amount of variance related to (or in the direction of) the component. In the previous example, the first component has variance 1.8 and the second 0.2, so that the first component represents or captures $1.8/2 = 0.9$ or 90% of the variance. On the basis of the eigenvalues, the number of interesting directions in the data can be evaluated by two rules of thumb. The first is that each eigenvalue should represent more variance than that of any of the observed variables. The second is the so-called elbow rule which states that when the first few eigenvalues are large and the remaining considerably smaller, then the first few are the most interesting.

Principal components analysis (PCA) is a descriptive method to analyze dependencies (correlations) between variables. If there are a few equally large eigenvalues, then there are equally a few directions in the data which summarize the most important variation among the gene expressions. In that case, it may be useful to explore simultaneously a two or three dimensional visualization of the genes and the patients in the coordinate system defined by the first two or three principal components (or eigenvectors). Furthermore, it can be rewarding to study the eigenvalues (weights of the eigenvectors) since these may reveal a structure in the data corresponding to variance along different orthogonal directions in the data. Lastly, the principal components contain less measurement error than the individual variables. For this reason, cluster analysis on the component scores of the principal components is often useful.

Example 2: Application to the Golub (1999) data. The first five eigenvalues from the correlation matrix of `golub` can be printed by the following:

```
> eigen(cor(golub))$values[1:5]
[1] 25.4382629  2.0757158  1.2484411  1.0713373  0.7365232
```

Because the eigenvalues are arranged in decreasing order, the sixth to the 38th are smaller than one. The first eigenvalue is by far the largest, indicating that the patients are dependent to a large extent. By applying the previous bootstrap methods to estimate 95% confidence intervals for the eigenvalues, we obtain the following intervals:

```
> data <- golub; p <- ncol(data); n <- nrow(data) ; nboot<-1000
> eigenvalues <- array(dim=c(nboot,p))
> for (i in 1:nboot){dat.star <- data[sample(1:n,replace=TRUE),]
+   eigenvalues[i,] <- eigen(cor(dat.star))$values}
> for (j in 1:p) print(quantile(eigenvalues[,j],c(0.025,0.975)))
    2.5%    97.5%
for (j in 1:5) cat(j,as.numeric(quantile(eigenvalues[,j],
 + c(0.025,0.975))),"\n" )
1 24.83581 26.00646
2 1.920871 2.258030
3 1.145990 1.386252
4 0.9917813 1.154291
5 0.6853702 0.7995948
```

Above, the null hypothesis of the eigenvalue being equal to one is accepted for the fourth component and rejected for the first three and the fifth. Thus, the fourth eigenvalue represents less variance than an individual variable.

The percentages of variance explained by the first two components can be computed with `sum(eigen(cor(golub))$values[1:2])/38*100`, which yields the amount 72.4052%:

```
> sum(eigen(cor(golub))$values[1:2])/38*100
[1] 72.40521
```

Thus, the first two components represent more than 72% of the variance in the data. Hence, the data can allow for a reduction in dimensions from thirty eight to two while still capturing over 72% of the total variance.

It can be checked that all the correlations between the patients are positive. This implies that large expression values on gene i co-vary positively with large deviations of gene j. The positivity of the correlations also implies that the weights of the first eigenvector have the same sign, so that these can be taken to be positive for all patients (Horn & Johnson, 1985). Unfortunately, this is not automatic in R. Therefore, caution is in order with respect to interpretation of the components. By using `-eigen(cor(golub))$vec[,1:2]` to print the weights to the screen, we observe that those correlations that correspond to the first component are indeed positive:

```
> -eigen(cor(golub))$vec[,1:2]
           [,1]         [,2]
 [1,] 0.1715179  0.104190383
 [2,] 0.1690830 -0.036887326
 [3,] 0.1650130  0.069108652
 [4,] 0.1726783  0.100701410
 [5,] 0.1659430  0.170952365
 [6,] 0.1668802  0.028349089
 [7,] 0.1686381  0.032390622
 [8,] 0.1602444  0.000506016
 [9,] 0.1648767  0.093593764
[10,] 0.1687937  0.023532832
[11,] 0.1653990  0.075375797
[12,] 0.1694389 -0.089380634
[13,] 0.1629073  0.233399833
[14,] 0.1661270  0.077938573
[15,] 0.1647690  0.237950957
[16,] 0.1720834  0.184071927
[17,] 0.1559293  0.078196673
[18,] 0.1600157  0.041608387
[19,] 0.1677201  0.114629349
[20,] 0.1491869  0.247148370
[21,] 0.1272725  0.201580255
[22,] 0.1620961 -0.014147556
[23,] 0.1643598  0.037858970
[24,] 0.1652554  0.210585669
[25,] 0.1659261 -0.044464961
[26,] 0.1690493  0.122286794
[27,] 0.1539690  0.021439143
[28,] 0.1689051 -0.189278784
[29,] 0.1541334 -0.174593292
[30,] 0.1516988 -0.243775745
[31,] 0.1691437 -0.165316026
[32,] 0.1682308 -0.150156325
[33,] 0.1452421 -0.344034852
[34,] 0.1675337 -0.157687854
[35,] 0.1638383 -0.130649438
```

```
[36,] 0.1508646 -0.277921314
[37,] 0.1476138 -0.344828867
[38,] 0.1520466 -0.222765750
```

We see that all weights of the first eigenvector are positive and have very similar size (all are between 0.13 and 0.17). Thus, the first principal component is almost equal to the sum of the variables (the correlation equals 0.9999). However, the weights of the second principal component have a different, very interesting pattern. Namely, almost all of the first 27 weights are positive while the last 11 weights are negative. Thus, the second principal component contrasts the ALL patients with the AML patients. Therefore, by contrasting ALL patients with AML patients, the next largest amount of variance is explained in the data. Hence, the AML-ALL distinction is discovered by the second principal component, which is in line with the findings of Golub et al. (1999).

Obviously the genes with the largest expression values from the first principal component can be printed. However, we will concentrate on the second principal component because it appears to be more directly related to the research presented in Golub et. al. (1999). The first ten and the last ten gene names with respect to the values on the second principal component can be printed as follows:

```
> pca <- princomp(golub, center = TRUE, cor=TRUE, scores=TRUE)
> o <- order(pca$scores[,2])
> golub.gnames[o[1:10],2]
> golub.gnames[o[3041:3051],2]
```

Many of these genes are related to leukemia as shown in Golub, et al., 1999.

Example 3: Biplot. A useful manner to plot both genes (cases) and patients (variables) is the biplot, which is based on a two-dimensional approximation (reduction) of the data very similar to principal components analysis (PCA). Below, we show how the two can be combined:

```
> biplot(princomp(data,cor=TRUE),pc.biplot=TRUE,cex=0.5,expand=0.8)
```

The resulting plot is given by Figure 7.13. The left and bottom axis refer to the gene expression (component) scores in black and the top and right to the patient scores in red, which are scaled to unit length by the specification `cor()`. It can be seen that the patients are clearly divided into two groups corresponding to the ALL (1-27) and AML (28-38) leukemia subtypes.

Example 4: Genes critical for S-phase. Golub et al. (1999) mention that among genes which are useful for tumor class prediction, there are genes that encode for proteins critical for S-phase cell cycle progression - such as Cyclin D3, Op18, and MCM3. We select genes which contain "CD", "Op", or "MCM" in their names and collect the corresponding row numbers:

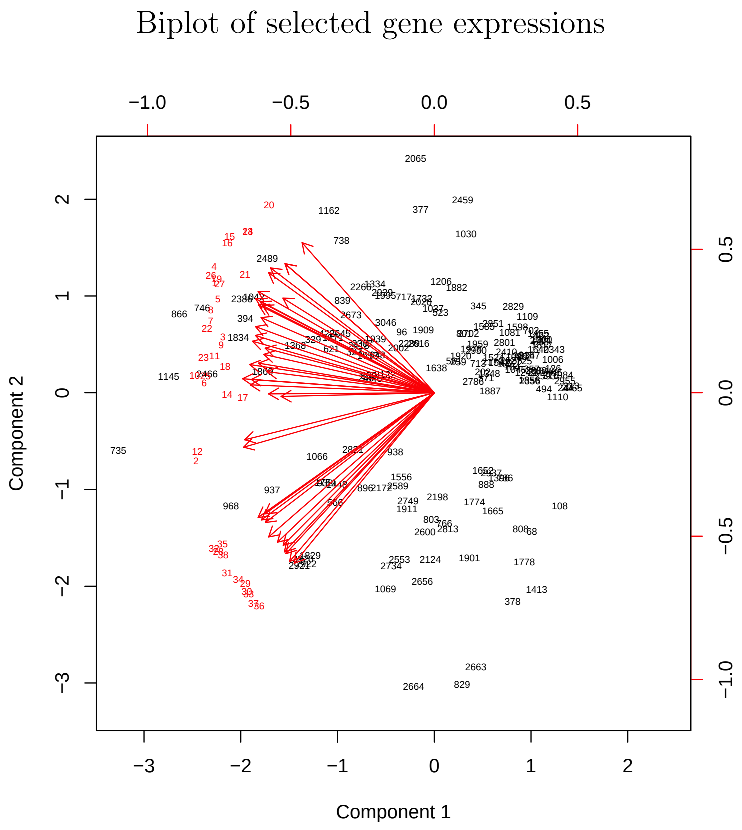

Figure 7.13: Biplot of selected gene expressions from the golub data. The patients are in red and the genes are in black. Notice that the AML patients (28-38) cluster together in the lower-left corner.

```
> data(golub, package = "multtest")
> golubFactor <- factor(golub.cl,levels=0:1, labels= c("ALL","AML"))
> o1 <- grep("CD",golub.gnames[,2])  # antigens
> o2 <- grep("Op",golub.gnames[,2])
> o3 <- grep("MCM",golub.gnames[,2])
> o <- c(o1,o2,o3)
> length(o)
[1] 110
```

This yields 110 genes. In order to select those that have an experimental effect, we use a two-sample t-test:

```
pt <- apply(golub, 1, function(x) t.test(x ~  golubFactor)$p.value)
oo <- o[pt[o]<0.01]
```

This yields 34 genes, of which the row numbers are selected in the vector `oo`. In order to identify genes in directions of large variation we use the scores on the first two principal components:

```
> Z <- as.matrix(scale(golub, center = TRUE, scale = TRUE))
> K <- eigen(cor(Z))
> P <- Z %*% -K$vec[,1:2]
> leu <- data.frame(P[oo,], row.names= oo)
> plot(leu,xlim=c(-10,15), ylim=c(-10,10), pch=19, cex=1.2, xlab="Principal Component 1",
    ↪ ylab="Principal Component 2", col="darkgreen")
> text(x = leu$X1, y=leu$X2, labels=rownames(leu), pos = 1, col="blue")
> fac <- as.integer(oo %in% o1) + 2 * as.integer(oo %in% o2) + 3 * as.integer(oo %in% o3)
> text(x = leu$X1, y=leu$X2, labels=fac, pos = 3, col="red")
>
```

The scores on the first two principal components of the selected genes are stored in the data frame `leu`. From the plotted component scores in Figure 7.14, it seems that there are several sub-clusters of genes. The genes that belong to these clusters can be identified by hiearchical cluster analysis:

```
> cl <- hclust(dist(leu,method="euclidian"),method="single")
> plot(cl,
+      lwd=3,
+      col="blue",
+      col.axis = "brown",
+      ylab="Distance",
+      xlab="Clustering of the expression of genes",
+      hang=-1,
+      main=NA,
+      sub=NA,
+      axes=FALSE)
> axis(side = 2, at = seq(0, 5, 1), col = "brown",labels = TRUE, lwd = 4)
```

From the resultant tree (dendrogram) in Figure 7.15 various clusters of genes are apparent that also appear in Figure 7.14.[3] The ordered genes can be obtained from the object `cl` as follows:

```
> a <- as.integer(rownames(leu)[cl$order])
> for (i in 1:length(a)) { cat(a[i],golub.gnames[a[i],2],"\n") }
1910 FCGR2B Fc fragment of IgG, low affinity IIb, receptor for (CD32)
2874 GB DEF = Fas (Apo-1, CD95)
```

[3]Unfortunately, some row numbers of genes are less readable because the points are very close.

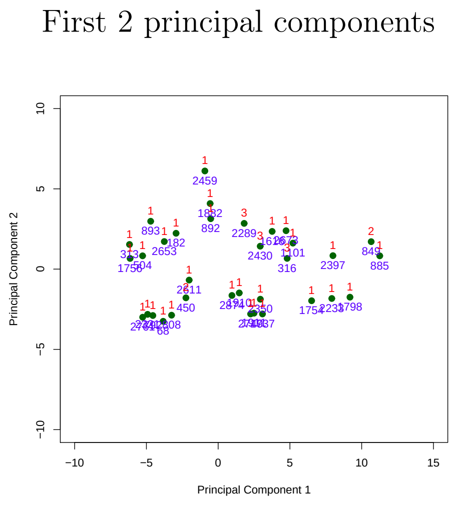

Figure 7.14: Scatter plot of selected genes with row labels on the first two principal components. 1=CD gene, 2=Op gene, 3=MCM gene.

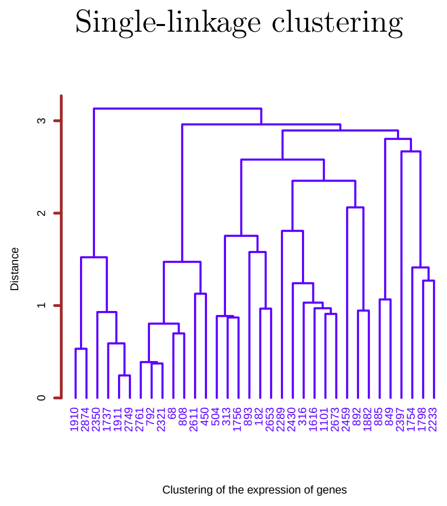

Figure 7.15: Single-linkage cluster diagram of selected gene expression values.

The cluster with rows 504, 313, 1756, and 893 consists of antigens (and contain "CD" in their gene names). The two probe sets for the gene "MCM3 Minichromosome maintenance deficient (S. cerevisiae) 3" with row numbers 2289 and 2430 also appear adjacent to each other. This illustrates that genes with similar functions may indeed be close with respect to their gene expression values.

7.5 Overview and concluding remarks

Unsupervised cluster analysis is used to explore for the existence of groupings in the data based upon a well-defined similarity metric. In this chapter, we focused on clusters of similarly expressioned genes within different leukemia subtypes. When groups are present, a k-means cluster analysis can be applied in combination with the bootstrap to estimate confidence intervals for the cluster means.

The correlation coefficient measures the degree of dependency between pairs of gene expression values. It can also be used to find gene expressions which are highly dependent with a phenotypic variable.

Principal components analysis (PCA) is very useful for finding directions in the data where the gene expression values vary maximally, see Jolliffe (2002) for a complete treatment of the principal component analysis. When these directions can be represented well by the first two components, a biplot helps to simultaneously visualize genes and patients. Thus, principal components analysis can be useful in identifying clusters of genes in a lower dimensional space.

7.6 Exercises

1. **Cluster analysis on Zyxin gene expression.**

   (a) Produce a scatter plot of the Zyxin gene expression values using different symbols for the two groups.

   (b) Use single-linkage cluster analysis to see whether Zyxin gene expression falls into two different clusters.

   (c) Use k-means cluster analysis on Zyxin gene expression with $k = 2$ without any initial values. Then re-do the k-means clustering using the mean Zyxin expression of the ALL and AML patients as the initial parameters (Note that this is cheating!). Do the two clusters reflect the diagnosis of the patient groups either with or without the initial parameters?

   (d) Did the clustering improve with the initial parameters? Did you get a perfect classifier? Why or why not?

   (e) Perform a bootstrap on the cluster means. Do the confidence intervals for the cluster means overlap?

2. **Gene expression similar to CCND3.** Recall that we did various analysis on the expression data of the CCND3 (Cyclin D3) gene of the Golub (1999) data.

   (a) Use `genefinder()` to find the ten genes with expression patterns most similar to CCND3 (Cyclin D3). Give their probe IDs as well as their biological names.

   (b) Produce 2 side-by-side plots of 4 separate side-by-side, vertical boxplots for CCND3 and the top 3 genes expressed most similarly to CCND3. The first set of 4 side-by-side boxplots should use the ALL expression values and the second set of 4 boxplots should use the AML expression values. Use `par(mfrow = c(1,2))` to specify 2 horizontal side-by-side plots. Plot the boxplots and then use `par(mfrow = c(1,1))` to return to single plots. How similar are the gene expression patterns for both the ALL and AML patients?

(c) Use `grep()` to find all the other genes that contain “Cyclin” in their names and compare their smallest distances to the distances found using `genefinder()` above. How do the distances differ?

3. **Outlier in MCM3 gene expression.** In the example for MCM3, a plot of the two sets of MCM3 expression values shows that there is an outlier.

   (a) Construct a 2D Scatter Plot of the two sets of MCM3 expression data and invent a manner to find the row number of the outlier.

   (b) Test for linear correlation with all the data, then remove the outlier, and then test the correlation coefficient again. How do the results differ?

   (c) Perform the bootstrap to construct confidence intervals of the correlation coefficients with and without the oultier. How do they differ?

4. **Cluster analysis on a portion of the Golub data.**

   (a) Use the `grep()` function with the string “oncogene” to select the oncogenes from the Golub data and plot the tree from a single-linkage cluster analysis of the patients using just the expression of the oncogenes.

   (b) Do you observe any meaningful AML or ALL clusters?

   (c) Use `grep("antigen’’)` to select the antigens and construct another tree. Do you see any meaningful clusters now?

   (d) Use `grep("receptor’’)` to select the receptor genes and construct another tree. Do you see any meaningful clusters now?

5. **Principal Components Analysis on part of the `ALL` data.**

(a) Construct an expression set with the patients with B-cell in stage B1, B2, and B3. Compute the corresponding ANOVA p-values for all the gene expressions. Construct the expression set with the p-values smaller than 0.001. Report the dimensionality of the data matrix with gene expressions that have significantly different means across the B1, B2, and B3 leukemia subtypes.

(b) Are all the correlations between the patients positive?

(c) Compute the eigenvalues of the correlation matrix. Report the largest five. Are the first three larger than one? What can you conclude about the first principal component?

(d) Program a bootstrap of the largest five eigenvalues. Report the bootstrap 95% confidence intervals and draw relevant conclusions.

(e) Use `prcomp()` and `autoplot()` to perform PCA analysis and then plot the patients in 2D space while using the first two principal components as the axes. Do you see any clustering of the B1, B2, or B3 leukemia subtypes?

6. **Some correlation matrices.**

$$\begin{bmatrix} 1 & -0.8 \\ -0.8 & 1 \end{bmatrix}, \quad \begin{bmatrix} 1 & 0.8 & 0.8 \\ 0.8 & 1 & 0.8 \\ 0.8 & 0.8 & 1 \end{bmatrix}, \quad \begin{bmatrix} 1 & -0.5 & -0.5 \\ -0.5 & 1 & -0.5 \\ -0.5 & -0.5 & 1 \end{bmatrix}$$

(a) Verify that the eigenvalues of the matrices are `1.8, 0.2`; `2.6, 0.2, 0.2`; and `1.500000e+00, 1.500000e+00, -7.644529e-17`.

(b) How much variance is captured by the first principal component corresponding to the second matrix?

(c) Verify that the first eigenvector of the second correlation matrix contains values with identical signs.