Chapter 6
Dimensionality Reduction

Now that we have most of the fundamentals covered in the previous chapters, we can delve into the first data analytic method, *dimension reduction*, which reduces the number of features when modeling a very large number of variables. Dimension reduction can help us extract a set of "uncorrelated" principal variables and reduce the complexity of the data. We are not simply picking some of the original variables. Rather, we are constructing new "uncorrelated" variables as functions of the old features.

Dimensionality reduction techniques enable exploratory data analyses by reducing the complexity of the dataset, still approximately preserving important properties, such as retaining the distances between cases or subjects. If we are able to reduce the complexity down to a few dimensions, we can then plot the data and untangle its intrinsic characteristics.

We will (1) start with a synthetic example demonstrating the reduction of a 2D data into 1D; (2) explain the notion of rotation matrices; (3) show examples of principal component analysis (PCA), singular value decomposition (SVD), independent component analysis (ICA) and factor analysis (FA); and (4) present a Parkinson's disease case-study at the end. The supplementary DSPA electronic materials for this chapter also include the theory and practice of t-Distributed Stochastic Neighbor Embedding (t-SNE), which represents high-dimensional data via projections into non-linear low-dimensional manifolds.

6.1 Example: Reducing 2D to 1D

We consider an example looking at twin heights. Suppose we simulate 1000 2D points that representing `normalized` individual heights, i.e., number of standard deviations from the mean height. Each 2D point represents a pair of twins. We will simulate this scenario using Bivariate Normal Distribution (Table 6.1 and Fig. 6.1).

I. D. Dinov, *Data Science and Predictive Analytics*,
https://doi.org/10.1007/978-3-319-72347-1_6

Table 6.1 Schematic data structure representation and indexing of twin heights

$\boldsymbol{Twin1_{Height}}$	$\boldsymbol{Twin2_{Height}}$
$\boldsymbol{y[1,1]}$	$\boldsymbol{y[1,2]}$
$\boldsymbol{y[2,1]}$	$\boldsymbol{y[2,2]}$
$\boldsymbol{y[3,1]}$	$\boldsymbol{y[3,2]}$
…	…
$\boldsymbol{y[500,1]}$	$\boldsymbol{y[500,2]}$

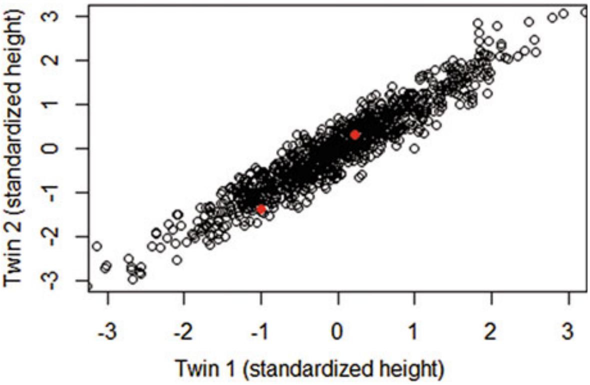


Fig. 6.1 Scatterplot of paired twin heights. The red points show the heights of the first two pairs of twins

```
library(MASS)

set.seed(1234)
n <- 1000
y=t(mvrnorm(n, c(0, 0), matrix(c(1, 0.95, 0.95, 1), 2, 2)))
```

$$y_{2\times 500}^{T} = \begin{bmatrix} y[1,] = Twin1_{Height} \\ \\ y[2,] = Twin2_{Height} \end{bmatrix} = BVN\left(\mu = \begin{bmatrix} Twin1_{Height} \\ \\ Twin2_{Height} \end{bmatrix}, \Sigma = \begin{bmatrix} 1 & 0.95 \\ \\ 0.95 & 1 \end{bmatrix}\right).$$

```
plot(y[1, ], y[2, ], xlab="Twin 1 (standardized height)",
     ylab="Twin 2 (standardized height)", xlim=c(-3, 3), ylim=c(-3, 3))
points(y[1, 1:2], y[2, 1:2], col=2, pch=16) # plot the first 2 points
```

These data may represent a fraction of the information included in a high-throughput neuroimaging genetics study of twins, like the pediatric study example shown here (http://wiki.socr.umich.edu/index.php/SOCR_Data_Oct2009_ID_NI).

Tracking the distances between any two samples can be accomplished using the `dist` function. For example, here is the distance between the two RED points in the Fig. 6.1:

```
d=dist(t(y))
as.matrix(d)[1, 2]

## [1] 2.100187
```

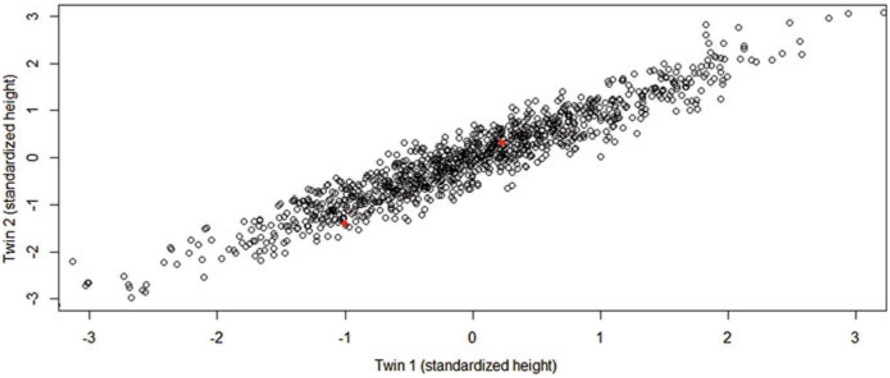


Fig. 6.2 Scatterplots of the raw twin heights

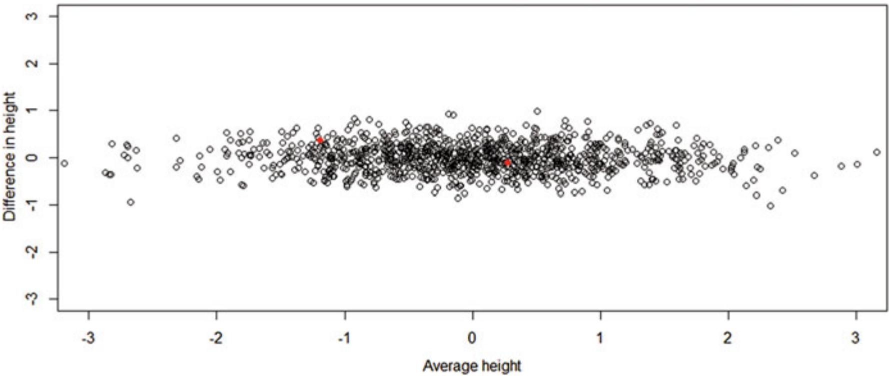


Fig. 6.3 Scatterplots of the transformed twin heights, compare to Fig. 6.2

To transform the 2D data to a simpler 1D plot, we can reduce the data to a 1D matrix (vector) approximately preserving the distances between the 2D points.

The 2D plot shows the Euclidean distance between the pair of red points, Fig. 6.2. The length of this line is the distance between the two points. In 2D, these lines tend to go along the direction of the diagonal. If we `rotate` the plot so that the diagonal is in the x-axis (Fig. 6.3):

```
z1 = (y[1, ]+y[2, ])/2 # the sum (or rather average)
z2 = (y[1, ]-y[2, ])   # the difference

z = rbind(z1, z2) #matrix now same dimensions as y

thelim <- c(-3, 3)
# par(mar=c(1, 2))
# par(mfrow=c(2,1))
plot(y[1, ], y[2, ], xlab="Twin 1 (standardized height)",
     ylab="Twin 2 (standardized height)",
     xlim=thelim, ylim=thelim)
points(y[1, 1:2], y[2, 1:2], col=2, pch=16)
```

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

plot(z[1, ], z[2, ], xlim=thelim, ylim=thelim, xlab="Average height", ylab="
Difference in height")
points(z[1, 1:2], z[2, 1:2], col=2, pch=16)
```

Of course, matrix linear algebra notation can be used to represent this affine transformation of the data. Here we can see that to get z we multiplied y by the matrix:

$$A = \begin{pmatrix} 1/2 & 1/2 \\ 1 & -1 \end{pmatrix} \Longrightarrow z = A \times y.$$

We can invert this transform by multiplying the result by the inverse matrix A^{-1} as follows:

$$A^{-1} = \begin{pmatrix} 1 & 1/2 \\ 1 & -1/2 \end{pmatrix} \Longrightarrow y = A^{-1} \times z.$$

You can try this in R:

```
A <- matrix(c(1/2, 1, 1/2, -1), nrow=2, ncol=2); A   # define a matrix

##      [,1] [,2]
## [1,]  0.5  0.5
## [2,]  1.0 -1.0

A_inv <- solve(A); A_inv  # inverse

##      [,1] [,2]
## [1,]    1  0.5
## [2,]    1 -0.5

A %*% A_inv # Verify result

##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

Note that this matrix transformation *did not* preserve distances, i.e., the matrix A is not a simple rotation in 2D:

```
d=dist(t(y)); as.matrix(d)[1, 2]   # distance between first two points of Y

## [1] 2.100187

d1=dist(t(z)); as.matrix(d1)[1, 2] # distance between first two points of
Z=A*Y

## [1] 1.541323
```

6.2 Matrix Rotations

One important question to ask is how we can identify transformations that preserve distances. In mathematics, transformations between metric spaces that are distance-preserving are called *isometries* (or congruences or congruent transformations).

First, let's test the MA transformation we used above (Fig. 6.3):

$$\left| \begin{array}{l} M = Y_1 - Y_2 \\ A = \dfrac{Y_1 + Y_2}{2}. \end{array} \right.$$

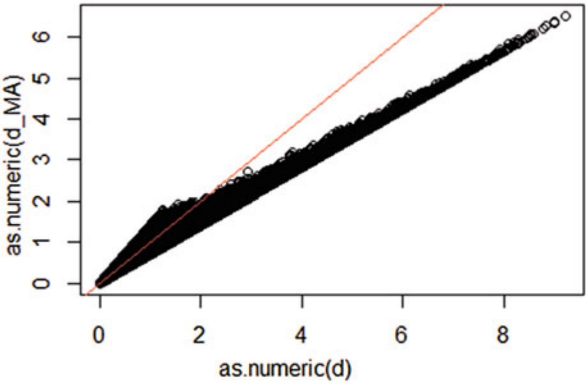
```
MA <- matrix(c(1/2, 1, 1/2, -1), 2, 2)

MA_z <- MA%*%y
d <- dist(t(y))
d_MA <- dist(t(MA_z))

plot(as.numeric(d), as.numeric(d_MA))
abline(0, 1, col=2)
```

Observe that this MA transformation is not an isometry – the distances are not preserved. Here is one example with $v1 = \begin{bmatrix} v1_x = 0 \\ v1_y = 1 \end{bmatrix}$, $v2 = \begin{bmatrix} v2_x = 1 \\ v2_y = 0 \end{bmatrix}$, which are distance $\sqrt{2}$ apart in their native space, but separated further by the transformation MA, $d(MA(v1), MA(v2)) = 2$.

Fig. 6.4 The above MA transformation is not an isometry. This scatterplot shows that the relation between the transformed (y-axis) and the native-space (x-axis) twin-pairs distances are not preserved



```
MA; t(MA); solve(MA); t(MA) - solve(MA)

##      [,1] [,2]
## [1,]  0.5  0.5
## [2,]  1.0 -1.0

##      [,1] [,2]
## [1,]  0.5    1
## [2,]  0.5   -1

##      [,1] [,2]
## [1,]    1  0.5
## [2,]    1 -0.5

##      [,1] [,2]
## [1,] -0.5  0.5
## [2,] -0.5 -0.5

v1 <- c(0,1); v2 <- c(1,0); rbind(v1,v2)

##    [,1] [,2]
## v1    0    1
## v2    1    0

euc.dist <- function(x1, x2) sqrt(sum((x1 - x2) ^ 2))
euc.dist(v1,v2)

## [1] 1.414214
v1_t <- MA %*% v1; v2_t <- MA %*% v2
euc.dist(v1_t,v2_t)

## [1] 2
```

More generally, if

$$\begin{pmatrix} Y_1 \\ Y_2 \end{pmatrix} \sim N\left(\begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \begin{pmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{pmatrix}\right).$$

Then,

$$Z = AY + \eta \sim BVN\left(\eta + A\mu, A\Sigma A^T\right).$$

Where BVN denotes bivariate normal distribution (see http://socr.umich.edu/HTML5/BivariateNormal/),

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, Y = (Y_1, Y_2)^T, \mu = (\mu_1, \mu_2), \Sigma = \begin{pmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{pmatrix}.$$

You can verify this by using the change of variable theorem. Thus, affine transformations preserve bivariate normality. However, in general, there is no means to guarantee isometry.

The question now is: *Under what additional conditions for a transformation matrix A, can we guarantee an isometry?*

Notice that,

$$d^2\left(P_i, P_j\right) = \sum_{k=1}^{T}\left(P_{jk} - P_{ik}\right)^2 = \|P\|^2 = P^T P,$$

where $P = (P_{j,1} - P_{i,1}, \ldots, P_{j,T} - P_{i,T})^T$, P_i and P_j is any two points in T dimensions.

Thus, the only requirement we need is $(AY)^T(AY) = Y^T Y$, *i. e.* , $A^T A = I$, which implies that A is an orthogonal (rotational) matrix.

Let's use a two dimension orthogonal matrix to illustrate this concept. Set $A = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$. It's easy to verify that A is an orthogonal (2D rotation) matrix.

The simplest way to test the isometry is to perform the linear transformation directly (Fig. 6.5).

```
A <- 1/sqrt(2)*matrix(c(1, 1, 1, -1), 2, 2)
z <- A%*%y
d <- dist(t(y))
d2 <- dist(t(z))

plot(as.numeric(d), as.numeric(d2))
abline(0, 1, col=2)
```

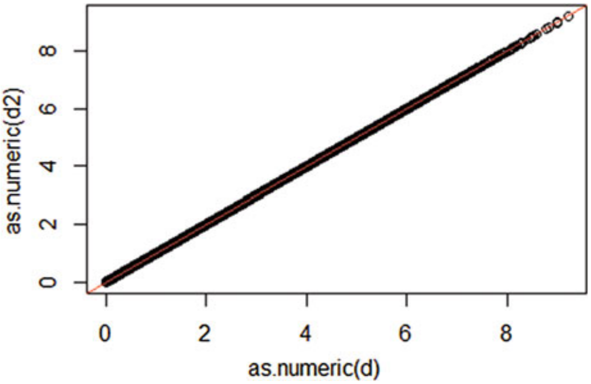
We can observe that the distances computed using the original data are preserved after the transformation. This transformation is called a rotation (isometry) of y. Note the difference compared to the earlier plot, Fig. 6.4.

An alternative method is to simulate from the joint distribution of $Z = (Z_1, Z_2)^T$. As we have mentioned above:

$$Z = AY + \eta \sim BVN\left(\eta + A\mu, A\Sigma A^T\right),$$

where $\eta = (0, 0)^T$, $\Sigma = \begin{pmatrix} 1 & 0.95 \\ 0.95 & 1 \end{pmatrix}$, $A = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$.

Fig. 6.5 The matrix A transformation above is distance preserving (i.e., an isometry), as illustrated by the perfect linear relation between the native-space and the transformed pairs of twin height distances



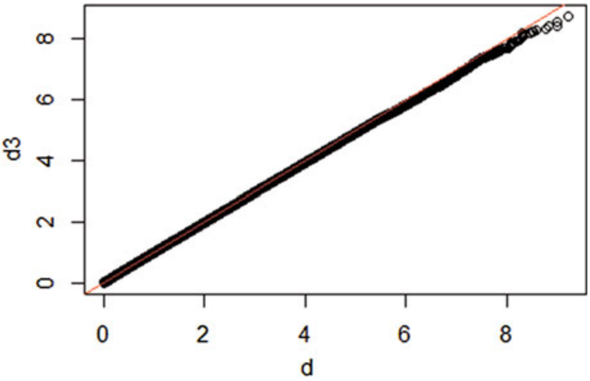


Fig. 6.6 QQ-plot of the distanced between twin heights (d) and distances between the simulated bivariate Normal distribution data (d3)

We can compute $A\Sigma A^T$ by hand or by using matrix multiplication in R:

```
sig <- matrix(c(1,0.95,0.95,1),nrow=2)
A%*%sig%*%t(A)

##      [,1] [,2]
## [1,] 1.95 0.00
## [2,] 0.00 0.05
```

$A\Sigma A^T$ represents the transformed variance-covariance matrix, $cov(z_1, z_2) = 0$. We can simulate z_1, z_2 independently from $z_1 \sim N(0{,}1.95)$ and $z_2 \sim N(0{,}0.05)$ (Note: independence and uncorrelation are equivalent for bivariate normal distribution) (Fig. 6.6).

```
set.seed(2017)
zz1 = rnorm(1000,0,sd = sqrt(1.95))
zz2 = rnorm(1000,0,sd = sqrt(0.05))
zz = rbind(zz1,zz2)
d3 = dist(t(zz))
qqplot(d,d3)
abline(a = 0,b=1,col=2)
```

We can observe that the distances computed using the original data and the simulated data are the same (Figs. 6.7 and 6.8).

```
thelim <- c(-3, 3)
#par(mfrow=c(2,1))
plot(y[1, ], y[2, ], xlab="Twin 1 (standardized height)",
     ylab="Twin 2 (standardized height)",
     xlim=thelim, ylim=thelim)
points(y[1, 1:2], y[2, 1:2], col=2, pch=16)
```

```
plot(z[1, ], z[2, ], xlim=thelim, ylim=thelim, xlab="Average height",
ylab="Difference in height")
points(z[1, 1:2], z[2, 1:2], col=2, pch=16)
```

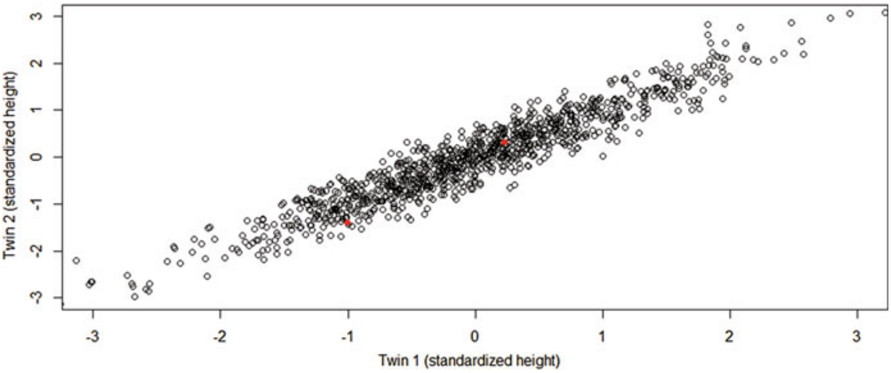


Fig. 6.7 Twin height scatterplot before rotation

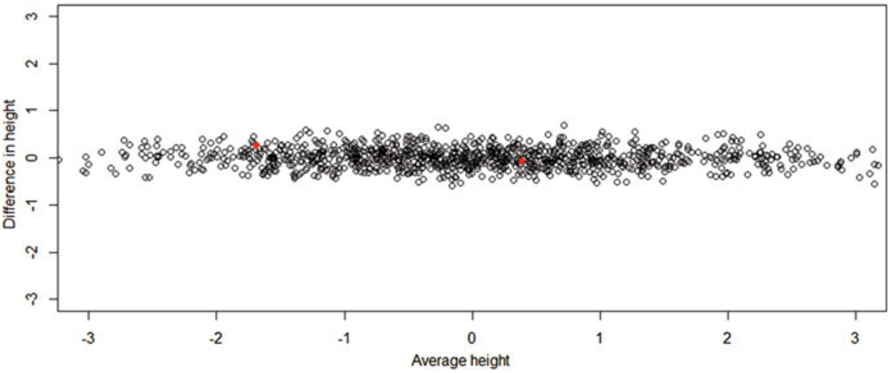


Fig. 6.8 Twin height scatterplot after the rotation

We applied this transformation and observed that the distances between points were unchanged after the rotation. This rotation achieves the goals of:

- Preserving the distances between points, and
- Reducing the dimensionality of the data (see plot reducing 2D to 1D).

Removing the second dimension and recomputing the distances, we get (Fig. 6.9):

```
d4 = dist(z[1, ]) ##distance computed using just the first dimension
plot(as.numeric(d), as.numeric(d4))
abline(0, 1)
```

The 1D distances provide a very good approximation to the actual 2D distances. This first dimension of the transformed data is called the first `principal component`. In general, this idea motivates the use of principal component analysis (PCA) and the singular value decomposition (SVD) to achieve dimensionality reduction.

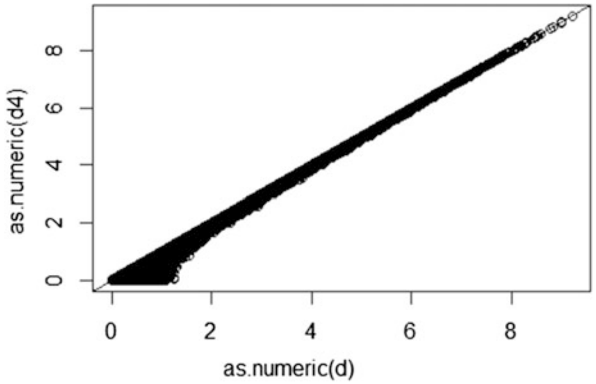


Fig. 6.9 Comparing the twin distances, computed using just one dimension, following the rotation transformation against the actual twin pair height distances. The strong linear relation suggests that measuring distances in the native space is equivalent to measuring distances in the transformed space, where we reduced the dimension of the data from 2D to 1D

6.3 Notation

In the notation above, the rows represent variables and columns represent cases. In general, rows represent cases and columns represent variables. Hence, in our example shown here, Y would be transposed to be a $N \times 2$ matrix. This is the most common way to represent the data: individuals in the rows, features in the columns. In genomics, it is more common to represent subjects/SNPs/genes in the columns. For example, genes are rows and samples are columns. The sample covariance matrix usually denoted with $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ and has cells representing covariance between two units. Yet, for this to be the case, we need the rows of **X** to represent the subjects and the columns to represent the variables, or features. Here, we have to compute, $\mathbf{Y}\mathbf{Y}^{\mathrm{T}}$ instead following the rescaling.

6.4 Summary (PCA vs. ICA vs. FA)

Principle Component Analysis (PCA), Independent Component Analysis (ICA), and Factor Analysis (FA) are similar strategies, seeking to identify a new basis (vectors representing the principal directions) that the data is projected against to maximize certain (specific to each technique) objective functions. These basis functions, or vectors, are just linear combinations of the original features in the data/signal.

The singular value decomposition (SVD), discussed later in this chapter, provides a specific matrix factorization algorithm that can be employed in various techniques to decompose a data matrix $X_{m \times n}$ as $U\Sigma V^T$, where U is an $m \times m$ real or complex unitary matrix ($UU^T = U^TU = I$), Σ is a $m \times n$ rectangular diagonal matrix of *singular values*, representing non-negative values on the diagonal, and V is an $n \times n$ unitary matrix (Table 6.2).

Table 6.2 Summary of some dimensionality reduction methods

Method	Assumptions	Cost function optimization	Applications
PCA	Gaussian signals	Aims to explain the variance in the original signal. Minimizes the covariance of the data and yields high-energy orthogonal vectors in terms of the signal variance. PCA looks for an orthogonal linear transformation that maximizes the variance of the variables	Relies on first and second moments of the measured data, which makes it useful when data features are close to Gaussian
ICA	No Gaussian signal assumptions	Minimizes higher-order statistics (e.g., third and fourth order skewness and kurtosis), effectively minimizing the *mutual information* of the transformed output. ICA seeks a linear transformation where the basis vectors are statistically independent, but neither Gaussian, orthogonal or ranked in order	Applicable for non-Gaussian, very noisy, or mixture processes composed of simultaneous input from multiple sources
FA	Approximately Gaussian data	Objective function relies on second order moments to compute likelihoods. FA *factors* are linear combinations that maximize the shared portion of the variance underlying *latent variables*, which may use a variety of optimization strategies (e.g., maximum likelihood)	PCA-generalization used to test a theoretical model of latent factors causing the observed features

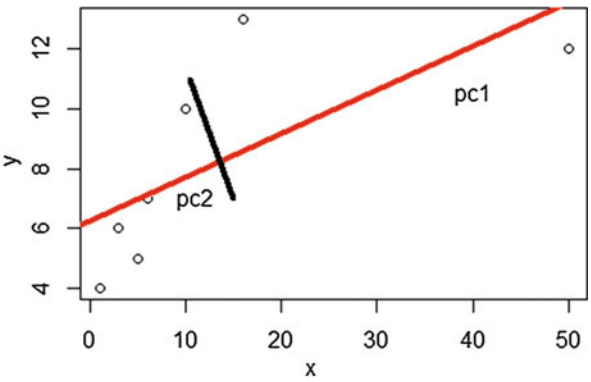
6.5 Principal Component Analysis (PCA)

PCA (principal component analysis) is a mathematical procedure that transforms a number of possibly correlated variables into a smaller number of uncorrelated variables through a process known as orthogonal transformation.

6.5.1 Principal Components

Let's consider the simplest situation where we have n observations $\{p_1, p_2, \ldots, p_n\}$ with two features $p_i = (x_i, y_i)$. When we draw them on a plot, we use the x-axis and y-axis for positioning. However, we can make our own coordinate system by principal components (Fig. 6.10).

Fig. 6.10 Schematic representation of the first two principal components (simulated data)



```
ex<-data.frame(x=c(1, 3, 5, 6, 10, 16, 50), y=c(4, 6, 5, 7, 10, 13, 12))
reg1<-lm(y~x, data=ex)
plot(ex)
abline(reg1, col='red', lwd=4)
text(40, 10.5, "pc1")
segments(10.5, 11, 15, 7, lwd=4)
text(11, 7, "pc2")
```

Illustrated on the graph, the first PC, pc_1 is a minimum distance fit in the feature space. The second PC is a minimum distance fit to a line perpendicular to the first PC. Similarly, the third PC would be a minimum distance fit to all previous PCs. In our case of a 2D space, two PC's is the most we can have. In higher dimensional spaces, we have to figure out how many PCs are needed to make the best performance.

In general, the formula for the first PC is $pc_1 = a_1^T X = \sum_{i=1}^{N} a_{i,1} X_i$ where X_i is a $n \times 1$ vector representing a column of the matrix X (complete design matrix with a total of n observations and N features). The weights $a_1 = \{a_{1,\ 1}, a_{2,\ 1}, \ldots, a_{N,\ 1}\}$ are chosen to maximize the variance of pc_1. According to this rule, the kth PC $pc_k = a_k^T X = \sum_{i=1}^{N} a_{i,k} X_i$, where $a_k = \{a_{1,\ k}, a_{2,\ k}, \ldots, a_{N,\ k}\}$ has to be constrained by more conditions:

1. Variance of pc_k is maximized
2. $Cov(pc_k, pc_l) = 0, \forall\ 1 \leq l < k$
3. $a_k^T a_k = 1$ (the weights vectors are unitary).

Let's figure out how to find a_1. To begin, we need to express the variance of our first principal component using the variance covariance matrix of X:

$$Var(pc_1) = E\left(pc_1^2\right) - (E(pc_1))^2 =$$
$$\sum_{i,j=1}^{N} a_{i,1} a_{j,1} E\left(x_i x_j\right) - \sum_{i,j=1}^{N} a_{i,1} a_{j,1} E(x_i) E\left(x_j\right) =$$
$$\sum_{i,j=1}^{N} a_{i,1} a_{j,1} S_{i,j},$$

where $S_{i,\ j} = E(x_i x_j) - E(x_i)E(x_j)$.

This implies $Var(pc_1) = a_1^T S a_1$, where $S = S_{i,\ j}$ is the covariance matrix of $X = \{X_1, \ldots, X_N\}$. Since a_1 maximized $Var(pc_1)$ and the constrain $a_1^T a_1 = 1$ holds, we can rewrite a_1 as:

$$a_1 = \arg\ \max_{a_1}\left(a_1^T S a_1 - \lambda\left(a_1^T a_1 - 1\right)\right).$$

Where the part after the subtraction should be trivial. Take the derivative of this expression w.r.t. a_1 and set the derivative to zero, which yields $(S - \lambda I_N)a_1 = 0$.

In Chap. 5 we showed that a_1 will correspond to the largest eigenvalue of S, the variance covariance matrix of X. Hence, pc_1 retains the largest amount of variation in the sample. Likewise, a_k is the kth largest eigenvalue of S.

PCA requires data matrix to have zero empirical means for each column. That is, the sample mean of each column has been shifted to zero.

Let's use a subset ($N = 33$) of Parkinson's Progression Markers Initiative (PPMI) database to demonstrate the relationship between S and PC loadings. First, we need to import the dataset into R and delete the patient ID column.

```
library(rvest)

wiki_url <-read_html("http://wiki.socr.umich.edu/index.php/SMHS_PCA_ICA_FA")
html_nodes(wiki_url, "#content")

pd.sub <- html_table(html_nodes(wiki_url, "table")[[1]])
summary(pd.sub)

##    Patient_ID   Top_of_SN_Voxel_Intensity_Ratio
##  Min.   :3001   Min.   :1.058
##  1st Qu.:3012   1st Qu.:1.334
##  Median :3029   Median :1.485
##  Mean   :3204   Mean   :1.532
##  3rd Qu.:3314   3rd Qu.:1.755
##  Max.   :3808   Max.   :2.149
##  Side_of_SN_Voxel_Intensity_Ratio    Part_IA         Part_IB
##  Min.   :0.9306                    Min.   :0.000   Min.   : 0.000
##  1st Qu.:0.9958                    1st Qu.:0.000   1st Qu.: 2.000
##  Median :1.1110                    Median :1.000   Median : 5.000
##  Mean   :1.1065                    Mean   :1.242   Mean   : 4.909
##  3rd Qu.:1.1978                    3rd Qu.:2.000   3rd Qu.: 7.000
##  Max.   :1.3811                    Max.   :6.000   Max.   :13.000
##     Part_II          Part_III
##  Min.   : 0.000   Min.   : 0.00
##  1st Qu.: 0.000   1st Qu.: 2.00
##  Median : 2.000   Median :12.00
##  Mean   : 4.091   Mean   :13.39
##  3rd Qu.: 6.000   3rd Qu.:20.00
##  Max.   :17.000   Max.   :36.00

pd.sub<-pd.sub[, -1]
```

Then, we need to center the `pdsub` by subtracting the average of all column means from each element. Next we change `pd.sub` to a matrix and get its variance covariance matrix, S. Now, we are able to calculate the eigenvalues and eigenvectors of S.

```
mu<-apply(pd.sub, 2, mean)
mean(mu)

## [1] 4.379068

pd.center<-as.matrix(pd.sub)-mean(mu)
S<-cov(pd.center)
eigen(S)

## $values
## [1] 1.315073e+02 1.178340e+01 6.096920e+00 1.424351e+00 6.094592e-02
## [6] 8.035403e-03
##
## $vectors
##              [,1]          [,2]         [,3]        [,4]        [,5]
## [1,] -0.007460885 -0.0182022093  0.016893318  0.02071859  0.97198980
## [2,] -0.005800877  0.0006155246  0.004186177  0.01552971  0.23234862
## [3,]  0.080839361 -0.0600389904 -0.027351225  0.99421646 -0.02352324
## [4,]  0.229718933 -0.2817718053 -0.929463536 -0.06088782  0.01466136
## [5,]  0.282109618 -0.8926329596  0.344508308 -0.06772403 -0.01764367
## [6,]  0.927911126  0.3462292153  0.127908417 -0.05068855  0.01305167
##              [,6]
## [1,] -0.232667561
## [2,]  0.972482080
## [3,] -0.009618592
## [4,]  0.003019008
## [5,]  0.006061772
## [6,]  0.002456374
```

The next step is to calculate the PCs using the `prcomp()` function in R. Note that we will use the uncentered version of the data and use `center=T` option. We stored the model information into `pca1`. Then `pca1$rotation` provides the loadings for each PC.

```
pca1<-prcomp(as.matrix(pd.sub), center = T)
summary(pca1)

## Importance of components:
##                            PC1    PC2     PC3     PC4    PC5     PC6
## Standard deviation     11.4677 3.4327 2.46919 1.19346 0.2469 0.08964
## Proportion of Variance  0.8716 0.0781 0.04041 0.00944 0.0004 0.00005
## Cumulative Proportion   0.8716 0.9497 0.99010 0.99954 1.0000 1.00000

pca1$rotation

##                                            PC1           PC2          PC3
## Top_of_SN_Voxel_Intensity_Ratio   0.007460885 -0.0182022093  0.016893318
## Side_of_SN_Voxel_Intensity_Ratio  0.005800877  0.0006155246  0.004186177
## Part_IA                          -0.080839361 -0.0600389904 -0.027351225
## Part_IB                          -0.229718933 -0.2817718053 -0.929463536
## Part_II                           -0.282109618 -0.8926329596  0.344508308
## Part_III                          -0.927911126  0.3462292153  0.127908417
##                                          PC4          PC5           PC6
## Top_of_SN_Voxel_Intensity_Ratio   0.02071859 -0.97198980 -0.232667561
## Side_of_SN_Voxel_Intensity_Ratio  0.01552971 -0.23234862  0.972482080
## Part_IA                           0.99421646  0.02352324 -0.009618592
## Part_IB                          -0.06088782 -0.01466136  0.003019008
## Part_II                          -0.06772403  0.01764367  0.006061772
## Part_III                         -0.05068855 -0.01305167  0.002456374
```

The loadings are just the eigenvectors times `-1`. This actually represents the same line in 6D dimensional space (we have six columns for the original data). The multiplier `-1` represents the opposite direction in the same line. For further comparisons, we can load the `factoextra` package to get the eigenvalues of PCs.

```
# install.packages("factoextra")
library("factoextra")

eigen<-get_eigenvalue(pca1); eigen

##          eigenvalue variance.percent cumulative.variance.percent
## Dim.1 1.315073e+02      87.159638589                    87.15964
## Dim.2 1.178340e+01       7.809737384                    94.96938
## Dim.3 6.096920e+00       4.040881920                    99.01026
## Dim.4 1.424351e+00       0.944023059                    99.95428
## Dim.5 6.094592e-02       0.040393390                    99.99467
## Dim.6 8.035403e-03       0.005325659                   100.00000
```

The eigenvalues correspond to the amount of the variation explained by each principal component (PC), which represesnts the eigenvalues for the S matrix.

To see detailed information about the variances that each PC explains, we utilize the `plot()` function. We can also visualize the PC loadings (Figs. 6.11, 6.12, and 6.13).

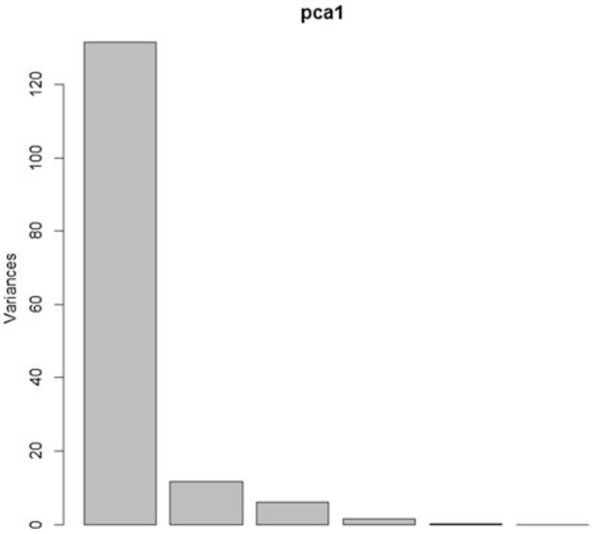
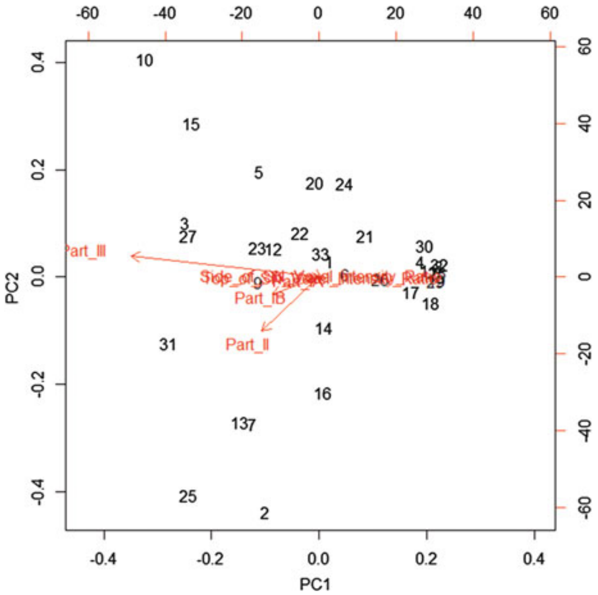


Fig. 6.11 Scree plot of the magnitude of the eigenvalues corresponding to the principal components

Fig. 6.12 A biplot, enhanced scatterplot, showing both points and vectors representing structure of the data in terms of the projections of the features onto the main two principal component directions



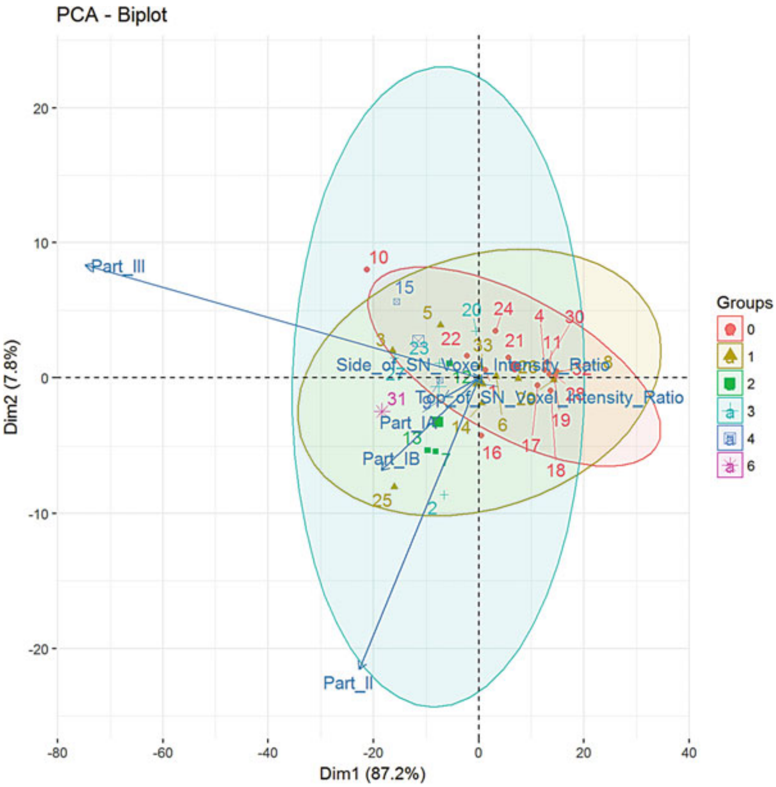


Fig. 6.13 A more elaborate biplot of the same Parkinson’s disease dataset

```
plot(pca1)
```

```
library(graphics)
biplot(pca1, choices = 1:2, scale = 1, pc.biplot = F)
```

```
library("factoextra")
# Data for the supplementary qualitative variables
qualit_vars <- as.factor(pd.sub$Part_IA)
head(qualit_vars)

## [1] 0 3 1 0 1 1
## Levels: 0 1 2 3 4 6

# for plots of individuals
# fviz_pca_ind(pca1, habillage = qualit_vars, addEllipses = TRUE,
ellipse.level = 0.68) +
#  theme_minimal()
# for Biplot of individuals and variables
fviz_pca_biplot(pca1, axes = c(1, 2), geom = c("point", "text"),
  col.ind = "black", col.var = "steelblue", label = "all",
  invisible = "none", repel = T, habillage = qualit_vars,
  palette = NULL, addEllipses = TRUE, title = "PCA - Biplot")
```

The histogram plot has a clear “elbow” point at the second PC. Two PCs explains about 95% of the total variation. Thus, we can use the first 2 PCs to represent the data. In this case, the dimension of the data is substantially reduced.

Here, `biplot` uses PC1 and PC2 as the axes and red vectors to represent the direction of variables after adding loadings as weights. It help us to visualize how the loadings are used to rearrange the structure of the data.

Next, let’s try to obtain a bootstrap test for the confidence interval of the explained variance (Fig. 6.14).

```
set.seed(12)
num_boot = 1000
bootstrap_it = function(i) {
  data_resample = pd.sub[sample(1:nrow(pd.sub),nrow(pd.sub),replace=TRUE), ]
  p_resample = princomp(data_resample,cor = T)
  return(sum(p_resample$sdev[1:3]^2)/sum(p_resample$sdev^2))
  }

pco = data.frame(per=sapply(1:num_boot, bootstrap_it))
quantile(pco$per, probs = c(0.025,0.975))

# specify 95-th %  Confidence Interval

##      2.5%     97.5%
## 0.8134438 0.9035291

corpp = sum(pca1$sdev[1:3]^2)/sum(pca1$sdev^2)
require(ggplot2)
plot = ggplot(pco, aes(x=pco$per)) +
  geom_histogram() + geom_vline(xintercept=corpp, color='yellow')+
  labs(title = "Percent Var Explained by the first 3 PCs") +
  theme(plot.title = element_text(hjust = 0.5))+
  labs(x='perc of var')
show(plot)

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

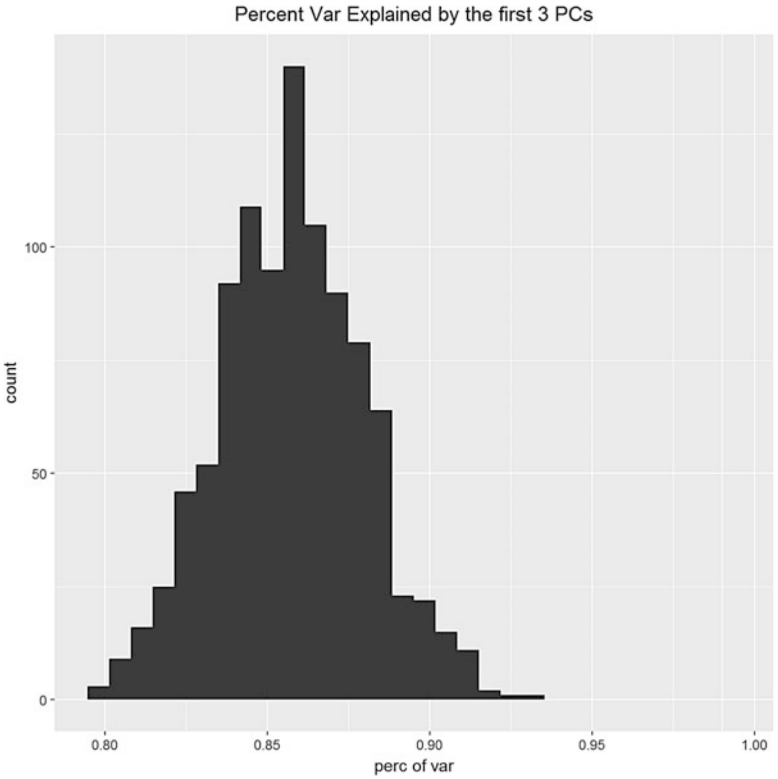


Fig. 6.14 A histogram plot illustrating the proportion of the energy of the original dataset accounted for by the first three principal components

6.6 Independent Component Analysis (ICA)

ICA aims to find basis vectors representing independent components of the original data. For example, this may be achieved by maximizing the norm of the fourth order normalized kurtosis, which iteratively projects the signal on a new basis vector, computes the objective function (e.g., the norm of the kurtosis) of the result, slightly adjusts the basis vector (e.g., by gradient ascent), and recomputes the kurtosis again. The end of this iterative process generates a basis vector corresponding to the highest (residual) kurtosis representing the next independent component.

The process of Independent Component Analysis is to maximize the statistical independence of the estimated components. Assume that each variable X_i is generated by a sum of n independent components.

$$X_i = a_{i,1}s_1 + \cdots + a_{i,n}s_n.$$

Here, X_i is generated by $s_1 : s_n$ and $a_{i,1} : a_{i,n}$ are the corresponding weights. Finally, we rewrite X as

$$X = As,$$

where $X = (X_1, \ldots, X_n)^T$, $A = (a_1, \ldots, a_n)^T$, $a_i = (a_{i,1}, \ldots, a_{i,n})$ and $s = (s_1, \ldots, s_n)^T$. Note that s is obtained by maximizing the independence of the components. This procedure is done by maximizing some *independence* objective function.

ICA assumes all of its components (s_i) are non-Gaussian and independent of each other.

We will now introduce the `fastICA` function in R.

```
fastICA(X, n.comp, alg.typ, fun, rownorm, maxit, tol)
```

- **X**: data matrix
- **n.comp**: number of components,
- **alg.type**: components extracted simultaneously (`alg.typ == "parallel"`) or one at a time (`alg.typ == "deflation"`)
- **fun**: functional form of F to approximate to neg-entropy,
- **rownorm**: whether rows of the data matrix X should be standardized beforehand
- **maxit**: maximum number of iterations
- **tol**: a positive scalar giving the tolerance at which the un-mixing matrix is considered to have converged.

Let's generate a correlated X matrix.

```
S <- matrix(runif(10000), 5000, 2)
S[1:10, ]

##              [,1]       [,2]
##  [1,] 0.19032887 0.92326457
##  [2,] 0.64582044 0.36716717
##  [3,] 0.09673674 0.51115358
##  [4,] 0.24813471 0.03997883
##  [5,] 0.51746238 0.03503276
##  [6,] 0.94568595 0.86846372
##  [7,] 0.29500222 0.76227787
##  [8,] 0.93488888 0.97061365
##  [9,] 0.89622932 0.62092241
## [10,] 0.33758057 0.84543862

A <- matrix(c(1, 1, -1, 3), 2, 2, byrow = TRUE)
X <- S %*% A # In R,  "*" and "%*%" indicate "scalar" and matrix multiplicat
ion, respectively!
cor(X)

##            [,1]       [,2]
## [1,]  1.0000000 -0.4563297
## [2,] -0.4563297  1.0000000
```

The correlation between two variables is −0.4563297. Then we can start to fit the ICA model.

```
# install.packages("fastICA")
library(fastICA)
a <- fastICA(X, 2, alg.typ = "parallel", fun = "logcosh", alpha = 1,
                 method = "C", row.norm = FALSE, maxit = 200,
                 tol = 0.0001)
```

To visualize how *correlated* the pre-processed data is and how *independent* the resulting S is, we can draw the following two plots (Fig. 6.15).

```
par(mfrow = c(1, 2))
plot(a$X, main = "Pre-processed data")
plot(a$S, main = "ICA components")
```

Finally, we can check the correlation of two components in the ICA result, S; it is nearly 0.

```
cor(a$S)

##               [,1]          [,2]
## [1,]  1.000000e+00 -7.677818e-16
## [2,] -7.677818e-16  1.000000e+00
```

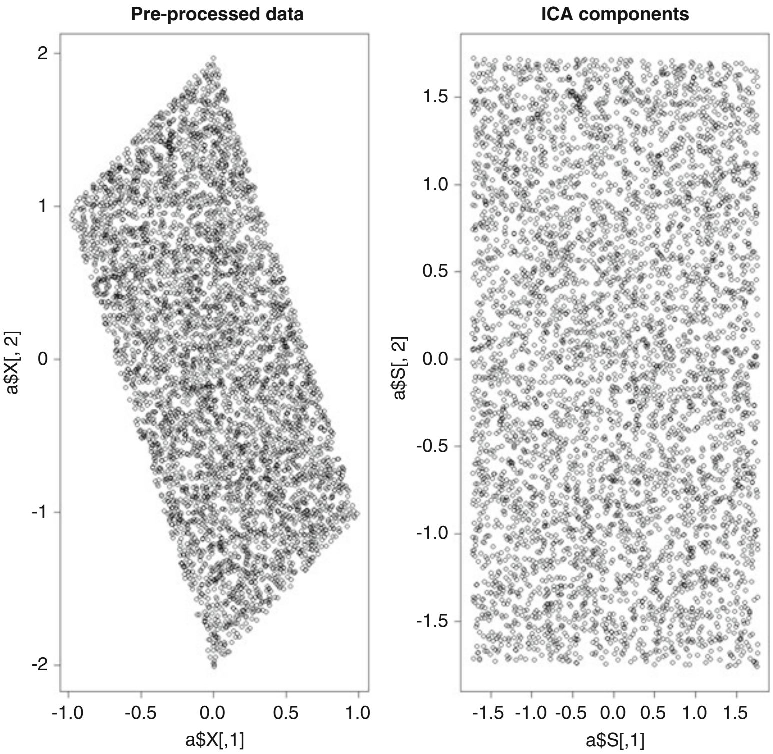


Fig. 6.15 Scatterplots of the raw data (left) illustrating intrinsic relation in the simulated bivariate data and the ICA-transformed data (right) showing random scattering

To do a more interesting example, we can use the `pd.sub` dataset (Parkinson's disease). It has six variables and the correlation is relatively high. After fitting the ICA model, the components are nearly independent.

```
cor(pd.sub)

##                                  Top_of_SN_Voxel_Intensity_Ratio
## Top_of_SN_Voxel_Intensity_Ratio                       1.00000000
## Side_of_SN_Voxel_Intensity_Ratio                      0.54747225
## Part_IA                                              -0.10144191
## Part_IB                                              -0.26966299
## Part_II                                              -0.04358545
## Part_III                                             -0.33921790
##                                  Side_of_SN_Voxel_Intensity_Ratio
## Top_of_SN_Voxel_Intensity_Ratio                         0.5474722
## Side_of_SN_Voxel_Intensity_Ratio                        1.0000000
## Part_IA                                                -0.2157587
## Part_IB                                                -0.4438992
## Part_II                                                -0.3766388
## Part_III                                               -0.5226128
##                                     Part_IA    Part_IB     Part_II
## Top_of_SN_Voxel_Intensity_Ratio  -0.1014419 -0.2696630 -0.04358545
## Side_of_SN_Voxel_Intensity_Ratio -0.2157587 -0.4438992 -0.37663875
## Part_IA                           1.0000000  0.4913169  0.50378157
## Part_IB                           0.4913169  1.0000000  0.57987562
## Part_II                           0.5037816  0.5798756  1.00000000
## Part_III                          0.5845831  0.6735584  0.63901337
##                                    Part_III
## Top_of_SN_Voxel_Intensity_Ratio  -0.3392179
## Side_of_SN_Voxel_Intensity_Ratio -0.5226128
## Part_IA                           0.5845831
## Part_IB                           0.6735584
## Part_II                           0.6390134
## Part_III                          1.0000000

a1<-fastICA(pd.sub, 2, alg.typ = "parallel", fun = "logcosh", alpha = 1,
                 method = "C", row.norm = FALSE, maxit = 200,
                 tol = 0.0001)
par(mfrow = c(1, 2))
cor(a1$X)

##             [,1]       [,2]       [,3]       [,4]        [,5]       [,6]
## [1,]  1.00000000  0.5474722 -0.1014419 -0.2696630 -0.04358545 -0.3392179
## [2,]  0.54747225  1.0000000 -0.2157587 -0.4438992 -0.37663875 -0.5226128
## [3,] -0.10144191 -0.2157587  1.0000000  0.4913169  0.50378157  0.5845831
## [4,] -0.26966299 -0.4438992  0.4913169  1.0000000  0.57987562  0.6735584
## [5,] -0.04358545 -0.3766388  0.5037816  0.5798756  1.00000000  0.6390134
## [6,] -0.33921790 -0.5226128  0.5845831  0.6735584  0.63901337  1.0000000

cor(a1$S)

##              [,1]         [,2]
## [1,] 1.000000e+00 1.088497e-15
## [2,] 1.088497e-15 1.000000e+00
```

Notice that we only have two ICA components instead of six variables, successfully reducing the dimension of the data.

6.7 Factor Analysis (FA)

Similar to ICA and PCA, FA tries to find components in the data. As a generalization of PCA, FA requires that the number of components is smaller than the original number of variables (or columns of the data matrix). FA optimization relies on iterative perturbations with full-dimensional Gaussian noise and maximum-likelihood estimation where every observation in the data represents a sample point in a subspace. Whereas PCA assumes the noise is spherical, Factor Analysis allows the noise to have an arbitrary diagonal covariance matrix and estimates the subspace as well as the noise covariance matrix.

Under FA, the centered data can be expressed in the following form:

$$x_i - \mu_i = l_{i,1}F_1 + \cdots + l_{i,k}F_k + \epsilon_i = LF + \epsilon_i,$$

where $i \in 1, \ldots, p, j \in 1, \ldots, k, k < p$ and ϵ_i are independently distributed error terms with zero mean and finite variance.

Let's do FA in R with function `factanal()`. According to PCA, our `pd.sub` dataset can explain 95% of variance with the first two principal components. This suggest that we might need two factors in FA. We can double check that by the following commands (Fig. 6.16).

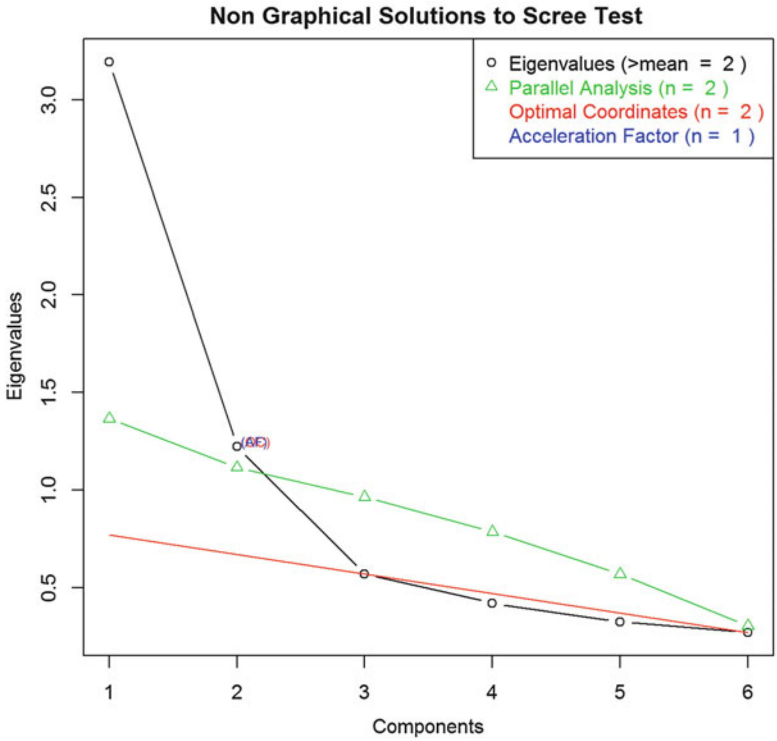


Fig. 6.16 Scree plots of various solutions

```
## Report For a nScree Class
##
## Details: components
##
##   Eigenvalues Prop Cumu Par.Analysis Pred.eig     OC Acc.factor     AF
## 1           3    1    1            1        1                NA (< AF)
## 2           1    0    1            1        1 (< OC)          1
## 3           1    0    1            1        0                 1
## 4           0    0    1            1        0                 0
## 5           0    0    1            1       NA                 0
## 6           0    0    1            0       NA                NA
##
##
##  Number of factors retained by index
##
##   noc naf nparallel nkaiser
## 1   2   1         2       2
```

Three out of four rules in Cattell's Scree test summary suggest we should use two factors. Thus, in function `factanal()` we use `factors=2` and the `varimax` rotation as performing arithmetic to obtain a new set of factor loadings. Oblique `promax` and `Procrustes rotation` (projecting the loadings to a target matrix with a simple structure) are two other commonly used matrix rotations.

```
fit<-factanal(pd.sub, factors=2, rotation="varimax")
# fit<-factanal(pd.sub, factors=2, rotation="promax") # the most popular obl
ique rotation; And fitting a simple structure
fit
```

```
## Call:
## factanal(x = pd.sub, factors = 2, rotation = "varimax")
##
## Uniquenesses:
##  Top_of_SN_Voxel_Intensity_Ratio Side_of_SN_Voxel_Intensity_Ratio
##                            0.018                            0.534
##                          Part_IA                          Part_IB
##                            0.571                            0.410
##                          Part_II                         Part_III
##                            0.392                            0.218
##
## Loadings:
##                                  Factor1 Factor2
## Top_of_SN_Voxel_Intensity_Ratio          0.991
## Side_of_SN_Voxel_Intensity_Ratio -0.417   0.540
## Part_IA                           0.650
## Part_IB                           0.726  -0.251
## Part_II                           0.779
## Part_III                          0.825  -0.318
##
##                Factor1 Factor2
## SS loadings      2.412   1.445
## Proportion Var   0.402   0.241
## Cumulative Var   0.402   0.643
##
## Test of the hypothesis that 2 factors are sufficient.
## The chi square statistic is 1.35 on 4 degrees of freedom.
## The p-value is 0.854
```

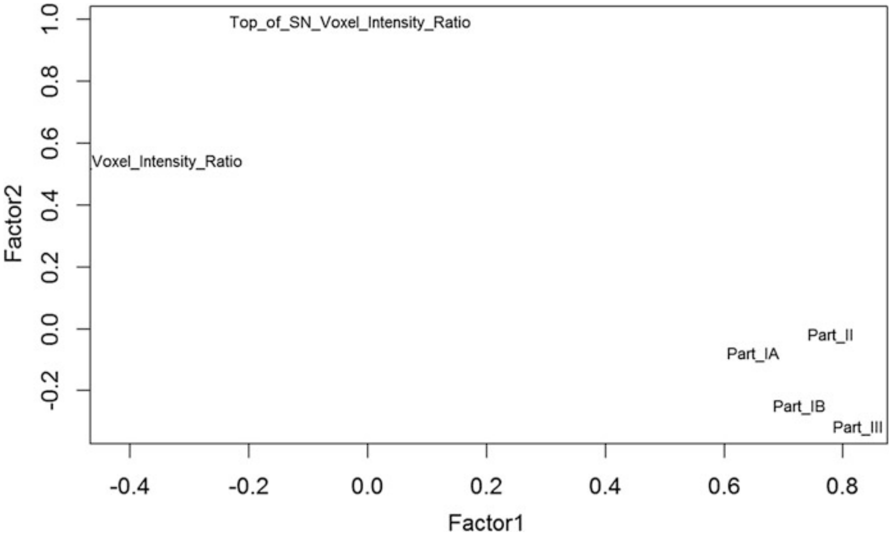


Fig. 6.17 Factor analysis results projecting the key features on the first two factor dimensions

Here the p-value 0.854 is very large, suggesting that we failed to reject the null-hypothesis that two factors are sufficient. We can also visualize the loadings for all the variables (Fig. 6.17).

```
load <- fit$loadings
plot(load, type="n") # set up plot
text(load, labels=colnames(pd.sub), cex=.7) # add variable names
```

This plot displays *factors 1* and *2* on the x-axis and y-axis, respectively.

6.8 Singular Value Decomposition (SVD)

SVD is a factorization of a real or complex matrix. If we have a data matrix X with n observation and p variables, it can be factorized into the following form:

$$X = UDV^T,$$

where U is a $n \times p$ unitary matrix, that $U^TU = I$, D is a $p \times p$ diagonal matrix, and V^T is a $p \times p$ unitary matrix, which is the conjugate transpose of the $n \times n$ unitary matrix, V. Thus, we have $V^TV = I$.

SVD is closely linked to PCA (when correlation matrix is used for calculation). U are the left singular vectors. D are the singular values. U gives PCA scores. V are the right singular vectors-PCA loadings.

We can compare the output from the `svd()` function and the `princomp()` function (another R function for PCA). Still, we are using the `pd.sub` dataset. Before the SVD, we need to scale our data matrix.

```
#SVD output
df<-nrow(pd.sub)-1
zvars<-scale(pd.sub)
z.svd<-svd(zvars)
z.svd$d/sqrt(df)

## [1] 1.7878123 1.1053808 0.7550519 0.6475685 0.5688743 0.5184536

z.svd$v

##             [,1]       [,2]        [,3]        [,4]       [,5]        [,6]
## [1,]  0.2555204 0.71258155 -0.37323594  0.10487773 -0.4773992  0.22073161
## [2,]  0.3855208 0.47213743  0.35665523 -0.43312945  0.5581867  0.04564469
## [3,] -0.3825033 0.37288211  0.70992668  0.31993403 -0.2379855 -0.22728693
## [4,] -0.4597352 0.09803466 -0.11166513 -0.79389290 -0.2915570 -0.22647775
## [5,] -0.4251107 0.34167997 -0.46424927  0.26165346  0.5341197 -0.36505061
## [6,] -0.4976933 0.06258370  0.03872473 -0.01769966  0.1832789  0.84438182

#PCA output
pca2<-princomp(pd.sub, cor=T)
pca2

## Call:
## princomp(x = pd.sub, cor = T)
##
## Standard deviations:
##    Comp.1    Comp.2    Comp.3    Comp.4    Comp.5    Comp.6
## 1.7878123 1.1053808 0.7550519 0.6475685 0.5688743 0.5184536
##
##  6  variables and  33 observations.

loadings(pca2)

##
## Loadings:
##                                  Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6
## Top_of_SN_Voxel_Intensity_Ratio -0.256 -0.713 -0.373 -0.105  0.477 -0.221
##Side_of_SN_Voxel_Intensity_Ratio -0.386 -0.472  0.357  0.433 -0.558
## Part_IA                           0.383 -0.373  0.710 -0.320  0.238  0.227
## Part_IB                           0.460        -0.112  0.794  0.292  0.226
## Part_II                           0.425 -0.342 -0.464 -0.262 -0.534  0.365
## Part_III                          0.498                      -0.183 -0.844
##
##                Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6
## SS loadings     1.000  1.000  1.000  1.000  1.000  1.000
## Proportion Var  0.167  0.167  0.167  0.167  0.167  0.167
## Cumulative Var  0.167  0.333  0.500  0.667  0.833  1.000
```

When the correlation matrix is used for calculation (`cor=T`), the *V* matrix of SVD contains the loadings of the PCA.

6.9 SVD Summary

Intuitively, the SVD approach $X = UDV^T$ represents a composition of the (centered!) data into three geometrical transformations: a rotation or reflection (U), a scaling (D), and a rotation or reflection (V). Here we assume that the data X stores samples/cases in rows and variables/features in columns. If these are reversed, then the interpretations of the U and V matrices reverse as well.

- The columns of V represent the directions of the principal axes, the columns of UD are the principal components, and the singular values in D are related to the eigenvalues of data variance-covariance matrix (Σ) via $\lambda_i = \frac{d_i^2}{n-1}$, where the eigenvalues λ_i capture the magnitude of the data variance in the respective PCs.
- The standardized scores are given by columns of $\sqrt{n-1}U$ and the corresponding loadings are given by columns of $\frac{1}{n-1}VD$. However, these "loadings" *are not* the principal directions. The requirement for X to be centered is needed to ensure that the covariance matrix $Cov(X) = \frac{1}{n-1}X^TX$.
- Alternatively, to perform PCA on the *correlation matrix* (instead of the covariance matrix), the columns of X need to be *scaled* (centered and standardized).
- To reduce the data dimensionality from p to $k < p$, we multiply the first k columns of U by the $k \times k$ upper-left corner of the matrix D to get an $n \times k$ matrix U_kD_k containing the first k PCs.
- Multiplying the first k PCs by their corresponding principal directions V_k^T reconstructs the original data from the first k PCs, $X_k{=}U_kD_kV_k^T$, with the lowest possible reconstruction error.
- Typically, we have more subjects/cases (n) than variables/features ($p < n$). As $U_{n \times n}$ and $V_{p \times p}$, the last $n - p > 0$ columns of U may be trivial (zeros). It's customary to drop the zero columns of U for $n \gg p$ to avid dealing with unnecessarily large (trivial) matrices.

6.10 Case Study for Dimension Reduction (Parkinson's Disease)

Step 1: Collecting Data

The data we will be using in this case study is the Clinical, Genetic and Imaging Data for Parkinson's Disease in the SOCR website. A detailed data explanation is available online http://wiki.socr.umich.edu/index.php/SOCR_Data_PD_BiomedBigMetadata. Let's import the data into R.

```
# Loading required package: xml2
wiki_url <- read_html("http://wiki.socr.umich.edu/index.php/SOCR_Data_PD_Bio
medBigMetadata")
html_nodes(wiki_url, "#content")

## {xml_nodeset (1)}
## [1] <div id="content" class="mw-body-primary" role="main">\n\t<a id="top
...

pd_data <- html_table(html_nodes(wiki_url, "table")[[1]])
head(pd_data); summary(pd_data)

##   Cases L_caudate_ComputeArea L_caudate_Volume R_caudate_ComputeArea
## 1     2                   597              767                   855
## 2     2                   597              767                   855
## 3     2                   597              767                   855
## 4     2                   597              767                   855
## 5     3                   604              873                   935
## 6     3                   604              873                   935
…

##   chr17_rs11868035_GT UPDRS_part_I UPDRS_part_II UPDRS_part_III Time
## 1                   0            1            12              1    0
## 2                   0            1            12              1    6
## 3                   0            1            12              1   12
## 4                   0            1            12              1   18
## 5                   1            0            19             22    0
## 6                   1            0            19             22    6

##      Cases        L_caudate_ComputeArea L_caudate_Volume
##  Min.   :  2.0   Min.   :525.0         Min.   :719.0
##  1st Qu.:158.0   1st Qu.:582.0         1st Qu.:784.0
##  Median :363.5   Median :600.0         Median :800.0
##  Mean   :346.1   Mean   :600.4         Mean   :800.3
##  3rd Qu.:504.0   3rd Qu.:619.0         3rd Qu.:819.0
##  Max.   :692.0   Max.   :667.0         Max.   :890.0
…

##  Min.   : 0.0
##  1st Qu.: 4.5
##  Median : 9.0
##  Mean   : 9.0
##  3rd Qu.:13.5
##  Max.   :18.0
```

Step 2: Exploring and Preparing the Data

To make sure that the data is ready for further modeling, we need to fix a few things. First, the Dx variable, or diagnosis, is a factor. We need to change it to a numeric variable. Second, we don't need the patient ID and time variable in the dimension reduction procedures.

```
pd_data$Dx <- gsub("PD", 1, pd_data$Dx)
pd_data$Dx <- gsub("HC", 0, pd_data$Dx)
pd_data$Dx <- gsub("SWEDD", 0, pd_data$Dx)
pd_data$Dx <- as.numeric(pd_data$Dx)
attach(pd_data)
pd_data<-pd_data[, -c(1, 33)]
```

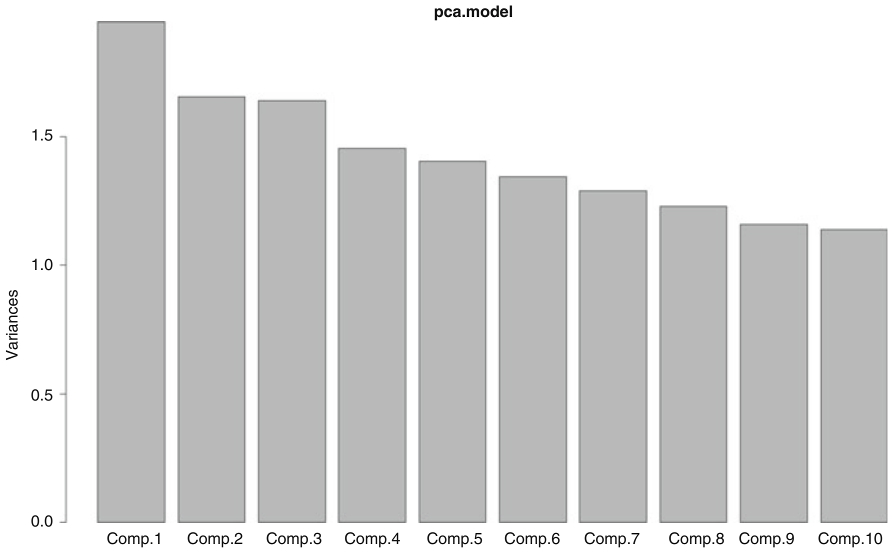


Fig. 6.18 Barplot illustrating the decay of the eigenvectors corresponding to the PCA linear transformation of the variables in the Parkinson's disease dataset (Figs. 6.19 and 6.20)

Step 3: Training a Model on the Data

1. **PCA**

Now we start the process of fitting a PCA model. Here we will use the `princomp()` function and use the correlation rather than the covariance matrix for calculation.

```
pca.model <- princomp(pd_data, cor=TRUE)
summary(pca.model) # pc loadings (i.e., igenvector columns)

## Importance of components:
##                            Comp.1     Comp.2     Comp.3    Comp.4
## Standard deviation     1.39495952 1.28668145 1.28111293 1.2061402
## Proportion of Variance 0.06277136 0.05340481 0.05294356 0.0469282
## Cumulative Proportion  0.06277136 0.11617617 0.16911973 0.2160479
##                           Comp.5     Comp.6   Comp.7     Comp.8    Comp.9
## Standard deviation    1.18527282 1.15961464 1.135510 1.10882348 1.0761943
## Proportion of Variance 0.04531844 0.04337762 0.041593 0.03966095 0.037361
## Cumulative Proportion  0.26136637 0.30474399 0.346337 0.38599794 0.423359
##                           Comp.10    Comp.11    Comp.12    Comp.13
## Standard deviation     1.06687730 1.05784209 1.04026215 1.03067437
## Proportion of Variance 0.03671701 0.03609774 0.03490791 0.03426741
## Cumulative Proportion  0.46007604 0.49617378 0.53108169 0.56534910
##                          Comp.14    Comp.15    Comp.16    Comp.17
## Standard deviation     1.0259684 0.99422375 0.97385632 0.96688855
## Proportion of Variance 0.0339552 0.03188648 0.03059342 0.03015721
## Cumulative Proportion  0.5993043 0.63119078 0.66178421 0.69194141
##                           Comp.18    Comp.19    Comp.20    Comp.21
## Standard deviation      0.92687735 0.92376374 0.89853718 0.88924412
## Proportion of Variance  0.02771296 0.02752708 0.02604416 0.02550823
## Cumulative Proportion   0.71965437 0.74718145 0.77322561 0.79873384
##                            Comp.22    Comp.23    Comp.24    Comp.25
## Standard deviation      0.87005195 0.86433816 0.84794183 0.82232529
## Proportion of Variance  0.02441905 0.02409937 0.02319372 0.02181351
## Cumulative Proportion   0.82315289 0.84725226 0.87044598 0.89225949
##                            Comp.26    Comp.27    Comp.28    Comp.29
## Standard deviation      0.80703739 0.78546699 0.77505522 0.76624322
## Proportion of Variance  0.02100998 0.01990188 0.01937776 0.01893963
## Cumulative Proportion   0.91326947 0.93317135 0.95254911 0.97148875
##                            Comp.30    Comp.31
## Standard deviation      0.68806884 0.64063259
## Proportion of Variance  0.01527222 0.01323904
## Cumulative Proportion   0.98676096 1.00000000

 plot(pca.model)
```

```
biplot(pca.model)
```

```
fviz_pca_biplot(pca.model, axes = c(1, 2), geom = "point",
  col.ind = "black", col.var = "steelblue", label = "all",
  invisible = "none", repel = F, habillage = pd_data$Sex,
  palette = NULL, addEllipses = TRUE, title = "PCA - Biplot")
```

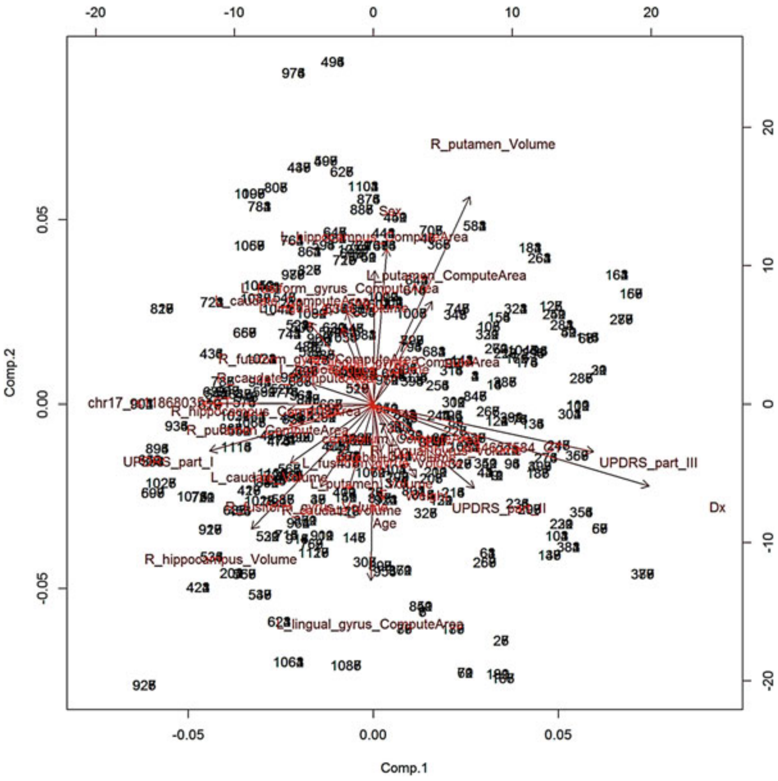


Fig. 6.19 Biplot of the PD variables onto the first two principle axes

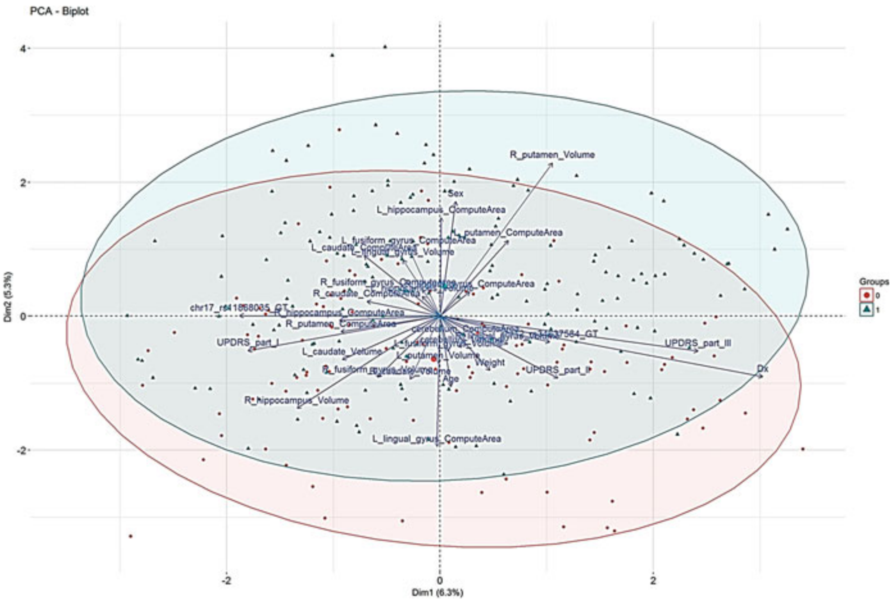


Fig. 6.20 Enhanced biplot of the PD data explicitly labeling the patients and control volunteers

We can see that in real world examples PCs do not necessarily have an “elbow” in the scree plot (Fig. 6.18). In our model, each PC explains about the same amount of variation. Thus, it is hard to tell how many PCs, or factors, we need to pick. This would be an ad hoc decision.

2. FA

Let’s set up a Cattell’s Scree test to determine the number of factors first.

```
ev <- eigen(cor(pd_data)) # get eigenvalues
ap <- parallel(subject=nrow(pd_data), var=ncol(pd_data), rep=100, cent=.05)
nS <- nScree(x=ev$values, aparallel=ap$eigen$qevpea)
summary(nS)

## Report For a nScree Class
##
## Details: components
##
##    Eigenvalues Prop Cumu Par.Analysis Pred.eig     OC Acc.factor     AF
## 1            2    0    0            1        2 (< OC)         NA (< AF)
## 2            2    0    0            1        2                 0
## 3            2    0    0            1        1                 0
## 4            1    0    0            1        1                 0
## 5            1    0    0            1        1                 0
…
```

```
## 30           0    0    1            1       NA                 0
## 31           0    0    1            1       NA                NA
##
##
##  Number of factors retained by index
##
##   noc naf nparallel nkaiser
## 1   1   1        14      14
```

Although the Cattell's Scree test suggest that we should use 14 factors, the real fit shows 14 is not enough. Previous PCA results suggest we need around 20 PCs to obtain a cumulative variance of 0.6. After a few trials, we find that 19 factors can pass the chi square test for sufficient number of factors at 0.05 level.

```
fa.model<-factanal(pd_data, 19, rotation="varimax")
fa.model
```

```
##
## Call:
## factanal(x = pd_data, factors = 19, rotation = "varimax")
##
## Uniquenesses:
##        L_caudate_ComputeArea             L_caudate_Volume
##                        0.840                        0.005
##        R_caudate_ComputeArea             R_caudate_Volume
##                        0.868                        0.849
##        L_putamen_ComputeArea             L_putamen_Volume
##                        0.791                        0.702
##        R_putamen_ComputeArea             R_putamen_Volume
##                        0.615                        0.438
##    L_hippocampus_ComputeArea         L_hippocampus_Volume
##                        0.476                        0.777
##    R_hippocampus_ComputeArea         R_hippocampus_Volume
##                        0.798                        0.522
##       cerebellum_ComputeArea            cerebellum_Volume
##                        0.137                        0.504
##  L_lingual_gyrus_ComputeArea       L_lingual_gyrus_Volume
##                        0.780                        0.698
##  R_lingual_gyrus_ComputeArea       R_lingual_gyrus_Volume
##                        0.005                        0.005
## L_fusiform_gyrus_ComputeArea      L_fusiform_gyrus_Volume
##                        0.718                        0.559
## R_fusiform_gyrus_ComputeArea      R_fusiform_gyrus_Volume
##                        0.663                        0.261
##                          Sex                       Weight
##                        0.829                        0.005
##                          Age                           Dx
##                        0.005                        0.005
##          chr12_rs34637584_GT          chr17_rs11868035_GT
##                        0.638                        0.721
##                 UPDRS_part_I                UPDRS_part_II
##                        0.767                        0.826
##               UPDRS_part_III
##                        0.616
```

```
##
## Loadings:
##                              Factor1 Factor2 Factor3 Factor4 Factor5
## L_caudate_ComputeArea
## L_caudate_Volume                                              0.980
## R_caudate_ComputeArea
## R_caudate_Volume
## L_putamen_ComputeArea
## L_putamen_Volume
## R_putamen_ComputeArea
## R_putamen_Volume
## L_hippocampus_ComputeArea
## L_hippocampus_Volume
## R_hippocampus_ComputeArea    -0.102
## R_hippocampus_Volume
## cerebellum_ComputeArea
## cerebellum_Volume
## L_lingual_gyrus_ComputeArea   0.107
## L_lingual_gyrus_Volume
## R_lingual_gyrus_ComputeArea                           0.989
## R_lingual_gyrus_Volume                0.983
## L_fusiform_gyrus_ComputeArea
## L_fusiform_gyrus_Volume
## R_fusiform_gyrus_ComputeArea
## R_fusiform_gyrus_Volume
## Sex                                          -0.111
## Weight                                        0.983
## Age
## Dx                            0.965
## chr12_rs34637584_GT                   0.124
## chr17_rs11868035_GT          -0.303
## UPDRS_part_I                 -0.260
## UPDRS_part_II
## UPDRS_part_III                0.332           0.104
##                              Factor6 Factor7 Factor8 Factor9 Factor10
## L_caudate_ComputeArea                -0.101
## L_caudate_Volume
…
##                 Factor1 Factor2 Factor3 Factor4 Factor5 Factor6 Factor7
## SS loadings      1.282   1.029   1.026   1.019   1.013   1.011   0.921
## Proportion Var   0.041   0.033   0.033   0.033   0.033   0.033   0.030
## Cumulative Var   0.041   0.075   0.108   0.140   0.173   0.206   0.235
##                 Factor8 Factor9 Factor10 Factor11 Factor12 Factor13
## SS loadings      0.838   0.782    0.687    0.647    0.615    0.587
## Proportion Var   0.027   0.025    0.022    0.021    0.020    0.019
## Cumulative Var   0.263   0.288    0.310    0.331    0.351    0.370
##                 Factor14 Factor15 Factor16 Factor17 Factor18 Factor19
## SS loadings       0.569    0.566    0.547    0.507    0.475    0.456
## Proportion Var    0.018    0.018    0.018    0.016    0.015    0.015
## Cumulative Var    0.388    0.406    0.424    0.440    0.455    0.470
##
## Test of the hypothesis that 19 factors are sufficient.
## The chi square statistic is 54.51 on 47 degrees of freedom.
## The p-value is 0.211
```

This data matrix has relatively low correlation. Thus, it is not suitable for ICA.

```
cor(pd_data)[1:10, 1:10]

##                           L_caudate_ComputeArea L_caudate_Volume
## L_caudate_ComputeArea               1.000000000       0.05794916
## L_caudate_Volume                    0.057949162       1.00000000
## R_caudate_ComputeArea              -0.060576361       0.01076372
## R_caudate_Volume                    0.043994457       0.07245568
## L_putamen_ComputeArea               0.009640983      -0.06632813
## L_putamen_Volume                   -0.064299184      -0.11131525
## R_putamen_ComputeArea               0.040808105       0.04504867
## R_putamen_Volume                    0.058552841      -0.11830387
## L_hippocampus_ComputeArea          -0.037932760      -0.04443615
## L_hippocampus_Volume               -0.042033469      -0.04680825
…

## L_caudate_ComputeArea                0.04080810      0.058552841
## L_caudate_Volume                     0.04504867     -0.118303868
## R_caudate_ComputeArea                0.07864348      0.007022844
## R_caudate_Volume                     0.05428747     -0.094336376
## L_putamen_ComputeArea                0.09049611      0.176353726
## L_putamen_Volume                     0.09093926     -0.057687648
## R_putamen_ComputeArea                1.00000000      0.052245264
## R_putamen_Volume                     0.05224526      1.000000000
## L_hippocampus_ComputeArea           -0.05508472      0.131800075
## L_hippocampus_Volume                -0.08866344     -0.001133570
##                           L_hippocampus_ComputeArea L_hippocampus_Volume
## L_caudate_ComputeArea                  -0.037932760          -0.04203347
## L_caudate_Volume                       -0.044436146          -0.04680825
## R_caudate_ComputeArea                   0.051359613           0.08578833
## R_caudate_Volume                        0.006123355          -0.07791361
## L_putamen_ComputeArea                   0.094604791          -0.06442537
## L_putamen_Volume                        0.025303302           0.04041557
## R_putamen_ComputeArea                  -0.055084723          -0.08866344
## R_putamen_Volume                        0.131800075          -0.00113357
## L_hippocampus_ComputeArea               1.000000000          -0.02633816
## L_hippocampus_Volume                   -0.026338163           1.00000000
```

6.11 Assignments: 6. Dimensionality Reduction

6.11.1 Parkinson's Disease Example

Apply principal component analysis (PCA), singular value decomposition (SVD), independent component analysis (ICA), and factor analysis (FA) to reduce the dimensionality of the PD data. Interpret the results.

6.11.2 Allometric Relations in Plants Example

Load Data

Load Allometric Relations in Plants data and perform a proper type conversion, e.g., convert “Province” and “Born”.

Dimensionality Reduction

- Apply Principal Component Analysis protocol.
  - Generate a data summary
  - Apply `prcomp`
  - Report the rotations (scores)
  - Display screen plot
  - Select the number of PCs and employ a bootstrap test
  - Apply `factoextra` to draw biplot and grouped by Province/Sites
- Perform SVD and ICA and compare the results of PCA.
  - Use these three variables `L, M, D` to perform ICA and show pair plots before ICA and after ICA. (Hint: `scatter3dplot()` may be helpful, which you saw in Chap. 5.)
- Perform factor analysis.
  - Use `require(nFactors)` to determine the number of the factors and show a scree plot as stated in notes
  - Use `factanal()` to apply FA and compare the rotation `varimax` and `promax`
  - Report the loadings and consider an appropriate visualization method.
- Interpret the findings in the context of the case-study.

References

Jolliffe, I.T. (2002) Principal Component Analysis, Springer.

Karhunen, J. and Hyvärinen, A. (2001) Independent Component Analysis, Wiley-Interscience.

Cattell, R.B. (1952) Factor analysis. New York: Harper.