Análisis de Componentes Principales (Principal Component Analysis, PCA)

Joaquín Amat Rodrigo j.amatrodrigo@gmail.com

Junio, 2017

Índice

Formato PDF: https://github.com/JoaquinAmatRodrigo/Estadistica-con-R

Introducción

Principal Component Analysis es un método estadístico que permite encontrar un espacio de menor dimensión que el espacio muestral inicial pero que contiene prácticamente la misma información que el espacio original. Supóngase que existe una muestra con n individuos cada uno con p variables (X_1, X_2, ..., X_p), es decir, el espacio muestral tiene p dimensiones. *PCA* permite encontrar un número de factores subyacentes $z < p$ que explican aproximadamente lo mismo que las p variables originales. Donde antes se necesitaban p valores para caracterizar a cada individuo, ahora bastan z valores. Cada una de estas z nuevas variables recibe el nombre de componente principal.

Principal Component Analysis (PCA) pertenece a la familia de técnicas conocida como *unsupervised learning*. Los métodos de *supervised learning* descritos en capítulos anteriores tienen el objetivo de predecir una variable respuesta Y a partir de una serie de predictores. Para ello, se dispone de las p características (X_1, X_2 ... X_p) y de la variable respuesta Y medidas en n observaciones. En el caso de *unsupervised learning*, la variable respuesta Y no se tiene en cuenta ya que el objetivo no es predecir Y sino extraer información empleando los predictores, por ejemplo, para identificar subgrupos. El principal problema al que se enfrentan los métodos de *unsupervised learning* es la dificultad para validar los resultados dado que no se dispone de una variable respuesta que permita contrastarlos.

El método de PCA permite "condensar" la información aportada por múltiples variables en solo unas pocas componentes, pero sigue siendo necesario disponer del valor de las variables originales para calcularlas.

Álgebra Matricial

En esta sección se describen dos de los conceptos matemáticos que se aplican en el PCA: *eigenvectors* y *eigenvalues*. Se trata simplemente de una descripción intuitiva con la única finalidad de que facilitar el entendimiento del cálculo de componentes principales.

Eigenvectors

Los *eigenvectors* son un caso particular de multiplicación entre una matriz y un vector. Obsérvese la siguiente multiplicación:

$$\begin{pmatrix} 2 & 3 \\ 2 & 1 \end{pmatrix} x \begin{pmatrix} 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 12 \\ 8 \end{pmatrix} = 4\ x \begin{pmatrix} 3 \\ 2 \end{pmatrix}$$

El vector resultante de la multiplicación es un múltiplo entero del vector original. Los *eigenvectors* de una matriz son todos aquellos vectores que, al multiplicarlos por dicha matriz, resultan en el mismo vector o en un múltiplo entero del mismo. Los *eigenvectors* tienen una serie de propiedades matemáticas específicas:

- Los *eigenvectors* solo existen para matrices cuadradas y no para todas. En el caso de que una matriz *n* x *n* tenga *eigenvectors*, el número de ellos es *n*.
- Si se escala un *eigenvector* antes de multiplicarlo por la matriz, se obtiene un múltiplo del mismo *eigenvector*. Esto se debe a que si se escala un vector multiplicándolo por cierta cantidad, lo único que se consigue es cambiar su longitud pero la dirección es la misma.
- Todos los *eigenvectors* de una matriz son perpendiculares (ortogonales) entre ellos, independientemente de las dimensiones que tengan.

Dada la propiedad de que multiplicar un *eigenvector* solo cambia su longitud pero no su naturaleza de *eigenvector*, es frecuente escalarlos de tal forma que su longitud sea 1. De este modo se consigue que todos ellos estén estandarizados. A continuación se muestra un ejemplo:

El *eigenvector* $\binom{3}{2}$ tiene una longitud de $\sqrt{3^2 + 2^2} = \sqrt{13}$. Si se divide cada dimensión entre la longitud del vector, se obtiene el *eigenvector* estandarizado con longitud 1.

$$\binom{3}{2} \div \sqrt{13} = \binom{3/\sqrt{13}}{2/\sqrt{13}}$$

Eigenvalue

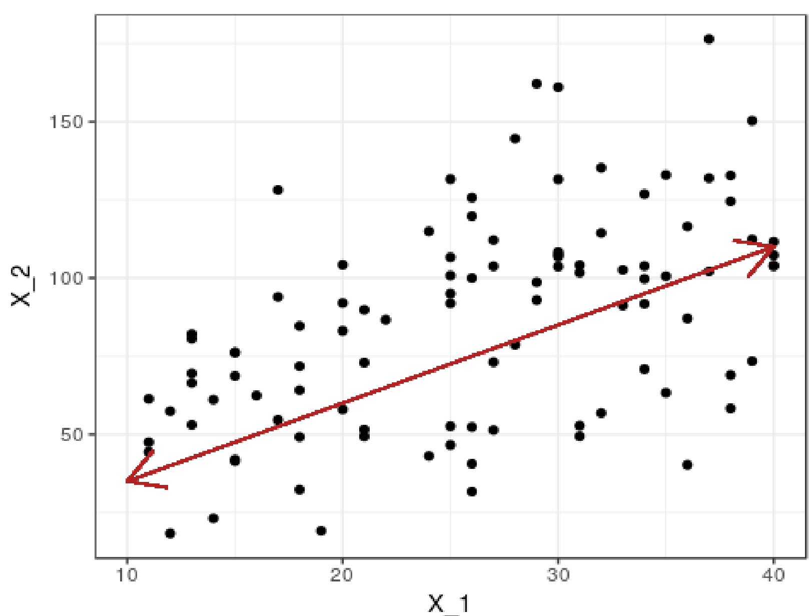
Cuando se multiplica una matriz por alguno de sus *eigenvectors* se obtiene un múltiplo del vector original, es decir, el resultado es ese mismo vector multiplicado por un número. Al valor por el que se multiplica el *eigenvector* resultante se le conoce como *eigenvalue*. A todo *eigenvector* le corresponde un *eigenvalue* y viceversa.

El método PCA, cada una de las componentes se corresponde con un *eigenvector*, y el orden de componente se establece por orden decreciente de *eigenvalue*. Así pues, la primera componente es el *eigenvector* con el *eigenvalue* asociado más alto.

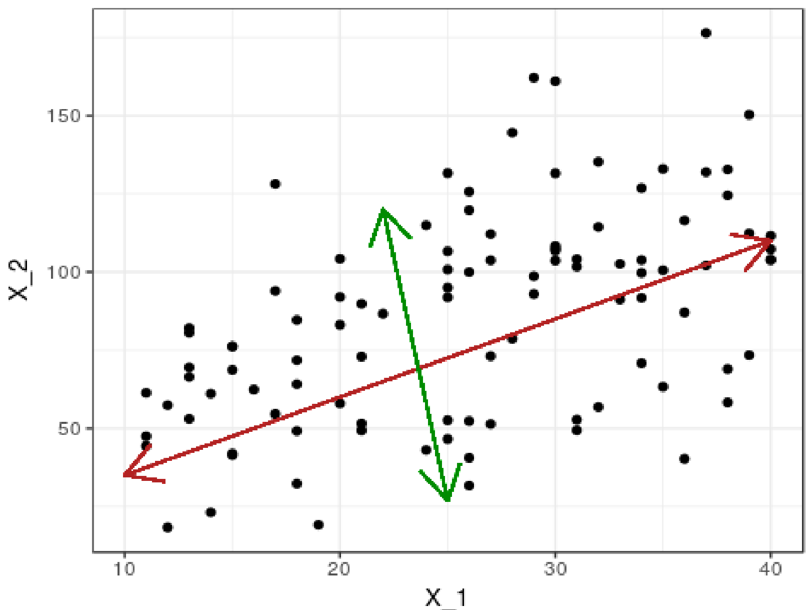
Interpretación geométrica de las componentes principales

Una forma de intuitiva de entender el proceso de PCA consiste en interpretar las componentes principales desde un punto de vista geométrico. Supóngase un conjunto de observaciones para las que se dispone de dos variables (X_1, X_2). El vector que define la primera componente principal (Z_1) sigue la dirección en la que las observaciones varían más (linea roja). La proyección de cada observación sobre esa dirección equivale al valor de la primera componente para dicha observación (*principal component scores*, z_{i1}).

```
library(ggplot2)
set.seed(435)
X_1 <- sample(x = 10:40, size = 100, replace = TRUE)
X_2 <- 2.5 * X_1 + 10
X_2 <- X_2 + rnorm(n = 100, mean = 10, sd = 30)
datos <- data.frame(X_1, X_2)
ggplot(data = datos, aes(x = X_1, y = X_2)) +
geom_point() +
geom_segment(aes(x = 10, y = 2.5 * 10 + 10, xend = 40, yend = 2.5 * 40 + 10),
                  colour = "firebrick", arrow = arrow(ends = "both")) +
theme_bw()
```



La segunda componente (Z_2) sigue la segunda dirección en la que los datos muestran mayor varianza y que no está correlacionada con la primera componente. La condición de no correlación entre componentes principales equivale a decir que sus direcciones son perpendiculares/ortogonales.



Cálculo de las componentes principales

Cada componente principal (Z_i) se obtiene por combinación lineal de las variables originales. Se pueden entender como nuevas variables obtenidas al combinar de una determinada forma las variables originales. La primera componente principal de un grupo de variables (X_1, X_2, ..., X_p) es la combinación lineal normalizada de dichas variables que tiene mayor varianza:

$$Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + \ldots + \phi_{p1}X_p$$

Que la combinación lineal sea normalizada implica que:

$$\sum_{j=1}^{p} \phi_{j1}^2 = 1$$

Los términos ϕ_{11}, ..., ϕ_{1p} reciben en el nombre de *loadings* y son los que definen a la componente. ϕ_{11} es el *loading* de la variable X_1 de la primera componente principal. Los *loadings* pueden interpretarse como el peso/importancia que tiene cada variable en cada componente y, por lo tanto, ayudan a conocer que tipo de información recoge cada una de las componentes.

Dado un set de datos X con n observaciones y p variables, el proceso a seguir para calcular la primera componente principal se inicia con la centralización de las variables (se resta a cada valor la media de la variable a la que pertenece). Con esto se consigue que todas las variables tengan media cero. A continuación se resuelve un problema de optimización para encontrar el valor de los *loadings* con los que se maximiza la varianza. Una forma de resolver esta optimización es mediante el cálculo de *eigenvector-eigenvalue* de la matriz de covarianzas. Una vez calculada la primera componente (Z_1) se calcula la segunda (Z_2) repitiendo el mismo proceso, pero añadiendo la condición de que la combinación lineal no pude estar correlacionada con la primera componente. Esto equivale a decir que Z_1 y Z_2 tienen que ser perpendiculares. EL proceso se repite de forma iterativa hasta calcular todas las posibles componentes (min(n-1, p)) o hasta que se decida detener el proceso. El orden de importancia de las componentes viene dado por la magnitud del *eigenvalue* asociado a cada *eigenvector*.

Escalado de las variables

El proceso de PCA identifica aquellas direcciones en las que la varianza es mayor. Como la varianza de una variable se mide en su misma escala elevada al cuadrado, si antes de calcular las componentes no se estandarizan todas las variables para que tengan media 0 y desviación estándar 1, aquellas variables cuya escala sea mayor dominarán al resto. De ahí que sea recomendable estandarizar siempre los datos.

Reproducibilidad de las componentes

El proceso de PCA genera siempre las mismas componentes principales independientemente del software utilizado, es decir, el valor de los *loadings* resultantes es el mismo. La única diferencia que puede darse es que el signo de todos los *loadings* esté invertido. Esto es así porque el vector de *loadings* determina la dirección de la componente, y dicha dirección es la misma independientemente del signo (la componente sigue una línea que se extiende en ambas direcciones). Del mismo modo, el valor específico de las componentes obtenido para cada observación (*principal component scores*) es siempre el mismo, a excepción del signo.

Influencia de outliers

Al trabajar con varianzas, el método PCA es altamente sensible a *outliers*, por lo que es altamente recomendable estudiar si los hay. La detección de valores atípicos con respecto a una determinada dimensión es algo relativamente sencillo de hacer mediante comprobaciones gráficas. Sin embargo, cuando se trata con múltiples dimensiones el proceso se complica. Por ejemplo, considérese un hombre que mide 2 metros y pesa 50 kg. Ninguno de los dos valores es atípico de forma individual, pero en conjunto se trataría de un caso muy excepcional. La distancia de *Mahalanobis* es una medida de distancia entre un punto y la media que se ajusta en función de la correlación entre dimensiones y que permite encontrar potenciales *outliers* en distribuciones multivariante.

Proporción de varianza explicada

Una de las preguntas más frecuentes que surge tras realizar un PCA es: ¿Cuánta información presente en el set de datos original se pierde al proyectar las observaciones en un espacio de menor dimensión? o lo que es lo mismo ¿Cuanta información es capaz de capturar cada una de las componentes principales obtenidas? Para contestar a estas preguntas se recurre a la proporción de varianza explicada por cada componente principal.

Asumiendo que las variables se han normalizado para tener media cero, la varianza total presente en el set de datos se define como

$$\sum_{j=1}^{p} V\,ar(X_j) = \sum_{j=1}^{p} \frac{1}{n} \sum_{i=1}^{n} x_{ij}^2$$

y la varianza explicada por la componente m es

$$\frac{1}{n} \sum_{i=1}^{n} z_{im}^2 = \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{j=1}^{p} \emptyset_{jm}\, x_{ij} \right)^2$$

Por lo tanto, la proporción de varianza explicada por la componente m viene dada por el ratio

$$\frac{\sum_{i=1}^{n} \left(\sum_{j=1}^{p} \phi_{jm} x_{ij} \right)^2}{\sum_{j=1}^{p} \sum_{i=1}^{n} x_{ij}^2}$$

Tanto la proporción de varianza explicada como la proporción de varianza explicada acumulada son dos valores de gran utilidad a la hora de decidir el número de de componentes principales a utilizar en los análisis posteriores. Si se calculan todas las componentes principales de un set de datos, entonces, aunque transformada, se está almacenando toda la información presente en los datos originales. El sumatorio de la proporción de varianza explicada acumulada de todas las componentes es siempre 1.

Número óptimo de componentes principales

Por lo general, dada una matriz de datos de dimensiones *n* x *p*, el número de componentes principales que se pueden calcular es como máximo de *n*-1 o *p* (el menor de los dos valores es el limitante). Sin embargo, siendo el objetivo del PCA reducir la dimensionalidad, suelen ser de interés utilizar el número mínimo de componentes que resultan suficientes para explicar los datos. No existe una respuesta o método único que permita identificar cual es el número óptimo de componentes principales a utilizar. Una forma de proceder muy extendida consiste en evaluar la proporción de varianza explicada acumulada y seleccionar el número de componentes mínimo a partir del cual el incremento deja de ser sustancial.

Ejemplo cálculo *eigenvectors* y *eigenvalues*

El siguiente es un ejemplo que muestra cómo se pueden calcular las componentes principales de un set de datos mediante la identificación de eigenvectors y eigenvalues por el método de covarianza. Con el objetivo de ser un ejemplo sencillo, el set de datos empleado tiene dos dimensiones (variables)

```
datos <- data.frame(X1 = c(2.5, 0.5, 2.2, 1.9, 3.1, 2.3, 2, 1, 1.5, 1.1),
                    X2 = c(2.4, 0.7, 2.9, 2.2, 3, 2.7, 1.6, 1.1, 1.6, 0.9))
datos
```

```
##      X1  X2
## 1  2.5 2.4
## 2  0.5 0.7
## 3  2.2 2.9
## 4  1.9 2.2
## 5  3.1 3.0
```

```
## 6  2.3 2.7
## 7  2.0 1.6
## 8  1.0 1.1
## 9  1.5 1.6
## 10 1.1 0.9
```

En primer lugar se resta a cada valor la media de la variable a la que pertenece. Con esto se consigue centralizar las variables y que su media sea 0.

```
datos_centrados <- datos
datos_centrados$X1 <- datos$X1 - mean(datos$X1)
datos_centrados$X2 <- datos$X2 - mean(datos$X2)
datos_centrados
```

```
##       X1    X2
## 1   0.69  0.49
## 2  -1.31 -1.21
## 3   0.39  0.99
## 4   0.09  0.29
## 5   1.29  1.09
## 6   0.49  0.79
## 7   0.19 -0.31
## 8  -0.81 -0.81
## 9  -0.31 -0.31
## 10 -0.71 -1.01
```

A continuación se calcula la matriz de correlaciones entre cada par de variables. Como en este ejemplo hay dos variables, el resultado es una matriz simétrica 2x2.

```
matriz_cov <- cov(datos_centrados)
matriz_cov
```

```
##           X1        X2
## X1 0.6165556 0.6154444
## X2 0.6154444 0.7165556
```

Dado que la matriz de covarianzas es cuadrada, se pueden obtener sus correspondientes *eigenvectors* y *eigenvalues*. La función `eigen()` calcula ambos y los almacena en una lista bajo el nombre de `vectors` y `values`. Los *eigenvalues* se devuelven en orden decreciente y los *eigenvectors* (estandarizados) se ordenan de izquierda a derecha acorde a sus *eigenvalues* asociados.

```
eigen <- eigen(matriz_cov)
eigen$values
```

```
## [1] 1.2840277 0.0490834
```

```
eigen$vectors
```

```
##           [,1]       [,2]
## [1,] 0.6778734 -0.7351787
## [2,] 0.7351787  0.6778734
```

Los *eigenvectors* ordenados de mayor a menor *eigenvalues* se corresponden con las componentes principales.

Una vez obtenidos los *eigenvectors* (componentes principales) se calcula el valor que toma cada componente para cada observación en función de las variables originales (*principal component scores*). Para ello, simplemente se tienen que multiplicar los *eigenvectors* transpuestos por los datos originales centrados y también transpuestos.

```
t_eigenvectors <- t(eigen$vectors)
t_eigenvectors
```

```
##            [,1]      [,2]
## [1,]  0.6778734 0.7351787
## [2,] -0.7351787 0.6778734
```

```
t_datos_centrados <- t(datos_centrados)
t_datos_centrados
```

```
##    [,1]  [,2] [,3] [,4] [,5] [,6]  [,7]  [,8]  [,9] [,10]
## X1 0.69 -1.31 0.39 0.09 1.29 0.49  0.19 -0.81 -0.31 -0.71
## X2 0.49 -1.21 0.99 0.29 1.09 0.79 -0.31 -0.81 -0.31 -1.01
```

```
# Producto matricial
pc_scores <- t_eigenvectors %*% t_datos_centrados
rownames(pc_scores) <- c("PC1", "PC2")

# Se vuelve a transponer para que los datos estén en modo tabla
t(pc_scores)
```

```
##               PC1         PC2
##  [1,]  0.82797019 -0.17511531
##  [2,] -1.77758033  0.14285723
##  [3,]  0.99219749  0.38437499
##  [4,]  0.27421042  0.13041721
##  [5,]  1.67580142 -0.20949846
##  [6,]  0.91294910  0.17528244
##  [7,] -0.09910944 -0.34982470
```

```
##  [8,] -1.14457216  0.04641726
##  [9,] -0.43804614  0.01776463
## [10,] -1.22382056 -0.16267529
```

En este ejemplo, al partir inicialmente de un espacio muestral con 2 dimensiones y haber calculado sus 2 componentes principales, no se ha reducido la dimensionalidad ni se ha perdido nada de información presente en los datos originales.

Cuando se calculan y conservan todos los *eigenvectors* (componentes principales), es posible recuperar de nuevo los valores iniciales. Solo es necesario invertir el proceso:

$$datos\ originales = (eigenvectors^{-1})\ x\ (principal\ component\ scores) + medias\ originales$$

Al estar incluidos todos los *eigenvectors* : $eigenvectors^{-1} = eigenvectors^{T}$

```
datos_recuperados <- t(eigen$vectors %*% pc_scores)
datos_recuperados[, 1] <- datos_recuperados[, 1] + mean(datos$X1)
datos_recuperados[, 2] <- datos_recuperados[, 2] + mean(datos$X2)

datos_recuperados
```

```
##       [,1] [,2]
##  [1,]  2.5  2.4
##  [2,]  0.5  0.7
##  [3,]  2.2  2.9
##  [4,]  1.9  2.2
##  [5,]  3.1  3.0
##  [6,]  2.3  2.7
##  [7,]  2.0  1.6
##  [8,]  1.0  1.1
##  [9,]  1.5  1.6
## [10,]  1.1  0.9
```

```
datos
```

```
##     X1  X2
## 1  2.5 2.4
## 2  0.5 0.7
## 3  2.2 2.9
## 4  1.9 2.2
## 5  3.1 3.0
## 6  2.3 2.7
## 7  2.0 1.6
## 8  1.0 1.1
## 9  1.5 1.6
## 10 1.1 0.9
```

En la mayoría de programas estadísticos, entre ellos `R`, existen funciones que calculan directamente las componentes principales y los *principal component scores* de cada observación sin tener que ir paso por paso.

Ejemplo PCA

El set de datos `USArrests` del paquete básico de `R` contiene el porcentaje de asaltos (Assault), asesinatos (Murder) y secuestros (Rape) por cada 100,000 habitantes para cada uno de los 50 estados de USA (1973). Además, también incluye el porcentaje de la población de cada estado que vive en zonas rurales (UrbanPoP).

```
data("USArrests")
head(USArrests)
```

```
##            Murder Assault UrbanPop Rape
## Alabama      13.2     236       58 21.2
## Alaska       10.0     263       48 44.5
## Arizona       8.1     294       80 31.0
## Arkansas      8.8     190       50 19.5
## California    9.0     276       91 40.6
## Colorado      7.9     204       78 38.7
```

El promedio de los datos muestra que hay tres veces más secuestros que asesinatos y 8 veces más asaltos que secuestros.

```
apply(X = USArrests, MARGIN = 2, FUN = mean)
```

```
##   Murder  Assault UrbanPop     Rape
##    7.788  170.760   65.540   21.232
```

La varianza es muy distinta entre las variables, en el caso de *Assault*, la varianza es varios órdenes de magnitud superior al resto.

```
apply(X = USArrests, MARGIN = 2, FUN = var)
```

```
##     Murder    Assault   UrbanPop       Rape
##   18.97047 6945.16571  209.51878   87.72916
```

Si no se estandarizan las variables para que tengan media cero y desviación estándar 1 antes de realizar el estudio PCA, la variable *Assault* dominará la mayoría de las componentes principales.

La función `prcomp()` es una de las múltiples funciones en R que realizan PCA. Por defecto, `prcomp()` centra las variables para que tengan media cero, pero si se quiere además que su desviación estándar sea de uno, hay que indicar `scale` `=` `TRUE`.

```
pca <- prcomp(USArrests, scale = TRUE)
names(pca)
```

```
## [1] "sdev"     "rotation" "center"   "scale"    "x"
```

Los elementos `center` y `scale` almacenados en el objeto `pca` contienen la media y desviación típica de las variables previa estandarización (en la escala original).

```
pca$center
```

```
##   Murder  Assault UrbanPop     Rape
##    7.788  170.760   65.540   21.232
```

```
pca$scale
```

```
##    Murder   Assault  UrbanPop      Rape
##  4.355510 83.337661 14.474763  9.366385
```

`rotation` contiene el valor de los *loadings* ϕ para cada componente (*eigenvector*). El número máximo de componentes principales se corresponde con el mínimo(n-1,p), que en este caso es $min(49,4) = 4$.

```
pca$rotation
```

```
##                 PC1        PC2        PC3         PC4
## Murder   -0.5358995  0.4181809 -0.3412327  0.64922780
## Assault  -0.5831836  0.1879856 -0.2681484 -0.74340748
## UrbanPop -0.2781909 -0.8728062 -0.3780158  0.13387773
## Rape     -0.5434321 -0.1673186  0.8177779  0.08902432
```

Analizar con detalle el vector de *loadings* que forma cada componente puede ayudar a interpretar que tipo de información recoge cada una de ellas. Por ejemplo, la primera componente es el resultado de la siguiente combinación lineal de las variables originales:

$$PC1 = -0.5358995\ Murder - 0.5831836\ Assault - 0.2781909\ UrbanPop - 0.5434321\ Rape$$

Los pesos asignados en la primera componente a las variables *Assault*, *Murder* y *Rape* son aproximadamente iguales entre ellos y bastante superiores al asignado a *UrbanPoP*, esto significa que la primera componente recoge mayoritariamente la información correspondiente a los delitos. En la segunda componente, es la variable *UrbanPoP* la que tiene con diferencia mayor peso, por lo que se corresponde principalmente con el nivel de urbanización del estado. Si bien en este ejemplo la interpretación de las componentes es bastante clara, no en todos los casos ocurre lo mismo.

La función `prcomp()` calcula automáticamente el valor de las componentes principales para cada observación (*principal component scores*) multiplicando los datos por los vectores de *loadings*. El resultado se almacena en la matriz x.

```
head(pca$x)
```

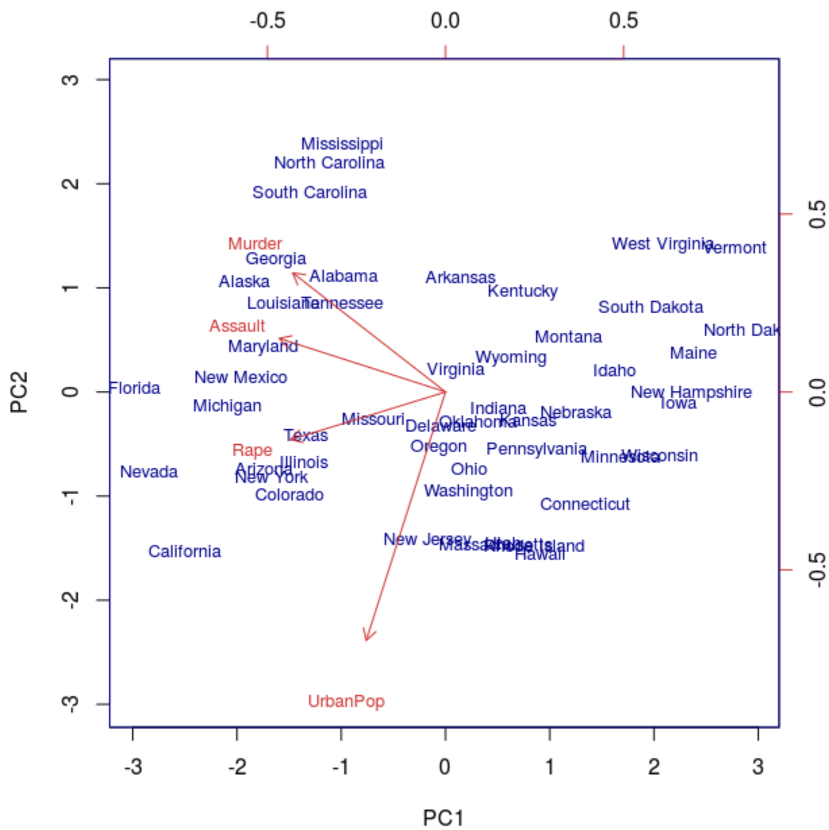
```
##                   PC1        PC2         PC3          PC4
## Alabama    -0.9756604  1.1220012 -0.43980366  0.154696581
## Alaska     -1.9305379  1.0624269  2.01950027 -0.434175454
## Arizona    -1.7454429 -0.7384595  0.05423025 -0.826264240
## Arkansas    0.1399989  1.1085423  0.11342217 -0.180973554
## California -2.4986128 -1.5274267  0.59254100 -0.338559240
## Colorado   -1.4993407 -0.9776297  1.08400162  0.001450164
```

```
dim(pca$x)
```

```
## [1] 50  4
```

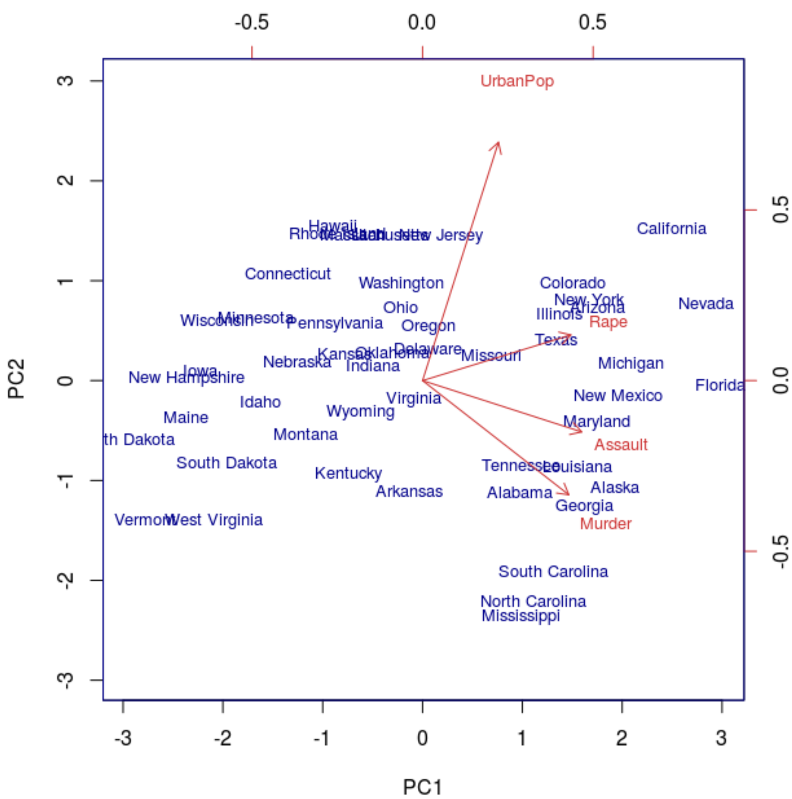
Mediante la función `biplot()` se puede obtener una representación bidimensional de las dos primeras componentes. Es recomendable indicar el argumento `scale = 0` para que las flechas estén en la misma escala que las componentes.

```
biplot(x = pca, scale = 0, cex = 0.8, col = c("blue4", "brown3"))
```



La imagen especular, cuya interpretación es equivalente, se puede obtener invirtiendo el signo de los *loadings* y de los *principal component scores*.

```
pca$rotation <- -pca$rotation
pca$x <- -pca$x
biplot(x = pca, scale = 0, cex = 0.8, col = c("blue4", "brown3"))
```



Una vez calculadas las componentes principales, se puede conocer la varianza explicada por cada una de ellas, la proporción respecto al total y la proporción de varianza acumulada.

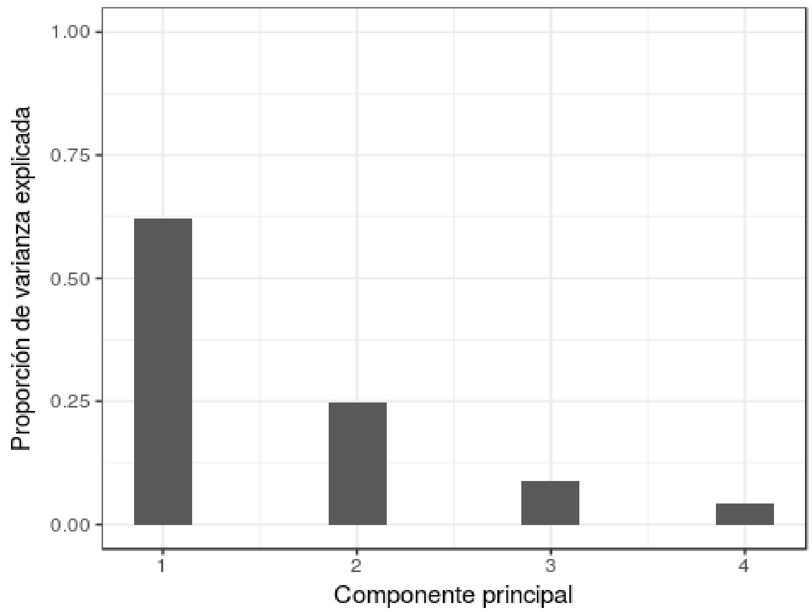
```
library(ggplot2)
pca$sdev^2
```

```
## [1] 2.4802416 0.9897652 0.3565632 0.1734301
```

```
prop_varianza <- pca$sdev^2/sum(pca$sdev^2)
prop_varianza
```

```
## [1] 0.62006039 0.24744129 0.08914080 0.04335752
```

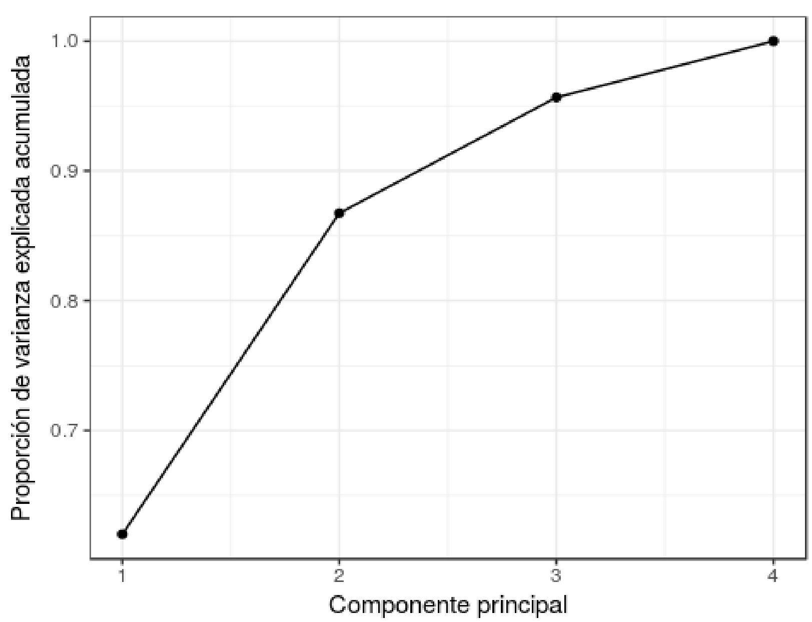
```
ggplot(data = data.frame(prop_varianza, pc = 1:4),aes(x = pc, y = prop_varianza)) +
geom_col(width = 0.3) +
 scale_y_continuous(limits = c(0, 1)) +
theme_bw() +
labs(x = "Componente principal", y = "Proporción de varianza explicada")
```




```
prop_varianza_acum <- cumsum(prop_varianza)
prop_varianza_acum
```

```
## [1] 0.6200604 0.8675017 0.9566425 1.0000000
```

```
ggplot(data = data.frame(prop_varianza_acum, pc = 1:4),
              aes(x = pc, y = prop_varianza_acum, group = 1)) +
geom_point() +
geom_line() +
theme_bw() +
labs(x = "Componente principal", y = "Proporción de varianza explicada acumulada")
```



En este caso, la primera componente explica el 62% de la varianza observada en los datos y la segunda el 24.7%. Las dos últimas componentes no superan por separado el 1% de varianza explicada. Si se empleasen únicamente las dos primeras componentes se conseguiría explicar el 83.8% de la varianza observada.

PCR: PCA aplicado a regresión lineal

El método *Principal Components Regression PCR* consiste en ajustar un modelo de regresión lineal por mínimos cuadrados empleando como predictores las componentes generadas a partir de un *Principal Component Analysis (PCA)*. De esta forma, con un número reducido de componentes se puede explicar la mayor parte de la varianza de los datos.

En los estudios observacionales, es frecuente disponer de un número elevado de variables que se pueden emplear como predictores. A pesar de ello, un alto número de predictores no implica necesariamente mucha información. Si las variables están correlacionadas entre ellas, la información que aportan es redundante y además viola la condición de no colinealidad necesaria en la regresión por mínimos cuadrados. Dado que el PCA es útil eliminando información redundante, si se emplean como predictores las componentes principales se puede mejorar el modelo de regresión. Es importante tener en cuenta que, si bien el *Principal Components Regression* reduce el número de predictores del modelo, no se puede considerar como un método de selección de variables ya que todas ellas se necesitan para el cálculo de las componentes. La identificación del número óptimo de componentes principales que se emplean como predictores en *PCR* puede identificarse por *cross validation*.

El método PCR no deja de ser un ajuste lineal por mínimos cuadrados que emplea componentes principales como predictores, para que sea válido se tienen que cumplir las condiciones requeridas para la regresión por mínimos cuadrados.

La cuantificación del contenido en grasa de la carne pude hacerse mediante técnicas de analítica química, sin embargo, este proceso es costoso en tiempo y recursos. Una posible alternativa para reducir costes y optimizar tiempo es emplear un espectrofotómetro (instrumento capaz de detectar la absorbancia que tiene un material a diferentes tipos de luz en función de sus características). Para comprobar su efectividad se mide el espectro de absorbancia de 100 longitudes de onda en 215 muestras de carne, cuyo contenido en grasa se obtiene también por análisis químico para poder comparar los resultados. El set de datos `meatspec` *del paquete* `faraway` *contiene toda la información.*

```
library(faraway)
data(meatspec)
dim(meatspec)
```

```
## [1] 215 101
```

El set de datos contiene 101 columnas. Las 100 primeras, nombradas como $V1$, ..., $V100$ recogen el valor de absorbancia para cada una de las 100 longitudes de onda analizadas, y la columna *fat* el contenido en grasa medido por técnicas químicas.

Para poder evaluar la capacidad predictiva del modelo, se dividen las observaciones disponibles en dos grupos: uno de entrenamiento para ajustar el modelo (80% de los datos) y uno de test (20% de los datos).

```
training <- meatspec[1:172, ]
test <- meatspec[173:215, ]
```

En primer lugar se ajusta un modelo incluyendo todas las longitudes de onda como predictores.

```
modelo <- lm(fat ~ ., data = training)
summary(modelo)
```

```
##
## Call:
## lm(formula = fat ~ ., data = training)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.09837 -0.35779  0.04555  0.38080  2.33860
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)      6.324      2.012   3.143 0.002439 **
## V1           12134.077   3659.798   3.316 0.001443 **
## V2          -12585.857   5971.891  -2.108 0.038605 *
## V3           -5107.556   9390.265  -0.544 0.588200
## V4           23880.493  17143.644   1.393 0.167977
## V5          -40509.555  22129.359  -1.831 0.071360 .
## V6           28469.416  19569.400   1.455 0.150134
## V7          -20901.082  12501.639  -1.672 0.098952 .
## V8            8369.465   7515.467   1.114 0.269193
## V9           -1539.328   5397.505  -0.285 0.776327
## V10           4706.267   7406.895   0.635 0.527217
## V11           7012.943  11720.620   0.598 0.551516
```

```
## V12           14891.444  20169.170   0.738 0.462749
## V13          -30963.902  26186.839  -1.182 0.240983
## V14           34338.612  22323.830   1.538 0.128444
## V15          -22235.237  13842.268  -1.606 0.112640
## V16           -7466.797   8558.172  -0.872 0.385890
## V17            6716.653   6561.805   1.024 0.309500
## V18           -2033.071   6741.330  -0.302 0.763851
## V19            8541.212   9419.998   0.907 0.367627
## V20           -1667.207  17300.433  -0.096 0.923500
## V21          -31972.494  24622.615  -1.299 0.198317
## V22           59526.389  27730.712   2.147 0.035244 *
## V23          -49241.388  23117.226  -2.130 0.036632 *
## V24           16184.597  16679.609   0.970 0.335180
## V25           12077.951  10751.912   1.123 0.265081
## V26          -12632.330   6774.573  -1.865 0.066361 .
## V27           -6298.837   7032.334  -0.896 0.373442
## V28           29625.988   9011.227   3.288 0.001573 **
## V29          -39374.835  13561.228  -2.903 0.004914 **
## V30           31251.427  18742.000   1.667 0.099829 .
## V31          -27238.189  21335.756  -1.277 0.205887
## V32           23009.543  19776.156   1.163 0.248522
## V33           -4584.373  14572.471  -0.315 0.753995
## V34           -5437.943  10344.728  -0.526 0.600754
## V35           -6128.931   8762.663  -0.699 0.486564
## V36            5599.605   6652.640   0.842 0.402776
## V37           -5569.160   6670.198  -0.835 0.406557
## V38              97.451   9291.480   0.010 0.991661
## V39           36021.407  12574.711   2.865 0.005488 **
## V40          -54273.400  17144.384  -3.166 0.002280 **
## V41           52084.876  21758.024   2.394 0.019318 *
## V42          -48458.089  23950.549  -2.023 0.046813 *
## V43           29334.488  20232.617   1.450 0.151500
## V44          -18282.834  13508.157  -1.353 0.180200
## V45           22110.934   9725.348   2.274 0.026020 *
## V46          -11735.692   6631.245  -1.770 0.081061 .
## V47            -514.521   3800.612  -0.135 0.892696
## V48            2551.480   6131.893   0.416 0.678592
## V49            3707.639   8970.401   0.413 0.680618
## V50          -25762.703  10934.783  -2.356 0.021236 *
## V51           46844.468  15367.852   3.048 0.003233 **
## V52          -47783.626  18069.344  -2.644 0.010065 *
## V53           26233.604  18822.491   1.394 0.167744
## V54              87.825  17403.836   0.005 0.995988
## V55           -8475.119  13232.005  -0.641 0.523908
## V56            3488.507   7228.428   0.483 0.630858
## V57           -1520.733   4988.093  -0.305 0.761355
## V58            2275.175   5495.630   0.414 0.680124
## V59           -5415.427   5721.475  -0.947 0.347099
## V60            7152.015   4754.317   1.504 0.136935
## V61           -4494.234   4512.937  -0.996 0.322702
```

```
## V62           3662.045   4811.634   0.761 0.449129
## V63          13993.987   7098.106   1.972 0.052563 .
## V64         -23252.133   8973.839  -2.591 0.011604 *
## V65           4373.731  10048.591   0.435 0.664695
## V66           4580.913  10146.146   0.451 0.653011
## V67           -837.676  10747.974  -0.078 0.938097
## V68          -7074.425  10852.430  -0.652 0.516587
## V69           9506.571   9739.256   0.976 0.332325
## V70          -2765.100   9519.031  -0.290 0.772295
## V71          -1125.135   8586.061  -0.131 0.896113
## V72          -7295.096   7489.488  -0.974 0.333341
## V73          17059.811   6522.093   2.616 0.010870 *
## V74          -9889.553   6543.945  -1.511 0.135162
## V75           -325.615   6125.973  -0.053 0.957759
## V76            782.219   5421.002   0.144 0.885677
## V77           8058.935   5793.416   1.391 0.168554
## V78         -15869.978   6448.208  -2.461 0.016282 *
## V79          21768.619   6435.678   3.382 0.001172 **
## V80         -28338.145   8180.874  -3.464 0.000906 ***
## V81           8523.317  10053.153   0.848 0.399384
## V82          22319.451  12098.046   1.845 0.069226 .
## V83         -17244.722  13991.685  -1.232 0.221829
## V84         -18325.836  14959.964  -1.225 0.224627
## V85          33345.457  13868.197   2.404 0.018808 *
## V86          -7955.157  14571.278  -0.546 0.586813
## V87          -7837.966  16141.553  -0.486 0.628762
## V88          -1815.552  17261.928  -0.105 0.916532
## V89            631.595  15684.751   0.040 0.967992
## V90          -2701.955  16187.612  -0.167 0.867911
## V91           4375.678  19400.005   0.226 0.822199
## V92          12925.188  16456.244   0.785 0.434816
## V93          -7441.235  12417.883  -0.599 0.550923
## V94          -2464.532  11815.234  -0.209 0.835366
## V95          -2090.635   9666.576  -0.216 0.829394
## V96          10912.352   9950.716   1.097 0.276505
## V97         -20331.405  11022.234  -1.845 0.069270 .
## V98           3948.443   8227.133   0.480 0.632753
## V99           6358.930   8652.372   0.735 0.464800
## V100          -263.365   4104.463  -0.064 0.949019
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.074 on 71 degrees of freedom
## Multiple R-squared:  0.997,  Adjusted R-squared:  0.9928
## F-statistic: 237.5 on 100 and 71 DF,  p-value: < 2.2e-16
```

El valor $R^2_{ajustado}$ obtenido es muy alto (0.9928) lo que indica que el modelo es capaz de predecir con gran exactitud el contenido en grasa de las observaciones con las que se ha entrenado. El hecho de que el modelo en conjunto sea significativo (p-value: < 2.2e-16), pero que muy pocos de los predictor lo sean a nivel individual, es un indicativo de una posible redundancia entre los predictores (colinealidad).

¿Cómo de bueno es el modelo prediciendo nuevas observaciones que no han participado en ajuste? Al tratarse de un modelo de regresión, la estimación del error de predicción se obtiene mediante el *Mean Square Error (MSE)*.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

```
# MSE empleando las observaciones de entrenamiento
training_mse <- mean((modelo$fitted.values - training$fat)^2)
training_mse
```

```
## [1] 0.4765372
```

```
# MSE empleando nuevas observaciones
predicciones <- predict(modelo, newdata = test)
test_mse <- mean((predicciones - test$fat)^2)
test_mse
```

```
## [1] 14.54659
```

Se observa que el modelo tiene un MSE muy bajo (0.48) cuando predice las mismas observaciones con las que se ha entrenado, pero 30 veces más alto (14.54) al predecir nuevas observaciones. Esto significa que el modelo no es útil, ya que el objetivo es aplicarlo para predecir el contenido en grasa de futuras muestras de carne. A este problema se le conoce como *overfitting*. Una de las causas por las que un modelo puede sufrir *overfitting* es la incorporación de predictores innecesarios, que no aportan información o que la información que aportan es redundante.

Se recurre en primer lugar a la selección de predictores mediante *stepwise selection* empleando el AIC como criterio de evaluación:

```
modelo_step_selection <- step(object = modelo, trace = FALSE)
# Número de predictores del modelo resultante
length(modelo_step_selection$coefficients)
```

```
## [1] 73
```

```
# Training-MSE
training_mse <- mean((modelo_step_selection$fitted.values - training$fat)^2)
training_mse
```

```
## [1] 0.5034001
```

```
# Test-MSE
predicciones <- predict(modelo_step_selection, newdata = test)
test_mse <- mean((predicciones - test$fat)^2)
test_mse
```

```
## [1] 12.88986
```

El proceso de *stepwise selection* devuelve como mejor modelo el formado por 73 de los 100 predictores disponibles. Al haber eliminado predictores del modelo, el *training MSE* siempre aumenta, en este caso de 0.48 a 0.05, pero el *test-MSE* se ha reducido a 12.88986.

Véase ahora el resultado si se ajusta el modelo empleando las componentes principales:

```
# Cálculo de componentes principales. Se excluye la columna con la variable
# respuesta *fat*
pca <- prcomp(training[, -101], scale. = TRUE)

# Se muestra la proporción de varianza explicada y acumulada de las 9
# primeras componentes
summary(pca)$importance[, 1:9]
```

```
##                            PC1      PC2       PC3       PC4        PC5
## Standard deviation     9.92492 1.043606 0.5357885 0.3312792 0.07898436
## Proportion of Variance 0.98504 0.010890 0.0028700 0.0011000 0.00006000
## Cumulative Proportion  0.98504 0.995930 0.9988000 0.9999000 0.99996000
##                               PC6        PC7        PC8         PC9
## Standard deviation     0.04974461 0.02700194 0.02059129 0.008603878
## Proportion of Variance 0.00002000 0.00001000 0.00000000 0.000000000
## Cumulative Proportion  0.99999000 0.99999000 1.00000000 1.000000000
```

El estudio de la proporción de varianza explicada muestra que la primera componente recoge la mayor parte de la información (98.5%), decayendo drásticamente la varianza en las sucesivas componentes.

Una vez obtenido el valor de las componentes para cada observación (*principal component scores*), puede ajustarse el modelo lineal empleando dichos valores junto con la variable respuesta que le corresponde a cada observación. Con la función `pcr()` del paquete `pls` se evita tener que codificar cada uno de los pasos intermedios.

Acorde a la proporción de varianza acumulada, emplear las 4 primeras componentes podría ser una buena elección, ya que en conjunto explican el 99.99100 % de varianza.

```
library(pls)
modelo_pcr <- pcr(formula = fat ~ ., data = training, scale. = TRUE, ncomp = 4)

# Test-MSE
predicciones <- predict(modelo_pcr, newdata = test, ncomp = 4)
test_mse <- mean((predicciones - test$fat)^2)
test_mse
```

```
## [1] 20.55699
```

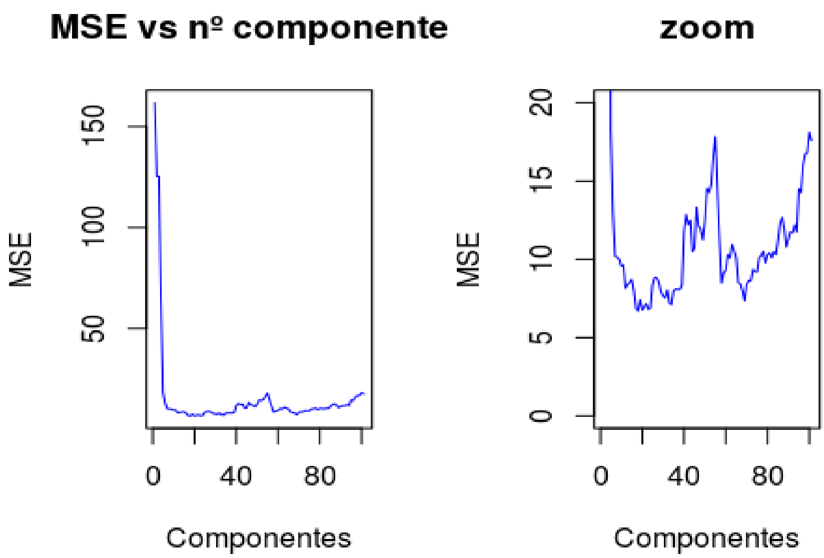
El *test-MSE* obtenido (20.56) para el modelo que emplea como predictores las 4 primeras componentes es mucho mayor que el obtenido con el modelo generado por *stepwise selection* (12.89) e incluso que el obtenido incluyendo todos los predictores (14.54659). Esto significa que, o bien el hecho de emplear componentes principales como predictores no es útil para este caso, o que el número de componentes incluido no es el adecuado.

La función `pcr()` incluye la posibilidad de recurrir a *cross validation* para identificar el número óptimo de componentes con el que se minimiza el *MSE*.

```
set.seed(123)
modelo_pcr <- pcr(formula = fat ~ ., data = training, scale. = TRUE, validation =
"CV")
modelo_pcr_CV <- MSEP(modelo_pcr, estimate = "CV")
which.min(modelo_pcr_CV$val)
```

```
## [1] 18
```

```
par(mfrow = c(1, 2))
plot(modelo_pcr_CV$val, main = "MSE vs nº componentes", type = "l", ylab = "MSE",
     col = "blue", xlab = "Componentes")
plot(modelo_pcr_CV$val, main = "zoom", type = "l", ylab = "MSE",
     xlab = "Componentes", col = "blue", ylim = c(0, 20))
```




```
# Test-MSE
predicciones <- predict(modelo_pcr, newdata = test, ncomp = 18)
test_mse <- mean((predicciones - test$fat)^2)
test_mse
```

```
## [1] 4.524698
```

El número óptimo de componentes principales identificado por *cross validation* es de 18. Empleando este número en la *PCR* se consigue reducir el *test-MSE* a 4.52, un valor muy por debajo del conseguido con los otros modelos.

PLS: PCA aplicado a regresión lineal

El método *Partial Least Squares (PLA)* es muy similar al *PCR* en cuanto que ambos emplean las componentes principales resultantes de un análisis *PCA* como predictores. La diferencia reside en que, mientras *PCR* ignora la variable respuesta *Y* para determinar las combinaciones lineales, *PLS* busca aquellas que, además de explicar la varianza observada, predicen *Y* lo mejor posible. Puede considerarse como una versión *supervised* de *PCR*.

La cuantificación del contenido en grasa de la carne pude hacerse mediante técnicas de analítica química, sin embargo, este proceso es costoso en tiempo y recursos. Una posible alternativa para reducir costes y optimizar tiempo es emplear un espectrofotómetro (instrumento capaz de detectar la absorbancia que tiene un material a diferentes tipos de luz en función de sus características). Para comprobar su efectividad se mide el espectro de *absorbancia de 100 longitudes de onda en 215 muestras de carne, cuyo contenido en grasa se obtiene también por análisis químico para poder comparar los resultados. El set de datos* `meatspec` *del paquete* `faraway` *contiene toda la información.*

```
library(faraway)
data(meatspec)
```

Igual que en el ejemplo anterior de *PCR*, para poder evaluar la capacidad predictiva del modelo, se dividen las observaciones disponibles en dos grupos: uno de entrenamiento para ajustar el modelo (80% de los datos) y uno de test (20% de los datos).

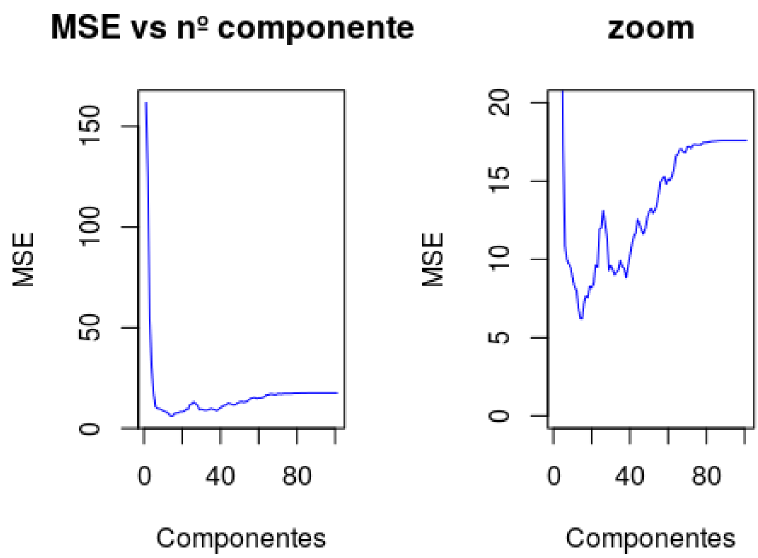
```
training <- meatspec[1:172, ]
test <- meatspec[173:215, ]
```

La función `plsr()` del paquete `pls` ajusta modelos por *Partial Least Squares* e incluye la posibilidad de recurrir a *cross validation* para identificar el número óptimo de componentes con el que se minimiza el *MSE*.

```
library(pls)
set.seed(123)
modelo_pls <- plsr(formula = fat ~ ., data = training, scale. = TRUE,
                   validation = "CV")
modelo_pls_CV <- MSEP(modelo_pls, estimate = "CV")
which.min(modelo_pls_CV$val)
```

```
## [1] 15
```

```
par(mfrow = c(1, 2))
plot(modelo_pls_CV$val, main = "MSE vs nº componentes", type = "l", ylab = "MSE",
     col = "blue", xlab = "Componentes")
plot(modelo_pls_CV$val, main = "zoom", type = "l", ylab = "MSE",
     xlab = "Componentes",  col = "blue", ylim = c(0, 20))
```




```
# Test-MSE
predicciones <- predict(modelo_pls, newdata = test, ncomp = 15)
test_mse <- mean((predicciones - test$fat)^2)
test_mse
```

```
## [1] 3.888104
```

Si se comparan los resultados obtenidos por *PCR* y *PLS* se observa que el número de componentes óptimo es inferior en *PLS*. Esto suele ser así ya que en el proceso de *PLS* se está incluyendo información adicional a través de la variable respuesta. Para este ejemplo, el método *PLS* consigue un *test-MSE* ligeramente inferior al obtenido por *PCR*.

Bibliografía

Introduction to Statistical Learning, Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani

A tutorial on Principal Components Analysis, Lindsay I Smith February 2002

Linear Models with R, Julian J.Faraway

https://en.wikipedia.org/wiki/Principal_component_analysis