Unsupervised Machine Learning

FSC

March 2, 2019

Contents

Aprendizaje no supervisado vs apredizaje supervisado

En data science solemos trabajar con un gran número de variables predictoras que potencialmente pueden tener una influencia en la variable que queremos predecir (outcome). Además, como ya habréis visto en las clases de aprendizaje supervisado es importante que los predictores no sean colineales por interpretabilidad del modelo y para mejorar su poder predictivo (RF,GLM,DL) o incluso para otros modelos como Naive Bayes se asume que las variables tienen que ser independientes entre si.

En algunas ocasiones ni siquiera tenemos muy claro cual es el mejor outcome a predecir para investigar un cierto problema. Por ejemplo, si queremos estimar la calidad de un producto podemos utilizar varias medidas para ello: el grado de satisfacción registrado por los consumidores medido por una encuesta, la evolución en las ventas durante un año, la evolución de las ventas de productos consumidores...

Por ello, un primer análisis exploratorio de los datos es importante. Y esto incluye variables predictoras y posibles outcomes. Hablamos de aprendizaje no supervisado cuando NO TENEMOS ETIQUETAS,

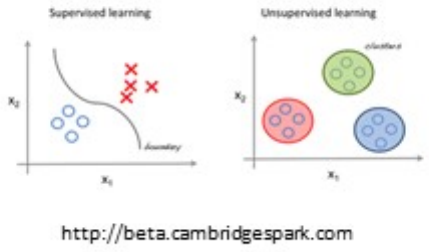


Figure 1: Supervised vs Unsupervised learning

simplemente buscamos identificar patrones en nuestros datos, tanto para las variables como para los individuos, de manera insesgada. No intentamos clasificar o predecir un outcome, solo encontrar patrones, como podemos ver en la Figura 1.

El material de esta clase ha sido elaborado con los recursos proporcionados en el libro: https://leanpub.com/dataanalysisforthelifesciences/

Distancia

Introducción

Un concepto clave en clasificacion no-supervisada es la distancia, ya que en general se trata de encontrar grupos de objetos que están "cerca" unos de otros, *whatever that means*. Por ejemplo, construyendo el árbol de la vida de los pokemons estamos implícitamente calculando distancias entre sus características

Para llegar a generar el árbol genealógico de los pokemons necesitamos crear una matriz en la que especifiquemos cómo de similares son unos de otros en función de varias características que hemos definido como: color, hábitat, categoría, tipo, aletas (SI/NO), cola (SI/NO), alas (SI/NO).

Los vectores de caracteristicas de cada pokemon quedarian asi:

```
pokemons=c("blastoise","charizard","primarina","delphox","pikachu")
blastoise=c("azul","laguna","marisco","agua","NO","SI","NO")
charizard=c("naranja","pradera","llama","fuego","NO","SI","SI")
primarina=c("azul","laguna","solista","agua","NO","SI","NO")
delphox=c("naranja","pradera","zorro","fuego","NO","SI","NO")
pikachu=c("amarillo","bosque","ratón","eléctrico","NO","SI","NO")
# all.pokemon=tibble(type=rep(pokemons,each=7),
#                        chars=c(blastoise,charizard,primarina,delphox,pikachu))
all.pokemon=data.frame(blastoise,charizard,primarina,delphox,pikachu)
rownames(all.pokemon)=c("color", "hábitat", "categoría", "tipo", "aletas", "cola", "alas")
```

Podemos generar un *heatmap* que resuma la información de los pokemon

```
library(ComplexHeatmap)
```

```
## Loading required package: grid
```

```
## ========================================
## ComplexHeatmap version 1.18.1
## Bioconductor page: http://bioconductor.org/packages/ComplexHeatmap/
```

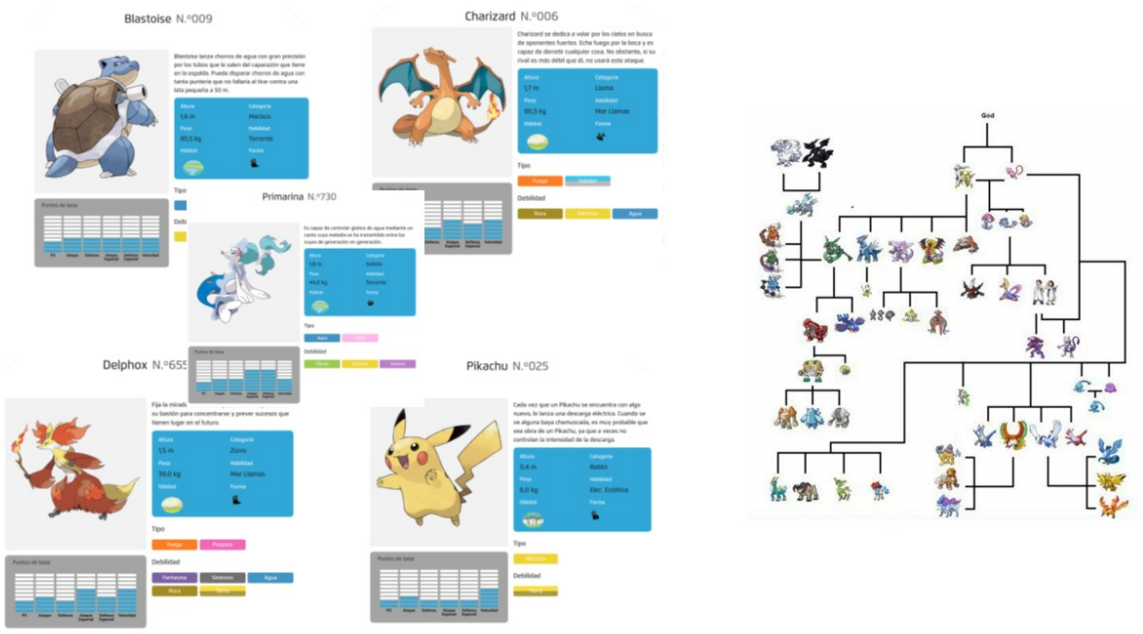
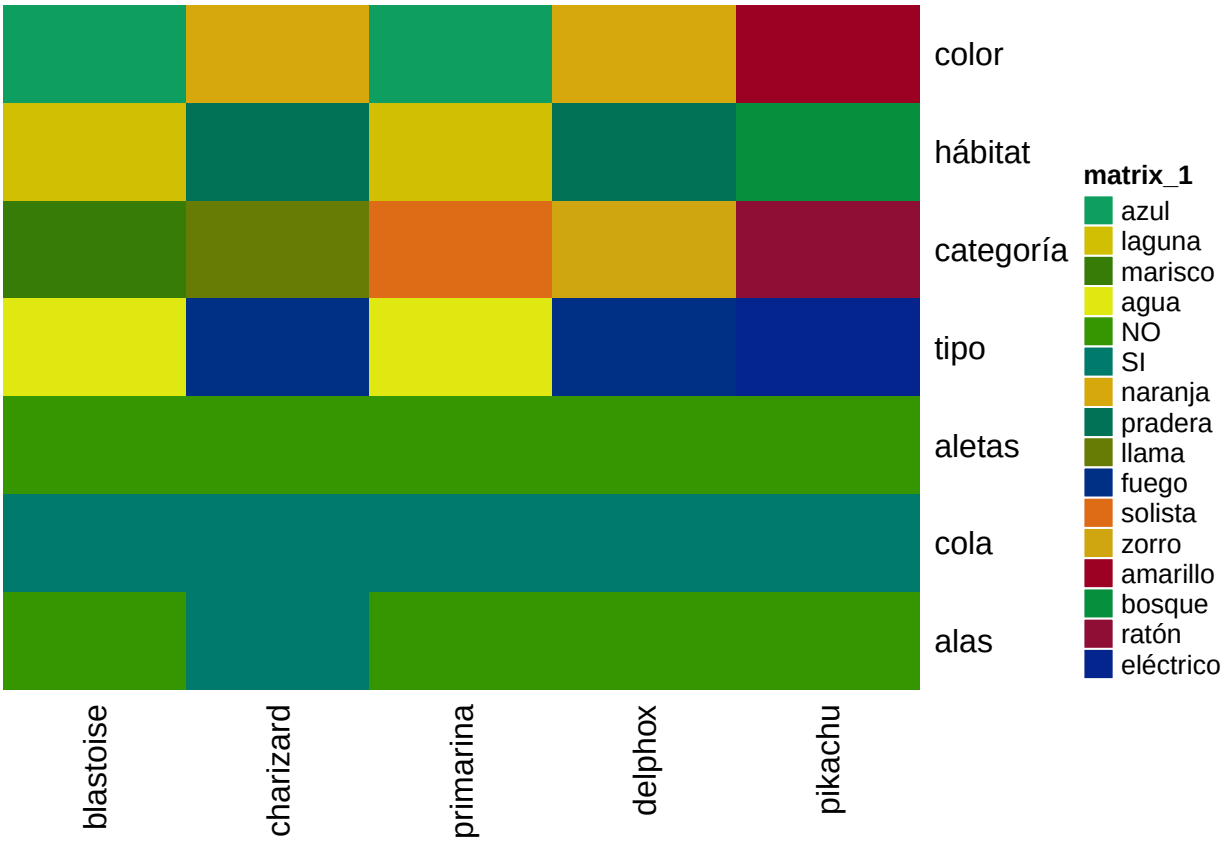


Figure 2: Clustering Pokemons.

```
## Github page: https://github.com/jokergoo/ComplexHeatmap
## Documentation: http://bioconductor.org/packages/ComplexHeatmap/
##
## If you use it in published research, please cite:
## Gu, Z. Complex heatmaps reveal patterns and correlations in multidimensional
##   genomic data. Bioinformatics 2016.
## ========================================
```

```
Heatmap(all.pokemon)
```

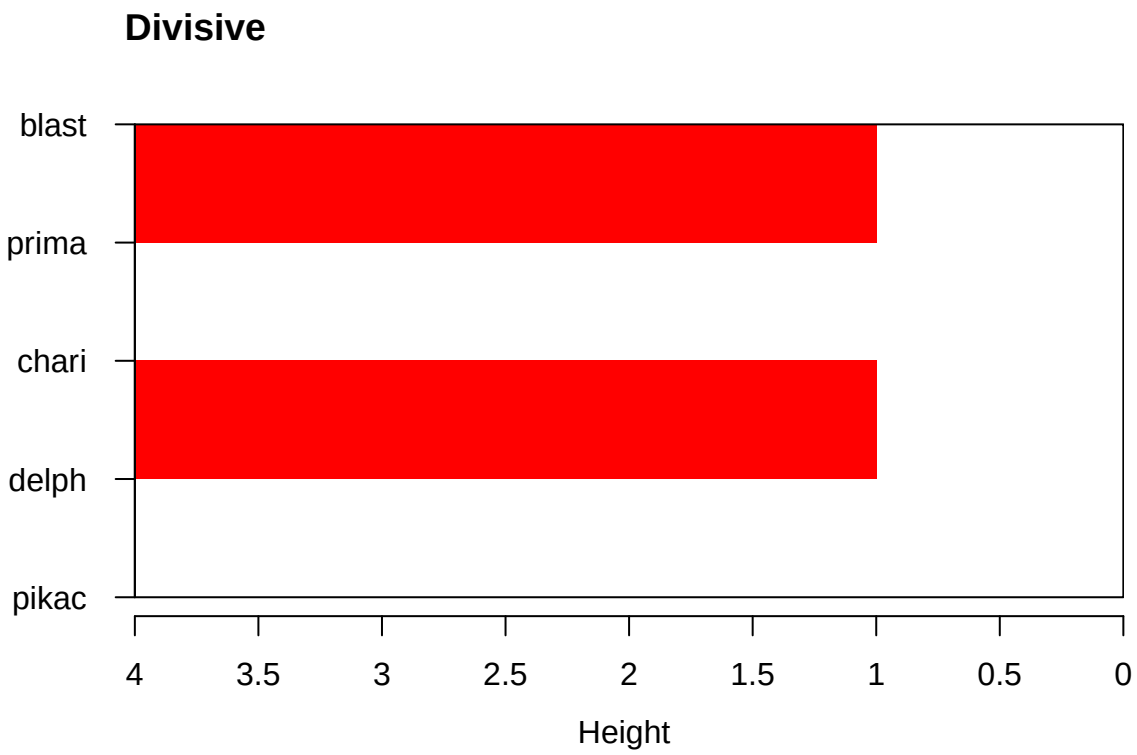


La distancia entre ellos vendrá dada por el número de características distintas que tienen:

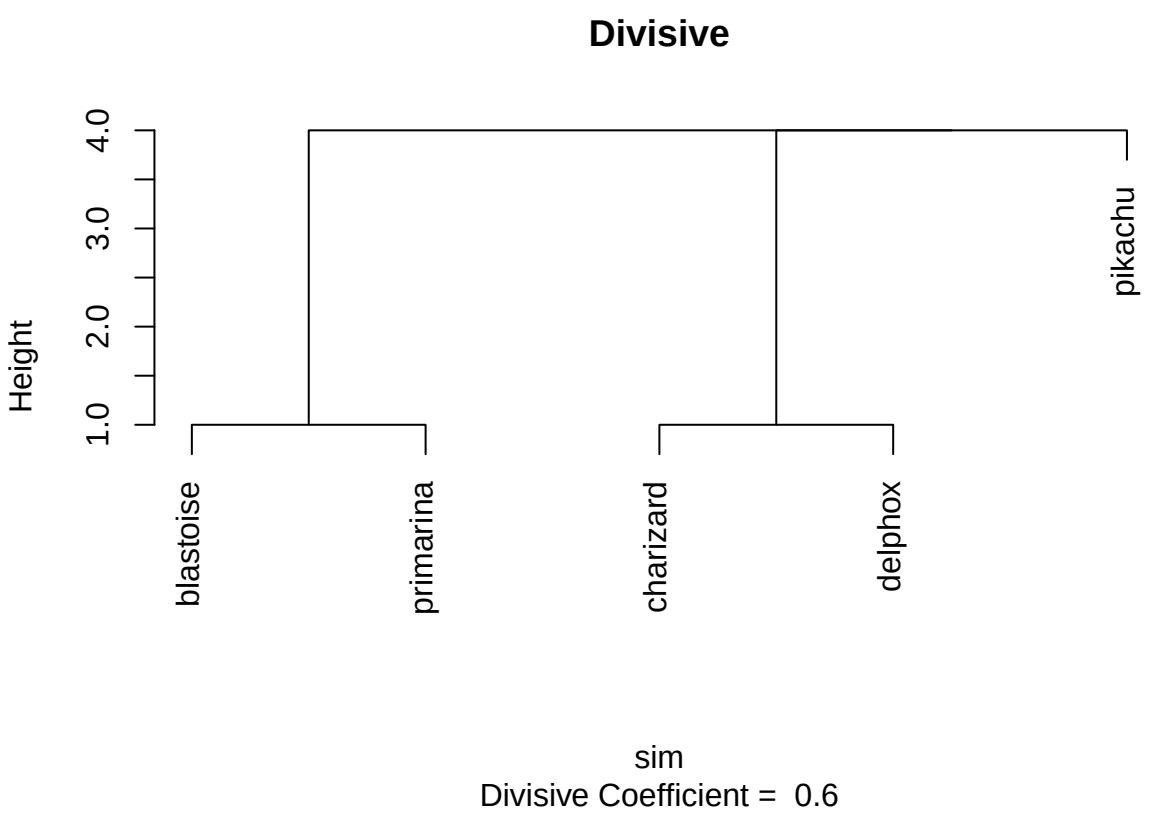
```
sim<-matrix(NA,5,5)
for (i in 1:5){
  for (j in 1:5){sim[i,j]<-length(setdiff(all.pokemon[,i],all.pokemon[,j]))}
}
colnames(sim)=pokemons
rownames(sim)=pokemons
```

Y podemos pintar cómo se organizan en función de esa matriz de distancias:

```
library(cluster)
divisive.clust <- diana(sim,
                  diss = TRUE, keep.diss = TRUE)
plot(divisive.clust, main = "Divisive")
```



Divisive Coefficient = 0.6



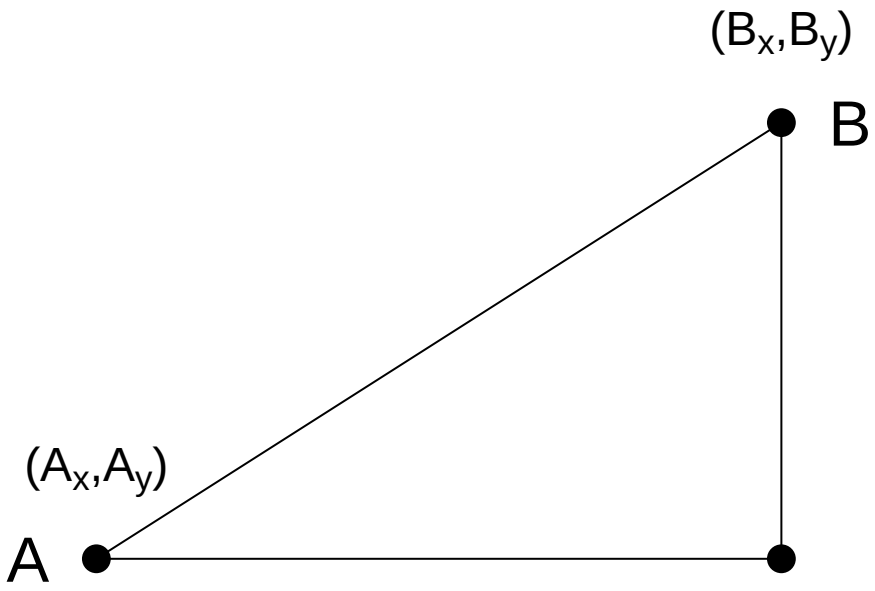
En un problema de clasificación supervisada las características de cada pokemon serían las *features*, *predictores* o *variables independientes* en terminología estadística tradicional, mientras que el tipo de pokemon sería la etiqueta *label*, *outcome* o *variable dependiente* Aquí lo único que nos interesa es saber qué pokemons forman parte de las mismas subfamilias o, de manera más general, *grupos* o *clústers*

Vamos a revisar brevemente algunos de los conceptos básicos de distancia:

Euclidean Distance

Si tenemos dos puntos A and B en un espacio cartesiano de 2 dimensiones:

```
## Warning: package 'rafalib' was built under R version 3.5.2
```



La distancia euclídea entre ellos es:

$$\sqrt{(A_x - B_x)^2 + (A_y - B_y)^2}$$

Distancia en más dimensiones

Vamos a utilizar un dataset con medidas de expresión génica para 22,215 genes en 189 muestras. Podemos bajarnos directamente el dataset:

Cada muestra representa la expresión de todos esos genes en 18 tejidos distintos. Para cada tejido tenemos un número diferente de pacientes.

```
library(tissuesGeneExpression)
data(tissuesGeneExpression)
dim(e) ##e contains the expression data
```

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

```
table(tissue) ##tissue[i] tells us what tissue is represented by e[,i]
```

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

Queremos descubrir qué muestras se parecen mas. Se parecerán más por individuo? O por tejido? También puede ser interesante encontrar genes con una expresión parecida a lo largo de los distintos tejidos o en ciertos

tejidos en particular.

Ahora no estamos en un plano cartesiano como antes, porque cada punto puede ser:

- una muestra: y por tanto tendría una dimension 22215, matemáticamente escrito

$(Y_{1,i}, \ldots, Y_{22215,i})^\top$

- un gen: que tendría una dimensión 189, u 8 si estamos considerando sólo la media de la expresión en cada tejido, matemáticamente escrito:

$(Y_{g,1}, \ldots, Y_{g,189})^\top$

Tenemos los puntos definidos. La distancia Euclídea entre dos muestras i y j si que se define como:

$$\text{dist}(i,j) = \sqrt{\sum_{g=1}^{22215} (Y_{g,i} - Y_{g,j})^2}$$

Y la distancia entre dos genes h y g será :

$$\text{dist}(h,g) = \sqrt{\sum_{i=1}^{189} (Y_{h,i} - Y_{g,i})^2}$$

Distancia usando matrices

La distancia entre dos muestras i y j se define como:

$$\text{dist}(i,j) = (\mathbf{Y}_i - \mathbf{Y}_j)^\top (\mathbf{Y}_i - \mathbf{Y}_j)$$

donde $\mathbf{Y}_i$ y $\mathbf{Y}_j$ para las columnas i y j.

Álgebra de matrices es la forma más eficiente computacionalmente para ejecutar los métodos de clustering que veremos en estas clases.

Examples

Calculemos la distancia entre las muestras 1 y 2 (riñones) de la matriz de expresión y a la muestra 87 (colon).

```
x <- e[,1]
y <- e[,2]
z <- e[,87]
sqrt(sum((x-y)^2))
```

```
## [1] 85.8546
```

```
sqrt(sum((x-z)^2))
```

```
## [1] 122.8919
```

Cómo cabría esperar los riñones están más cerca entre ellos que del colon. Utilizando álgebra de matrices:

```
sqrt( crossprod(x-y) )
```

```
##          [,1]
## [1,] 85.8546
```

```
sqrt( crossprod(x-z) )
```

```
##           [,1]
## [1,] 122.8919
```

Si queremos calcular todas las distancias a la vez usaremos la función `dist`. Esta función computa por defecto la distancia entre filas. Por lo tanto tenemos que transponer la matriz (darle la vuelta) para que calcule la distancia entre columnas.

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

```
## [1] "dist"
```

El output es un objeto de tipo dist que necesitamos forzar en una matriz para acceder a sus filas y columnas:

```
as.matrix(d)[1,2]
```

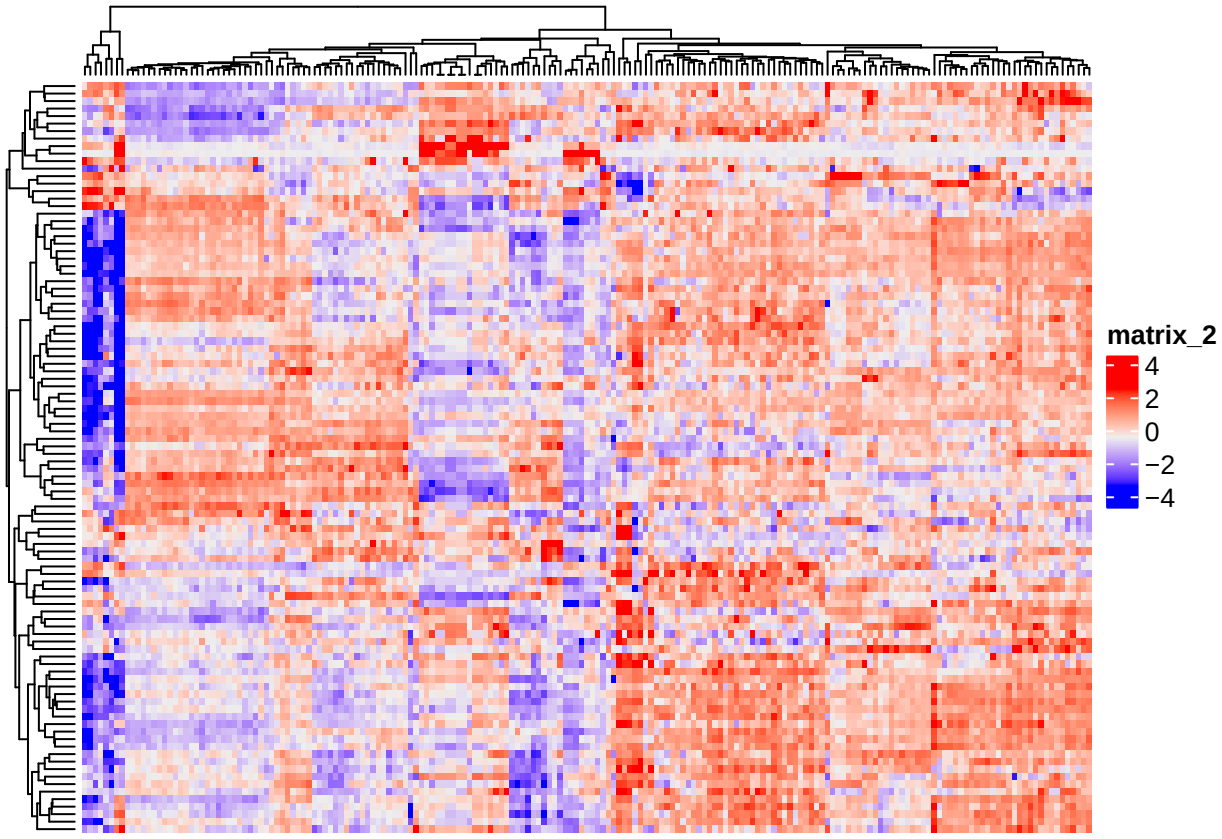
```
## [1] 85.8546
```

```
as.matrix(d)[1,87]
```

```
## [1] 122.8919
```

Anticipando lo que viene después así quedaría el clustering jerárquico de la matriz de expresión génica (restringiendonos a los 100 primeros genes por problemas de memoria):

```
library(ComplexHeatmap)
Heatmap(t(scale(t(e[1:100,]))),show_row_names = F,show_column_names = F)
```



IMPORTANTE: Si calculas dist(e) calculará la distancia dos a dos de las 22500 filas, lo cual puede romper tu sesión de R

Técnicas de reducción de la dimensionalidad

Motivación

Como ya hemos visto anteriormente, la visualizacion de los datos es esencial en datascience. El problema es que no podemos ver nada en una dimensión superior a 3D, lo cual equivaldria a tres variables. Imaginemos un estudio de calidad de producto a 1000 individuos en el que tenemos medidas las respuestas a 100 preguntas de un cuestionario. Para comparar esta información dos a dos si no estamos seguros de que las relaciones entre los datos son lineales (en cuyo caso la correlación sería completamente meaningless) necesitaríamos crear 4950 scatterplots y explorarlos, lo cual es por completo imposible. Y crear un sólo plot es del todo imposible porque cada punto, que representa las respuestas a una pregunta, es 1000-dimensional (el número de individuos que han respondido).

Por ello necesitamos reducir la dimensionalidad de nuestros datos a tan sólo 2 o 3 dimensiones, que es aquello que la mente humana puede percibir, con la menor pérdida de información posible. Una de las cosas más importante a preservar es la distancia entre las variables. Singular Value Decomposition (SVD) es el método numérico matemático que utilizaremos en Principal Component Analysis, una de las formas más conocidas de reducción de la dimensionalidad. Esta técnica se basa sin embargo en relaciones lineales entre variables que en datasets de gran dimensionalidad y heterogeneidad puede no ser lo más apropiado. Para estos otros casos revisaremos una nueva técnica llamada t-sne.

Pero antes de eso, veamos un ejemplo

Ejemplo: 2D to 1D

Queremos explorar la altura de hermanos gemelos. Para cada pareja de gemelos simulamos la desviación con respecto a la media de la altura de cada gemelo. Es decir, si uno mide 172 cm y el otro 168, tendríamos dos medidas (2,-2) que indicarían la desviación en cm para cada una de las mediciones respecto a la media 170.

Este tipo de distribución se llama normal multivariante y en realidad son dos normales correlacionadas. La correlación entre ambas variables es de 0.95.

El vector 1 representa la desviación respecto a la media del gemelo 1 y el vector 2, la desviación respecto a la media del gemelo 2. Hemos marcado en naranja la diferencia en altura para los dos gemelos de la pareja 1 y de la pareja 2 y estamos interesados en conocer la distancia entre estos dos puntos. Cómo la calculamos? Es muy fácil usando R.

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

```
## [1] 1.140897
```

Pero, que pasaría si dibujar en 2 dimensiones fuera muy complejo? Cómo podríamos sumarizar la información de las dos dimensiones en una sola, preservando la distancia entre los puntos?

Imaginemos que hay una línea que une dos puntos cualquiera del scatterplot. El tamaño de la línea es la distancia entre ambos puntos. Como esas líneas son todas pequeñas desviaciones de la diagonal podemos probar a rotar el gráfico de manera que la diagonal se convierta en el eje de las x. Así tendríamos un plot MA que representa la diferencia entre los valores (M) en el eje Y y la media de los valores en el eje x (A)

```
z1 = (y[1,]+y[2,])/2 #the sum
z2 = (y[1,]-y[2,])   #the difference
z = rbind( z1, z2) #matrix now same dimensions as y
thelim <- c(-3,3)
mypar(1,2)
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)
```

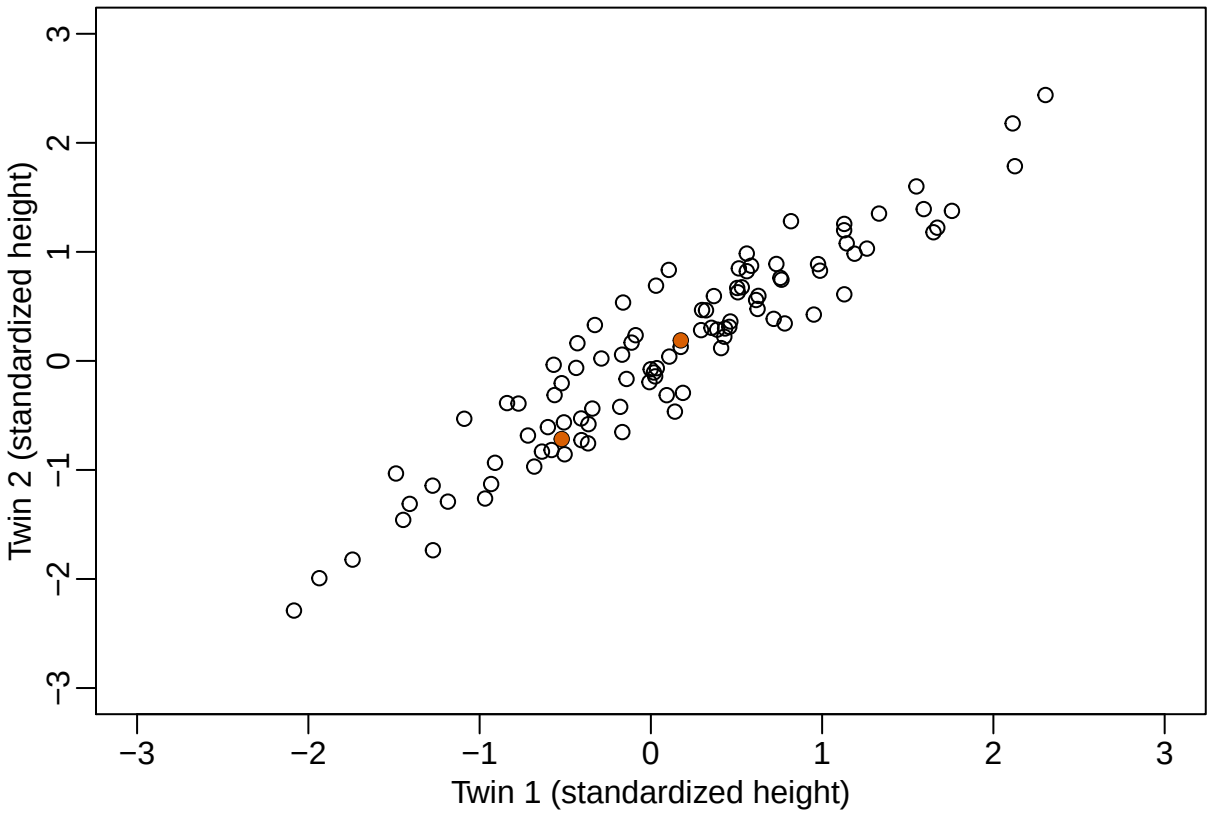
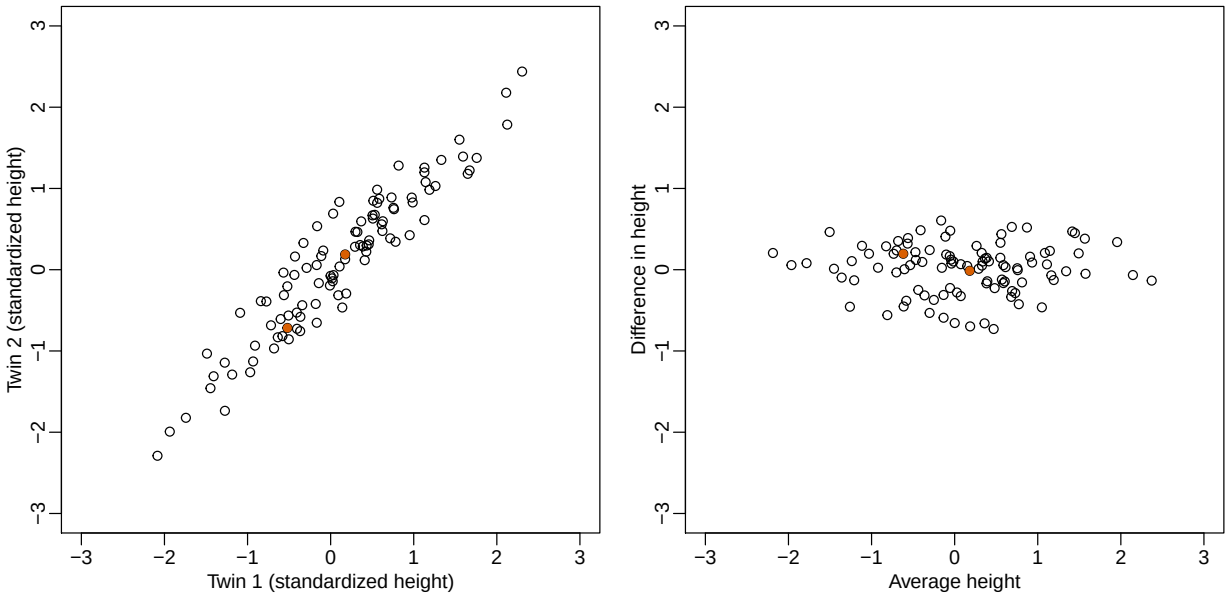


Figure 3: Simulated twin pair heights.

```r
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)
```



La nueva matriz z que también tiene dos dimensiones (difference & average) en realidad la hemos obtenido multiplicando la matriz y original por una matriz A fija de la forma:

$$A = \begin{pmatrix} 1/2 & 1/2 \\ 1 & -1 \end{pmatrix} \implies z = Ay$$

Podemos volver a transformar los datos a su forma original multiplicando por A^{-1}:

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

Rotaciones

Habeis visto que hemos realizado una rotación de los datos. Esto en matemáticas se hace multiplicando matrices de unas ciertas características. Lo bueno es que se mantiene la distancia entre los puntos, que es lo que realmente nos interesa de este gráfico. Otra transformacion adicional de los datos que tampoco modifica la distancia entre puntos es un re-escalado. Esto se hace multiplicando por un escalar (valor único). La desviación estandard de las nuevas variables M y A son:

$$\text{sd}[Z_1] = \text{sd}[(Y_1 + Y_2)/2] = \frac{1}{\sqrt{2}}\sigma \text{ and } \text{sd}[Z_2] = \text{sd}[Y_1 - Y_2] = \sqrt{2}\sigma$$

Por lo que si cambiamos la transformación anterior a:

$$A = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

la desviación standard de cada columna de Y es igual a la varianza de las columnas de Z. Como para A se cumple que $A^{-1}A = I$ decimos que A es una matriz *ortogonal*, lo que garantiza que las distancias se preserven:

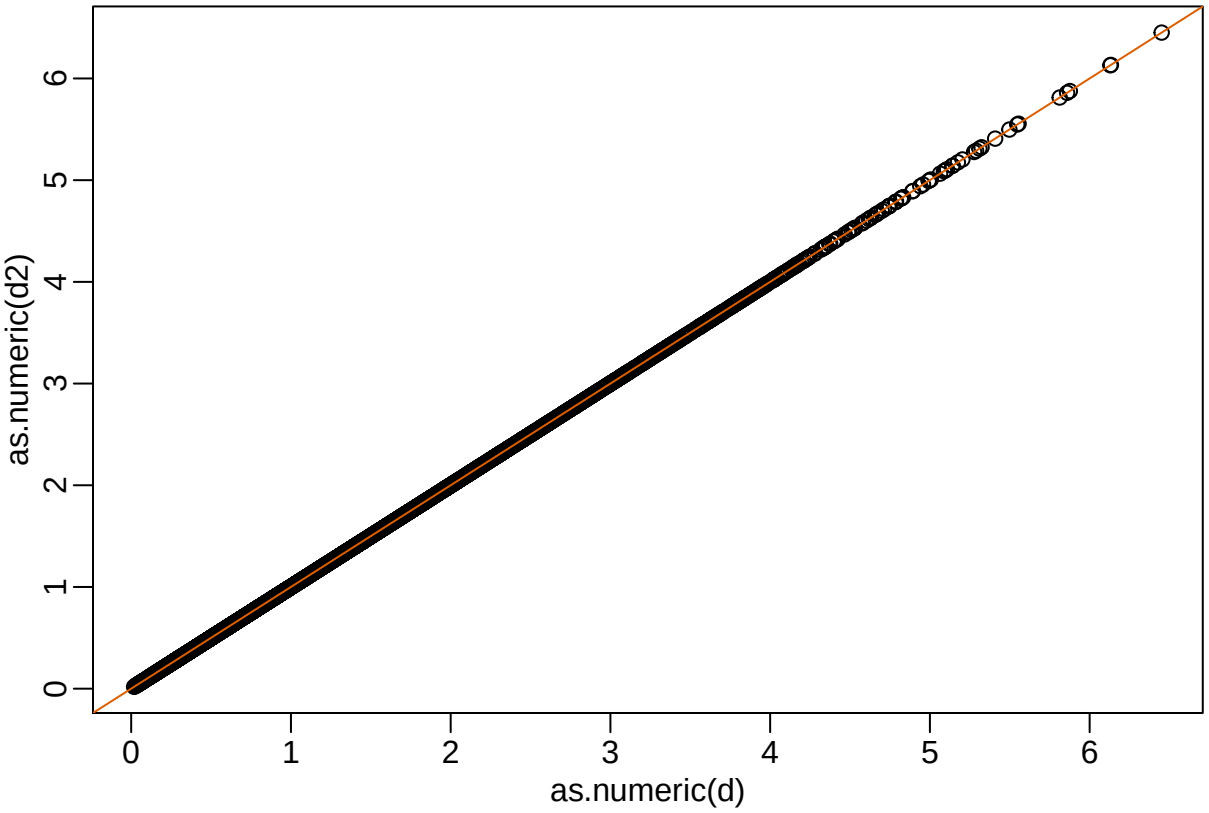


Figure 4: Distance computed from original data and after rotation is the same.

```
A <- 1/sqrt(2)*matrix(c(1,1,1,-1),2,2)
z <- A%*%y
d <- dist(t(y))
d2 <- dist(t(z))
mypar(1,1)
plot(as.numeric(d),as.numeric(d2)) #as.numeric turns distances into long vector
abline(0,1,col=2)
```

Esta transformación se llama *rotacion* de `y`.

```
mypar(1,2)
thelim <- c(-3,3)
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)
```

Si recordais hicimos esto para intentar tener una forma de ver los datos en una sola dimension preservando la caracteristica que mas nos interesa: la distancia entre los puntos. De manera intuitiva sugerimos esta transformacion porque vimos que mas o menos las lineas que unían dos puntos cualquiera eran pequeñas desviaciones de la identidad. Vamos a prescindir de la segunda dimensión de `z` que era A y vamos a volver a computar las distancias:

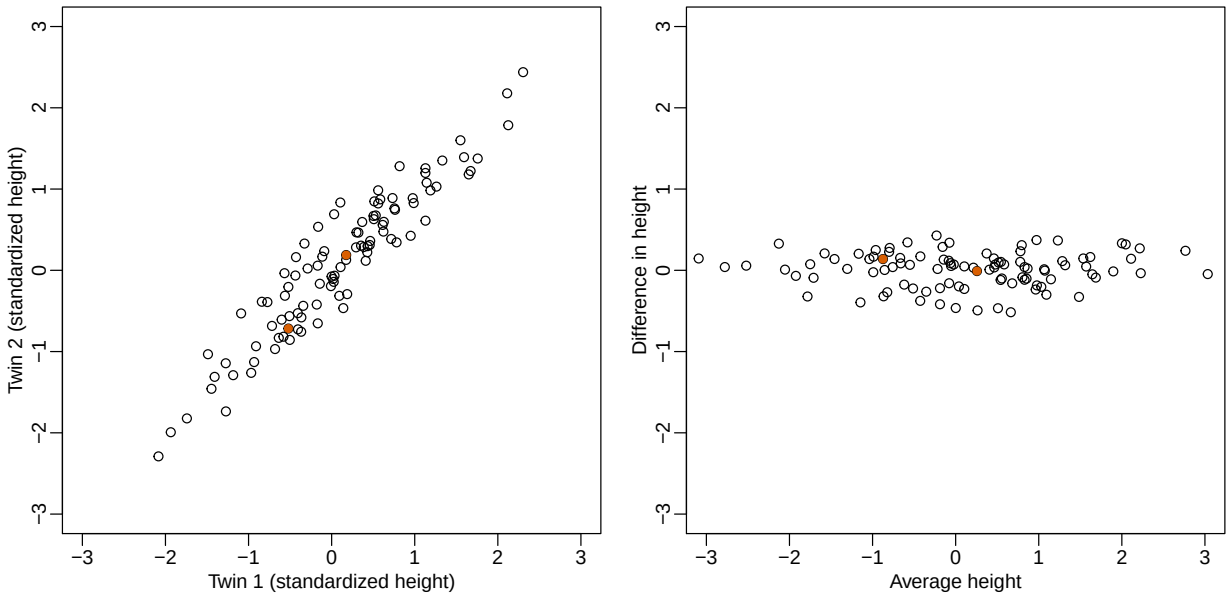


Figure 5: Twin height scatterplot (left) and after rotation (right).

```
d3 = dist(z[1,]) ##distance computed using just first dimension
mypar(1,1)
plot(as.numeric(d),as.numeric(d3))
abline(0,1)
```

La distancia calculada sólo sobre M nos proporciona una muy buena aproximación habiendo reducido las dimensiones de 2 a 1. La primera dimensión del dato transformado es su primera *componente principal*. Así funciona el *análisis de componentes principales* **(PCA)** y su generalización *Singular Value Decomposition* **(SVD)**

Principal Component Analysis

Example: Twin heights

Vamos a generar ahora un dataset de alturas distinto al anterior:

```
set.seed(1988)
library(MASS)
n <- 100
x <- rbind(mvrnorm(n / 2, c(69, 69), matrix(c(9, 9 * 0.9, 9 * 0.92, 9 * 1), 2, 2)),
           mvrnorm(n / 2, c(55, 55), matrix(c(9, 9 * 0.9, 9 * 0.92, 9 * 1), 2, 2)))
```

Hemos simulado una distribución normal bivariante, lo cual puede verse con un gráfico:

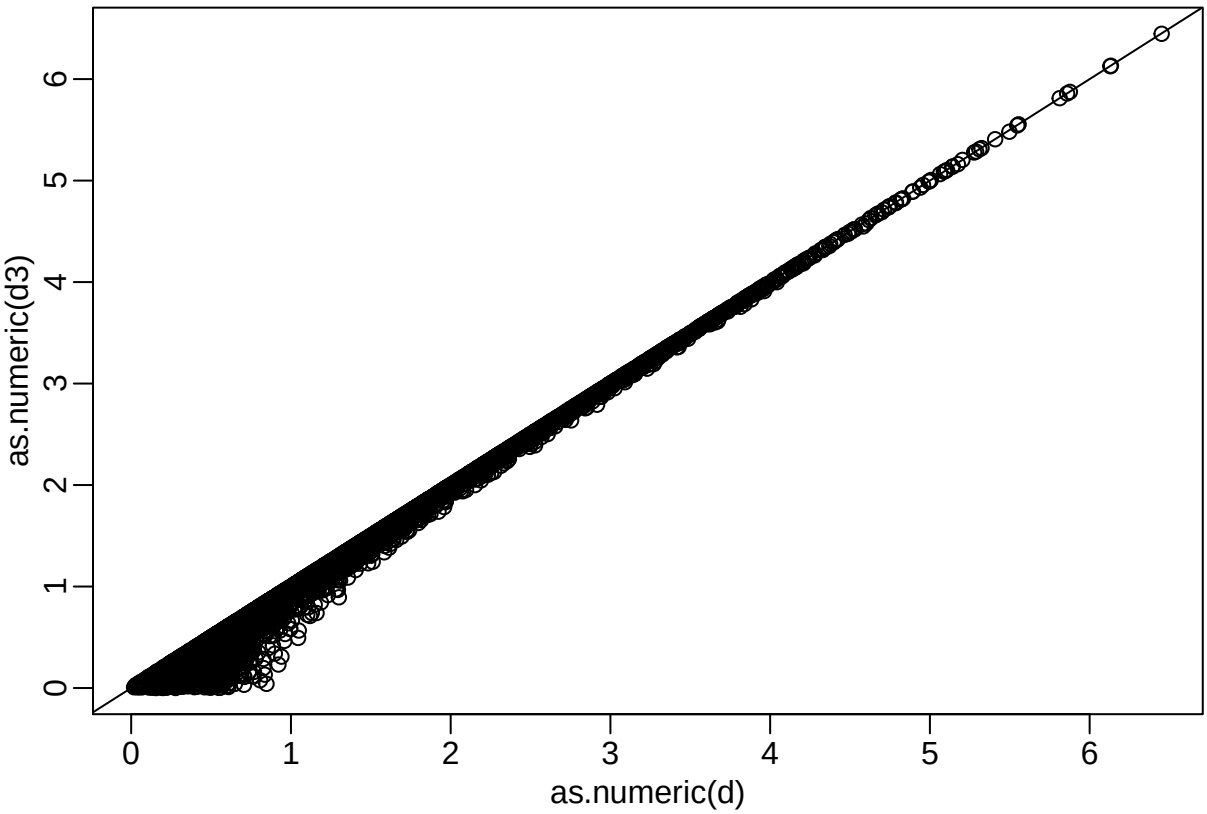
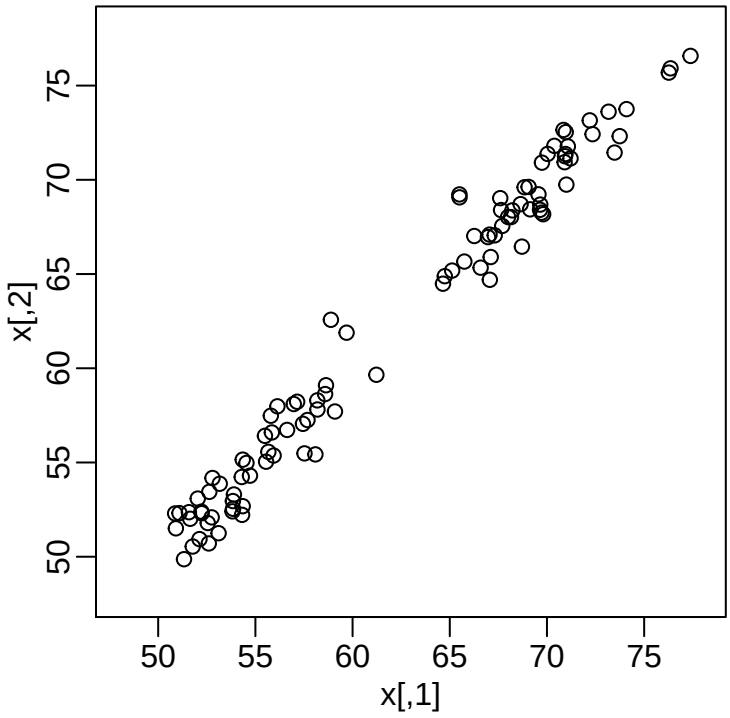
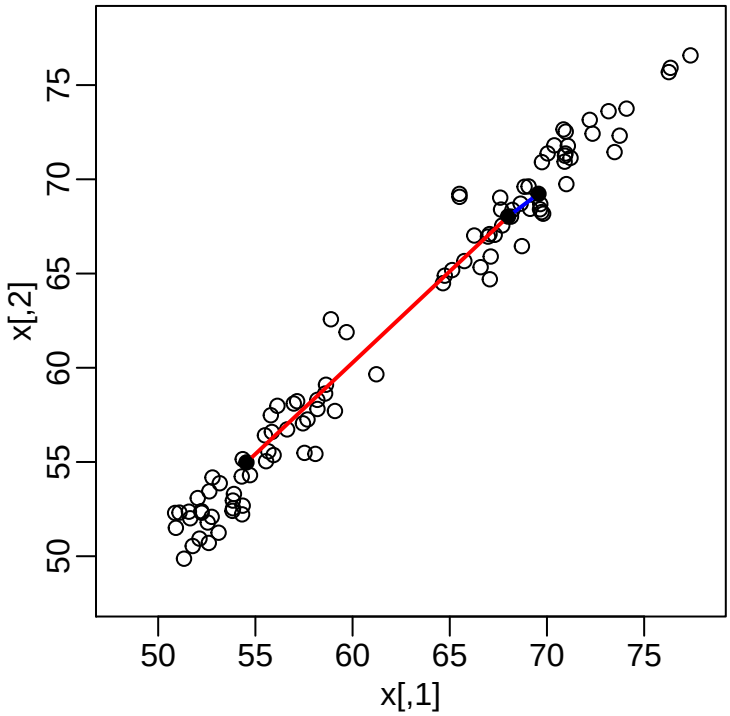


Figure 6: Distance computed with just one dimension after rotation versus actual distance.



Nuestras variables son otra vez vectores de dimension 2, con dos alturas. Vamos a ver si con sólo una dimension somos capaces de identificar las dos clases que tenemos: adultos y niños.



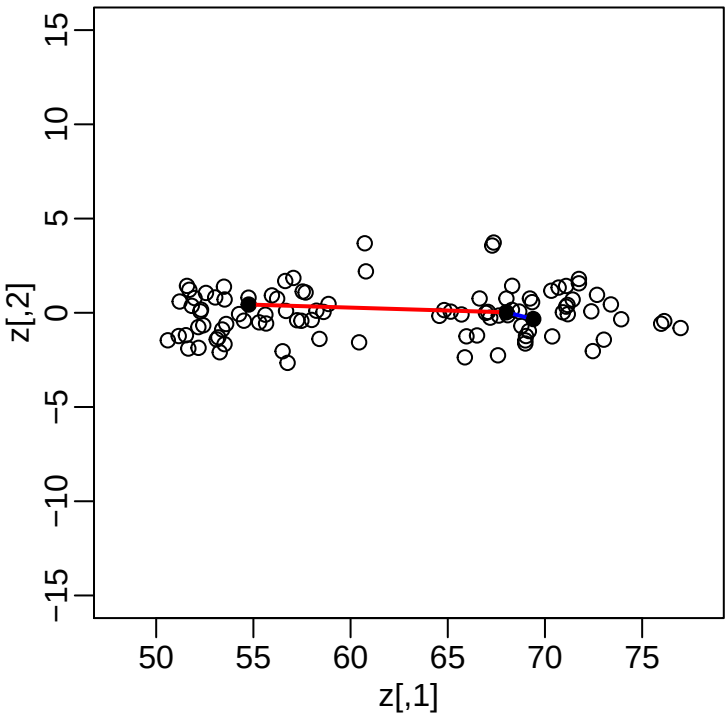

```
## [1] 1.975417
```

```
## [1] 18.74467
```

La linea roja nos marca la distancia entre los dos puntos azules. Usando la transformación que ya conocemos:

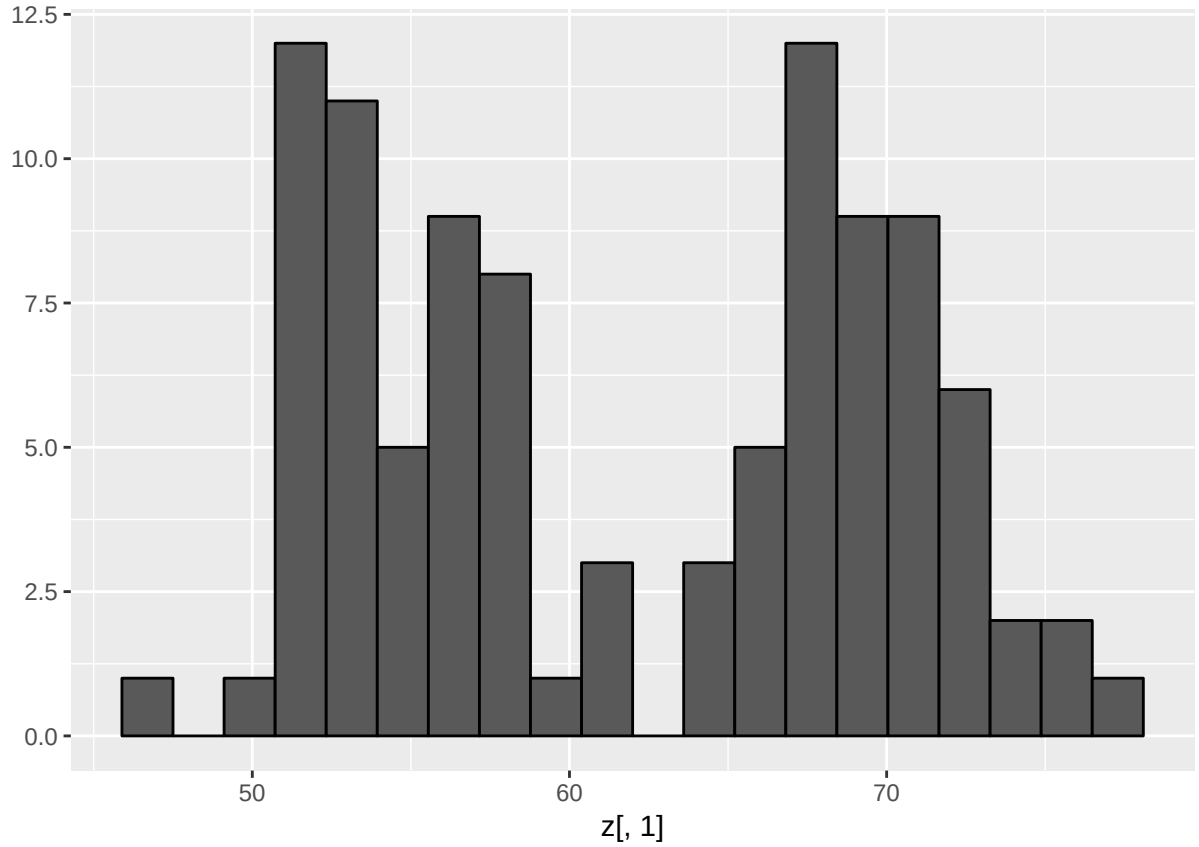
```
z  <- cbind((x[,2] + x[,1])/2,  x[,2] - x[,1])
```

Vemos que la media es suficiente para explicar los datos y que la distancia permanece igual:



Utilizando la misma transformación que en el primer ejemplo podemos sumarizar dos medidas en una. Si exploramosla primera dimensión de Z vemos:

```
library(tidyverse)
qplot(z[,1], bins = 20, color = I("black"))
```



Lo que nos muestra que hay dos tipos de medidas: de niños y de adultos. La razon por la que todo salió tan bien fue que las variables de X estaban muy correlacionadas:

```
cor(x[,1], x[,2])
```

```
## [1] 0.9880317
```

y la transformación produjo dos variables no correlacionadas:

```
cor(z[,1], z[,2])
```

```
## [1] 0.08762082
```

Esto es algo relativamente comun en problemas de machine learning. En estos casos PCA puede usarse como procedimiento de feature selection.

‘

Fundamentos

La variabilidad total del dataset de la altura de gemelos sería la suma de la suma de cuadrados de cada columna. Si cada columna está centrada en 0 la suma de los cuadrados equivaldria a a suma de las varianzas de cada columna.

$$v_1 + v_2, \text{ with } v_1 = \frac{1}{N}\sum_{i=1}^{N} X_{i,1}^2 \text{ and } v_2 = \frac{1}{N}\sum_{i=1}^{N} X_{i,2}^2$$

los calculamos con:

```r
colMeans(x^2)
```

```
## [1] 3903.942 3901.795
```

Es interesante ver que la variable Z recoge el 99% de la variabilidad del dataset mientras que para x ambas variables tienen mas o menos la misma variabilidadof the variability is included in just the first dimensions:

```r
colMeans(z^2)
```

```
## [1] 3902.481221    1.550253
```

```r
v <- colMeans(z^2)
v/sum(v)
```

```
## [1] 0.9996029096 0.0003970904
```

La *primera componente principal* de una matriz X is la transformacion lineal ortogonal (su inversa es su traspuesta) que maximiza la variabilidad. La función `prcomp` produce exactamente esto:

```r
pca <- prcomp(x)
pca$rotation
```

```
##             PC1        PC2
## [1,] -0.7022354  0.7119448
## [2,] -0.7119448 -0.7022354
```

Es una matriz ortogonal (su traspuesta es igual a su inversa)

```r
library(matlib)
```

```
## Warning: package 'matlib' was built under R version 3.5.2
```

```r
t(pca$rotation)
```

```
##           [,1]       [,2]
## PC1 -0.7022354 -0.7119448
## PC2  0.7119448 -0.7022354
```

```r
inv(pca$rotation)
```

```
##
## [1,] -0.7022354 -0.7119448
## [2,]  0.7119448 -0.7022354
```

Siempre podemos encontrar esa transformacion lineal, para cualquier numero de dimensiones.

Para una matriz multidimensional X con p columnas podemos encontrar una transformación que crea Z que mantiene la distancia entre filas pero para la que las nuevas columnas tienen una varianza decreciente.

Interpretación de las componentes principales

El vector ortogonal que maximiza la suma de cuadrados:

$$(\mathbf{u}_1^\top \mathbf{Y})^\top (\mathbf{u}_1^\top \mathbf{Y})$$

$\mathbf{u}_1^\top \mathbf{Y}$ es la *primera componente principal*. Los *weights* $\mathbf{u}$ que se usan para obtenerla son los *loadings*. Si pensáramos en términos de rotaciones la primera componente sería la *direction* , que son las nuevas coordenadas.

Ahora repetiríamos el ejercicio de maximización para la suma de los cuadrados de los residuos:

$$\mathbf{r} = \mathbf{Y} - \mathbf{u}_1^\top \mathbf{Y}\mathbf{v}_1$$

La *segunda componente principal* es un vector de la forma:

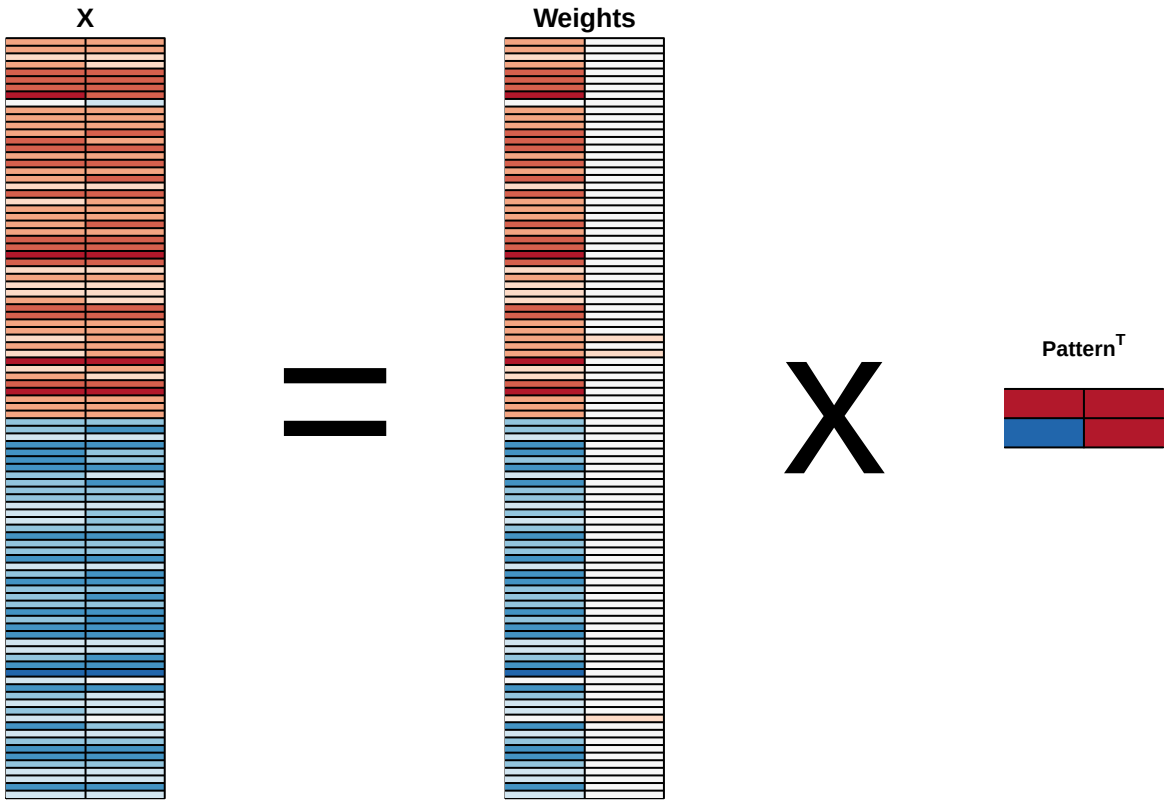
$$\mathbf{v}_2^\top \mathbf{v}_2 = 1$$

$$\mathbf{v}_2^\top \mathbf{v}_1 = 0$$

que maximiza $(\mathbf{r}\mathbf{v}_2)^\top \mathbf{r}\mathbf{v}_2$.

Si Y es $N \times m$ podemos encontrar m componentes principales.

Es muy interesante ver cómo esa primera componente sumariza los datos. Rojo representa valores positivos y azul negativos:

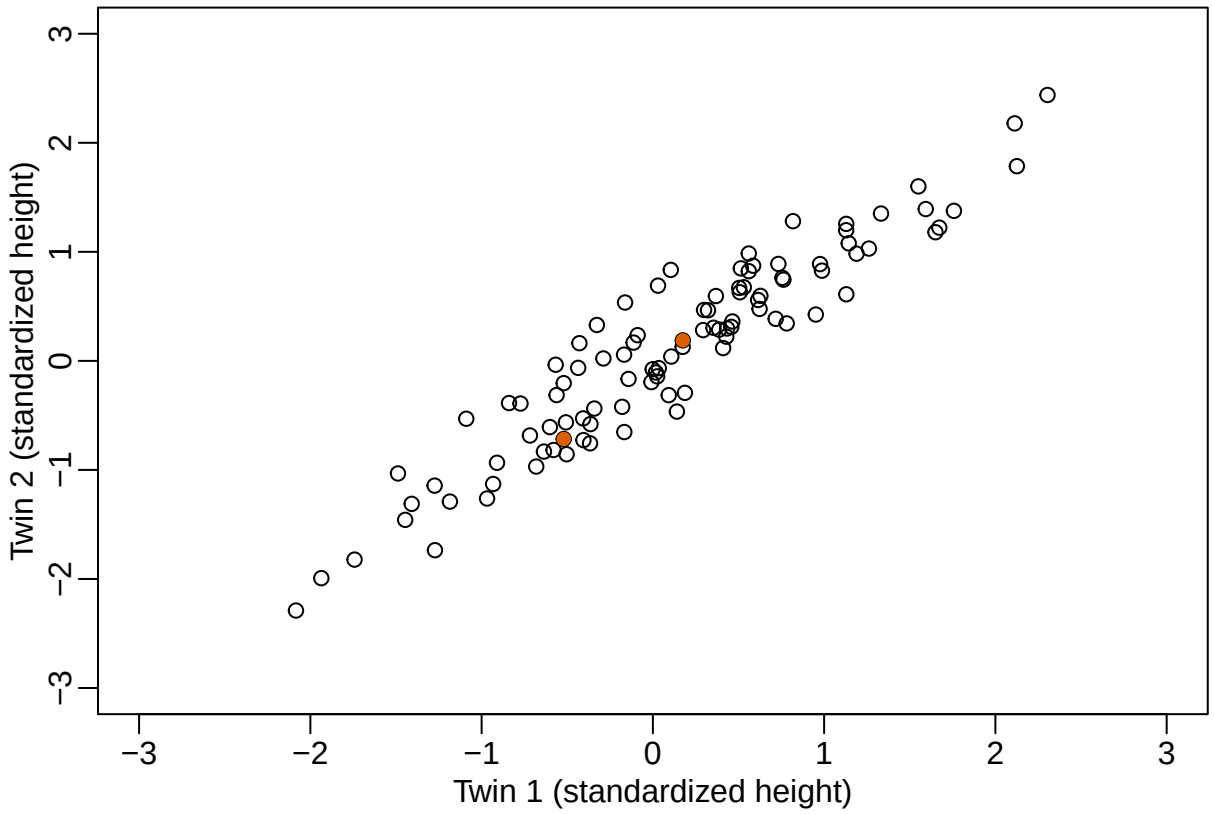


prcomp

R tiene una función específica para hacer esto.

```
library(rafalib)
library(MASS)
n <- 100
set.seed(1)
Y=t(mvrnorm(n,c(0,0), matrix(c(1,0.95,0.95,1),2,2)))
mypar()
```

```
thelim <- c(-3,3)
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)
```




```
pc<-prcomp(t(Y))
```

que, si los datos están estandarizados produce el mismo resultado que SVD.

```
s <- svd( Y - rowMeans(Y) )
mypar(1,2)
for(i in 1:nrow(Y) ){
  plot(pc$x[,i], s$d[i]*s$v[,i])
}
```

Podemos encontrar los pesos con los que se combinan las variables haciendo:

```
pc$rotation
```

```
##            PC1        PC2
## [1,] 0.7072304  0.7069831
## [2,] 0.7069831 -0.7072304
```

y la varianza explicada en:

```
pc$sdev
```

```
## [1] 1.2542672 0.2141882
```

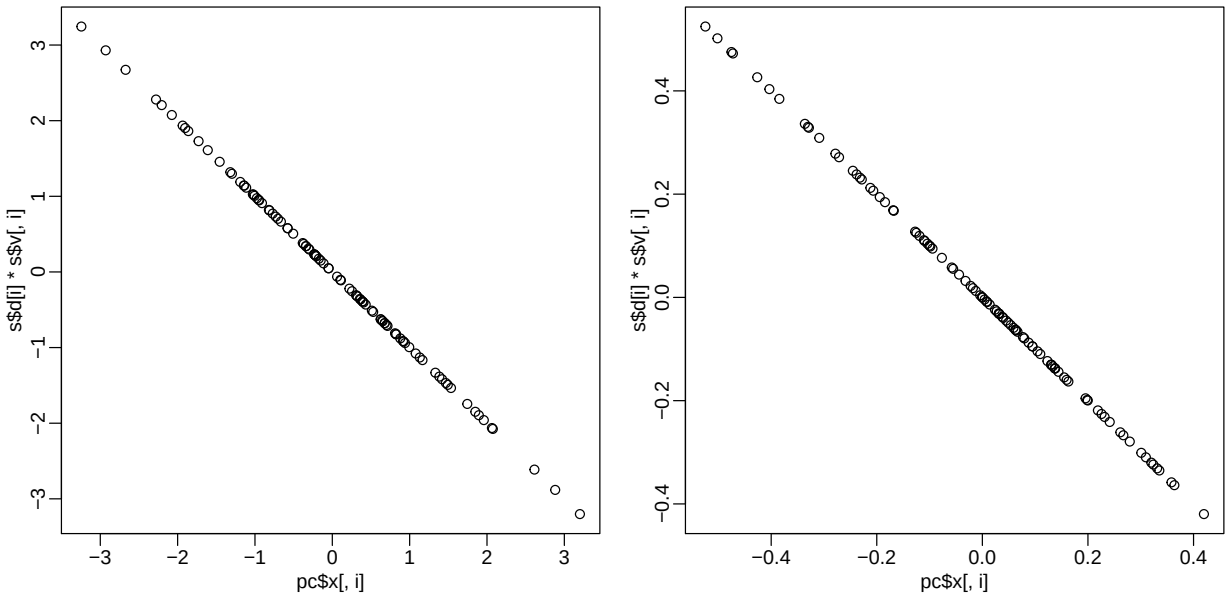


Figure 7: Plot showing SVD and prcomp give same results.

NOTA: Usamos la transpuesta de `Y` porque `prcomp` asume el orden: units/samples en filas y los predictores/features en las columnas

SVD y PCA en práctica usando R

Usaremos los datos de expresión génica en tejidos para ver cómo aplicar todo esto en práctica

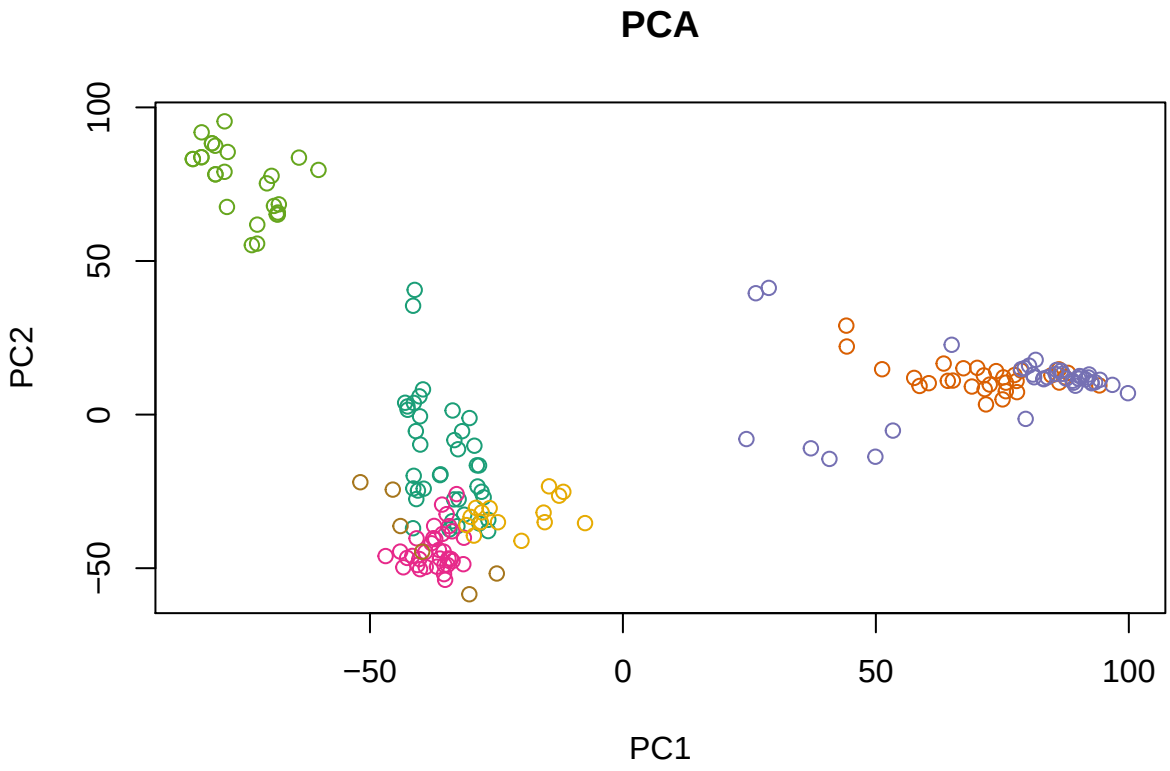
```
library(devtools)
#install_github("dagdata","genomicsclass")
#library(dagdata)
data(tissuesGeneExpression)
library(rafalib)
group <- as.fumeric(tab$Tissue)
```

En primer lugar queremos ver el PCA de las muestras:

```
x <- t(e)
pc <- prcomp(x)
# ?prcomp
names(pc)
```

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

```
plot(pc$x[,1], pc$x[,2], col=group, main="PCA", xlab="PC1", ylab="PC2")
```

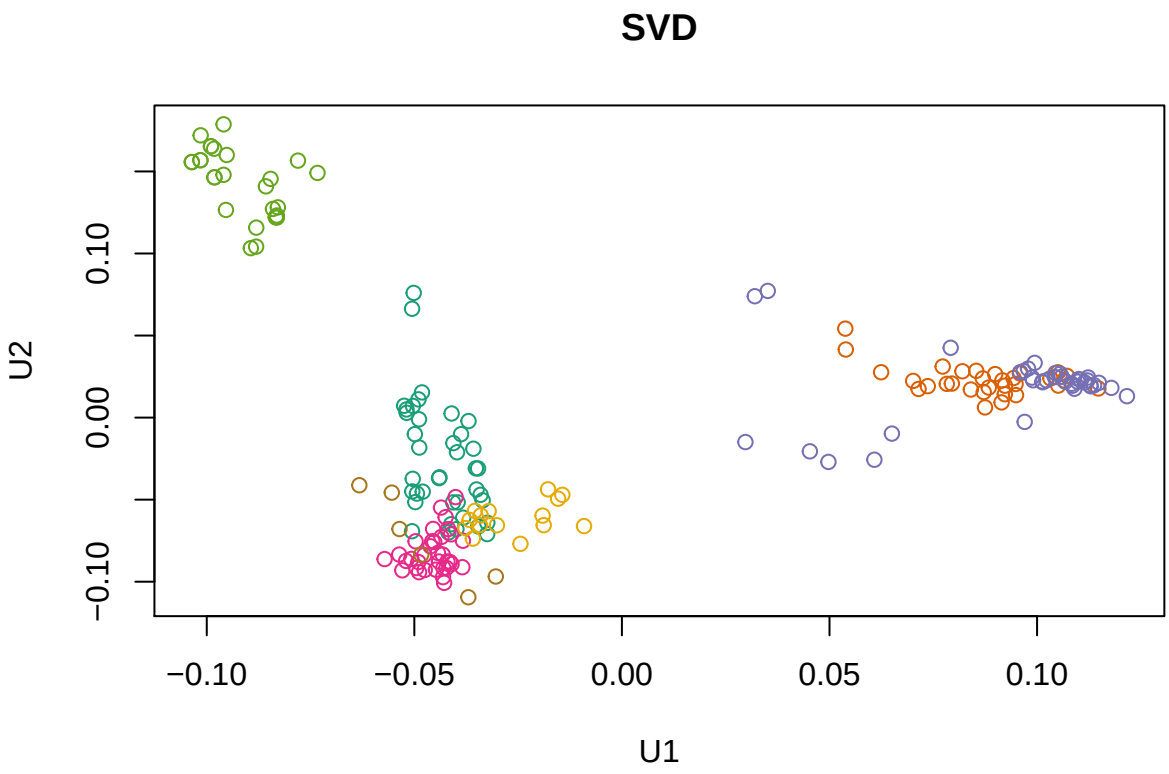


PCA es equivalente a calcular SVD en los datos centrados por columnas (una vez traspuesta nuetra matriz x tiene los genes en las columnas) Podemos usar la función *sweep* para trabajar con filas y columnas eficientemente.

```
cx <- sweep(x, 2, colMeans(x), "-")
sv <- svd(cx)
names(sv)
```

```
## [1] "d" "u" "v"
```

```
plot(sv$u[,1], sv$u[,2], col=group, main="SVD", xlab="U1", ylab="U2")
```



Las columnas de U de la descomposicion SVD se corresponden con las componentes principales de `x` en el PCA. Además la matriz V del SVD es equivalente a hacer la matriz de `rotacion` que nos devuelve`prcomp`.

```
sv$v[1:5,1:5]
```

```
##               [,1]         [,2]         [,3]         [,4]          [,5]
## [1,]  0.0046965883 -0.013275124  0.002087159  0.017092900  0.0006956308
## [2,] -0.0021622800 -0.002211639  0.001543377 -0.003345557 -0.0034158660
## [3,] -0.0030945339  0.005870061  0.001686328  0.003890002  0.0019032363
## [4,] -0.0007355177 -0.002001858 -0.002752526  0.001776287  0.0192205309
## [5,]  0.0010132625  0.001214547 -0.001561327  0.003349323 -0.0012379970
```

```
pc$rotation[1:5,1:5]
```

```
##                     PC1          PC2          PC3          PC4
## 1007_s_at  0.0046965883 -0.013275124  0.002087159  0.017092900
## 1053_at   -0.0021622800 -0.002211639  0.001543377 -0.003345557
## 117_at    -0.0030945339  0.005870061  0.001686328  0.003890002
## 121_at    -0.0007355177 -0.002001858 -0.002752526  0.001776287
## 1255_g_at  0.0010132625  0.001214547 -0.001561327  0.003349323
##                     PC5
## 1007_s_at  0.0006956308
## 1053_at   -0.0034158660
## 117_at     0.0019032363
## 121_at     0.0192205309
## 1255_g_at -0.0012379970
```

Los elementos de la diagonal de D de la SVD son proporcionales a las desviaciones estandard del PCA con la

excepcion de que las std de`prcomp` son muestrales (con $n/(n-1)$ correction). Los elementos de D son la suma de cuadrados de las componentes principales sin dividir por el tamaño muestral.

```
head(sv$d^2)
```

```
## [1] 673418.29 285393.11 182527.20 127667.15 108576.10  81998.66
```

```
head(pc$sdev^2)
```

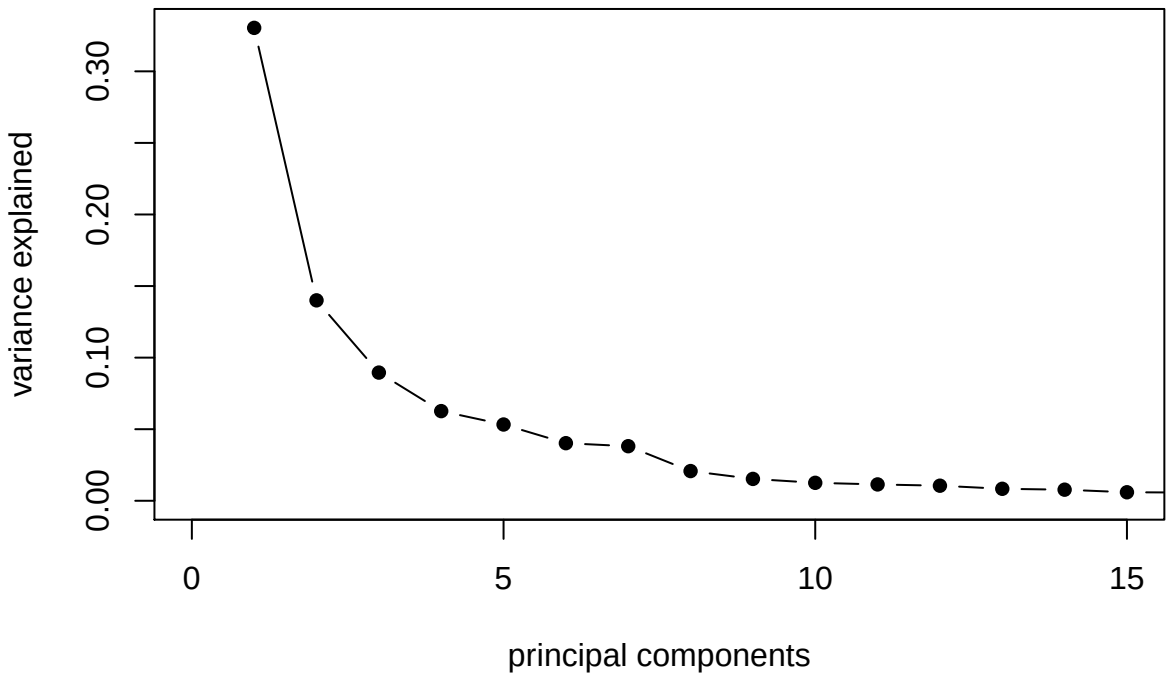
```
## [1] 3582.0122 1518.0485  970.8894  679.0806  577.5325  436.1631
```

```
head(sv$d^2 / (ncol(e) - 1))
```

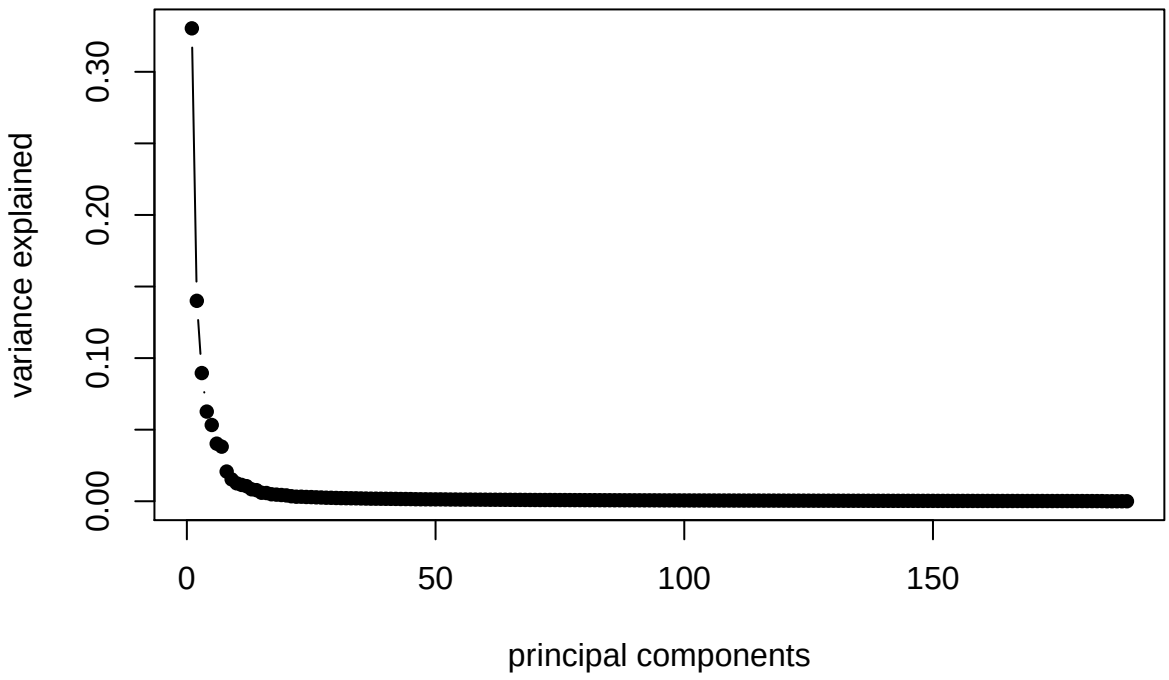
```
## [1] 3582.0122 1518.0485  970.8894  679.0806  577.5325  436.1631
```

Si dividimos la varianza por la suma obtenemos un plot del radio de varianza explicada por cada componente principal:

```
plot(sv$d^2 / sum(sv$d^2), xlim=c(0,15), type="b", pch=16,
     xlab="principal components",
     ylab="variance explained")
```

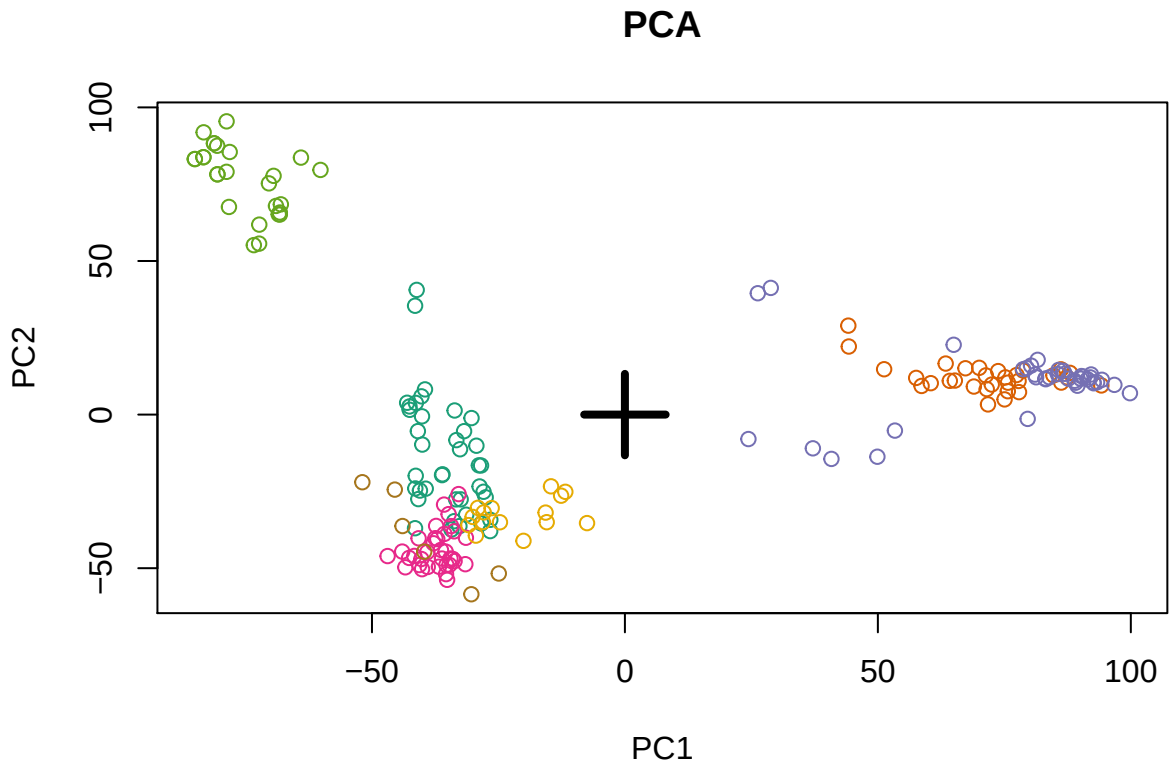



```
plot(sv$d^2 / sum(sv$d^2), type="b", pch=16,
     xlab="principal components",
     ylab="variance explained")
```

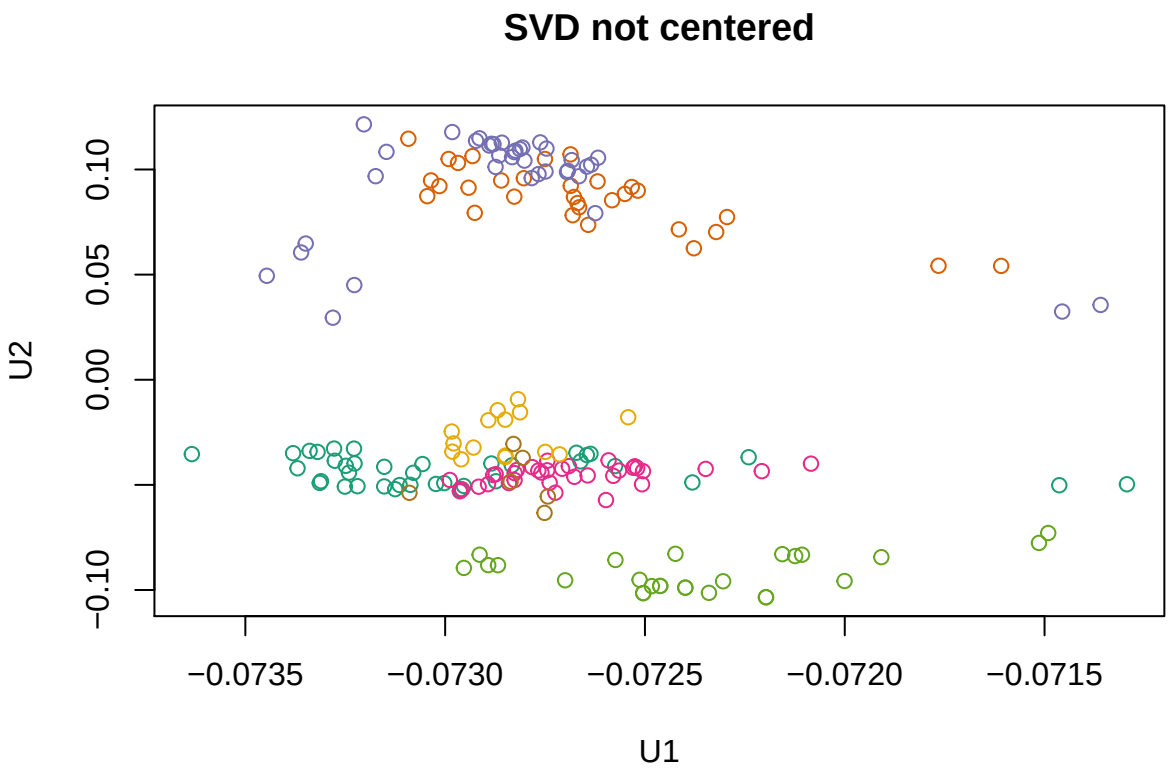


Si no hubiéramos centrado los datos antes de hacer `svd` el plot habría sido algo distinto:

```
svNoCenter <- svd(x)
plot(pc$x[,1], pc$x[,2], col=group, main="PCA", xlab="PC1", ylab="PC2")
points(0,0,pch=3,cex=4,lwd=4)
```



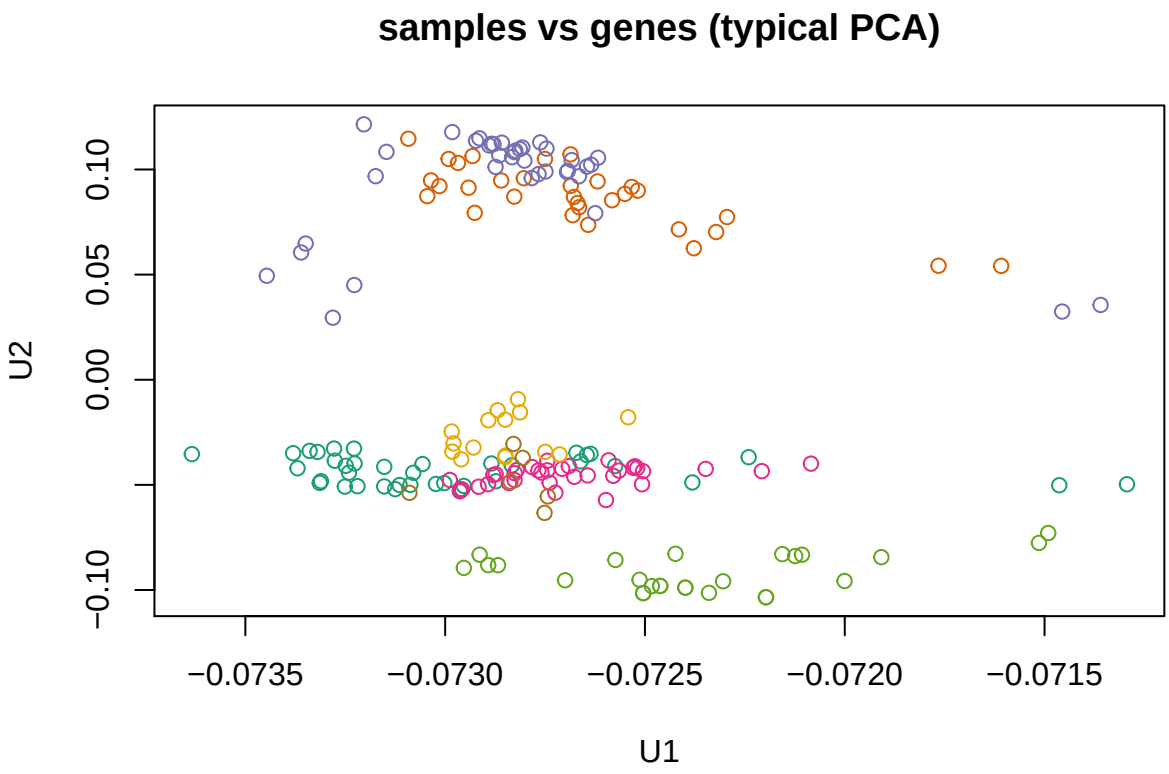

```
plot(svNoCenter$u[,1], svNoCenter$u[,2], col=group, main="SVD not centered", xlab="U1", ylab="U2")
```



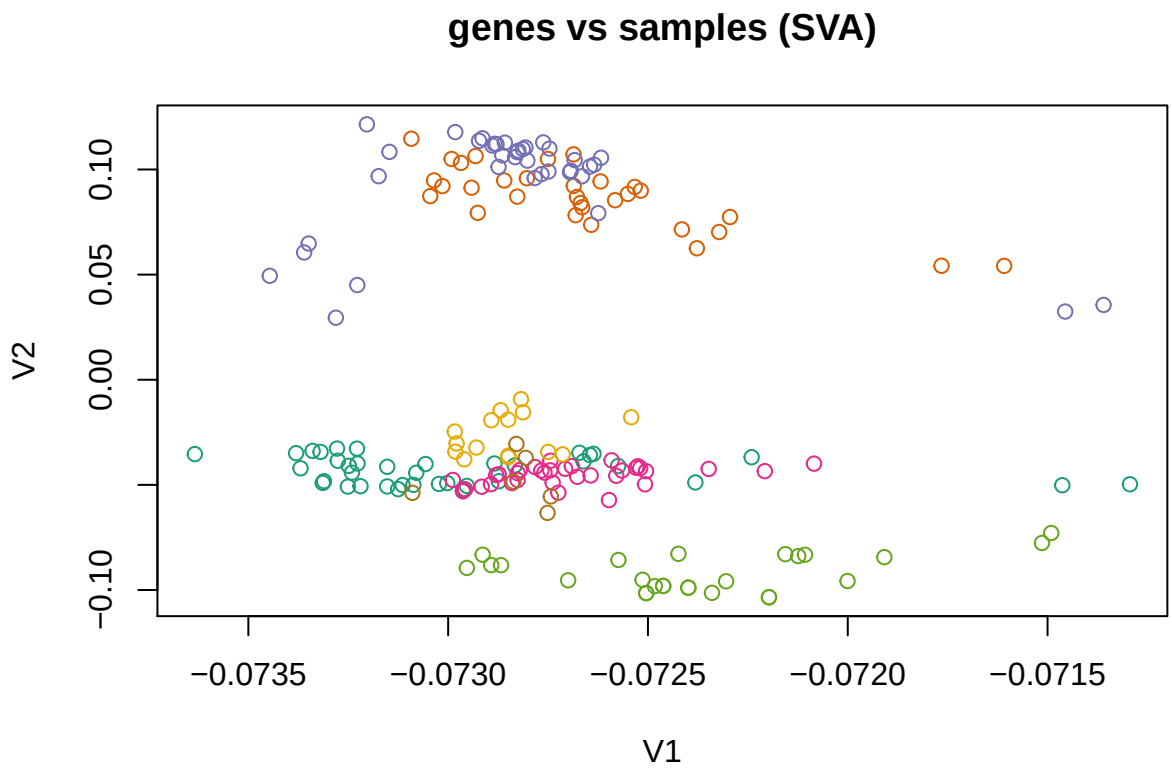
SVA: Surrogate Variable Analysis

Si usamos SVD sobre la matriz en la que las muestras son las columnas estamos realizando SVA. Es equivalente al SVD de la traspuesta (las muestras son las filas) si no hacemos centering

```
sv2 <- svd(t(e))
plot(sv2$u[,1], sv2$u[,2], col=group, main="samples vs genes (typical PCA)", xlab="U1", ylab="U2")
```




```
sv1 <- svd(e)
plot(sv1$v[,1], sv1$v[,2], col=group, main="genes vs samples (SVA)", xlab="V1", ylab="V2")
```



Como lo hagamos dependerá de nuestra pregunta.

Aplicaciones de PCA

Iris Dataset: PCA to detect and remove collinearities

El ejemplo Iris contiene medidas botánicas para tres tipos de plantas diferentescon muchas observaciones para cada tipo. La pregunta final seria intentar construir un clasificador basado en estos cinco predictores para identificar los tipos de planta. Es esto posible? Primero queremos ver si los tres tipos de planta que hay se diferencian los unos de otros suficientemente en base a estos 5 predictores o si necesitamos recoger más datos.

```
names(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
## [5] "Species"
```

```
head(iris)
```

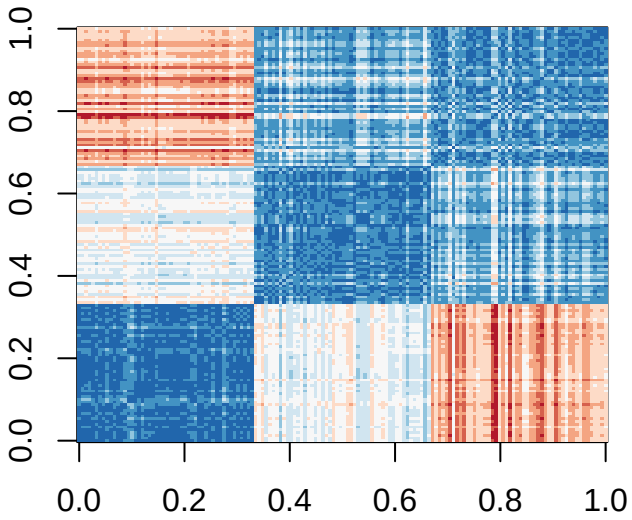
```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
dim(iris)
```

```
## [1] 150   5
```

Vamos a calcular la distancia entre especies como hicimos con el ejemplo de los pokemon. Las cuatro primeras variables son numéricas, la quinta no y por eso la quitamos para calcular la distancia.

```
x <- iris[,1:4] %>% as.matrix()
d <- dist(x)
image(as.matrix(d), col = rev(RColorBrewer::brewer.pal(9, "RdBu")))
```



Parece que los predictores se agrupan en dos grupos en lugar de en tres. Vamos a ver si nuestros predictores estan correlacionados.

```
cor(x)
```

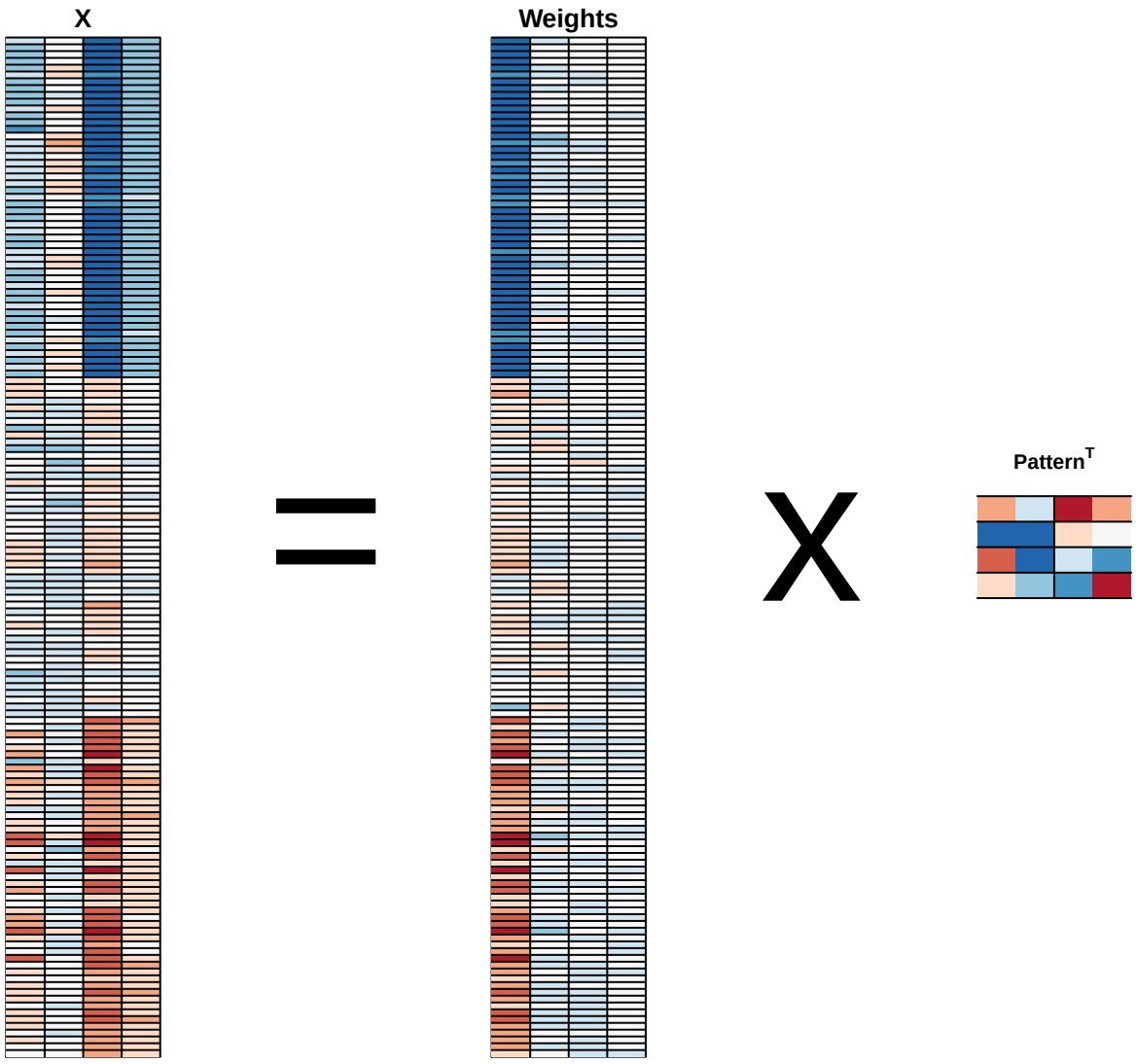
```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length    1.0000000  -0.1175698    0.8717538   0.8179411
## Sepal.Width    -0.1175698   1.0000000   -0.4284401  -0.3661259
## Petal.Length    0.8717538  -0.4284401    1.0000000   0.9628654
## Petal.Width     0.8179411  -0.3661259    0.9628654   1.0000000
```

Vamos a aplicar PCA a la matriz de datos para ver si podemos ver nuestros datos en dos dimensiones en lugar de en las cinco que tenemos. Vemos cuanta variabilidad se explica con cada componente usando la función *summary*:

```
pca <- prcomp(x)
summary(pca)
```

```
## Importance of components:
##                           PC1     PC2    PC3     PC4
## Standard deviation     2.0563 0.49262 0.2797 0.15439
## Proportion of Variance 0.9246 0.05307 0.0171 0.00521
## Cumulative Proportion  0.9246 0.97769 0.9948 1.00000
```

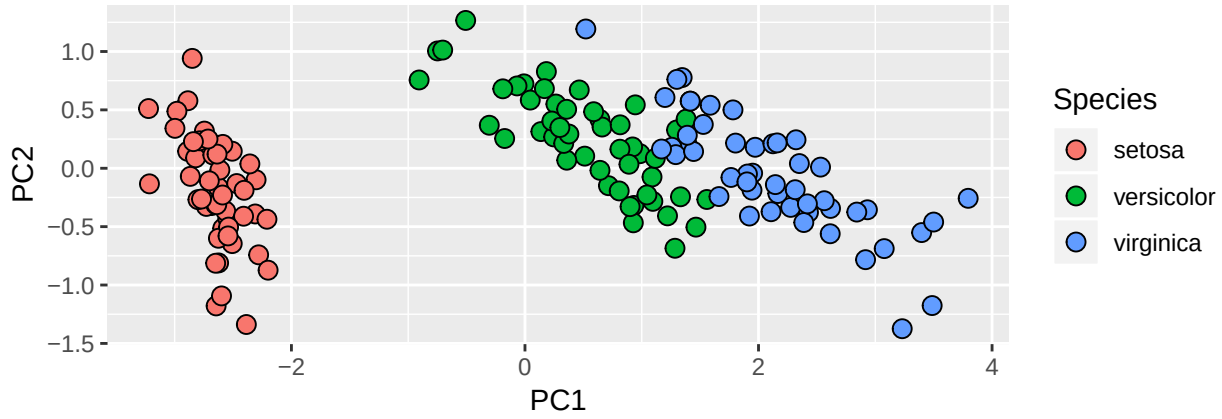
La primera componente representa el 92% de la variabilidad y la segunda le añade otro 5% mas. La aproximación a dos dimensiones debería de ser suficientemente buena. Visualizamos los resultados del PCA:



El primer *pattern* consiste en Sepal Length, Petal Length and Petal Width (rojo) en una direccion y Sepal Width en la otra (azul). El segundo patrón tiene Sepal Length y Petal Width en una direccion (azul) y las otras dos variables en la otra (rojo). Se ve claramente como los pesos de la primera componente principal son los que dominan todo, separando el primer tercio del heatmap (setosa) del segundo y el tercero (versicolor and virginica). La segunda columna de pesos separa ligeramente versicolor (rojo) de virgnica (azul).

Pintamos la PCA

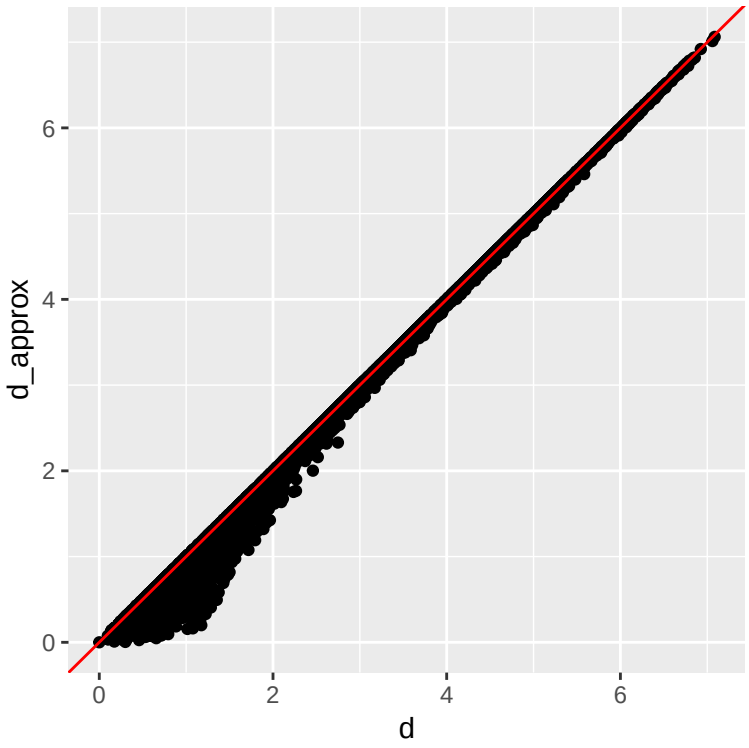
```
data.frame(pca$x[,1:2], Species=iris$Species) %>%
  ggplot(aes(PC1,PC2, fill = Species))+
  geom_point(cex=3, pch=21) +
  coord_fixed(ratio = 1)
```



Las dos primeras componentes preservan la distancia:

```
d_approx <- dist(pca$x[, 1:2])
qplot(d, d_approx) + geom_abline(color="red")
```

```
## Don't know how to automatically pick scale for object of type dist. Defaulting to continuous.
## Don't know how to automatically pick scale for object of type dist. Defaulting to continuous.
```



MNIST Example

Usemos un ejemplo en el que tenemos la intensidad de 784 pixels en imágenes de números escritos a mano. Podemos hacer algo con esto? Podemos crear nuevos algoritmos de ML con menos variables? Let's load the data:

```
library(dslabs)
```

```
## Warning: package 'dslabs' was built under R version 3.5.2
```

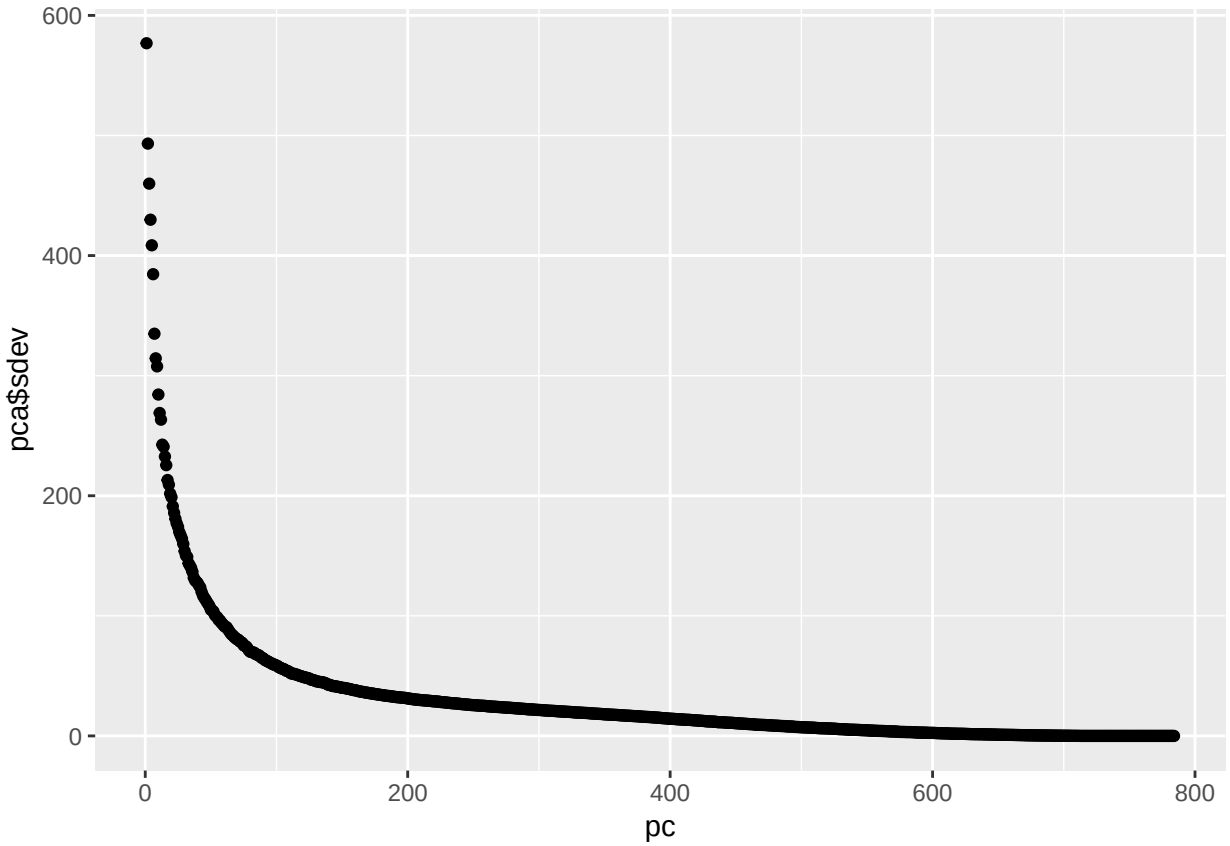
```
if(!exists("mnist")) mnist <- read_mnist()
```

Este dataset se divide en 60.000 números para training y 10.000 para test. Como los pixels están muy cerca podemos esperar que estén correlacionados y que por tanto podamos usar reducción de la dimensionalidad con ellos.

Probemos con PCA. Llevará un tiempo...

```
col_means <- colMeans(mnist$test$images)
pca <- prcomp(mnist$train$images)
```

```
pc <- 1:ncol(mnist$test$images)
qplot(pc, pca$sdev)
```



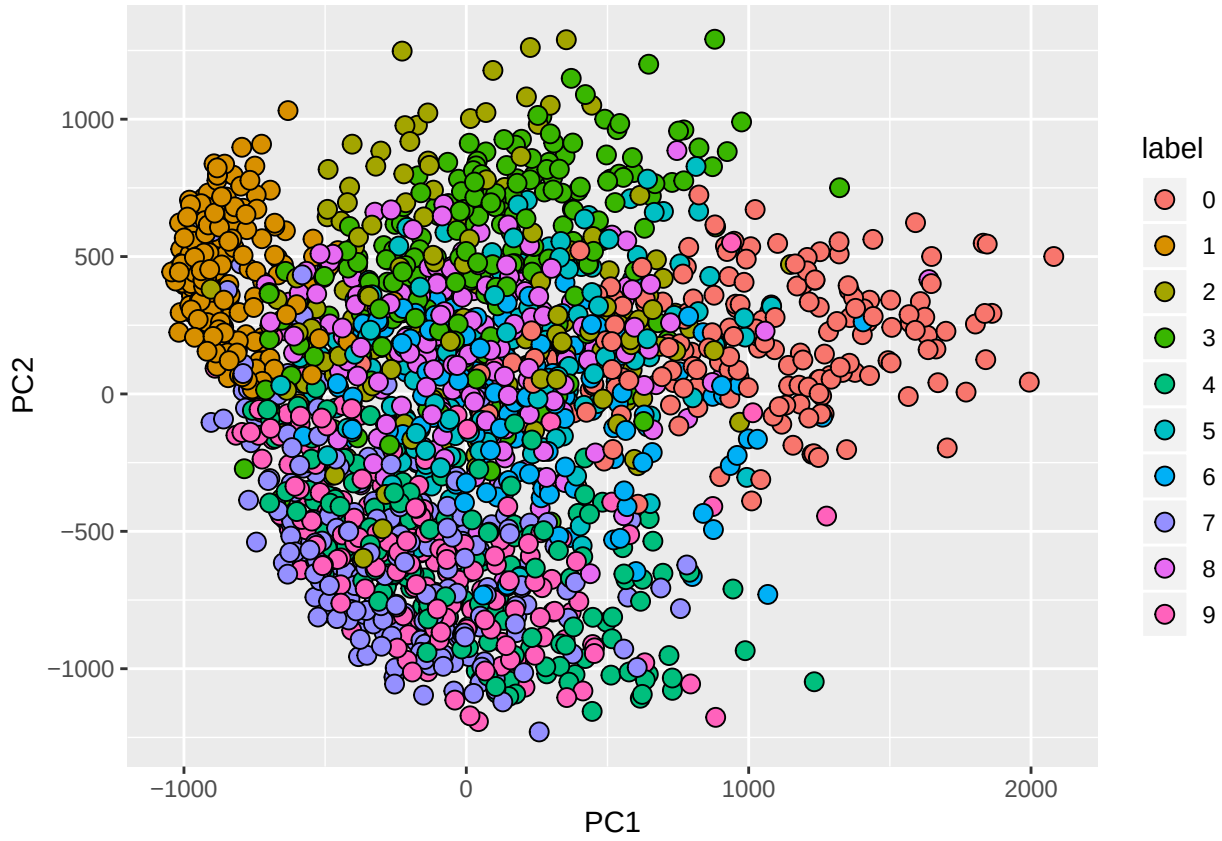
Las primeras componentes explican mucha parte de la variabilidad:

```
summary(pca)$importance[,1:5]
```

```
##                             PC1       PC2       PC3       PC4       PC5
## Standard deviation     576.82291 493.23822 459.89930 429.85624 408.56680
## Proportion of Variance   0.09705   0.07096   0.06169   0.05389   0.04869
## Cumulative Proportion    0.09705   0.16801   0.22970   0.28359   0.33228
```

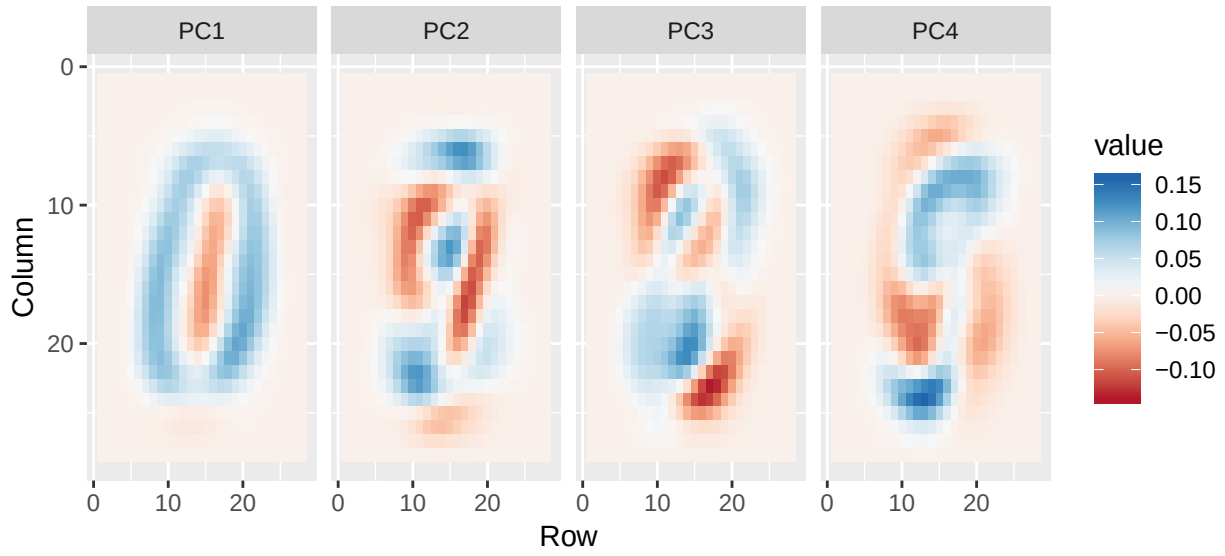
Tomando una muestra de 2000 dígitos:

```
data.frame(PC1 = pca$x[,1], PC2 = pca$x[,2],
           label=factor(mnist$train$label)) %>%
  sample_n(2000) %>%
  ggplot(aes(PC1, PC2, fill=label))+
  geom_point(cex=3, pch=21)
```



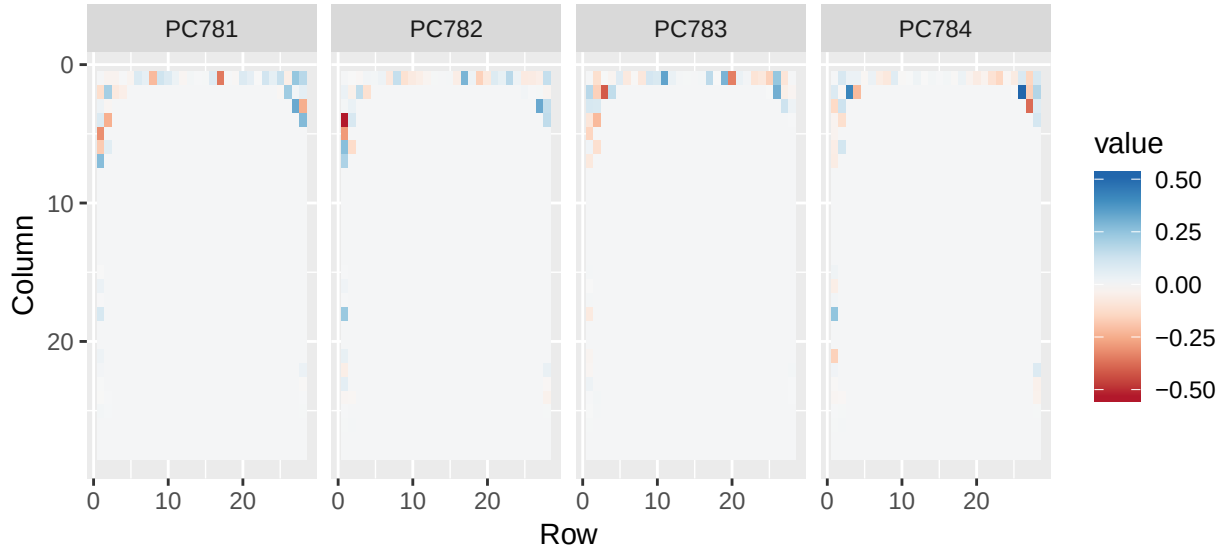
Vamos a ver los pesos:

```
library(RColorBrewer)
tmp <- lapply( c(1:4,781:784), function(i){
    expand.grid(Row=1:28, Column=1:28) %>%
      mutate(id=i, label=paste0("PC",i),
             value = pca$rotation[,i])
})
tmp <- Reduce(rbind, tmp)
tmp %>% filter(id<5) %>%
  ggplot(aes(Row, Column, fill=value)) +
  geom_raster() +
  scale_y_reverse() +
  scale_fill_gradientn(colors = brewer.pal(9, "RdBu")) +
  facet_wrap(~label, nrow = 1)
```



Las PCs con menor varianza aparecen en las esquinas:

```
tmp %>% filter(id>5) %>%
  ggplot(aes(Row, Column, fill=value)) +
  geom_raster() +
  scale_y_reverse() +
  scale_fill_gradientn(colors = brewer.pal(9, "RdBu")) +
  facet_wrap(~label, nrow = 1)
```



Apliquemos ahora la transformacion obtenida en el training al test y corramos K-nearest neighbors en ese data set de dimensionalidad reducida.

36 dimensiones explican el 80% de los datos:

First fit the model:

```
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
k <- 36
x_train <- pca$x[,1:k]
y <- factor(mnist$train$labels)
fit <- knn3(x_train, y)
```

Transformamos el test:

```
x_test <- sweep(mnist$test$images, 2, col_means) %*% pca$rotation
x_test <- x_test[,1:k]
```

Predecimos:

```
y_hat <- predict(fit, x_test, type = "class")
confusionMatrix(y_hat, factor(mnist$test$labels))$overall["Accuracy"]
```

```
## Accuracy
##   0.9744
```

Ejercicios

Vamos a explorar el dataset `tissue_gene_expression`

```r
data("tissue_gene_expression")
dim(tissue_gene_expression$x)
```

Queremos saber que muestras de esas 189 se parecen mas en base a la expresion de esos 500 genes. Pinta las dos primeras componentes con colores que representen el tipo de tejido.

2. Los predictores (genes) para cada muestra se han medido con la misma máquina. Ver si existe algun sesgo en las muestras calculando la media de las observaciones de cada muestra y pintandolas contra la primera componente, coloreadas por el tipo de tejido. Cual es la correlacion?
3. Se ve un sesgo. Re-calcula la PCA despues de haber centrado los datos.
4. Dibuja un boxplot mostrando el valor de cada tejido en las diez primeras componentes.
5. Dibuja el % de varianza explicada por cada PC. Hint: use the `summary` function.

Non-lineal projections: t-sne

t-sne utiliza probabilidades condicionales en lugar de distancias euclídeas.

Si comparamos el PCA y el t-sne para el data set MNIST vemos como t-sne es capaz de capturar en 2 dimensiones lo que PCA en muchas mas

```
library(dplyr)
library(Rtsne)
library(dslabs)
if(!exists("mnist")) mnist <- read_mnist()

train_full <- mnist$train$images
train <- sample_frac(as.data.frame(train_full),0.33)
```

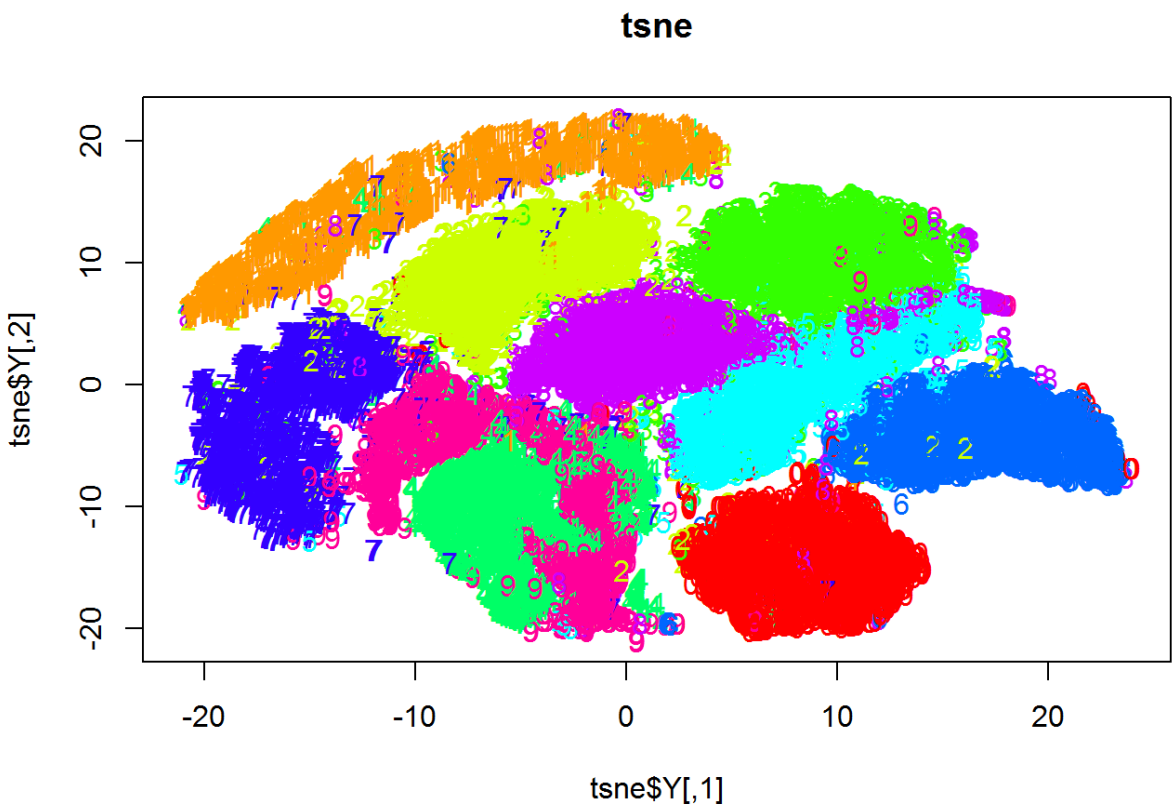


Figure 8: Supervised vs Unsupervised learning

```
## Curating the database for analysis with both t-SNE and PCA
label<-mnist$train$label
## for plotting
colors = rainbow(10)
names(colors) = unique(label)

## Executing the algorithm on curated data
tsne <- Rtsne(train[,-1], dims = 2,
              perplexity=30,
              verbose=TRUE,
              max_iter = 500,
              col=label)
exeTimeTsne<- system.time(Rtsne(train[,-1], dims = 2, perplexity=30, verbose=TRUE, max_iter = 500))

plot(tsne$Y, t='n', main="tsne")

text(tsne$Y, labels=label, col=colors[label])
```

Clustering

Utilizaremos el dataset de expresión génica:

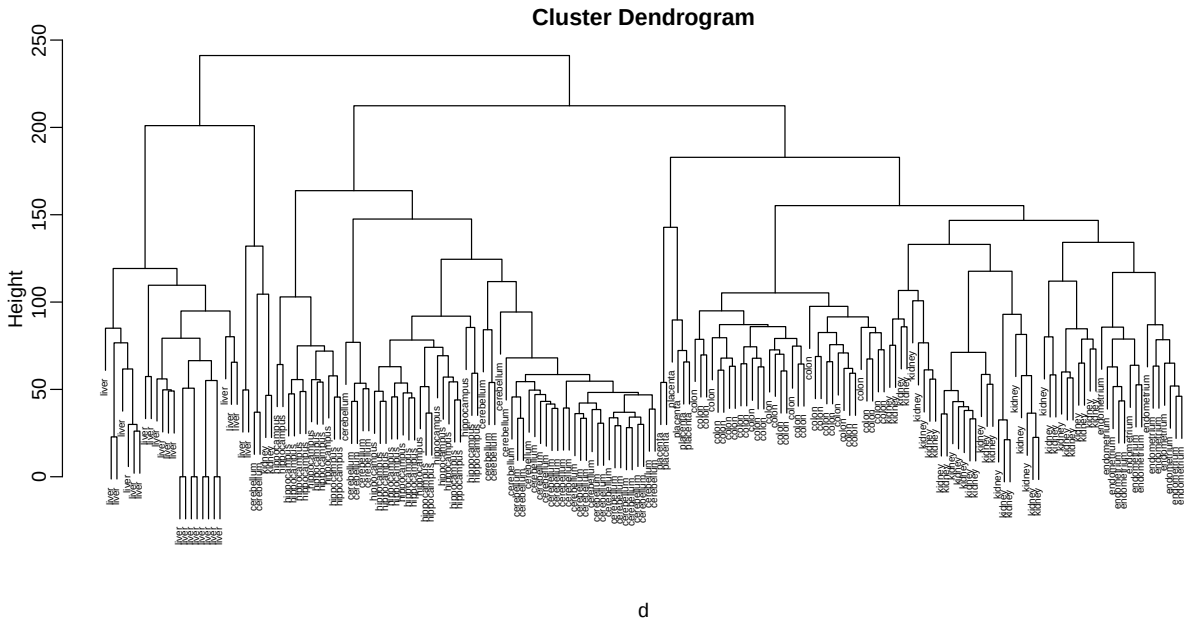


Figure 9: Dendrogram showing hierarchical clustering of tissue gene expression data.

```
library(tissuesGeneExpression)
data(tissuesGeneExpression)
```

Calculamos la distancia entre muestras

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

Hierarchical clustering

Sabemos la distancia, pero... como se relacionan entre ellos? Clustering jierarquico compara dos a dos los perfiles y comienza agregando aquellos mas parecidos. COn ese nuevo perfil de dos, busca el tercero mas cercano y asi iterativamente.

La funcion *hclust* genera los grupos a partir de las distancias. Ese objeto (un árbol) puede ser después pintado.

```
library(rafalib)
mypar()
hc <- hclust(d)
hc
```

```
##
## Call:
## hclust(d = d)
##
## Cluster method   : complete
## Distance         : euclidean
## Number of objects: 189
```

```
plot(hc,labels=tissue,cex=0.5)
```

Podemos identificar las muestras agrupadas por tejido? Usamos `myplclust` para colorear.

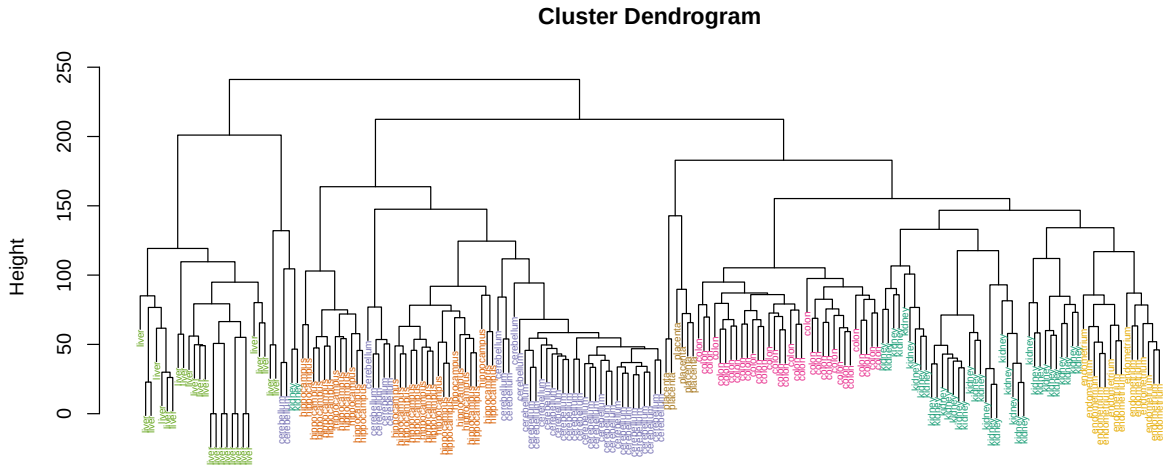


Figure 10: Dendrogram showing hierarchical clustering of tissue gene expression data with colors denoting tissues.

```
myplclust(hc, labels=tissue, lab.col=as.fumeric(tissue), cex=0.5)
```

Parece que si. Pero necesitamos definir clusters. . . cortando el árbol. Dibujamos una linea a 120 y vemos que grupos se desconectan:

```
myplclust(hc, labels=tissue, lab.col=as.fumeric(tissue),cex=0.5)
abline(h=120)
```

Vemos cual es el overlap de estos grupos con sus grupos reales:

```
hclusters <- cutree(hc, h=120)
table(true=tissue, cluster=hclusters)
```

```
##              cluster
## true           1  2  3  4  5  6  7  8  9 10 11 12 13 14
##   cerebellum   0  0  0  0 31  0  0  0  2  0  0  5  0  0
##   colon        0  0  0  0  0  0 34  0  0  0  0  0  0  0
##   endometrium  0  0  0  0  0  0  0  0  0  0 15  0  0  0
##   hippocampus  0  0 12 19  0  0  0  0  0  0  0  0  0  0
##   kidney       9 18  0  0  0 10  0  0  2  0  0  0  0  0
##   liver        0  0  0  0  0  0  0 24  0  2  0  0  0  0
##   placenta     0  0  0  0  0  0  0  0  0  0  0  0  2  4
```

`cutree` identifica los clusters para un número determinado de clusters:

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

```
##              cluster
## true           1  2  3  4  5  6  7  8
##   cerebellum   0  0 31  0  0  2  5  0
##   colon        0  0  0 34  0  0  0  0
```

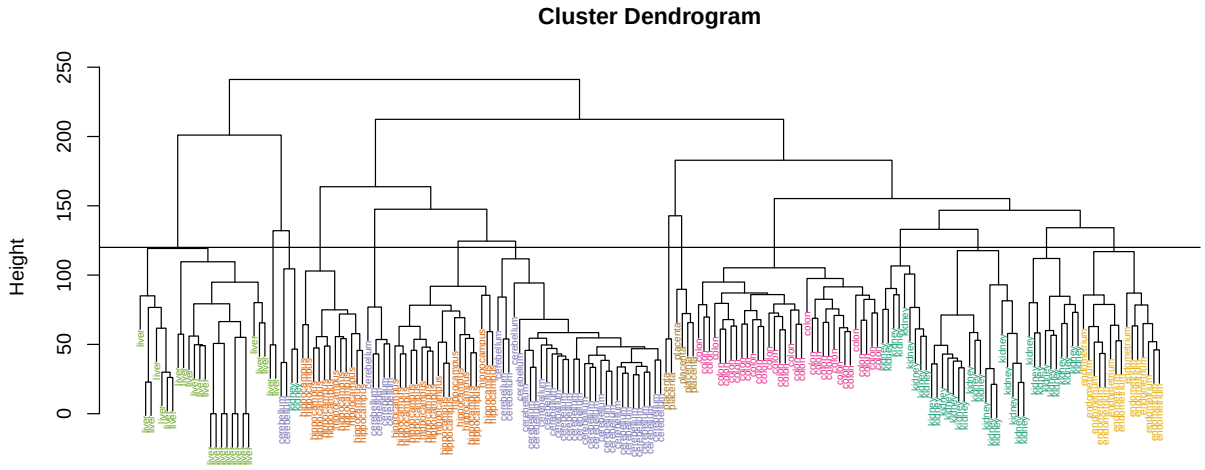


Figure 11: Dendrogram showing hierarchical clustering of tissue gene expression data with colors denoting tissues. Horizontal line defines actual clusters.

```
##   endometrium 15  0  0  0  0  0  0  0
##   hippocampus  0 12 19  0  0  0  0  0
##   kidney      37  0  0  0  0  2  0  0
##   liver        0  0  0  0 24  2  0  0
##   placenta     0  0  0  0  0  0  0  6
```

En general tenemos una asociacion tejido-cluster.

K-means

k-means detecta no asociaciones entre muestras sino directamente grupos de muestras (o de genes):

```
set.seed(1)
km <- kmeans(t(e[1:2,]), centers=7)
names(km)
```

```
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

```
mypar(1,2)
plot(e[1,], e[2,], col=as.fumeric(tissue), pch=16)
plot(e[1,], e[2,], col=km$cluster, pch=16)
```

El color representa el tejido real en el primer plot y en el segundo el color representa el cluster definido por kmeans. No ha funcionado muy bien:

```
table(true=tissue,cluster=km$cluster)
```

```
##              cluster
## true           1  2  3  4  5  6  7
##   cerebellum   0  1  8  0  6  0 23
##   colon        2 11  2 15  4  0  0
```

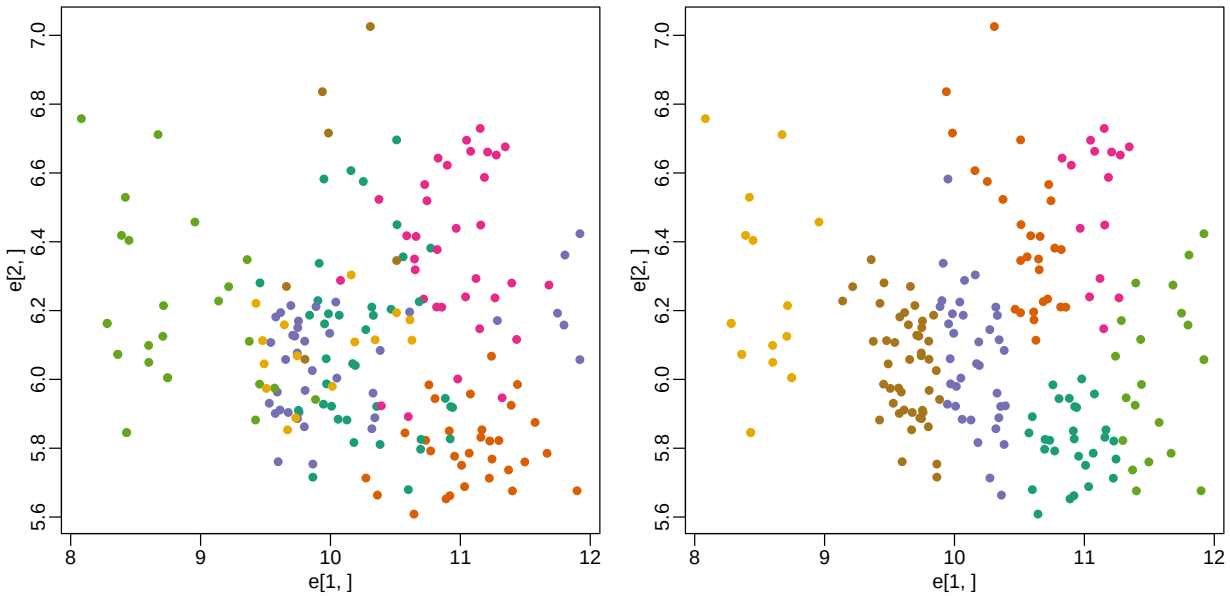


Figure 12: Plot of gene expression for first two genes (order of appearance in data) with color representing tissue (left) and clusters found with kmeans (right).

```
##   endometrium  0  3  4  0  0  0  8
##   hippocampus 19  0  2  0 10  0  0
##   kidney       7  8 20  0  0  0  4
##   liver        0  0  0  0  0 18  8
##   placenta     0  4  0  0  0  0  2
```

Tiene sentido que usar solo 2 genes no sea suficiente. Si usamos todos los genes y lo visualizamos con MDS:

```
km <- kmeans(t(e), centers=7)
mds <- cmdscale(d)
mypar(1,2)
plot(mds[,1], mds[,2])
plot(mds[,1], mds[,2], col=km$cluster, pch=16)
```

By tabulating the results, we see that we obtain a similar answer to that obtained with hierarchical clustering.

```
table(true=tissue,cluster=km$cluster)
```

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

Heatmaps

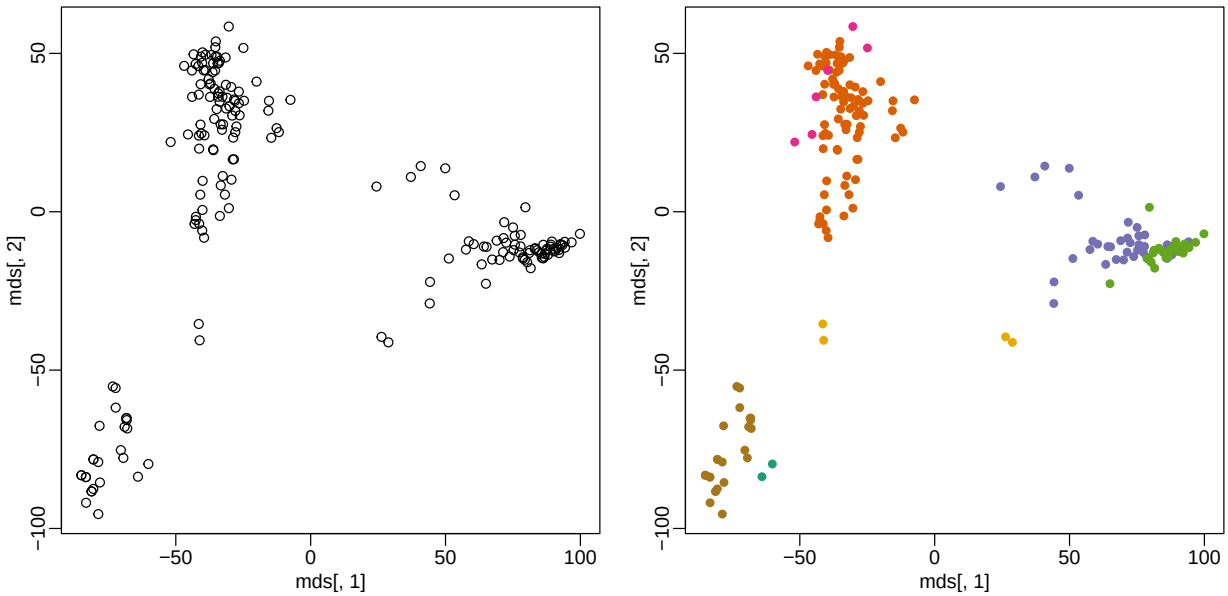


Figure 13: Plot of gene expression for first two PCs with color representing tissues (left) and clusters found using all genes (right).

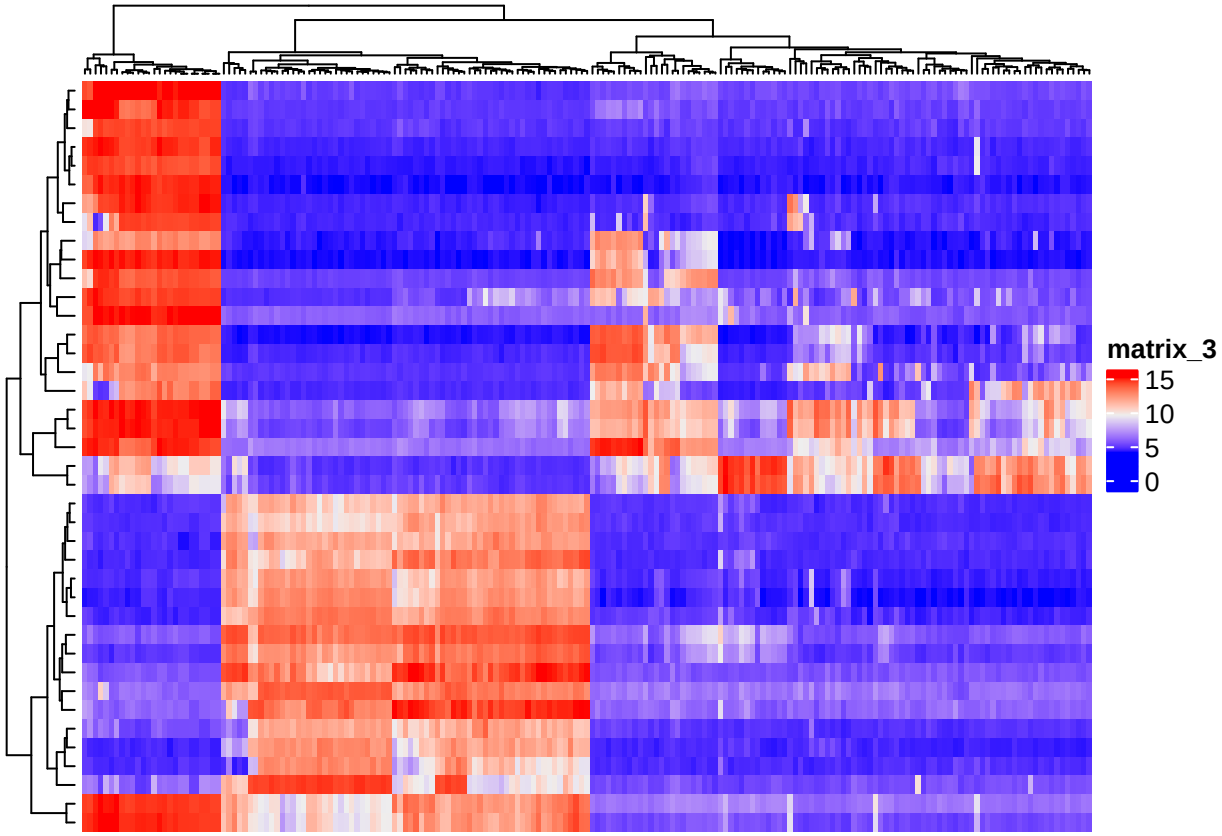
```
library(RColorBrewer)
hmcol <- colorRampPalette(brewer.pal(9, "GnBu"))(100)
```

Nos quedamos con los genes mas informativos, los que tienen una varianza muy alta

```
library(genefilter)
rv <- rowVars(e)
idx <- order(-rv)[1:40]
```

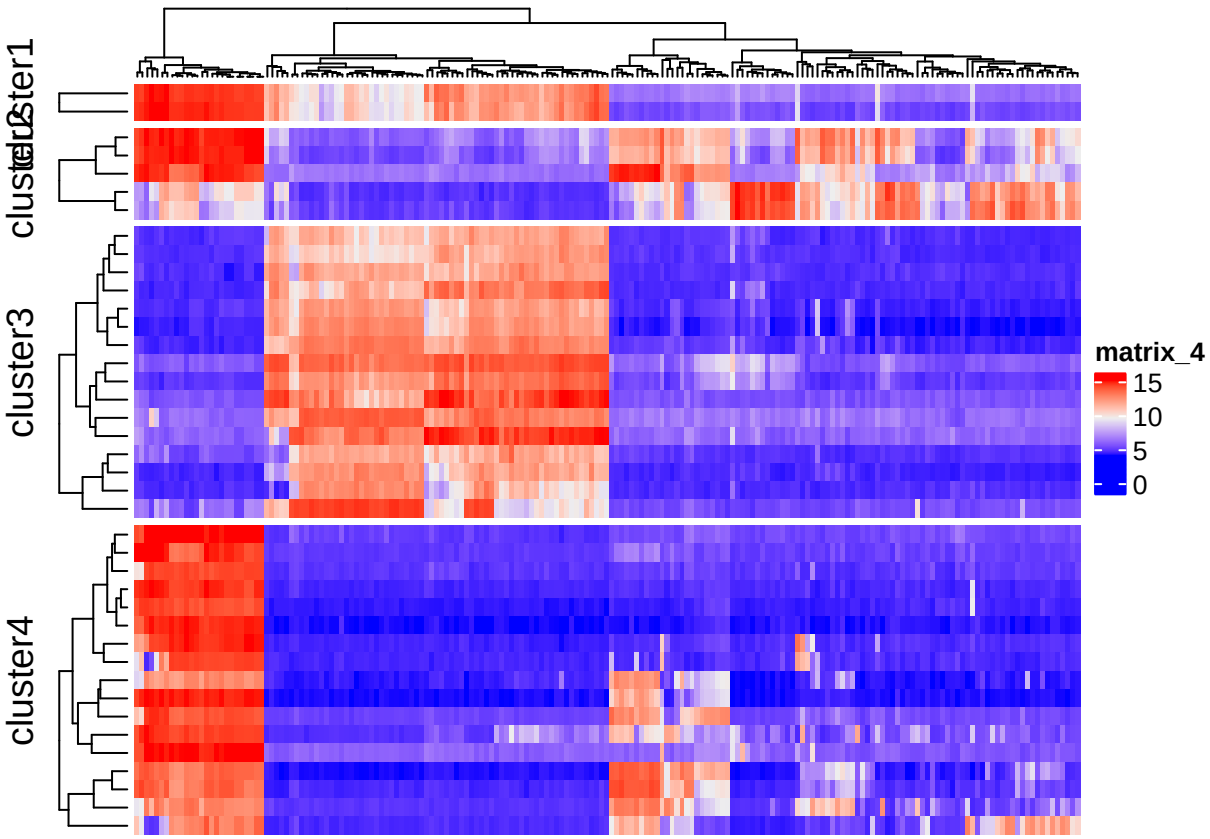
Usamos la libreria ComplexHeatmap para crear un heatmap:

```
library(ComplexHeatmap)
Heatmap(e[idx,],show_column_names = F,show_row_names = F)
```



Podemos a la vez hacerlo con kmeans

```
Heatmap(e[idx,],show_column_names = F,show_row_names = F,km=4)
```


cluster1
cluster2
cluster3
cluster4
matrix_4
15
10
5
0