reduccion de dimenciones

luis manuel

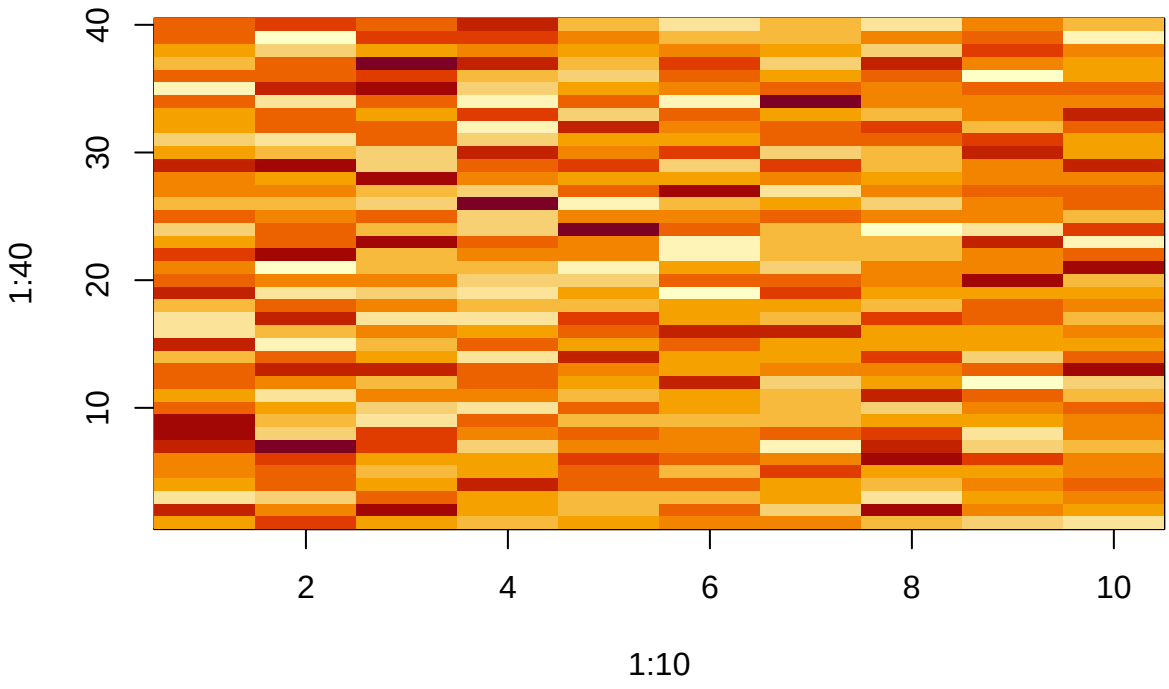
17/12/2020

En esta lección discutiremos el análisis de componentes principales (PCA) y la descomposición de valores singulares (SVD), dos técnicas importantes y relacionadas de reducción de dimensiones. Esto último implica procesos que encuentran subconjuntos de variables en conjuntos de datos que contienen sus esencias. El PCA y el SVD se utilizan tanto en la fase exploratoria como en la etapa de análisis de modelado más formal. Nos centraremos en la fase exploratoria y tocaremos brevemente algunas de las teorías subyacentes.

Comenzaremos con un ejemplo motivador: datos aleatorios.

Esta es dataMatrix, una matriz de 400 números normales aleatorios (media 0 y desviación estándar 1). Lo mostramos con la imagen del comando R. Ejecute el encabezado de comando R con dataMatrix como argumento para ver cómo se ve dataMatrix.

```
set.seed(12345)
dataMatrix <- matrix(rnorm(400), nrow = 40)
image(1:10, 1:40, t(dataMatrix)[, nrow(dataMatrix):1])
```

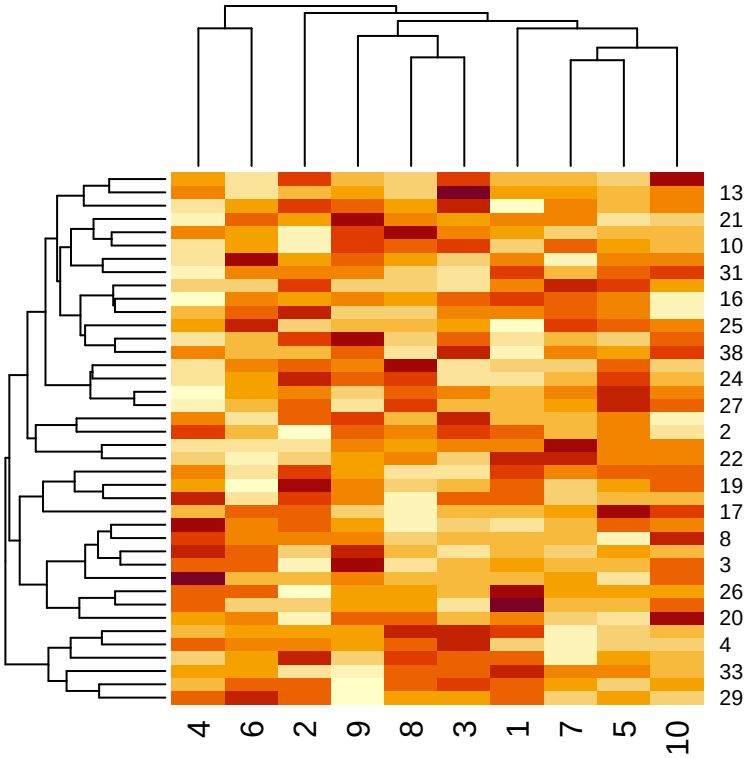


```
head(dataMatrix)
```

```
##            [,1]       [,2]       [,3]       [,4]       [,5]       [,6]
## [1,]  0.5855288  1.1285108  0.6453831  1.5448636 -0.4876385 -1.4361457
## [2,]  0.7094660 -2.3803581  1.0431436  1.3214520  0.3031512 -0.6292596
## [3,] -0.1093033 -1.0602656 -0.3043691  0.3221516 -0.2419740  0.2435218
## [4,] -0.4534972  0.9371405  2.4771109  1.5309551 -0.4817336  1.0583622
## [5,]  0.6058875  0.8544517  0.9712207 -0.4212397 -0.9918029  0.8313488
## [6,] -1.8179560  1.4607294  1.8670992 -1.1588210 -0.2806491  0.1052118
##            [,7]       [,8]       [,9]       [,10]
## [1,] -0.7000758 -1.5138641  0.3803157 -0.37582344
## [2,] -0.5674016  0.1642810  0.6051368 -1.81283376
## [3,] -0.2613939 -0.8708652  1.0196741  0.28860021
## [4,] -1.0638850  1.5933290  0.4749430 -0.18962258
## [5,] -0.1063687  0.6465975 -2.1859464  0.01786021
## [6,]  0.7711037  0.3573697  0.9331922  0.65043024
```

Entonces vemos que dataMatrix tiene 10 columnas (y por lo tanto 40 filas) de números aleatorios. La imagen aquí parece bastante aleatoria. Veamos cómo se agrupan los datos. Ejecute el mapa de calor del comando R con dataMatrix como único argumento.

```
heatmap(dataMatrix)
```

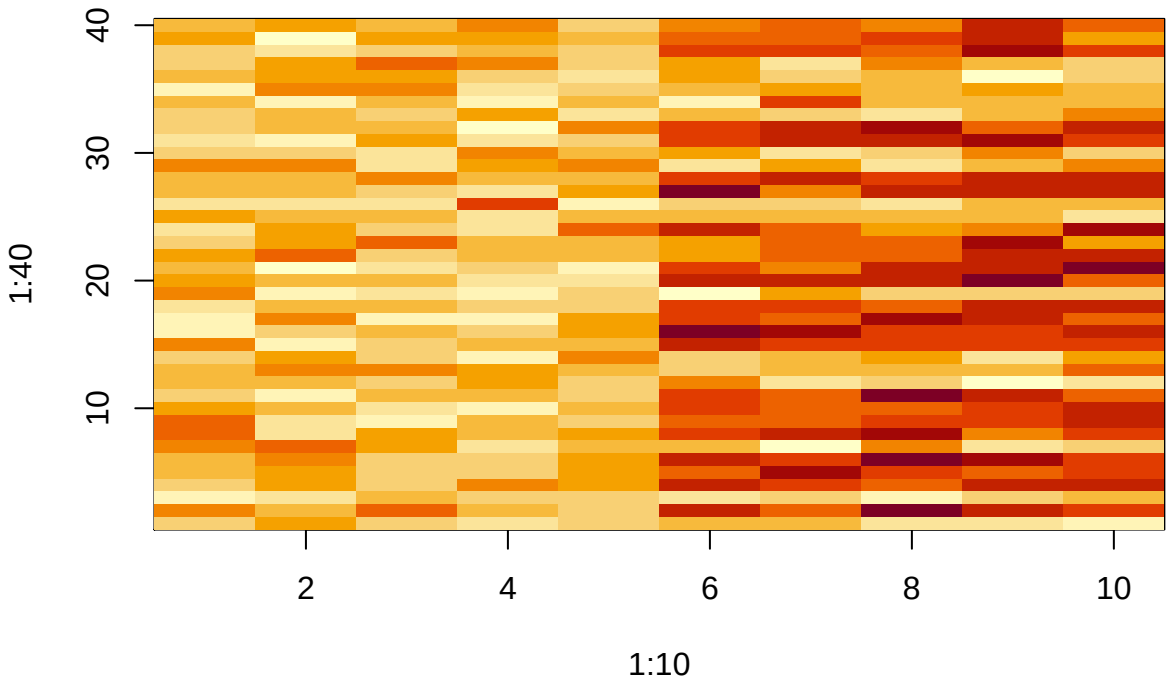


Podemos ver que incluso con la agrupación que proporciona el mapa de calor, permutando las filas (observaciones) y columnas (variables) de forma independiente, los datos todavía parecen aleatorios.

```r
set.seed(678910)
for(i in 1:40){
  # flip a coin
  coinFlip <- rbinom(1,size=1,prob=0.5)
  # if coin is heads add a common pattern to that row
  if(coinFlip){
    dataMatrix[i,] <- dataMatrix[i,] + rep(c(0,3),each=5)
  }
}
```

So in rows affected by the coin flip, the 5 left columns will still have a mean of 0 but the right 5 columns will have a mean closer to 3.

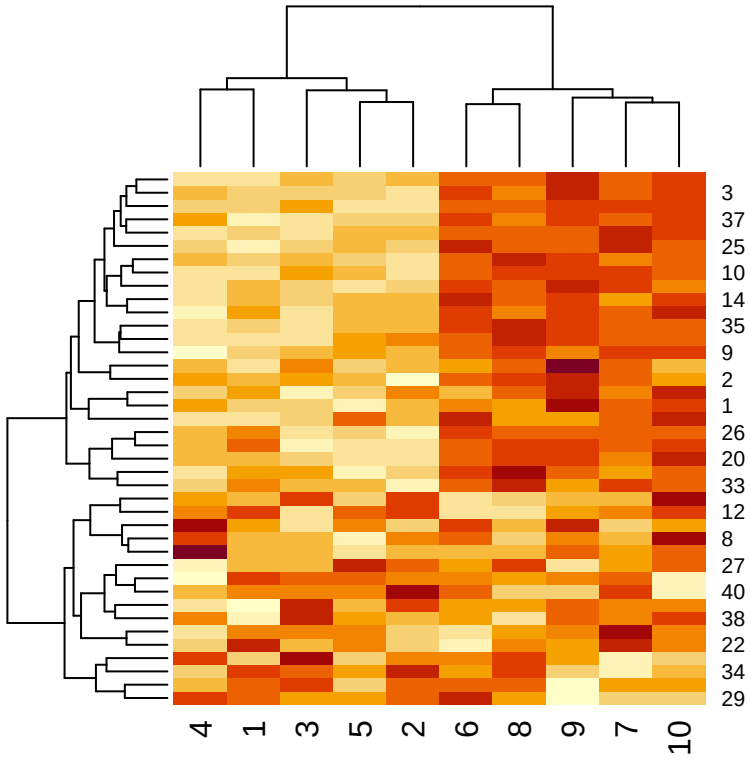
```r
image(1:10, 1:40, t(dataMatrix)[, nrow(dataMatrix):1])
```



Aquí está la imagen de la matriz de datos alterada después de que se haya agregado el patrón. El patrón es claramente visible en las columnas de la matriz. La mitad derecha es más amarilla o más caliente, lo que indica valores más altos en la matriz.

Ahora ejecute el mapa de calor del comando R nuevamente con dataMatrix como su único argumento. Esto realizará un análisis de conglomerados jerárquico en la matriz.

```r
heatmap(dataMatrix)
```



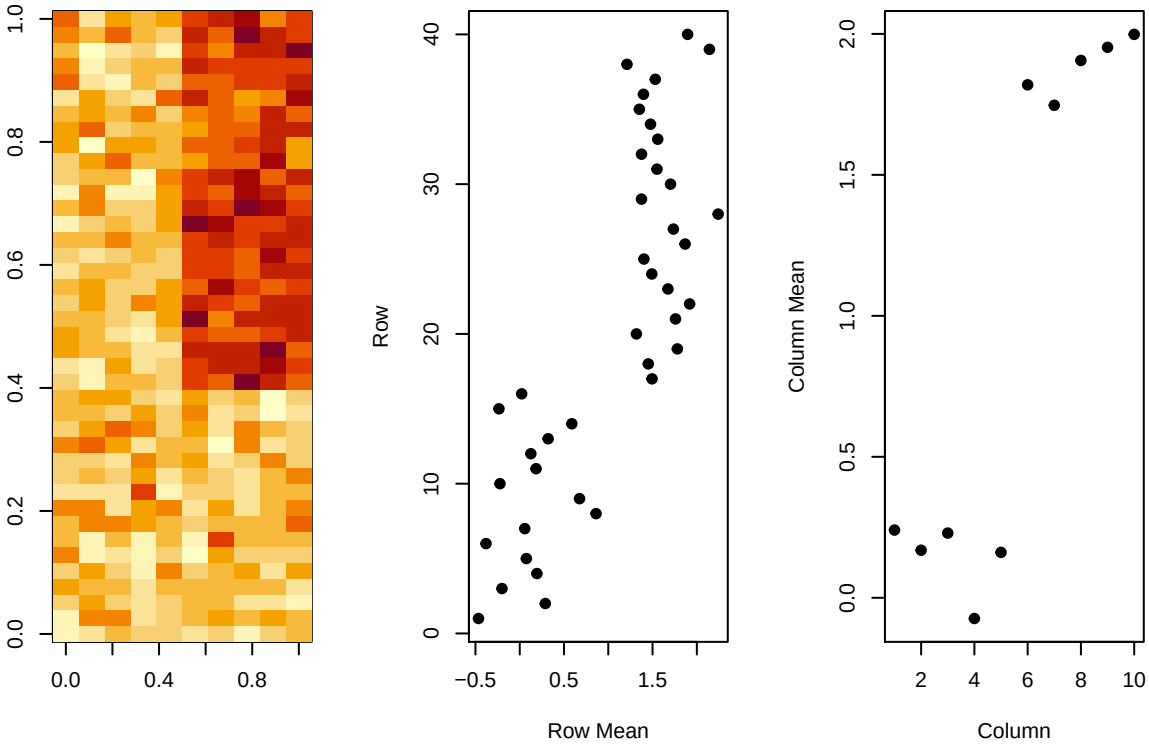
Nuevamente vemos el patrón en las columnas de la matriz. Como se muestra en el dendrograma en la parte superior de la pantalla, estos se dividen en 2 grupos, las columnas con números inferiores (1 a 5) y las columnas con números más altos (6 a 10). Recuerde del código en addPatt.R que para las filas seleccionadas por el coinflip, las últimas 5 columnas tenían 3 agregadas. Las filas todavía parecen aleatorias.

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
hh <- dist(dataMatrix) %>% hclust
dataMatrixOrdered <- dataMatrix[hh$order, ]
par(mfrow = c(1, 3))

## Complete data
image(t(dataMatrixOrdered)[, nrow(dataMatrixOrdered):1])
```

```r
## Show the row means
plot(rowMeans(dataMatrixOrdered), 40:1, , xlab = "Row Mean", ylab = "Row", pch = 19)

## Show the column means
plot(colMeans(dataMatrixOrdered), xlab = "Column", ylab = "Column Mean", pch = 19)
```



Ahora considere esta imagen. A la izquierda hay una imagen similar al mapa de calor de dataMatix que acaba de trazar. Es un gráfico de imagen de la salida de hclust (), una función de agrupación jerárquica aplicada a dataMatrix. El amarillo indica valores "más calientes" o más altos que el rojo. Esto es consistente con el patrón que aplicamos a los datos (aumentando los valores para algunas de las columnas más a la derecha).

La pantalla central muestra la media de cada una de las 40 filas (a lo largo del eje x). Las filas se muestran en el mismo orden que las filas de la matriz térmica de la izquierda. La pantalla de la derecha muestra la media de cada una de las 10 columnas. Aquí los números de las columnas están a lo largo del eje xy sus medias a lo largo de la y.

Vemos inmediatamente la conexión entre la parte amarilla (más caliente) de la imagen del grupo y los medios de la fila superior, ambos en la parte superior derecha de las pantallas. De manera similar, las medias de las columnas de mayor valor están en la mitad derecha de esa pantalla y las medias de las columnas inferiores están en la mitad izquierda.

Ahora hablaremos un poco de teoría. Suponga que tiene miles de variables multivariantes X_1, . . . , X_n. Por multivariado nos referimos a que cada X_i contiene muchos componentes, es decir, X_i = (X_ {i1}, . . . , X_ {im}. Sin embargo, estas variables (observaciones) y sus componentes pueden estar correlacionados entre sí.

Como científicos de datos, nos gustaría encontrar un conjunto más pequeño de variables multivariadas que no estén correlacionadas Y que expliquen tanta varianza (o variabilidad) de los datos como sea posible. Este

es un enfoque estadístico.

En otras palabras, nos gustaría encontrar la mejor matriz creada con menos variables (es decir, una matriz de rango inferior) que explique los datos originales. Esto está relacionado con la compresión de datos.

Dos soluciones relacionadas a estos problemas son PCA, que significa Análisis de componentes principales y SVD, Descomposición de valores singulares. Esto último simplemente significa que expresamos una matriz X de observaciones (filas) y variables (columnas) como el producto de otras 3 matrices, es decir, X = UDV ˆ t. Este último término (V ˆ t) representa la transposición de la matriz V.

Aquí, U y V tienen columnas ortogonales (no correlacionadas). Las columnas de U son los vectores singulares de la izquierda de X y las columnas de V son los vectores singulares de la derecha de X. D es una matriz diagonal, por lo que queremos decir que todas sus entradas que no están en la diagonal son 0. Las entradas diagonales de D son el singular valores de X.

Para ilustrar esta idea, creamos una matriz de ejemplo simple llamada mat. Míralo ahora.

```
mat<-matrix(c(1,2,2,5,3,7),nrow = 2,ncol = 3)
```

Entonces mat es una matriz de 2 por 3. Por suerte para nosotros, R proporciona una función para realizar la descomposición de valores singulares. Se llama, como era de esperar, svd. Llámalo ahora con un solo argumento, mat.

```
a<-svd(mat)
a
```

```
## $d
## [1] 9.5899624 0.1806108
##
## $u
##            [,1]       [,2]
## [1,] -0.3897782 -0.9209087
## [2,] -0.9209087  0.3897782
##
## $v
##            [,1]       [,2]
## [1,] -0.2327012 -0.7826345
## [2,] -0.5614308  0.5928424
## [3,] -0.7941320 -0.1897921
```

Vemos que la función devuelve 3 componentes, d que contiene 2 elementos diagonales, u, una matriz de 2 por 2, yv, una matriz de 3 por 2. Almacenamos las entradas diagonales en una matriz diagonal para usted, diag, y también almacenamos uyv en las variables matu y matv respectivamente. Multiplique matu por diag por t (matv) para ver lo que obtiene. (Esta última expresión representa la transposición de matv en R). Recuerde que en R la multiplicación de matrices requiere que use el operador% *%.

```
diag<-matrix(c(9.589962,0,0,0.1806108),2,2)
matu <-a$u
matv<-a$v
matu %*% diag %*% t(matv)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    2    5    7
```

Ahora hablaremos un poco sobre PCA, Análisis de componentes principales, "un método simple y no paramétrico para extraer información relevante de conjuntos de datos confusos". Estamos citando aquí un artículo muy agradable y conciso sobre este tema que se puede encontrar en http://arxiv.org/pdf/1404.1100.pdf. El artículo de Jonathon Shlens de Google Research se llama Tutorial sobre análisis de componentes principales.

Básicamente, PCA es un método para reducir un conjunto de datos de alta dimensión a sus elementos esenciales (no perder información) y explicar la variabilidad en los datos. No entraremos en los detalles matemáticos aquí (R tiene una función para realizar PCA), pero debe saber que SVD y PCA están estrechamente relacionados.

Lo demostraremos ahora. Primero tenemos que escalar mat, nuestra matriz de datos de ejemplo simple. Esto significa que restamos la media de la columna de cada elemento y dividimos el resultado por la desviación estándar de la columna. Por supuesto, R tiene un comando, escala, que hace esto por ti. Run svd on scale of mat.

```
svd(scale(mat))
```

```
## $d
## [1] 1.732051 0.000000
##
## $u
##            [,1]      [,2]
## [1,] -0.7071068 0.7071068
## [2,]  0.7071068 0.7071068
##
## $v
##           [,1]       [,2]
## [1,] 0.5773503 -0.5773503
## [2,] 0.5773503  0.7886751
## [3,] 0.5773503 -0.2113249
```

Ahora ejecute el programa R prcomp en scale (mat). Esto le dará los componentes principales del tapete. Vea si le resultan familiares.

```
prcomp(scale(mat))
```

```
## Standard deviations (1, .., p=2):
## [1] 1.732051 0.000000
##
## Rotation (n x k) = (3 x 2):
##            PC1        PC2
## [1,] 0.5773503 -0.5773503
## [2,] 0.5773503  0.7886751
## [3,] 0.5773503 -0.2113249
```

Observe que los componentes principales de la matriz escalada, que se muestran en el componente Rotación de la salida de prcomp, SON las columnas de V, los valores singulares de la derecha. Por tanto, el PCA de una matriz escalada produce la matriz V (vectores singulares derechos) de la misma matriz escalada.

Ahora que cubrimos la teoría, regresemos a nuestra matriz más grande de datos aleatorios en la que habíamos agregado un patrón fijo para algunas filas seleccionadas por coinflips. El patrón cambió efectivamente los medios de las filas y columnas.

Aquí hay una imagen que muestra la relación entre PCA y SVD para esa matriz más grande. Hemos trazado 10 puntos (5 están agrupados en la esquina inferior izquierda). Las coordenadas x son los elementos del primer componente principal (salida de prcomp), y las coordenadas y son los elementos de la primera columna de V, el primer vector singular derecho (obtenido al ejecutar svd). Vemos que todos los puntos se encuentran en la línea de 45 grados representada por la ecuación y = x. Entonces, la primera columna de V ES el primer componente principal de nuestra matriz de datos más grande.

Para demostrar que no lo estamos inventando, ejecutamos svd en dataMatrix y almacenamos el resultado en el objeto svd1. Esto tiene 3 componentes, d, u y v. Mire ahora la primera columna de V. Se puede ver usando la notación svd1 $ v [, 1].

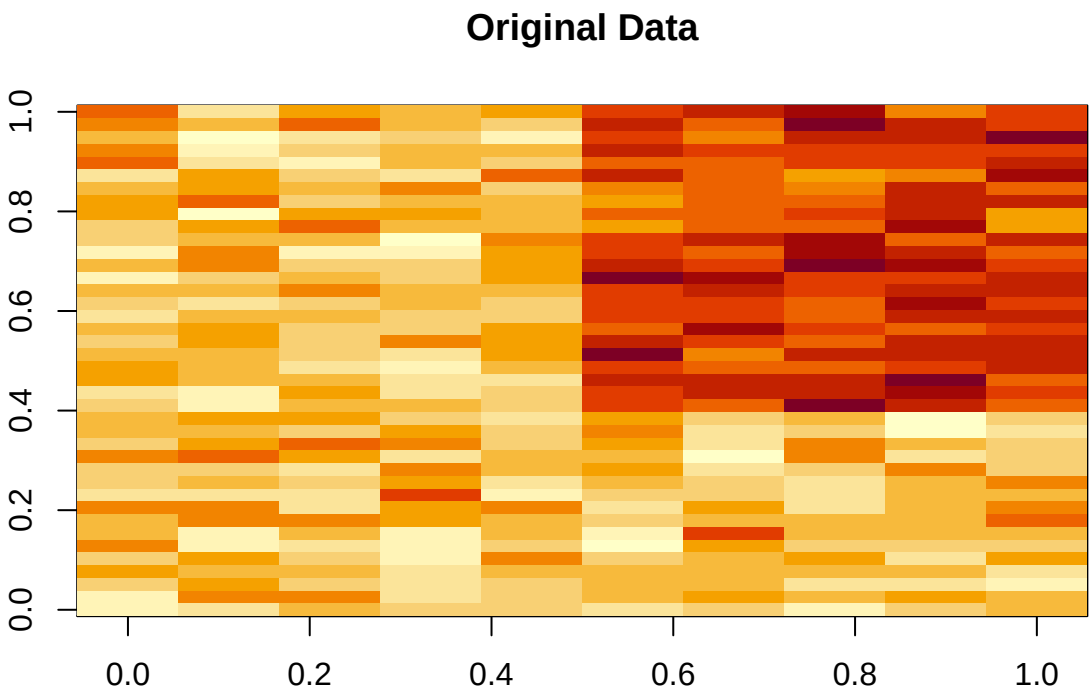
```
svd1<-svd(dataMatrix)
svd1$v[,1]
```

```
##  [1] -0.037286927  0.003956243 -0.029594100  0.026764952 -0.042066296
##  [6] -0.436054252 -0.405997744 -0.453694237 -0.480551794 -0.451094580
```

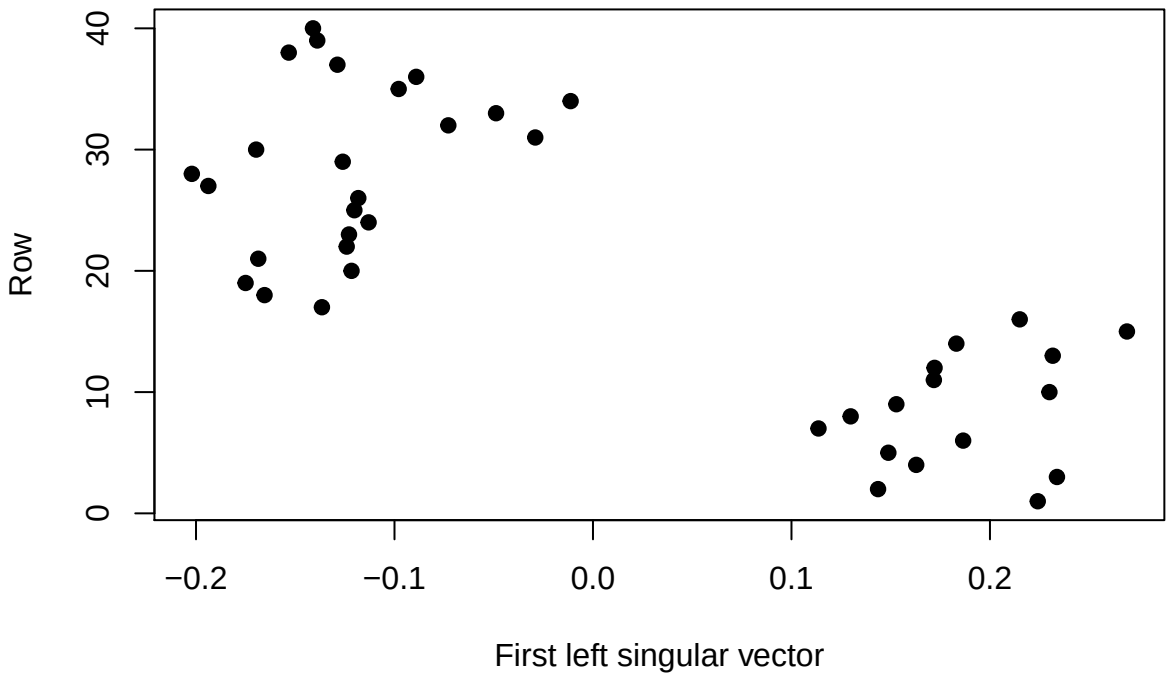
¿Ves cómo estos valores corresponden a los trazados? Cinco de las entradas están ligeramente a la izquierda del punto (-0.4, -0.4), dos más son negativas (a la izquierda de (0,0)) y tres son positivas (a la derecha de (0,0)).

Aquí mostramos nuevamente la matriz de datos agrupados a la izquierda. Junto a él, hemos trazado la primera columna de la matriz U asociada con la matriz de datos escalados. Este es el primer vector singular IZQUIERDO y está asociado con las medias FILA de los datos agrupados. Puede ver la clara separación entre las medias de las 24 filas superiores (alrededor de -0,2) y las 16 inferiores (alrededor de 0,2). No los mostramos, pero tenga en cuenta que las otras columnas de U no muestran este patrón con tanta claridad.

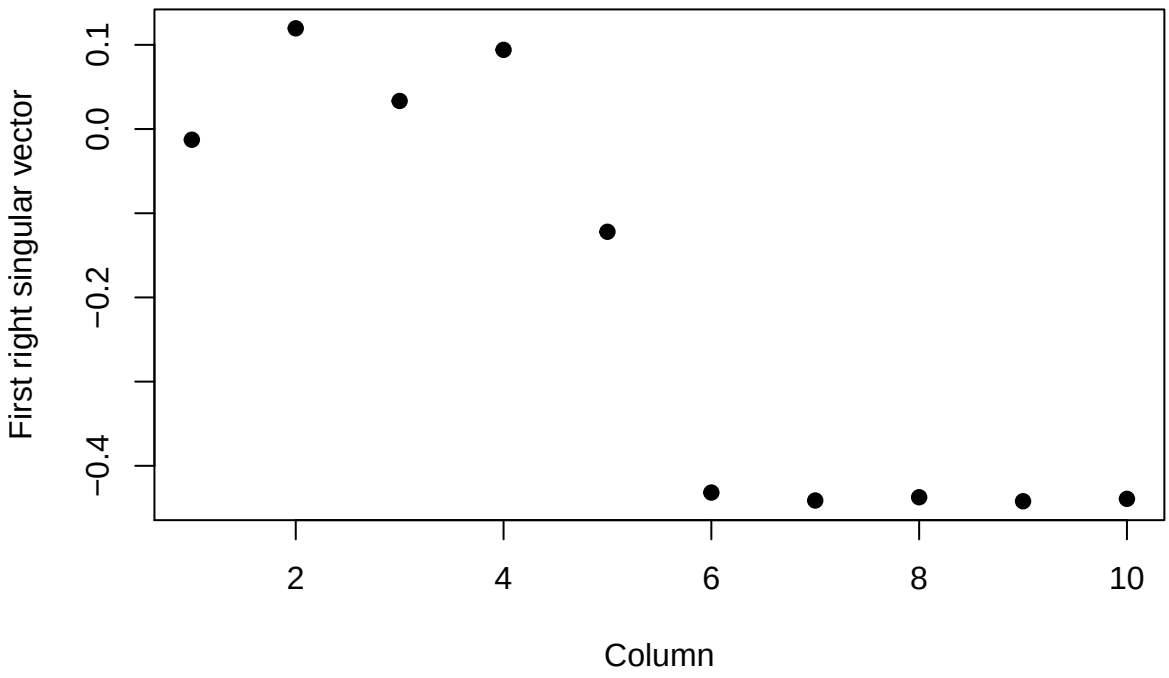
```
svd1 <- svd(scale(dataMatrixOrdered))
image(t(dataMatrixOrdered)[, nrow(dataMatrixOrdered):1], main = "Original Data")
```




```
plot(svd1$u[, 1], 40:1, , ylab = "Row", xlab = "First left singular vector", pch = 19)
```




```
plot(svd1$v[, 1], xlab = "Column", ylab = "First right singular vector", pch = 19)
```



La pantalla de la derecha muestra la primera columna de la matriz V asociada con la matriz de datos agrupados y escalados. Este es el primer vector singular DERECHO y está asociado con las medias COLUMNA de los datos agrupados. Puede ver la clara separación entre las medias de las 5 columnas de la izquierda (entre -0,1 y 0,1) y las medias de las 5 columnas de la derecha (todas por debajo de -0,4). Al igual que con los vectores singulares de la izquierda, las otras columnas de V no muestran este patrón tan claramente como lo hace este primero.

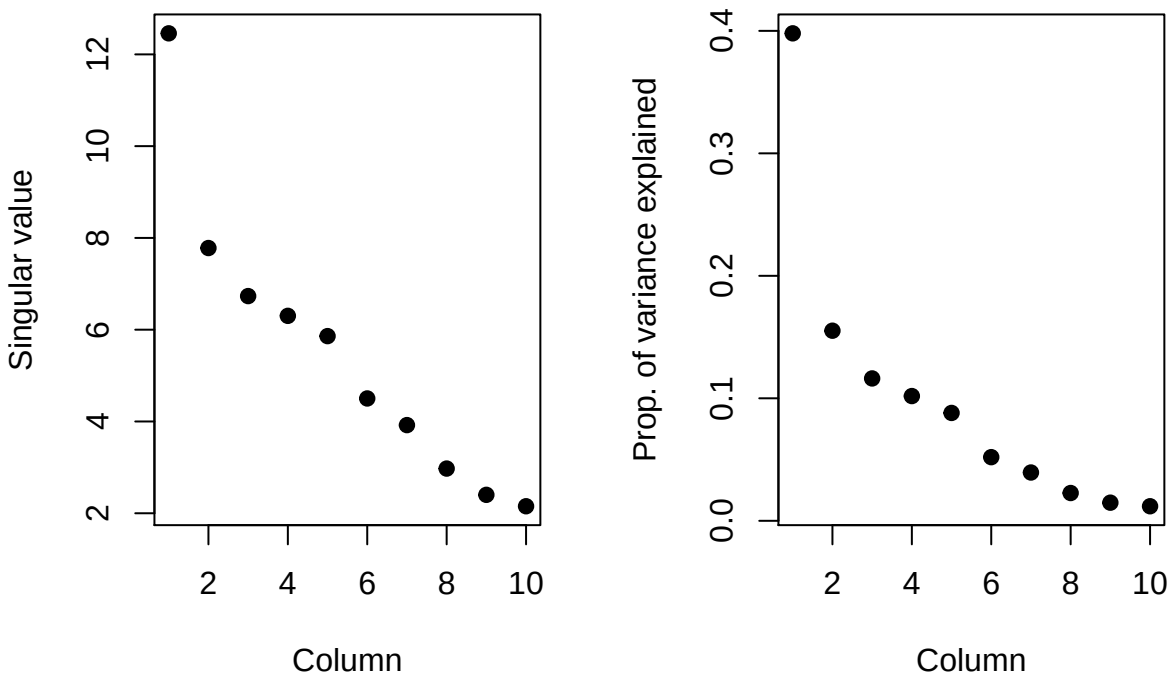
Entonces, la descomposición de valores singulares recogió automáticamente estos patrones, las diferencias en las medias de las filas y columnas.

¿Por qué las primeras columnas de las matrices U y V eran tan especiales? Pues da la casualidad de que la matriz D de la SVD explica este fenómeno. Es un aspecto de la SVD llamado varianza explicado. Recuerde que D es la matriz diagonal intercalada entre U y V ˆ t en la representación SVD de la matriz de datos. Las entradas diagonales de D son como pesos para las columnas U y V que tienen en cuenta la variación de los datos. Se dan en orden decreciente de mayor a menor. Mire estas entradas diagonales ahora. Recuerde que están almacenados en svd1 $ d.

```
svd1$d
```

```
##  [1] 12.458121  7.779798  6.732595  6.301878  5.860013  4.501826  3.921267
##  [8]  2.973909  2.401470  2.152848
```

```
 par(mfrow = c(1, 2))
svd1 <- svd(scale(dataMatrixOrdered))
plot(svd1$d, xlab = "Column", ylab = "Singular value", pch = 19)
plot(svd1$d^2/sum(svd1$d^2), xlab = "Column", ylab = "Prop. of variance explained", pch = 19)
```



Aquí hay una visualización de estos valores (a la izquierda). El primero (12,46) es significativamente más grande que los demás. Como no tenemos ninguna unidad especificada, a la derecha hemos trazado la proporción de la varianza que representa cada entrada. Vemos que la primera entrada representa aproximadamente el 40% de la varianza en los datos. Esto explica por qué las primeras columnas de las matrices U y V respectivamente mostraron los patrones distintivos en las medias de las filas y columnas con tanta claridad.

Ahora le mostraremos otro ejemplo simple de cómo la SVD explica la varianza. Hemos creado una matriz de 40 por 10, matriz constante. Utilice el encabezado de comando R con constantMatrix como argumento para ver las filas superiores.

```
constantMatrix <- dataMatrixOrdered * 0
for (i in 1:dim(dataMatrixOrdered)[1]) {
  constantMatrix[i, ] <- rep(c(0, 1), each = 5) }
svd2 <- svd(constantMatrix)
```

```
head(constantMatrix)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    0    0    0    0    0    1    1    1    1     1
## [2,]    0    0    0    0    0    1    1    1    1     1
## [3,]    0    0    0    0    0    1    1    1    1     1
## [4,]    0    0    0    0    0    1    1    1    1     1
## [5,]    0    0    0    0    0    1    1    1    1     1
## [6,]    0    0    0    0    0    1    1    1    1     1
```

El resto de las filas se parecen a estas. Puede ver que las 5 columnas de la izquierda son todas ceros y las 5

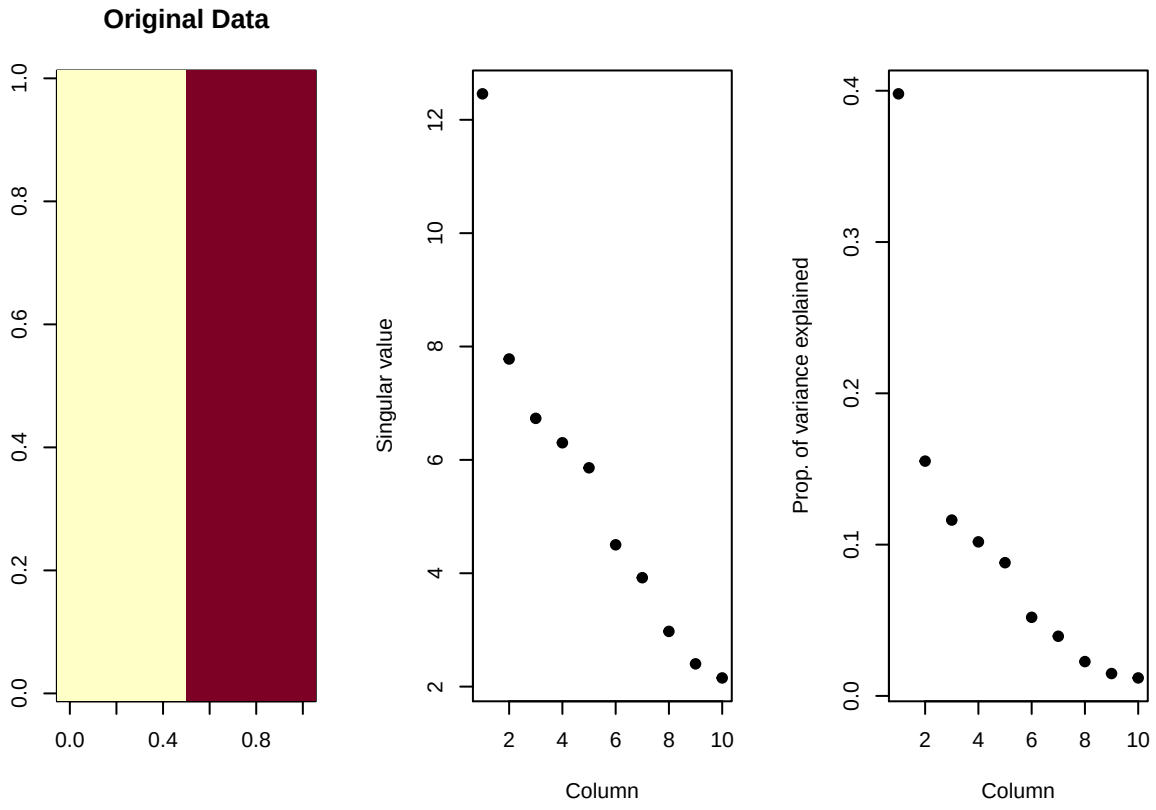
columnas de la derecha son todas 1. Ejecutamos svd con constantMatrix como su argumento y almacenamos el resultado en svd2. Mire el componente diagonal, d, de svd2 ahora

```
svd2$d
```

```
##  [1] 1.414214e+01 1.293147e-15 2.515225e-16 8.585184e-31 9.549693e-32
##  [6] 3.330034e-32 2.022600e-46 4.362170e-47 1.531252e-61 0.000000e+00
```

veamos la siguiente imagen

```
par(mfrow = c(1, 3))
image(t(constantMatrix)[, nrow(constantMatrix):1], main = "Original Data")
plot(svd1$d, xlab = "Column", ylab = "Singular value", pch = 19)
plot(svd1$d^2/sum(svd1$d^2), xlab = "Column", ylab = "Prop. of variance explained",pch = 19)
```



Entonces, la primera entrada domina con mucho a las demás. Aquí, la imagen de la izquierda muestra el mapa de calor de constantMatrix. Puede ver en qué se diferencian las columnas de la izquierda de las de la derecha. La gráfica del medio muestra los valores de los valores singulares de la matriz, es decir, los elementos diagonales que son las entradas de svd2 $ d. Nueve de estos son 0 y el primero está un poco por encima de 14. El tercer gráfico muestra la proporción del total que representa cada elemento diagonal.

Entonces, ¿qué significa esto? Básicamente, los datos son unidimensionales. Solo 1 pieza de información, es decir, en qué columna se encuentra una entrada, determina su valor.

Ahora regresemos a nuestra matriz de datos aleatoria de 40 por 10 y consideremos un ejemplo un poco más complicado en el que le agregamos 2 patrones. Nuevamente, elegiremos qué filas ajustar usando coinflips.

Específicamente, para cada una de las 40 filas lanzaremos 2 monedas. Si el primer coinflip es cara, agregaremos 5 a cada entrada en las 5 columnas de la derecha de esa fila, y si el segundo coinflip es cara, agregaremos 5 solo a las columnas pares de esa fila.

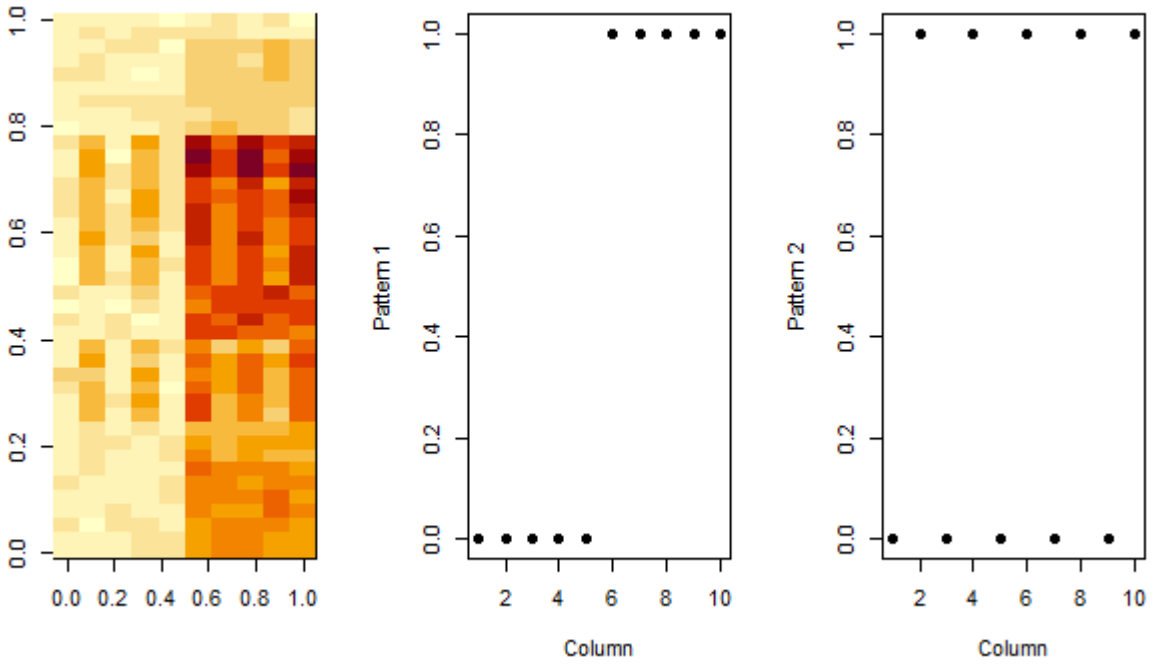


Figure 1: A caption

Así que aquí está la imagen de la matriz de datos escalada a la izquierda. Podemos ver ambos patrones, la clara diferencia entre las 5 columnas de la izquierda y las 5 de la derecha, pero también, un poco menos visible, el patrón alterno de las columnas. Los otros gráficos muestran los patrones reales que se agregaron a las filas afectadas. El gráfico del medio muestra la verdadera diferencia entre las columnas izquierda y derecha, mientras que el gráfico de la derecha muestra la verdadera diferencia entre los números impares y | columnas pares.

La pregunta es: "¿Puede nuestro análisis detectar estos patrones solo a partir de los datos?" Veamos qué muestra la SVD. Como estamos interesados en los patrones en las columnas, veremos los dos primeros vectores singulares derechos (columnas de V) para ver si muestran alguna evidencia de los patrones.

Aquí vemos los 2 vectores singulares de la derecha trazados junto a la imagen de la matriz de datos. La gráfica del medio muestra la primera columna de V y la gráfica de la derecha la segunda. La gráfica del medio muestra que las últimas 5 columnas tienen entradas más altas que las primeras 5. Esto recoge, o al menos alude, al primer patrón que agregamos en el que afectó las últimas 5 columnas de la matriz. El gráfico de la derecha, que muestra la segunda columna de V, parece más aleatorio. Sin embargo, una inspección más cercana muestra que las entradas se alternan o rebotan hacia arriba y hacia abajo a medida que se mueve de izquierda a derecha. Esto sugiere el segundo patrón que agregamos en el que afectó solo a columnas pares de filas seleccionadas.

Para ver esto más de cerca, observe las 2 primeras columnas del componente v. Almacenamos la salida de SVD en el objeto svd svd2.

```
svd2$v[,1:2]
```

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

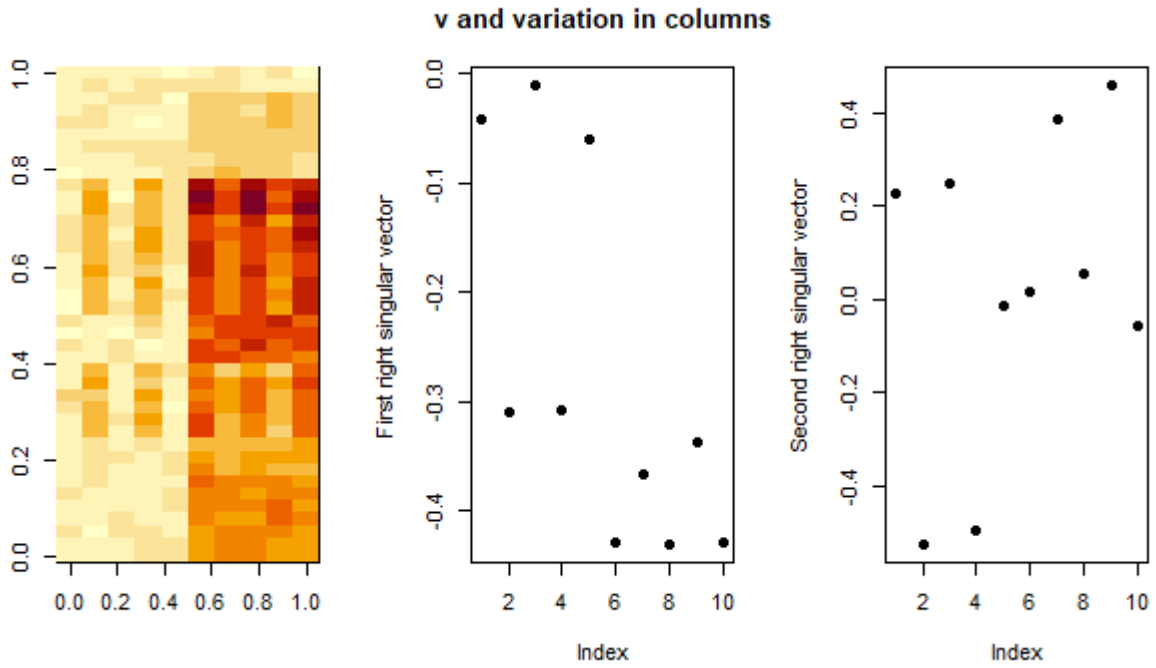


Figure 2: A caption

```
##  [1,]  0.000000e+00  0.000000e+00
##  [2,]  0.000000e+00 -9.853762e-02
##  [3,]  0.000000e+00  3.330669e-16
##  [4,]  1.388571e-47  0.000000e+00
##  [5,] -5.834077e-62  1.540744e-33
##  [6,]  4.472136e-01  8.900743e-01
##  [7,]  4.472136e-01 -2.225186e-01
##  [8,]  4.472136e-01 -2.225186e-01
##  [9,]  4.472136e-01 -2.225186e-01
## [10,]  4.472136e-01 -2.225186e-01
```

Al ver las 2 columnas una al lado de la otra, vemos que los valores en ambas columnas aumentan y disminuyen alternativamente. Sin embargo, sabíamos que debíamos buscar este patrón, por lo que es probable que no lo haya notado si no hubiera sabido si estaba allí. Este ejemplo está destinado a mostrarle que es difícil ver patrones, incluso los más sencillos.

Ahora mire las entradas de la matriz diagonal d que resulta del svd. Recuerde que almacenamos esta salida para usted en el objeto svd svd2.

```
svd2$d
```

```
##  [1] 1.414214e+01 1.293147e-15 2.515225e-16 8.585184e-31 9.549693e-32
##  [6] 3.330034e-32 2.022600e-46 4.362170e-47 1.531252e-61 0.000000e+00
```

Vemos que el primer elemento, 14,55, domina a los demás. Aquí está la gráfica de estos elementos diagonales de d. La izquierda muestra las entradas numéricas y la derecha muestra el porcentaje de varianza que explica cada entrada.

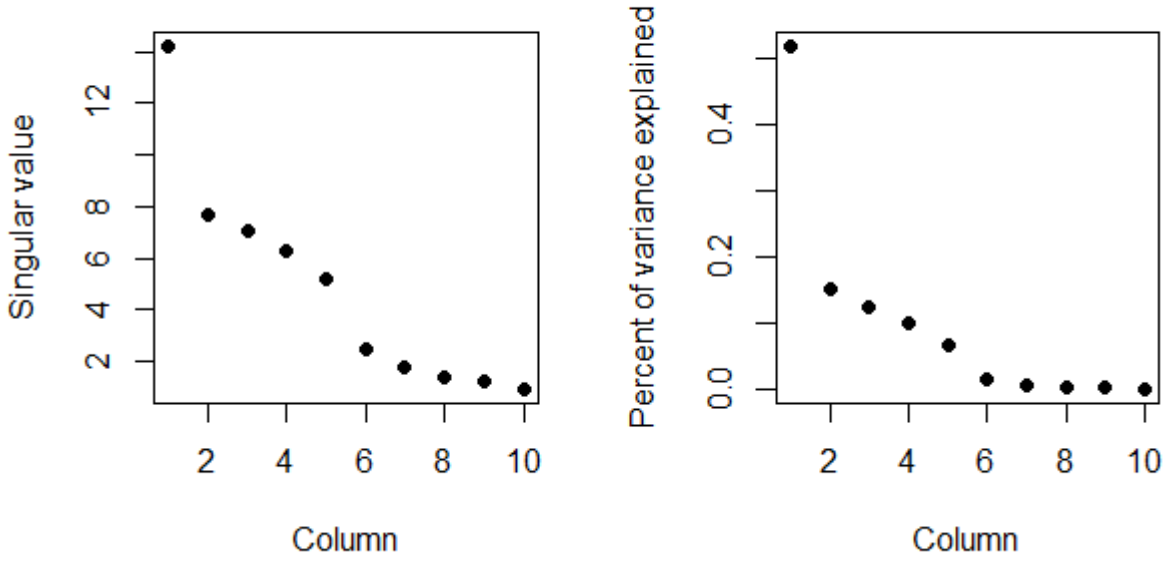


Figure 3: A caption

Entonces, el primer elemento que mostró la diferencia entre las mitades izquierda y derecha de la matriz representa aproximadamente el 50% de la variación en la matriz, y el segundo elemento que tomó el patrón alterno representa el 18% de la variación. Los elementos restantes representan porcentajes más pequeños de variación. Esto indica que el primer patrón es mucho más fuerte que el segundo. Además, los dos patrones se confunden, por lo que es más difícil separarlos y verlos con claridad. Esto es lo que suele ocurrir con los datos reales.

Ahora probablemente esté convencido de que SVD y PCA son geniales y útiles como herramientas de análisis, pero un problema con ellos que debe conocer es que no pueden lidiar con datos FALTANTES. Ninguno de ellos funcionará si falta algún dato en la matriz. (Obtendrá mensajes de error de R en rojo si lo intenta.) Los datos faltantes no son inusuales, por lo que afortunadamente tenemos formas de solucionar este problema. Uno que solo mencionaremos se llama imputación de datos.

Esto usa los k vecinos más cercanos para calcular los valores a usar en lugar de los datos faltantes. Es posible que desee especificar un número entero k que indique cuántos vecinos desea promediar para crear este valor de reemplazo. El paquete de bioconductores (http://bioconductor.org) tiene un paquete de imputación que puede utilizar para completar los datos que faltan. Una función específica en él es impute.knn.

Pasaremos ahora a un ejemplo final del poder de la descomposición de valores singulares y el análisis de componentes principales y cómo funcionan como técnica de compresión de datos.

Considere este archivo de imagen de baja resolución que muestra una cara. Usaremos SVD y veremos cómo los primeros componentes contienen la mayor parte de la información en el archivo, por lo que es posible que no sea necesario almacenar una matriz enorme.

Los datos de la imagen se almacenan en la matriz faceData. Ejecute el comando R dim en faceData para ver qué tan grande es.

```
faceData<-read.csv("C:/Users/luism/Documents/faceData.csv")
dim(faceData)
```

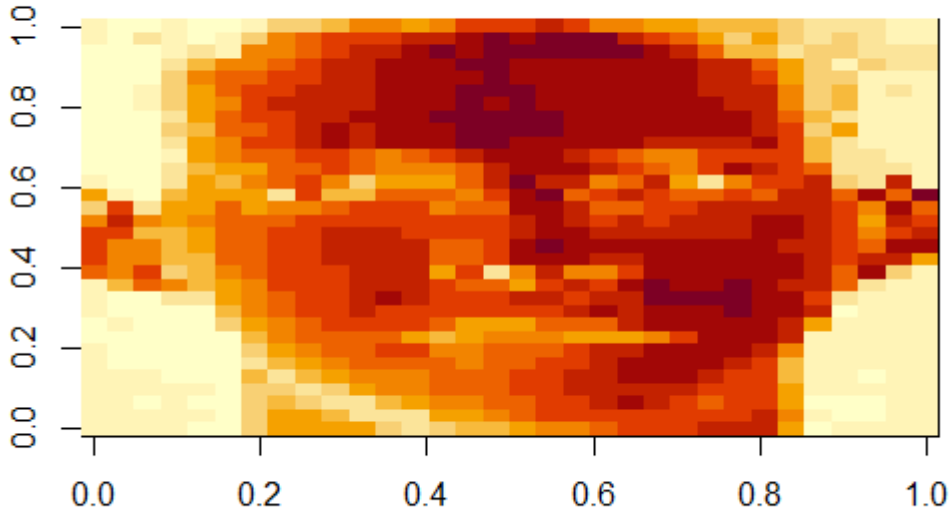
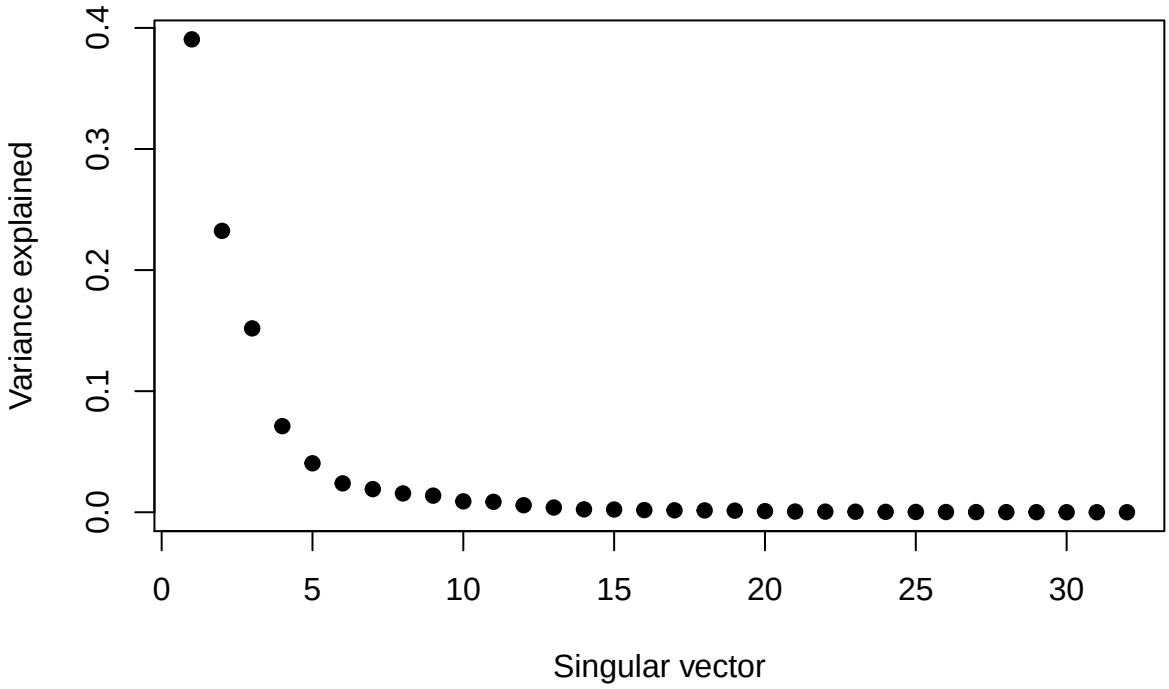


Figure 4: A caption

```
## [1] 32 33
```

```
svd1 <- svd(scale(faceData))
plot(svd1$d^2/sum(svd1$d^2), pch = 19, xlab = "Singular vector", ylab = "Variance explained")
```



Así que no es un archivo tan grande, pero queremos mostrarte cómo usar lo que aprendiste en esta lección. Hemos hecho el SVD y lo almacenamos en el objeto svd1 para usted. Aquí está la gráfica de la varianza explicada.2

Entonces, el 40% de la variación en la matriz de datos se explica por el primer componente, el 22% por el segundo, y así sucesivamente. Parece que la mayor parte de la variación está contenida en los primeros 10 componentes. ¿Cómo podemos comprobar esto? ¿Podemos intentar crear una imagen aproximada utilizando solo unos pocos componentes?

Recuerde que la matriz de datos X es el producto de 3 matrices, es decir, X = UDV ˆ t. Esto es precisamente lo que obtiene cuando ejecuta svd en la matriz X.

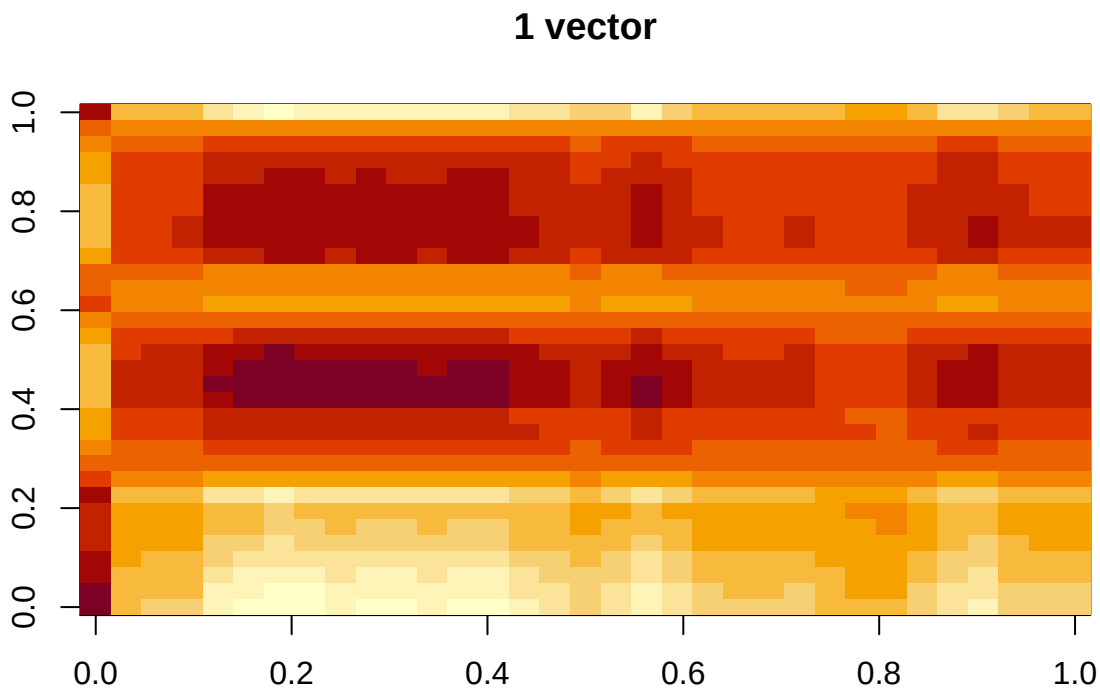
Supongamos que creamos el producto de piezas de estos, digamos las primeras columnas de U y V y el primer elemento de D. La primera columna de U se puede interpretar como una matriz de 32 por 1 (recuerde que faceData era una matriz de 32 por 32) , entonces podemos multiplicarlo por el primer elemento de D, una matriz de 1 por 1, y obtener un resultado de matriz de 32 por 1. Podemos multiplicar eso por la transposición de la primera columna de V, que es el primer componente principal. (Tenemos que usar la transposición de la columna de V para convertirla en una matriz de 1 por 32 para poder hacer la multiplicación de matrices correctamente).

Por desgracia, así es como lo hacemos en teoría, pero en R, usar solo un elemento de d significa que es una constante. Entonces tenemos que hacer la multiplicación de matrices con el operador% % *y la multiplicación por la constante (svd1 $ d [1]) con el operador de multiplicación regular* .

Pruebe esto ahora y ponga el resultado en la variable a1. Recuerde que svd1 $ u, svd1 $ d y svd1 $ v contienen toda la información que necesita. TENGA EN CUENTA que debido a las peculiaridades de la conversión de R, si primero hace la multiplicación escalar con el operador * (antes de la multiplicación de matriz con el operador% *%) DEBE encerrar los 2 argumentos (svd1 $ u [, 1] y svd1 $ d [1]) entre paréntesis.

```
a1 <- (svd1$u[,1] * svd1$d[1]) %*% t(svd1$v[,1])
```

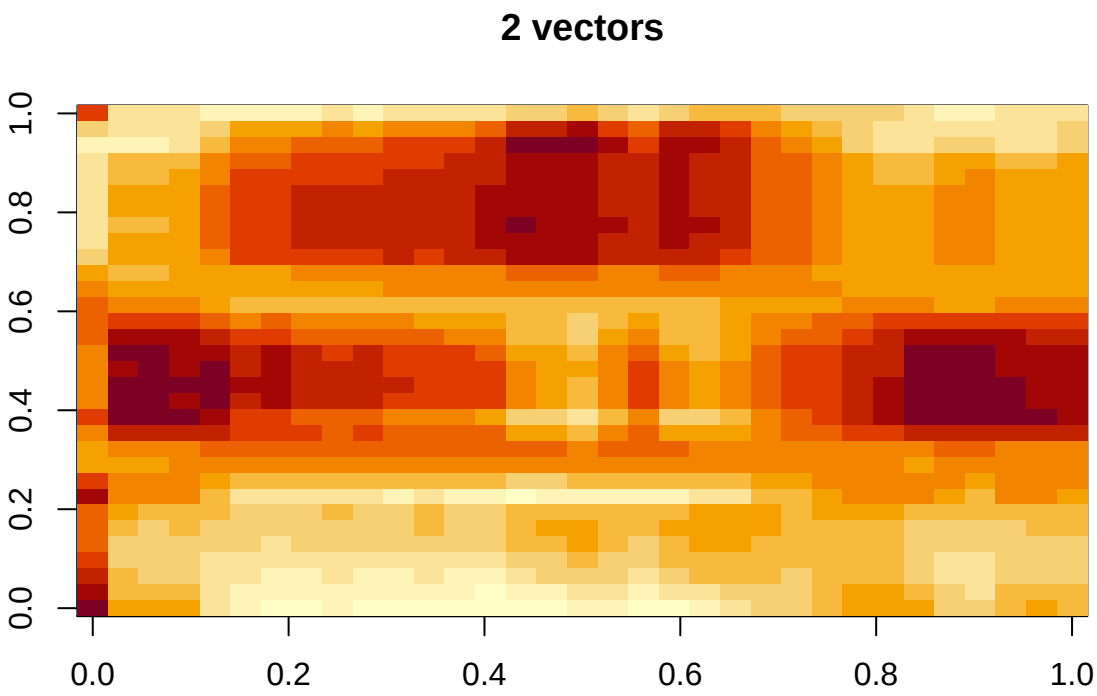
```
image(t(a1)[, nrow(a1):1], main = "1 vector")
```



Puede que no parezca mucho, pero es un buen comienzo. Ahora intentaremos el mismo experimento, pero esta vez usaremos 2 elementos de cada uno de los 3 términos SVD.

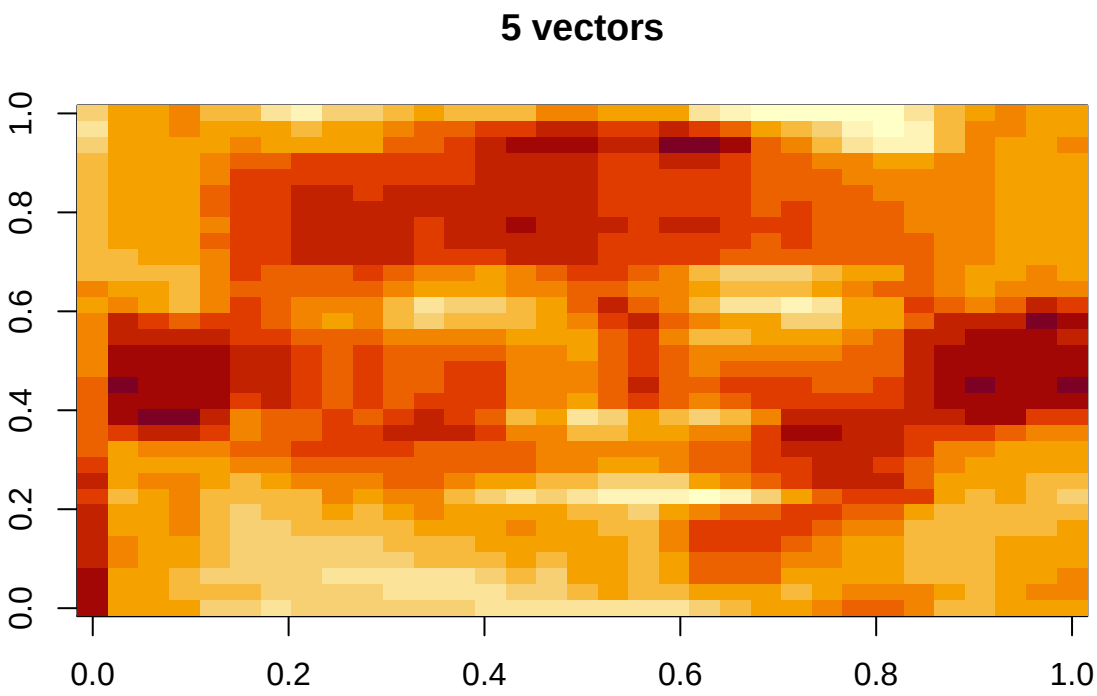
Cree la matriz a2 como el producto de las 2 primeras columnas de svd1 $ u, una matriz diagonal usando los 2 primeros elementos de svd1 $ d y la transposición de las 2 primeras columnas de svd1 $ v. Dado que todos sus multiplicandos son matrices, debe usar solo el operador% *% Y NO necesita paréntesis. Además, debe utilizar la función R diag con svd1 $ d [1: 2] como único argumento para crear la matriz diagonal adecuada. Recuerde, la multiplicación de matrices NO es conmutativa, por lo que debe colocar los multiplicandos en el orden correcto. Utilice la notación 1: 2 y no la c (m: n), es decir, la función de concatenar, al especificar las columnas.

```
a2 <- svd1$u[,1:2] %*% diag(svd1$d[1:2]) %*% t(svd1$v[,1:2])
image(t(a2)[, nrow(a2):1], main = "2 vectors")
```



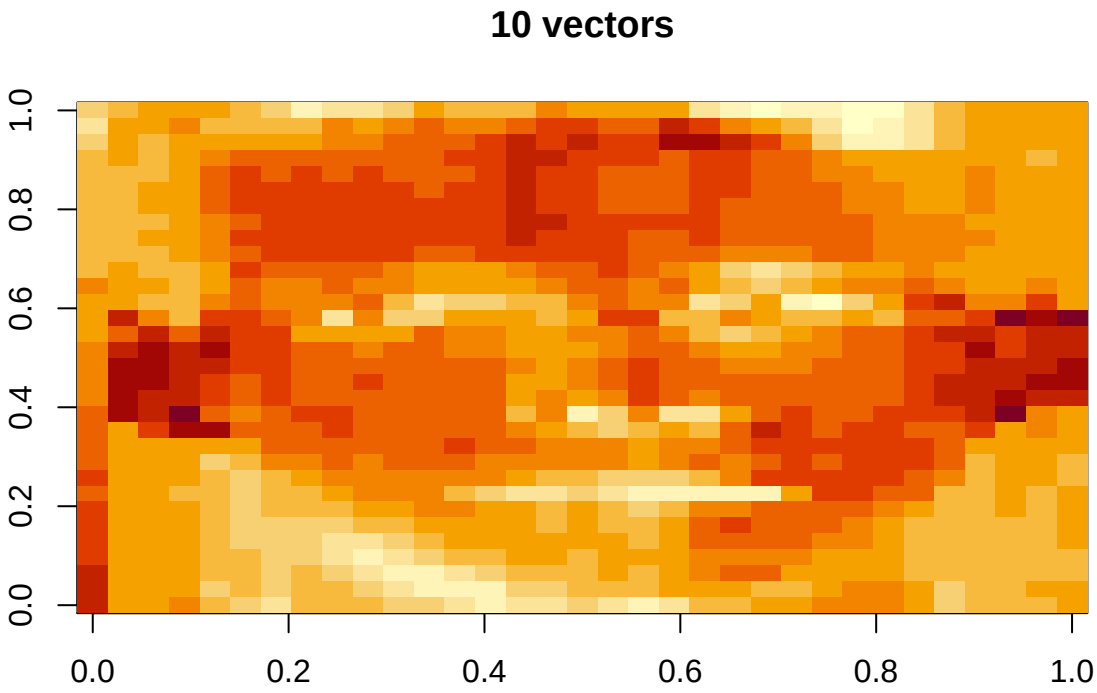
Estamos empezando a ver un poco más de detalle, y tal vez si entrecierras los ojos veas una mueca en la boca. Ahora veamos qué imagen resulta con 5 componentes. De nuestra gráfica de la varianza explicada, 5 componentes cubrieron un porcentaje considerable de la variación. Para guardar la escritura, use la flecha hacia arriba para recuperar el comando que creó a2 y reemplace a2 y la flecha de asignación con la llamada a myImage, y cambie las tres ocurrencias de 2 a 5.

```
a5 <- svd1$u[,1:5] %*% diag(svd1$d[1:5]) %*% t(svd1$v[,1:5])
image(t(a5)[, nrow(a5):1], main = "5 vectors")
```



Ciertamente mucho mejor. Claramente aparece una cara con ojos, nariz, oídos y boca reconocibles. Nuevamente, use la flecha hacia arriba para recuperar el último comando (llamando a myImage con un argumento de producto de matriz) y cambie los 5 por 10. Veremos cómo se ve esta imagen.

```
a10 <- svd1$u[,1:10] %*% diag(svd1$d[1:10]) %*% t(svd1$v[,1:10])
image(t(a10)[, nrow(a10):1], main = "10 vectors")
```



Eso es bastante parecido al original, que era de baja resolución para empezar, pero puede ver que 10 componentes realmente capturan la esencia de la imagen. La descomposición de valores singulares es una buena forma de aproximar datos sin tener que almacenar mucho.

Cerraremos ahora con algunos comentarios. Primero, al reducir las dimensiones, debe prestar atención a las escalas en las que se miden las diferentes variables y asegurarse de que todos sus datos estén en unidades consistentes. En otras palabras, las escalas de sus datos son importantes. En segundo lugar, los componentes principales y los valores singulares pueden mezclar patrones reales, como vimos en nuestro ejemplo simple de 2 patrones, por lo que encontrar y separar los patrones reales requiere algo de trabajo de detective. Hagamos una revisión rápida ahora.