DELVALLE
GRUPO EDUCATIVO

tarea2

Grupo #8
Osman Emanuel de León García - 23428
María José Yee Vidal - 231193
Andrés Rafael Chivalán Marroquín - 21534

2026-02-24

Introducción

El aprendizaje no supervisado es una de las ramas principales de la minería de datos y del aprendizaje automático. Su objetivo es descubrir patrones, estructuras o relaciones ocultas en los datos sin utilizar etiquetas o resultados previamente conocidos.

Estas técnicas se emplean principalmente para la exploración y comprensión de grandes volúmenes de información, permitiendo tareas como la segmentación de elementos similares, la reducción de la complejidad de los datos, la detección de anomalías y el descubrimiento de asociaciones frecuentes. Su importancia radica en que facilita la generación de conocimiento a partir de datos no estructurados o sin clasificar, apoyando la toma de decisiones y la identificación de patrones que no son evidentes a simple vista.

Objetivos

- Diferenciar los diferentes tipos de formas
- Aprender cuando aplicar los diferentes algoritmos
- Aprender a interpretar las respuestas
- Aprender a filtrar las respuestas

Cargar librerías

```
library(dplyr)
library(tidyr)
library(Matrix)
library(ggplot2)
library(Rtsne)
library(umap)
library(edfReader)
library(fastICA)
```

Cargar el dataset

El archivo `u.data` contiene los ratings en formato:

user_id | item_id | rating | timestamp

```
ratings <- read.table("data/u.data",
                      sep = "\t",
                      header = FALSE)

colnames(ratings) <- c("user_id", "item_id", "rating", "timestamp")

head(ratings)
```

```
##   user_id item_id rating timestamp
## 1     196     242      3 881250949
## 2     186     302      3 891717742
## 3      22     377      1 878887116
## 4     244      51      2 880606923
## 5     166     346      1 886397596
## 6     298     474      4 884182806
```

```
dim(ratings)
```

```
## [1] 100000      4
```

Construcción de la matriz Usuario × Película

Para aplicar SVD es necesario convertir los datos en una matriz donde:

- Filas → usuarios
- Columnas → películas
- Valores → ratings

```
rating_matrix <- ratings %>%
  select(user_id, item_id, rating) %>%
  pivot_wider(names_from = item_id,
              values_from = rating,
              values_fill = 0)

user_ids <- rating_matrix$user_id

rating_matrix <- as.matrix(rating_matrix[,-1])

dim(rating_matrix)
```

```
## [1]  943 1682
```

Aplicación de SVD

```
svd_result <- svd(rating_matrix)

str(svd_result)
```

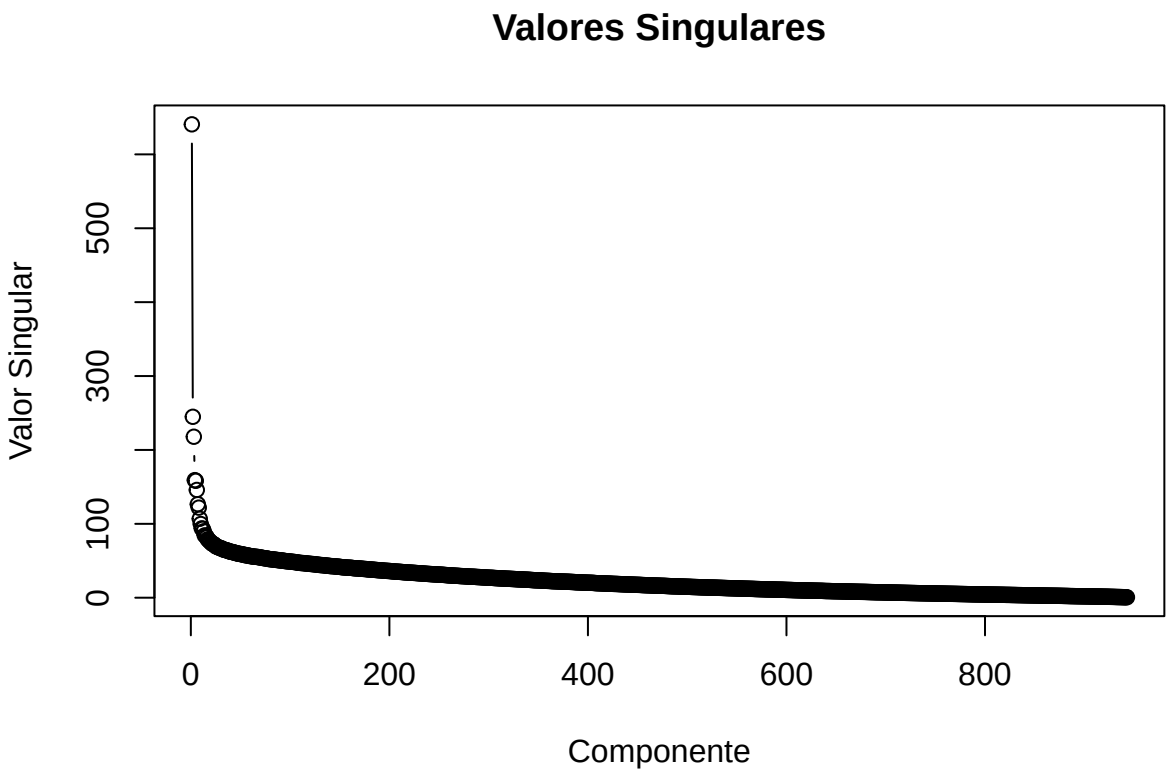
```
## List of 3
##  $ d: num [1:943] 641 245 218 159 158 ...
##  $ u: num [1:943, 1:943] 0.00853 0.01678 0.03319 0.05149 0.00376 ...
##  $ v: num [1:1682, 1:943] 0.01714 0.05084 0.00232 0.02539 0.02081 ...
```

El resultado contiene:

- U → factores asociados a usuarios
- D → valores singulares (importancia de componentes)
- V → factores asociados a películas

Visualización de valores singulares

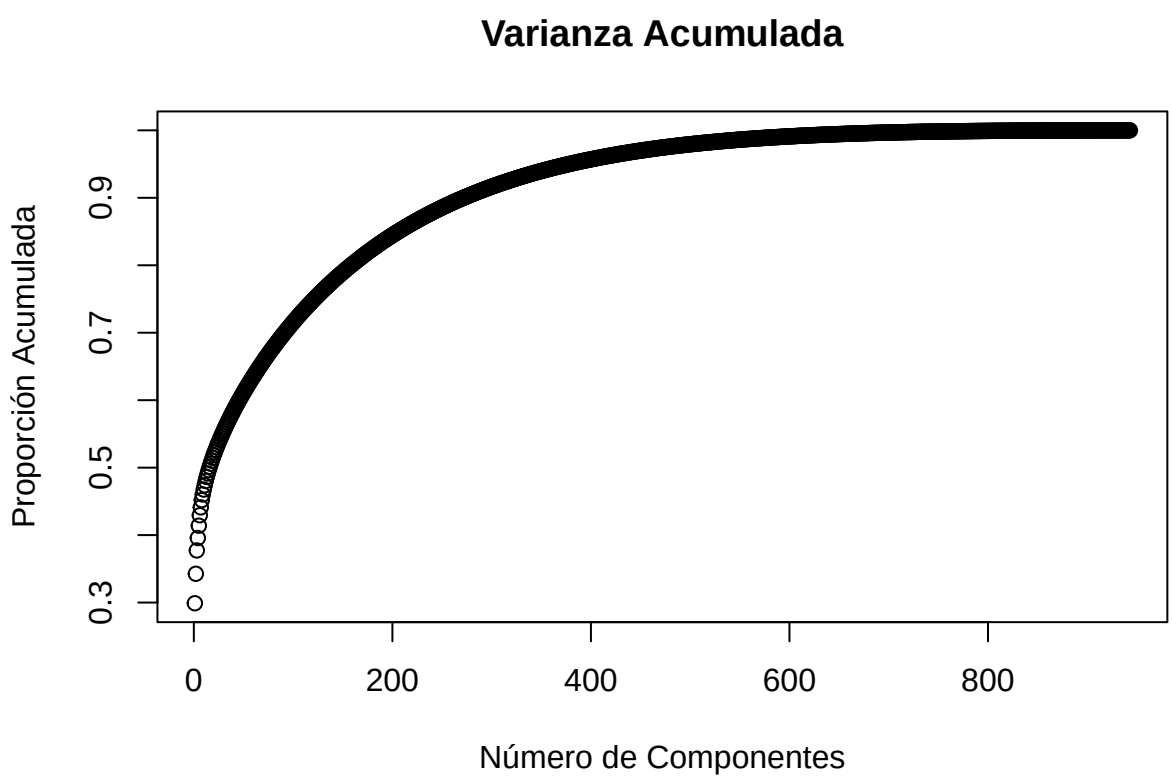
```
plot(svd_result$d,
     type = "b",
     main = "Valores Singulares",
     xlab = "Componente",
     ylab = "Valor Singular")
```



Los primeros valores singulares concentran mayor información.

Varianza explicada

```
variance_explained <- svd_result$d^2 / sum(svd_result$d^2)

plot(cumsum(variance_explained),
     type = "b",
     main = "Varianza Acumulada",
     xlab = "Número de Componentes",
     ylab = "Proporción Acumulada")
```




```
# ¿Cuántos componentes explican 80% y 90%?

components_80 <- which(cumsum(variance_explained) >= 0.80)[1]
components_90 <- which(cumsum(variance_explained) >= 0.90)[1]

components_80
```

```
## [1] 159
```

```
components_90
```

```
## [1] 272
```

Esta gráfica permite identificar cuántos componentes explican la mayor parte de la variabilidad del sistema.

Reducción de dimensionalidad (SVD truncado)

Seleccionamos k componentes principales.

```
k <- 20

U_k <- svd_result$u[,1:k]
D_k <- diag(svd_result$d[1:k])
V_k <- svd_result$v[,1:k]

approx_matrix <- U_k %*% D_k %*% t(V_k)

dim(approx_matrix)
```

```
## [1]  943 1682
```

Error de reconstrucción

```
reconstruction_error <- norm(rating_matrix - approx_matrix, type = "F")

reconstruction_error
```

```
## [1] 812.8734
```

Un menor error indica mejor aproximación usando k componentes.

Comparación parcial

```
original_sample <- rating_matrix[1:5,1:5]
approx_sample <- approx_matrix[1:5,1:5]

original_sample
```

```
##      242 302 377 51 346
## [1,]   3   0   0  0   0
## [2,]   0   3   0  0   0
## [3,]   0   0   1  0   0
## [4,]   0   0   0  2   0
## [5,]   0   0   0  0   1
```

```
approx_sample
```

```
##            [,1]       [,2]         [,3]         [,4]        [,5]
## [1,]  0.7812385  0.9246899  0.028632014  0.053027436  0.02573357
## [2,]  0.4013566  1.0595446  0.020533813  0.367327869  0.24444772
## [3,]  0.2933030  0.2532724  0.090011330 -0.130733185 -0.10310155
## [4,] -0.3766809 -0.8599377 -0.005875941  2.191204870  0.15378501
## [5,] -0.0871198  1.5710042  0.005971020  0.002561477  0.54427996
```

Interpretación de resultados

La gráfica de los valores singulares muestra que los primeros componentes tienen valores mucho mayores que el resto. Esto indica que una gran parte de la información del sistema de calificaciones está concentrada en pocas dimensiones. En otras palabras, aunque el dataset tiene muchas películas, las preferencias de los usuarios pueden resumirse en un número menor de factores importantes.

En la gráfica de varianza acumulada se observa que los primeros componentes explican un alto porcentaje de la variabilidad total. Esto significa que no es necesario usar todas las dimensiones originales para representar adecuadamente la información, ya que una parte significativa puede capturarse con menos componentes.

Al reconstruir la matriz utilizando solo 20 componentes, se obtiene un error de reconstrucción moderado. Esto indica que existe cierta pérdida de información, pero la estructura general del sistema de ratings se mantiene. Esto demuestra que el comportamiento de los usuarios puede representarse mediante factores latentes que resumen patrones de preferencia.

t-SNE

Cargar dataset

```r
data <- read.csv("data/data.csv")

head(data)
```

```
##         id diagnosis radius_mean texture_mean perimeter_mean area_mean
## 1   842302         M       17.99        10.38         122.80    1001.0
## 2   842517         M       20.57        17.77         132.90    1326.0
## 3 84300903         M       19.69        21.25         130.00    1203.0
## 4 84348301         M       11.42        20.38          77.58     386.1
## 5 84358402         M       20.29        14.34         135.10    1297.0
## 6   843786         M       12.45        15.70          82.57     477.1
##   smoothness_mean compactness_mean concavity_mean concave.points_mean
## 1         0.11840          0.27760         0.3001             0.14710
## 2         0.08474          0.07864         0.0869             0.07017
## 3         0.10960          0.15990         0.1974             0.12790
## 4         0.14250          0.28390         0.2414             0.10520
## 5         0.10030          0.13280         0.1980             0.10430
## 6         0.12780          0.17000         0.1578             0.08089
##   symmetry_mean fractal_dimension_mean radius_se texture_se perimeter_se
## 1        0.2419                0.07871    1.0950     0.9053        8.589
## 2        0.1812                0.05667    0.5435     0.7339        3.398
## 3        0.2069                0.05999    0.7456     0.7869        4.585
## 4        0.2597                0.09744    0.4956     1.1560        3.445
## 5        0.1809                0.05883    0.7572     0.7813        5.438
## 6        0.2087                0.07613    0.3345     0.8902        2.217
##   area_se smoothness_se compactness_se concavity_se concave.points_se
## 1  153.40      0.006399        0.04904      0.05373           0.01587
## 2   74.08      0.005225        0.01308      0.01860           0.01340
## 3   94.03      0.006150        0.04006      0.03832           0.02058
## 4   27.23      0.009110        0.07458      0.05661           0.01867
## 5   94.44      0.011490        0.02461      0.05688           0.01885
## 6   27.19      0.007510        0.03345      0.03672           0.01137
##   symmetry_se fractal_dimension_se radius_worst texture_worst perimeter_worst
## 1     0.03003             0.006193        25.38         17.33          184.60
## 2     0.01389             0.003532        24.99         23.41          158.80
## 3     0.02250             0.004571        23.57         25.53          152.50
## 4     0.05963             0.009208        14.91         26.50           98.87
## 5     0.01756             0.005115        22.54         16.67          152.20
## 6     0.02165             0.005082        15.47         23.75          103.40
##   area_worst smoothness_worst compactness_worst concavity_worst
## 1     2019.0           0.1622            0.6656          0.7119
## 2     1956.0           0.1238            0.1866          0.2416
## 3     1709.0           0.1444            0.4245          0.4504
## 4      567.7           0.2098            0.8663          0.6869
## 5     1575.0           0.1374            0.2050          0.4000
## 6      741.6           0.1791            0.5249          0.5355
##   concave.points_worst symmetry_worst fractal_dimension_worst  X
## 1               0.2654         0.4601                0.11890 NA
## 2               0.1860         0.2750                0.08902 NA
## 3               0.2430         0.3613                0.08758 NA
## 4               0.2575         0.6638                0.17300 NA
## 5               0.1625         0.2364                0.07678 NA
## 6               0.1741         0.3985                0.12440 NA
```

```r
dim(data)
```

```
## [1] 569  33
```

Limpieza de datos

Eliminamos columnas que no aportan información al modelo.

```r
# Ver nombres de columnas
colnames(data)
```

```
##  [1] "id"                      "diagnosis"
##  [3] "radius_mean"             "texture_mean"
##  [5] "perimeter_mean"          "area_mean"
##  [7] "smoothness_mean"         "compactness_mean"
##  [9] "concavity_mean"          "concave.points_mean"
## [11] "symmetry_mean"           "fractal_dimension_mean"
## [13] "radius_se"               "texture_se"
## [15] "perimeter_se"            "area_se"
## [17] "smoothness_se"           "compactness_se"
## [19] "concavity_se"            "concave.points_se"
## [21] "symmetry_se"             "fractal_dimension_se"
## [23] "radius_worst"            "texture_worst"
## [25] "perimeter_worst"         "area_worst"
## [27] "smoothness_worst"        "compactness_worst"
## [29] "concavity_worst"         "concave.points_worst"
## [31] "symmetry_worst"          "fractal_dimension_worst"
## [33] "X"
```

```r
# Eliminar columna id
data <- data %>%
  select(-id)

# Eliminar columnas completamente vacías (si existen)
data <- data[, colSums(is.na(data)) < nrow(data)]
diagnosis <- data$diagnosis

data_numeric <- data %>%
  select(-diagnosis)
```

Escalamiento de datos

```r
data_scaled <- scale(data_numeric)
```

Aplicación de t-SNE

```r
set.seed(123)

tsne_result <- Rtsne(data_scaled,
                     dims = 2,
                     perplexity = 30,
                     verbose = TRUE,
                     max_iter = 500)
```

```
## Performing PCA
```

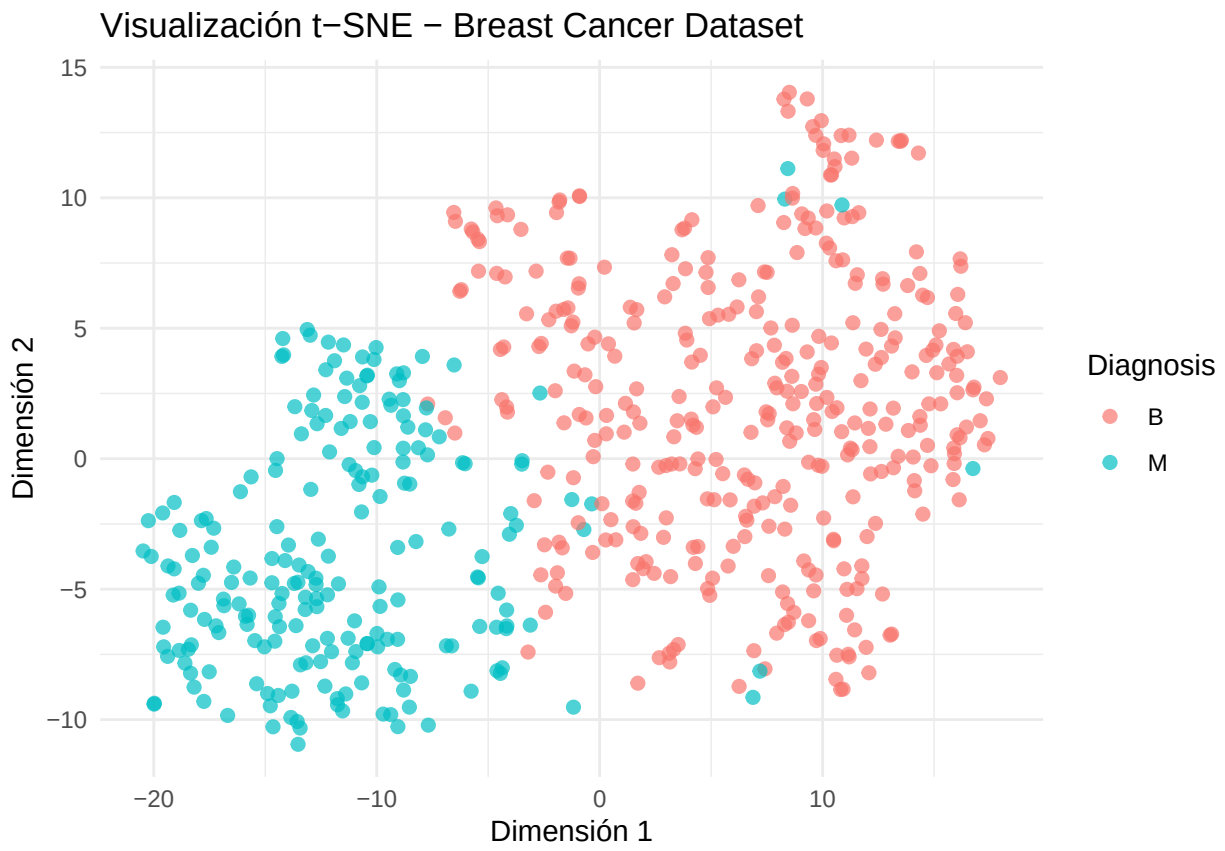
```
## Read the 569 x 30 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 30.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
## Done in 0.17 seconds (sparsity = 0.224116)!
## Learning embedding...
## Iteration 50: error is 62.155687 (50 iterations in 0.09 seconds)
## Iteration 100: error is 59.863724 (50 iterations in 0.07 seconds)
## Iteration 150: error is 59.869459 (50 iterations in 0.05 seconds)
## Iteration 200: error is 59.865602 (50 iterations in 0.04 seconds)
## Iteration 250: error is 59.873559 (50 iterations in 0.06 seconds)
## Iteration 300: error is 1.098313 (50 iterations in 0.05 seconds)
## Iteration 350: error is 1.026303 (50 iterations in 0.05 seconds)
## Iteration 400: error is 1.004124 (50 iterations in 0.05 seconds)
## Iteration 450: error is 0.995080 (50 iterations in 0.05 seconds)
## Iteration 500: error is 0.991102 (50 iterations in 0.05 seconds)
## Fitting performed in 0.55 seconds.
```

Crear dataframe para visualización

```
tsne_df <- data.frame(
  Dim1 = tsne_result$Y[,1],
  Dim2 = tsne_result$Y[,2],
  Diagnosis = diagnosis
)
```

Visualización

```
ggplot(tsne_df, aes(x = Dim1, y = Dim2, color = Diagnosis)) +
  geom_point(size = 2, alpha = 0.7) +
  labs(title = "Visualización t-SNE - Breast Cancer Dataset",
       x = "Dimensión 1",
       y = "Dimensión 2") +
  theme_minimal()
```



Interpretación de resultados

La visualización obtenida mediante t-SNE muestra una clara tendencia a la formación de dos grupos principales, correspondientes a los diagnósticos benigno (B) y maligno (M). Esto indica que las variables del dataset contienen información suficiente para diferenciar ambos tipos de tumores.

Se observa que los puntos malignos tienden a concentrarse en una región específica del espacio bidimensional, mientras que los benignos ocupan otra zona distinta. Aunque existe cierto solapamiento entre ambos grupos, la separación general es evidente.

Dado que t-SNE preserva relaciones locales entre observaciones, esta agrupación sugiere que los pacientes con características similares tienden a compartir el mismo diagnóstico. Esto demuestra la utilidad del algoritmo como herramienta de visualización para explorar la estructura interna de datos de alta dimensionalidad.

Aplicación de UMAP

```
set.seed(123)

umap_result <- umap(data_scaled)
```

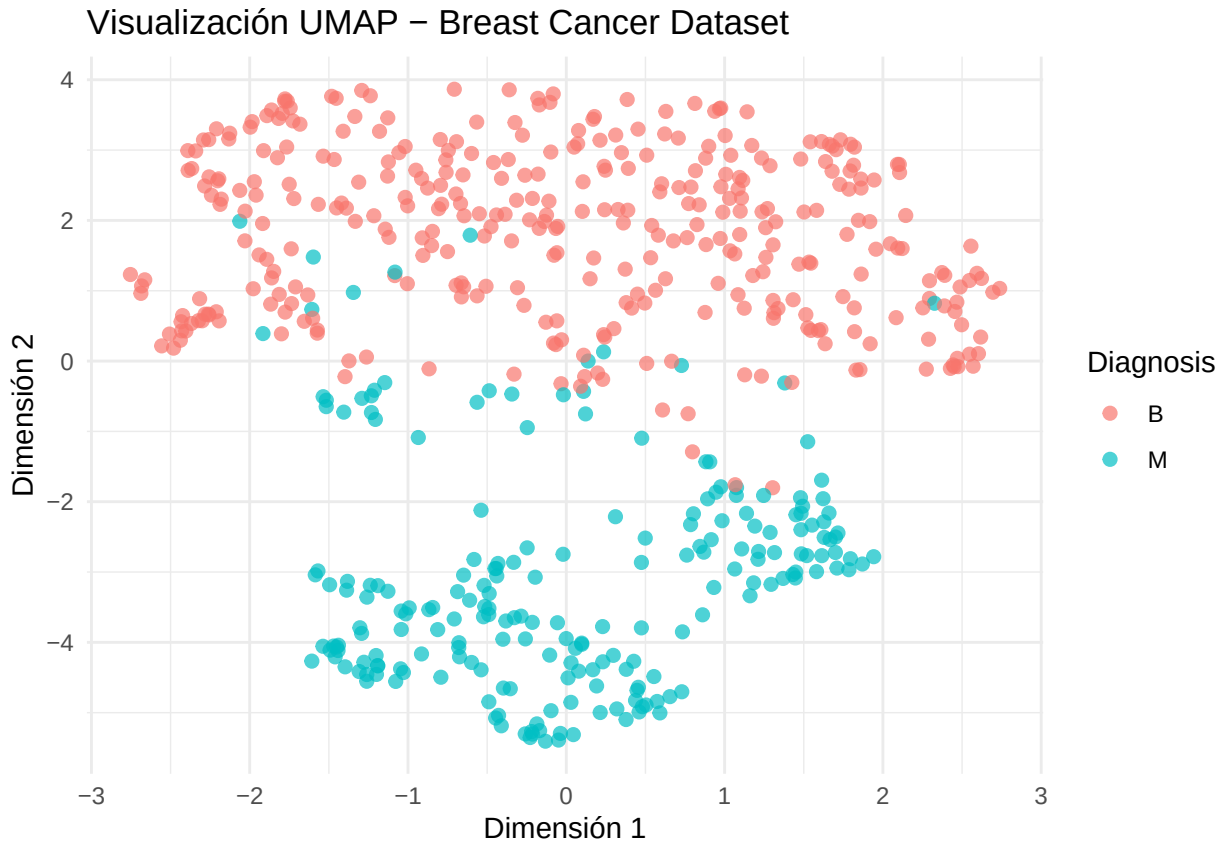
Crear dataframe para visualización

```
umap_df <- data.frame(
  Dim1 = umap_result$layout[,1],
```

```
  Dim2 = umap_result$layout[,2],
  Diagnosis = diagnosis
)
```

Visualización

```
ggplot(umap_df, aes(x = Dim1, y = Dim2, color = Diagnosis)) +
  geom_point(size = 2, alpha = 0.7) +
  labs(title = "Visualización UMAP - Breast Cancer Dataset",
       x = "Dimensión 1",
       y = "Dimensión 2") +
  theme_minimal()
```



Interpretación de resultados

La proyección obtenida mediante UMAP muestra una separación clara entre los casos benignos y malignos. A diferencia de t-SNE, UMAP tiende a preservar no solo las relaciones locales entre puntos, sino también parte de la estructura global del conjunto de datos.

En la visualización se observa que los dos grupos presentan menor solapamiento y una estructura más compacta, lo que sugiere que UMAP captura de forma eficiente la geometría interna de los datos.

Esto confirma que las características clínicas medidas en el dataset permiten diferenciar ambos diagnósticos y

demuestra que UMAP es una herramienta efectiva para la reducción de dimensionalidad y visualización de datos complejos.

Cargar señal ECG convertida

```
ecg <- read.csv("data/100_ecg.csv")

dim(ecg)
```

```
## [1] 650000      2
```

```
head(ecg)
```

```
##       X0     X1
## 1 -0.145 -0.065
## 2 -0.145 -0.065
## 3 -0.145 -0.065
## 4 -0.145 -0.065
## 5 -0.145 -0.065
## 6 -0.145 -0.065
```

Seleccionar una porción de la señal

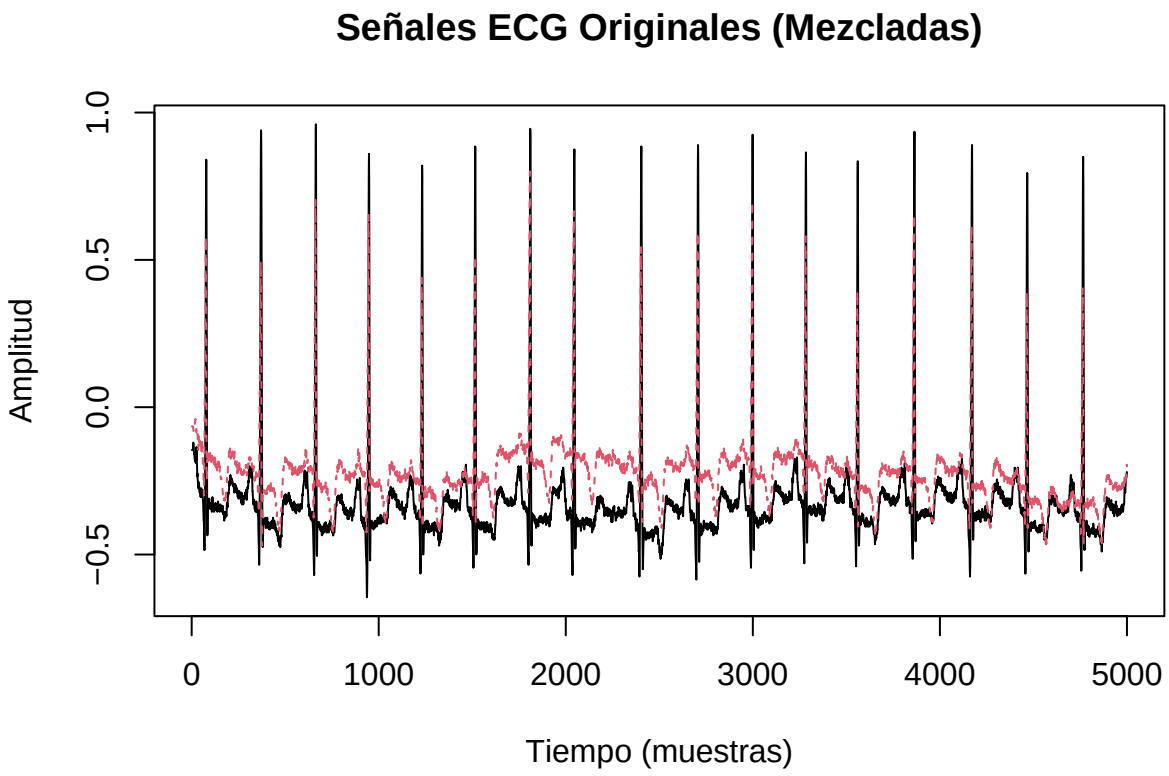
(Usamos solo las primeras 5000 muestras para simplificar el análisis)

```
signals <- as.matrix(ecg[1:5000, ])

dim(signals)
```

```
## [1] 5000    2
```

Visualizar señales originales

```
matplot(signals, type="l",
        main="Señales ECG Originales (Mezcladas)",
        xlab="Tiempo (muestras)",
        ylab="Amplitud")
```



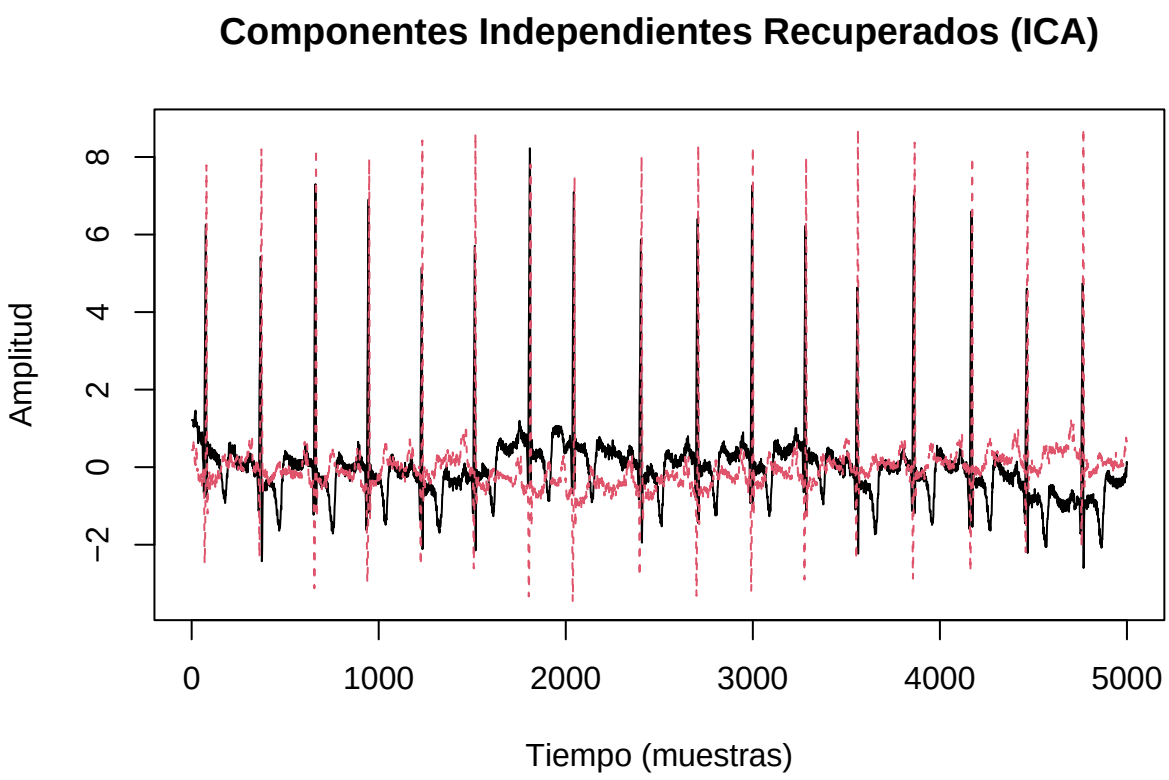
Aplicar ICA

```
set.seed(123)

ica_result <- fastICA(signals, n.comp = ncol(signals))

components <- ica_result$S

dim(components)
```

```
## [1] 5000    2
```

Visualizar componentes independientes

```
matplot(components, type="l",
        main="Componentes Independientes Recuperados (ICA)",
        xlab="Tiempo (muestras)",
        ylab="Amplitud")
```



Interpretación de resultados

Las señales ECG originales corresponden a registros reales del MIT-BIH Arrhythmia Database. Estas señales representan mediciones eléctricas cardíacas registradas mediante múltiples derivaciones, las cuales pueden contener mezclas de diferentes fuentes fisiológicas y posibles artefactos.

Al aplicar ICA, el algoritmo busca descomponer las señales observadas en componentes estadísticamente independientes. En la visualización de los componentes recuperados se observan patrones diferenciados, lo que indica que ICA logró separar parcialmente las fuentes subyacentes presentes en la señal original.

Este resultado demuestra la utilidad de ICA en el análisis de señales biomédicas, ya que permite identificar estructuras ocultas y separar posibles fuentes independientes a partir de mezclas lineales observadas.

Comparaciones generales

SVD (Singular Value Decomposition)

La SVD (Descomposición en Valores Singulares) es un método algebraico que descompone una matriz en componentes ortogonales ordenados por importancia.

Ventajas

- Método determinista (mismo resultado en cada ejecución).
- Base matemática sólida.
- Eficiente para datos lineales y matrices grandes.

- Muy útil en compresión de datos y análisis de texto (ej. LSA).
- Preserva la mayor varianza posible en menos dimensiones.

Limitaciones

- Solo captura relaciones lineales.
- No preserva estructuras locales complejas.
- Puede ser sensible a datos no escalados.
- Componentes a veces difíciles de interpretar.

t-SNE (t-Distributed Stochastic Neighbor Embedding)

t-SNE es un algoritmo no lineal diseñado principalmente para visualización de datos de alta dimensión en 2D o 3D.

Ventajas

- Excelente para visualizar clusters.
- Preserva muy bien relaciones locales.
- Revela estructuras complejas no lineales.
- Muy popular en análisis de datos biológicos y NLP.

Limitaciones

- Alto costo computacional.
- No preserva bien distancias globales.
- No es determinista (puede variar entre ejecuciones).
- Difícil de usar para nuevos datos sin reentrenar.
- Principalmente útil para visualización, no para modelado posterior.

UMAP (Uniform Manifold Approximation and Projection)

UMAP es un método no lineal basado en teoría de variedades y grafos, orientado tanto a visualización como a reducción eficiente de dimensionalidad.

Ventajas

- Más rápido que t-SNE.
- Preserva mejor la estructura global y local.
- Escalable a grandes volúmenes de datos.
- Permite transformar nuevos datos sin recalcular todo.
- Buen equilibrio entre interpretación y rendimiento.

Limitaciones Requiere ajuste de hiperparámetros.

Puede generar agrupamientos artificiales si no se configura adecuadamente.

Menor estabilidad si los datos tienen mucho ruido.

ICA (Independent Component Analysis)

Independent Component Analysis es una técnica que busca separar señales en componentes estadísticamente independientes.

Ventajas

- Útil para separación de señales (ej. audio, EEG).
- Identifica fuentes independientes ocultas.
- Captura estructuras no gaussianas.
- Interpretación clara cuando se conocen las fuentes subyacentes.

Limitaciones

- Supone independencia estadística (no siempre realista).
- Sensible al ruido.
- No ordena componentes por importancia.
- No está orientado específicamente a visualización.

Conclusiones

Principales Aprendizajes Aprendimos como utilizar los adgoritmos, como interpretarlos y como identificar sus casos de uso, con lo cual aprendimos también a preparar los datos de manera en que sean más coherentes respectivos a los casos de uso.

Dificultades encontradas

Reflexiones Estos algoritmos son bastante útiles aunque son limitados en cuanto a sus usos, en referencia en que en un caso normal los datos deverían de ser depurados de cierta manera para que los algoritmos no tengan problemas generales o que el ruido interfiera mucho con los mismos.