

Modelización de Computación Predictiva

ÁNGEL MORA BONILLA Y DOMINGO LÓPEZ RODRÍGUEZ

Modelización de Computación Predictiva

Ángel Mora Bonilla y Domingo López Rodríguez

Tabla de contenidos

Introducción	1
Motivación del Libro	1
Estructura del Contenido	1
Prerrequisitos	1
1 El paradigma de la investigación reproducible ...	3
1.1 Ingeniería de Proyectos de Datos	3
1.2 El Taller del Ingeniero: R y RStudio	5
1.3 Ingeniería de Datos con <code>tidyverse</code>	7
1.4 Programación funcional con <code>purrr</code>	12
1.5 Ejercicios propuestos	13
I Matrices y álgebra lineal	
2 Resolución de grandes sistemas de ecuaciones ...	17
2.1 El problema de la escala	17
2.2 Métodos directos y factorización	17
2.3 Análisis detallado de matrices con estructura	23
2.4 Estabilidad numérica y condicionamiento	29
2.5 Métodos iterativos para sistemas grandes	30
2.6 Práctica Final: Simulación de Red de Calor	34
2.7 Ejercicios Adicionales	35
3 Autovalores, Autovectores y Descomposición Singular	37
3.1 El método de las potencias (the power method)	37
3.2 El Algoritmo QR (espectro completo)	40
3.3 Descomposición en Valores Singulares (SVD)	42

4	Algoritmo PageRank	43
4.1	El algoritmo PageRank	45
4.2	Relación con el álgebra lineal	48
4.3	¿Cómo hacerlo en R?	49
4.4	Problemas en la web real	50
4.5	El factor de amortiguación (damping factor)	51
5	Análisis de componentes principales	53
5.1	Motivación: la maldición de la dimensionalidad	53
5.2	Fundamento teórico	54
5.3	Caso Práctico: Crimen en EEUU (USArrests)	55
5.4	Visualización Avanzada e Interpretación	58
5.5	Verificación: PCA Manual (“Bajo el capó”)	60
5.6	Ejercicios	61

II**Análisis de redes sociales**

6	Introducción al Análisis de Redes Sociales	65
6.1	¿Por qué SNA?	65
6.2	SNA en el ecosistema R	65
6.3	Fundamentos Teóricos (Repaso)	66
7	Visualización y Manipulación de Grafos	67
7.1	Fundamentos de Representación	67
7.2	Métodos de igraph para la Construcción	68
7.3	Estructuras Canónicas y Generación	69
7.4	Estilización y Metadatos	70
7.5	Exportación de Datos	71
8	Métricas y Análisis Estructural	73
8.1	Medidas de Centralidad	73
8.2	Análisis de Cohesión y Estructura Global	75
8.3	Análisis de Aristas y Comunidades	75
8.4	Otras Medidas de Interés	76

III**Sistemas recomendadores y predictivos**

9	Sistemas de Recomendación	79
9.1	El Concepto del Long Tail	79
9.2	Filtrado Colaborativo (Collaborative Filtering)	79
9.3	Factorización de Matrices (SVD)	83
10	85

Introducción

Bienvenidos al curso de **Modelización para Computación Predictiva**. Este libro ha sido diseñado para servir como referencia integral para estudiantes de ingeniería que buscan dominar los fundamentos matemáticos y computacionales detrás de los algoritmos modernos de análisis de datos.

Motivación del Libro

En la era del Big Data, no es suficiente con saber “ejecutar” un algoritmo desde una librería. Un ingeniero debe comprender la **maquinaria interna**: por qué funciona un método, cuándo falla y cómo optimizarlo. Este texto tiende un puente entre:

1. **Fundamentos Matemáticos Rigurosos:** Álgebra Lineal computacional, Teoría de Grafos y Cálculo matricial.
2. **Implementación Práctica:** Uso de R y el ecosistema `tidyverse` para traducir ecuaciones a código eficiente y reproducible.
3. **Aplicaciones Reales:** Desde el algoritmo PageRank de Google hasta sistemas de recomendación y análisis de redes sociales.

Estructura del Contenido

El libro está organizado en bloques temáticos que construyen el conocimiento de forma progresiva:

- **Fundamentos de Ingeniería de Datos:** Antes de modelar, aprendemos a estructurar proyectos reproducibles y a dominar R (Capítulo 1).
- **Álgebra Lineal Computacional:** Abordamos el problema de la escala. Cómo resolver sistemas de ecuaciones gigantes y entender la descomposición de matrices (Autovalores, SVD, PCA) que sustentan la reducción de dimensión (Capítulos 2, 3, 4).
- **Análisis de Redes (SNA):** Una inmersión profunda en la teoría de grafos aplicada. Desde la visualización hasta métricas de centralidad y detección de comunidades (Capítulos 5, 6, 7).
- **Sistemas de Recomendación:** Aplicamos lo aprendido para construir motores que predicen preferencias de usuarios (Capítulo 8).
- **Series Temporales:** Modelización de datos secuenciales para predicción (Capítulo 9).

Prerrequisitos

- **Matemáticas:** Conocimientos básicos de álgebra lineal (matrices, vectores) y cálculo.

- **Programación:** Familiaridad básica con conceptos de algoritmos. No se requiere experiencia previa en R, ya que se enseña desde cero en el primer capítulo.

1

El paradigma de la investigación reproducible

En la ingeniería y la ciencia de datos moderna, existe una crisis silenciosa: la incapacidad de reproducir resultados pasados. Un estudio o modelo predictivo no es válido si solo funciona en el ordenador de quien lo creó.

Para combatir esto, abandonamos las herramientas manuales (“point-and-click”) en favor de flujos de trabajo basados en código.

Definición: Reproducibilidad vs Replicabilidad

Aunque a veces se usan indistintamente, existe un matiz técnico importante. Hablamos de **reproducibilidad** cuando un investigador independiente tiene acceso a los *mismos* datos y al *mismo* código, y es capaz de computar exactamente los *mismos* resultados; este es el estándar mínimo de ingeniería que perseguimos. Por otro lado, la **replicabilidad** es un estándar científico más alto, donde un investigador independiente recoge *nuevos* datos experimentales bajo las mismas condiciones y llega a las *mismas* conclusiones científicas. Este libro se centra en garantizar la primera: la reproducibilidad total mediante código robusto y autocontenido.

1.1 Ingeniería de Proyectos de Datos

Antes de escribir una sola línea de código de análisis, un ingeniero debe establecer una infraestructura sólida. El caos en los archivos es la causa #1 de la “muerte” de los proyectos a largo plazo.

1.1.1 Gestión profesional de proyectos

El pecado de las rutas absolutas

Jamás utilice rutas que dependan de su usuario específico:

```
# MALA PRÁCTICA: Ruta absoluta  
# Solo funciona en EL ordenador de "Juan"  
datos <- read.csv("C:/Users/Juan/Documentos/Tesis/Analisis_Datos/data.csv")
```

Si envía este script a su profesor o colega, fallará. Si cambia de ordenador, fallará.

La solución: RStudio Projects (.Rproj)

Un **Proyecto de RStudio** es un archivo que define la “raíz” de su trabajo. Al abrir el `.Rproj`, R entiende automáticamente que esa carpeta es el “hogar”, permitiendo usar **rutas relativas**:

```
# BUENA PRÁCTICA: Ruta relativa al proyecto
# Funciona en CUALQUIER ordenador (Windows, Mac, Linux)
datos <- read_csv("data/ventas_2024.csv")
```

Estructura de carpetas estándar

Para mantener el orden, sugerimos una jerarquía estándar que separe claramente los insumos de los productos. En la raíz del proyecto (`nombre_proyecto/`) se debe ubicar el archivo `.Rproj`. Los datos originales e inmutables, aquellos que nunca deben editarse manualmente para preservar la integridad de la fuente, se almacenan en `data-raw/`. Una vez procesados y limpios, los datasets listos para el análisis se guardan en `data/`. El código ejecutable, ya sean funciones auxiliares o scripts de limpieza, reside en `scripts/`, mientras que cualquier archivo generado por el código, como gráficos o tablas exportadas, debe dirigirse a `output/`. Finalmente, los documentos narrativos como este libro se organizan en `chapters/`.

1.1.2 El control de versiones: Git y GitHub

En ingeniería de software, trabajar sin control de versiones es impensable. En ciencia de datos, debería serlo también. **Git** es un sistema que registra los cambios en nuestro código a lo largo del tiempo, permitiéndonos volver a versiones anteriores (como un “Deshacer” infinito) y trabajar en equipo sin sobrescribir el trabajo de otros.

El flujo de trabajo en Git se basa en cuatro conceptos fundamentales que operan en ciclo. Todo comienza en el **Repositorio (Repo)**, que es la carpeta de su proyecto vigilada por el sistema. Cuando realizamos cambios significativos, creamos una instantánea del estado del proyecto mediante un **Commit**; piense en esto como un punto de guardado seguro al que siempre se puede volver. Para colaborar o respaldar el trabajo, enviamos estos commits locales a la nube (GitHub) mediante un **Push**. Inversamente, para actualizar nuestro entorno local con los cambios que otros hayan subido a la nube, ejecutamos un **Pull**.

Git en RStudio

RStudio tiene una integración excelente con Git. En el panel superior derecho, encontrará la pestaña **Git**. Desde ahí puede marcar archivos (*Stage*), hacer *Commit* y sincronizar con *Push/Pull* sin tocar la terminal.

1.1.3 Gestión de dependencias con `renv`

Un script que funciona hoy puede no funcionar mañana si los paquetes se actualizan. `renv` es un paquete que crea un entorno de librería **aislado** para

cada proyecto.

El uso de `renv` sigue un ciclo de vida predecible. Primero, se inicializa el proyecto con `renv::init()`, lo que configura la infraestructura para usar librerías privadas y aisladas del sistema. A partir de ahí, el trabajo continúa normalmente instalando paquetes con `install.packages()`. La diferencia clave es que, periódicamente, debemos guardar el estado de nuestras dependencias mediante `renv::snapshot()`, lo cual genera una “lista de ingredientes” exacta (el archivo `renv.lock`). Si cambiamos de ordenador o un colega necesita ejecutar nuestro código, el comando `renv::restore()` leerá ese archivo y reconstruirá el entorno instalando exactamente las mismas versiones de los paquetes, garantizando así la reproducibilidad técnica.

1.1.4 Comunicación científica con Quarto

Quarto es la evolución de RMarkdown. Permite mezclar narrativa (texto), código y resultados (gráficos/tablas) en un solo documento vivo.

Anatomía de un documento

Todo archivo `.qmd` comienza con un encabezado YAML (metadatos) entre tres guiones:

```
title: "Mi Análisis"  
format: html
```

A continuación, escribimos texto en Markdown (usando **negrita**, # Títulos, etc.) e insertamos **Code Chunks** para el análisis:

```
```{r}  
#| label: fig-dispersion
#| fig-cap: "Relación entre variables"
plot(x, y)
```
```

Referencias cruzadas y citas

Quarto gestiona automáticamente la numeración de figuras y ecuaciones, un requisito indispensable en textos académicos.

- **Figuras:** Al etiquetar el chunk con `#| label: fig-algo`, podemos citarlo en el texto escribiendo `@fig-algo`.
- **Bibliografía:** Usando un archivo `.bib`, podemos citar fuentes con `[@autor2024]`.

1.2 El Taller del Ingeniero: R y RStudio

Ahora que tenemos una metodología de trabajo robusta, entremos en el detalle de las herramientas específicas.

1.2.1 Entorno para ciencia de datos

Para trabajar profesionalmente, distinguimos tres componentes:

1. **El Motor (R):** El lenguaje de programación que realiza los cálculos.
2. **El Volante (RStudio):** El Entorno de Desarrollo Integrado (IDE) que nos permite gestionar el código cómodamente.
3. **El Ecosistema (tidyverse):** Un conjunto de librerías modernas que comparten una gramática y filosofía común.

Una Breve Historia de R



R fue creado en 1993 por **Ross Ihaka** y **Robert Gentleman** en la Universidad de Auckland, Nueva Zelanda. Se basaron en el lenguaje S (desarrollado en Bell Labs). El nombre “R” proviene en parte de las iniciales de sus dos creadores y en parte como un juego de palabras sobre el nombre de S. Hoy en día, es un proyecto colaborativo global mantenido por el “R Core Team”.

1.2.2 Los 4 paneles de RStudio

RStudio divide su interfaz en cuatro cuadrantes críticos. Es vital entender qué hace cada uno para no perder datos.

1. **Source (Editor - Arriba Izqda):** Su “cuaderno de laboratorio”. Aquí se escriben los scripts (.R) y documentos (.qmd). **Lo que se escribe aquí, se guarda.**
2. **Console (Abajo Izqda):** Donde R ejecuta las órdenes. **Advertencia:** Es efímero. Si cierra RStudio, lo que haya escrito aquí desaparece. Úselo solo para pruebas rápidas.
3. **Environment (Arriba Dcha):** La memoria RAM visible. Muestra las variables y tablas cargadas.
4. **Files/Plots/Packages (Abajo Dcha):** Su explorador de archivos y visor de resultados gráficos.

Configuración Profesional

Vaya a Tools > Global Options y asegúrese de desmarcar “*Restore .RData into workspace at startup*”. Esto obliga a que su código sea capaz de regenerar todo el análisis desde cero cada vez que inicia sesión, garantizando la reproducibilidad.

1.2.3 Fundamentos del lenguaje R

Antes de usar herramientas avanzadas, debemos comprender la sintaxis base.

Asignación y estilo

En R, creamos variables asignando valores con el operador flecha <-.

```
x <- 10
nombre <- "Modelización"
```

Guía de estilo

El código se lee más veces de las que se escribe. Siga estas reglas:

- **snake_case:** Use minúsculas y guiones bajos (`precio_unitario`).
- **Nombres descriptivos:** Evite `x`, `y`, `temp`. Use `tasa_interes`, `pib_real`.
- **Sin espacios:** `variable nueva` dará error. Use `variable_nueva`.

R maneja fundamentalmente tres tipos de datos atómicos. Los números se representan como tipo **Numeric (Double)**, que abarca tanto enteros como reales (ej. 1.5, 200). El texto se maneja como tipo **Character**, siendo obligatorio el uso de comillas simples o dobles. Finalmente, el tipo **Logical** es crucial para el control de flujo y el filtrado de datos, admitiendo únicamente los valores `TRUE` o `FALSE`.

Vectores

La estructura mínima en R es el vector. Se crea con la función `c()` (concatenar).

```
precios <- c(100, 120, 150)
ciudades <- c("Málaga", "Sevilla", "Granada")
```

La magia de R es la **vectorización**: podemos operar sobre todo el vector a la vez sin bucles.

```
# Aplicar IVA a todos los precios simultáneamente
precios_con_iva <- precios * 1.21
```

1.3 Ingeniería de Datos con tidyverse

El paquete `dplyr` (parte de `tidyverse`) proporciona una “gramática” consistente para la manipulación de datos. Se basa en **verbos** que realizan acciones humanas sobre los datos.

El Arquitecto del Tidyverse



Hadley Wickham, Estadístico Jefe en Posit, revolucionó R al crear paquetes como `ggplot2` y `dplyr`. Su filosofía se basa en que “los datos ordenados (tidy data) son todos iguales, pero cada conjunto de datos desordenado lo es a su manera”. El `tidyverse` es impulsado en gran medida por Hadley Wickham. Su idea central es que las herramientas de manipulación de datos deben seguir una “gramática”. Así como una gramática lingüística nos permite combinar palabras para crear frases complejas, la “gramática de manipulación de datos” (implementada en `dplyr`) nos da *verbos* que se pueden combinar para construir análisis complejos de forma legible.

Cargaremos el paquete y el dataset de ejemplo:

```
library(tidyverse)
data("starwars") # Dataset incluido en dplyr
```

1.3.1 El operador tubería (the pipe |>)

La tubería nos permite leer el código de izquierda a derecha, encadenando operaciones. Pasa el resultado de la izquierda como **primer argumento** de la función de la derecha.

```
# Lógica: Datos -> Filtrar -> Seleccionar -> Mostrar
starwars |>
  filter(species == "Droid") |>
  select(name, homeworld)
```

1.3.2 Verbo 1: `select()` (selección de columnas)

Permite escoger, renombrar y reordenar columnas. Es más potente de lo que parece gracias a los **ayudantes de selección** (*selection helpers*).

- **Selección básica:** `select(nombre, altura, peso)`
- **Exclusión:** `select(-nombre)` (Todas menos nombre).
- **Rangos:** `select(name:mass)` (Desde name hasta mass).

Las funciones auxiliares o “wildcards” nos permiten seleccionar columnas basándonos en patrones de texto en lugar de nombres exactos. Por ejemplo, `starts_with("color")` y `ends_with("color")` seleccionarán todas las variables que comiencen o terminen con ese prefijo/sufijo (como `hair_color`, `skin_color`). `contains("_")` buscará cualquier coincidencia parcial. Quizás la más potente sea `where(is.numeric)`, que selecciona columnas basándose en su tipo de dato, permitiendo operaciones masivas sobre todas las variables numéricas del dataset.

```
# Ejemplo avanzado
starwars |>
  select(name, where(is.numeric)) |>
  head(3)
```

1.3.3 Verbo 2: filter() (selección de filas)

Filtrá las observaciones que cumplen una condición lógica. Si la condición es TRUE, la fila se queda; si es FALSE o NA, se va.

Las condiciones se construyen utilizando operadores lógicos estándar. Para la igualdad exacta usamos == y para la desigualdad !=. Las comparaciones numéricas se realizan con los operadores clásicos >, <, =>, =<. Para combinar múltiples condiciones, el operador & (AND) exige que se cumplan ambas, mientras que | (OR) requiere que se cumpla al menos una. Un operador extremadamente útil en ciencia de datos es %in%, que verifica la pertenencia a un conjunto (por ejemplo, país %in% c("España", "Francia")).

```
# Filtrado complejo
starwars |>
  filter(
    species == "Human",                      # Humanos
    height > 180,                            # Altos
    (homeworld == "Tatooine" | homeworld == "Naboo") # De estos planetas
  )

# Versión elegante con %in%
starwars |>
  filter(homeworld %in% c("Tatooine", "Naboo", "Alderaan"))
```

El Peligro de NA

`filter()` elimina los NA automáticamente. Si desea conservarlos, debe pedirlo explícitamente: `filter(is.na(mass) | mass > 100)`.

1.3.4 Verbo 3: arrange() (ordenación)

Ordena las filas. Por defecto es ascendente. Use `desc()` para descendente.

```
starwars |>
  arrange(desc(mass), height) # Ordena por masa (desc) y desempata por altura (asc)
```

1.3.5 Verbo 4: mutate() (creación/edición de variables)

Crea nuevas columnas o modifica las existentes manteniendo el mismo número de filas.

Usos Avanzados:

1. **Matemáticas:** `mutate(imc = mass / (height/100)^2)`

2. Lógica Condicional (`if_else`):

```
mutate(tipo = if_else(mass > 100, "Pesado", "Ligero"))
```

3. Múltiples Casos (`case_when`):

La alternativa vectorizada a los `if-else` anidados.

```
starwars |>
  select(name, height) |>
  mutate(categoría_altura = case_when(
    height < 100 ~ "Pequeño",
    height < 180 ~ "Mediano",
    height >= 180 ~ "Alto",
    TRUE ~ "Desconocido" # Captura los NA o restos
  ))
```

1.3.6 Verbo 5: `summarise()` y `group_by()` (agregación)

Esta pareja es el motor del análisis de datos (estrategia “Split-Apply-Combine”).

1. **`group_by()`:** No cambia los datos visualmente, pero añade “etiquetas” invisibles de grupo.
2. **`summarise()`:** Colapsa cada grupo en una sola fila calculando estadísticos.

Dentro de `summarise`, podemos utilizar cualquier función que tome un vector y devuelva un solo valor. Las más comunes incluyen medidas de tendencia central y dispersión como `mean()`, `median()`, `sd()`, `min()` y `max()`. Para contar frecuencias usamos `n()` (número de filas en el grupo) y `n_distinct()` (número de valores únicos). También son útiles `first()` y `last()` para extraer valores posicionales específicos.

```
# Análisis completo por especie
starwars |>
  group_by(species) |>
  summarise(
    total_personajes = n(),
    altura_media = mean(height, na.rm = TRUE),
    peso_maximo = max(mass, na.rm = TRUE),
    planetas_unicos = n_distinct(homeworld)
  ) |>
  filter(total_personajes > 1) |> # Descartamos especies únicas
  arrange(desc(altura_media))
```

Desagrupar

El resultado de un `summarise()` suele eliminar la última capa de agrupación, pero es buena práctica añadir `.groups = "drop"` o usar `ungroup()` al final para evitar comportamientos inesperados en análisis posteriores.

1.3.7 Caso de Estudio: Pipeline Completo de Ingeniería de Datos

Para visualizar el poder del `tidyverse` en un contexto real, imaginemos que recibimos datos “sucios” y necesitamos generar un reporte limpio. Simularemos este escenario usando el dataset `starwars` y una tabla auxiliar ficticia de transacciones.

El Problema: Queremos analizar el “Índice de Masa Corporal (IMC) promedio por especie” y cruzarlo con una tabla de “costes de mantenimiento médico” por especie, para identificar las especies más costosas de mantener.

Paso 0: Preparar los datos (Simulación) Creamos una tabla auxiliar de costes médicos.

```
# Tabla simulada de costes médicos por especie
costes_medicos <- tibble(
  species = c("Human", "Droid", "Wookiee", "Yoda's species"),
  coste_anual = c(500, 100, 200, 5000)
)
```

La Pipeline de Ingeniería:

1. **Limpieza:** Eliminamos personajes sin masa o altura.
2. **Transformación:** Calculamos el IMC.
3. **Agregación:** Calculamos el IMC medio por especie.
4. **Enriquecimiento (Join):** Cruzamos con la tabla de costes médicos.
5. **Resultado:** Ordenamos por coste.

```
reporte_final <- starwars |>
  # 1. Limpieza preliminar
  select(name, species, height, mass) |>
  filter(!is.na(height), !is.na(mass), !is.na(species)) |>

  # 2. Feature Engineering (Crear variable IMC)
  mutate(
    height_m = height / 100,
    imc = mass / (height_m^2)
  ) |>

  # 3. Agregación (Reducir dimensionalidad)
  group_by(species) |>
  summarise(
    n_sujetos = n(),
    imc_medio = mean(imc),
    peso_total = sum(mass)
  ) |>
  ungroup() |>

  # 4. Enriquecimiento de datos (Left Join)
  # Unimos nuestra tabla calculada con la tabla externa de costes
```

```

left_join(costes_medicos, by = "species") |>

# 5. Limpieza final de NAs generados por el join
# (Si no hay dato de coste, asumimos 0 o un valor base)
mutate(coste_anual = replace_na(coste_anual, 0)) |>

# 6. Ordenación para reporte
arrange(desc(coste_anual))

# Visualizar el resultado
print(reporte_final)

# A tibble: 31 x 5
  species      n_sujetos imc_medio peso_total coste_anual
  <chr>          <int>    <dbl>      <dbl>        <dbl>
1 Yoda's species     1      39.0       17      5000
2 Human            20      24.9      1626.       500
3 Wookiee          2      23.2       248       200
4 Droid             4      32.7       279       100
5 Aleena            1      24.0       15         0
6 Besalisk          1      26.0       102       0
7 Cerean            1      20.9       82         0
8 Clawdite          1      19.5       55         0
9 Dug               1      31.9       40         0
10 Ewok              1      25.8       20         0
# i 21 more rows

```

Este bloque de código representa el 80% del trabajo diario de un científico de datos: tomar datos crudos, limpiarlos, transformarlos y enriquecerlos para el análisis final.

1.3.8 Verbo extra: `slice()` y sus variantes

Existen variantes de `slice` para necesidades específicas: `slice(1:5)` selecciona filas por su índice posicional; `slice_max(mass, n = 3)` extrae las filas con los valores más altos de una variable; y `slice_sample(n = 5)` realiza un muestreo aleatorio de filas, útil para bootstrapping o revisiones rápidas.

1.4 Programación funcional con purrr

Hasta ahora hemos manipulado dataframes. Pero a menudo necesitamos realizar la misma operación sobre múltiples objetos (ej. leer 50 archivos Excel, generar 100 gráficos, entrenar modelos con distintos parámetros).

Los bucles `for` en R son conocidos por ser verbosos y, si no se gestionan bien, lentos. El paquete `purrr` (también del tidyverse) nos ofrece la familia de funciones `map()`.

La función básica es `map(.x, .f)`, donde `.x` es un vector o lista y `.f` es la función a aplicar a cada elemento.

La familia de funciones `map` varía según el tipo de salida deseada. La función base `map()` devuelve siempre una lista, lo cual es la estructura más flexible pero a veces difícil de manejar. Si sabemos que el resultado debe ser un vector numérico, usamos `map_dbl()` (double); si esperamos texto, `map_chr()`. Para ingeniería de datos, la variante más útil suele ser `map_df()`, que intenta combinar los resultados de cada iteración en un único dataframe rectangular.

```
# Ejemplo: Calcular la media de cada columna numérica de 'starwars'
starwars |>
  select(where(is.numeric)) |>
  map_dbl(mean, na.rm = TRUE)
```

Lectura masiva de archivos

El uso más potente de `purrr` en ingeniería de datos es la carga masiva. Supongamos que tenemos una carpeta con archivos de ventas mensuales (`ventas_ene.csv`, `ventas_feb.csv`, ...).

```
# código conceptual
list.files("data/ventas", pattern = "*.csv", full.names = TRUE) |>
  map_df(read_csv)
# ¡En una línea hemos leído y unido todos los archivos!
```

1.5 Ejercicios propuestos

Para consolidar el flujo de trabajo reproducible, realice los siguientes ejercicios.

Ejercicio 1.1 Configuración de un proyecto.

1. Cree un nuevo proyecto llamado `Practica_Reproducible`.
2. Cree las carpetas `data` y `scripts`.
3. Guarde un script que lea un archivo CSV ficticio usando una ruta relativa.
4. Mueva la carpeta del proyecto a otra ubicación y verifique que sigue funcionando.

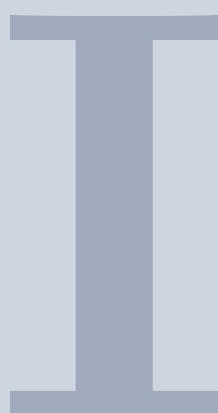


Ejercicio 1.2 Usando el dataset `starwars`:

1. Filtre los personajes que no tienen color de piel (`skin_color`) definido (es decir, `NA`).
2. Agrupe por `homeworld` (planeta de origen).
3. Calcule la masa media de los habitantes de cada planeta.
4. Ordene el resultado de mayor a menor masa media.

Ejercicio 1.3 Dado el vector de crecimiento: `x <- c(2.5, -1.0, 0.5, -0.2, 3.0)`.

1. Cree un vector lógico que sea TRUE cuando el crecimiento sea negativo.
2. Use ese vector lógico para extraer los valores numéricos de los períodos de recesión.



Matrices y álgebra lineal

2 Resolución de grandes sistemas de ecuaciones 17

- 2.1 El problema de la escala
- 2.2 Métodos directos y factorización
- 2.3 Análisis detallado de matrices con estructura
- 2.4 Estabilidad numérica y condicionamiento
- 2.5 Métodos iterativos para sistemas grandes
- 2.6 Práctica Final: Simulación de Red de Calor
- 2.7 Ejercicios Adicionales

3 Autovalores, Autovectores y Descomposición Singular 37

- 3.1 El método de las potencias (the power method)
- 3.2 El Algoritmo QR (espectro completo)
- 3.3 Descomposición en Valores Singulares (SVD)

4 Algoritmo PageRank 43

- 4.1 El algoritmo PageRank
- 4.2 Relación con el álgebra lineal
- 4.3 ¿Cómo hacerlo en R?
- 4.4 Problemas en la web real
- 4.5 El factor de amortiguación (damping factor)

5 Análisis de componentes principales 53

- 5.1 Motivación: la maldición de la dimensionalidad
- 5.2 Fundamento teórico
- 5.3 Caso Práctico: Crimen en EEUU ([USArrests](#))
- 5.4 Visualización Avanzada e Interpretación
- 5.5 Verificación: PCA Manual (“Bajo el capó”)
- 5.6 Ejercicios

2

Resolución de grandes sistemas de ecuaciones

2.1 El problema de la escala

En el análisis de datos moderno, la modelización predictiva y la ingeniería, raramente nos enfrentamos a sistemas de 3x3 ecuaciones. Lo habitual es tratar con sistemas donde el número de variables, n , puede ser de miles, millones o incluso miles de millones.

Resolver un sistema $A\mathbf{x} = \mathbf{b}$ con $n = 10.000$ es trivial para un ordenador moderno si la matriz A es densa (casi todas sus entradas son no nulas). Pero, ¿qué ocurre si $n = 10.000.000$?

Si A es una matriz densa de $10^7 \times 10^7$, necesitaríamos almacenar $(10^7)^2 = 10^{14}$ valores. Si cada valor ocupa 8 bytes (un `numeric` en R), el espacio de memoria requerido sería:

$$10^{14} \text{ valores} \times 8 \text{ bytes/valor} \approx 8 \times 10^{14} \text{ bytes} \approx 800 \text{ Terabytes}$$

Ningún ordenador personal puede almacenar esa matriz en RAM. Además, los métodos directos clásicos, como la eliminación de Gauss, tienen un coste computacional de $O(n^3)$ operaciones. Para $n = 10^7$, esto es 10^{21} operaciones, una cantidad intratable.

La solución a este problema proviene de dos frentes complementarios. Por un lado, aprovechamos que la mayoría de los sistemas del mundo real son **matrices dispersas**, lo que nos permite usar estructuras de datos que almacenan solo los valores no nulos. Por otro lado, recurrimos a **métodos numéricos avanzados**, como los iterativos (Jacobi, Gauss-Seidel), que refinan una solución aproximada en lugar de intentar resolver el sistema de forma directa y costosa.

En este capítulo, exploraremos las herramientas teóricas y computacionales en R para abordar ambos frentes.

2.2 Métodos directos y factorización

Los métodos directos resuelven el sistema en un número finito y predecible de pasos. El más conocido es la **Eliminación de Gauss**, que transforma el sistema en uno triangular superior equivalente, fácil de resolver mediante **sustitución regresiva**.

Carl Friedrich Gauss



Carl Friedrich Gauss (1777-1855) no inventó el método (ya se conocía en China en el siglo I a.C.), pero lo popularizó y formalizó en Europa. Su contribución clave fue el análisis sistemático de los errores de medición, lo que le llevó al método de mínimos cuadrados, intrínsecamente ligado a la resolución de sistemas.

En R, la función `solve()` es una “caja negra” altamente optimizada que implementa un método directo.

```
A <- matrix(c(4, 1, -1, 1, 6, 2, -1, 2, 5), nrow = 3, byrow = TRUE)
b <- c(6, 15, 12)

# Resolver Ax = b
x <- solve(A, b)
print(x)

[1] 1.651685 1.516854 2.123596

# Verificación (debería dar b)
A %*% x

[,1]
[1,]    6
[2,]   15
[3,]   12
```

2.2.1 El coste de `solve()`: factorización LU

La función `solve(A, b)` no calcula la inversa A^{-1} y luego multiplica $A^{-1}b$, ya que calcular la inversa es computacionalmente más caro e inestable.

En su lugar, la mayoría de los *solvers* modernos (incluido el de R) se basan en una **factorización de la matriz**. La más común es la **factorización LU**.

Definición 2.1 — Factorización LU. Para una matriz A (bajo ciertas condiciones), la factorización LU encuentra una matriz triangular inferior L (*Lower*), una matriz triangular superior U (*Upper*) y (a menudo) una matriz de permutación P (que reordena las filas para estabilidad numérica) tales que:

$$PA = LU$$

Resolver $A\mathbf{x} = \mathbf{b}$ se convierte en $PA\mathbf{x} = Pb$, es decir, $LU\mathbf{x} = Pb$. Este problema se resuelve en dos pasos (mucho más rápidos):

Resolver el sistema $LU\mathbf{x} = Pb$ se convierte en un proceso de dos pasos muy eficiente. Primero, realizamos una **sustitución progresiva** para resolver $Ly = Pb$ y encontrar \mathbf{y} . Acto seguido, usamos ese resultado en una **sustitución regresiva** para resolver $Ux = \mathbf{y}$ y hallar finalmente \mathbf{x} .

¿Por qué factorizar?

La factorización ($O(n^3)$) es la parte cara. Las sustituciones (progresiva y regresiva) son muy rápidas ($O(n^2)$).

Si necesitas resolver el mismo sistema A con muchos vectores \mathbf{b} diferentes ($A\mathbf{x}_1 = \mathbf{b}_1$, $A\mathbf{x}_2 = \mathbf{b}_2$, ...), calculas la factorización $PA = LU$ *una sola vez* y luego reutilizas L y U para cada \mathbf{b}_i con un coste muy bajo.

En R, usamos el paquete `Matrix` para obtener esta descomposición explícitamente.

■ Ejemplo 2.1 — Ejemplo: Factorización LU en R.

```
library(Matrix)

A <- matrix(c(2, 2, 1, 4, 3, 3, 8, 7, 7), nrow = 3, byrow = TRUE)
b <- c(1, 1, 1)

# Obtenemos la descomposición LU (con pivoteo)
decomp_lu <- lu(A)
print(decomp_lu)

LU factorization of Formal class 'denseLU' [package "Matrix"] with 4 slots
..@ x      : num [1:9] 8 0.5 0.25 7 -0.5 -0.5 7 -0.5 -1
..@ perm   : int [1:3] 3 2 3
..@ Dim    : int [1:2] 3 3
..@ Dimnames:List of 2
... $ : NULL
... $ : NULL

# Extraemos las matrices L, U y P directamente del objeto
res <- Matrix::expand(decomp_lu)
L <- res$L
U <- res$U
P <- res$P # P es una matriz de permutación

# Verificación: P %*% A debe ser igual a L %*% U
print(P %*% A)

3 x 3 Matrix of class "dgeMatrix"
 [,1] [,2] [,3]
```

```

[1,]    8    7    7
[2,]    4    3    3
[3,]    2    2    1
print(L %*% U)

3 x 3 Matrix of class "dgeMatrix"
[,1] [,2] [,3]
[1,]    8    7    7
[2,]    4    3    3
[3,]    2    2    1

# --- Resolución manual usando la factorización ---

# 1. Aplicamos la permutación a b
b_perm <- P %*% b

# 2. Resolvemos Ly = b_perm (sustitución progresiva)
# Usamos 'solve' que es inteligente y detecta que L es triangular
y <- solve(L, b_perm)
print(y)

3 x 1 Matrix of class "dgeMatrix"
[,1]
[1,]  1.0
[2,]  0.5
[3,]  1.0

# 3. Resolvemos Ux = y (sustitución regresiva)
x <- solve(U, y)
print(x)

3 x 1 Matrix of class "dgeMatrix"
[,1]
[1,]    1
[2,]    0
[3,]   -1

# Comparar con la solución directa de R
solve(A, b)

[1]  1  0 -1

```

■

2.2.2 Factorización de Cholesky

Un caso especial muy importante en estadística y *machine learning* (ej. en regresión de Mínimos Cuadrados Ordinarios) es cuando la matriz A es **simétrica y definida positiva**.

Definición 2.2 — Factorización de Cholesky. Si A es simétrica y definida positiva, existe una única matriz triangular inferior L tal que:

$$A = LL^T$$

Esto es computacionalmente más rápido (aproximadamente la mitad de operaciones que LU) y más estable numéricamente.

En R, la función `chol()` calcula la factorización de Cholesky.

¡Cuidado con `chol()`!

Por convención histórica, `chol(A)` en R devuelve la matriz triangular *superior* R tal que $A = R^T R$. Si queremos la L de la definición $A = LL^T$, debemos tomar la transpuesta del resultado: `L <- t(chol(A))`.

■ **Ejemplo 2.2 — Ejemplo: Cholesky en R.**

```
# A es simétrica y definida positiva
A <- matrix(c(4, 2, -2, 2, 10, 2, -2, 2, 5), nrow = 3, byrow = TRUE)
b <- c(6, 15, 12)
```

```
# 1. Calcular la descomposición
R <- chol(A) # R es triangular SUPERIOR
print(R)
```

```
[,1] [,2]      [,3]
[1,]    2     1 -1.000000
[2,]    0     3  1.000000
[3,]    0     0  1.732051
```

```
# Verificación: R^T * R debe ser A
t(R) %*% R
```

```
[,1] [,2] [,3]
[1,]    4     2    -2
[2,]    2    10     2
[3,]   -2     2     5
```

```
# --- Resolución manual usando Cholesky ---
# A*x = b => (R^T * R) * x = b => R^T * (R*x) = b
# Paso 1: Resolver R^T * y = b (sustitución progresiva)
#   (Usamos fwdSolve en R base, o solve)
y <- solve(t(R), b)
```

```
# Paso 2: Resolver R * x = y (sustitución regresiva)
x <- solve(R, y)
print(x)
```

```
[1] 3.2777778 0.1111111 3.6666667
```

```
# Comparar con la solución directa
solve(A, b)
```

```
[1] 3.2777778 0.1111111 3.6666667
```

■

2.2.3 Aplicación Estelar: Regresión Lineal por Mínimos Cuadrados

La factorización de Cholesky no es solo una curiosidad teórica; es el motor de cálculo de millones de modelos estadísticos cada día. Cuando ajustamos una regresión lineal $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$, buscamos el vector de coeficientes $\hat{\boldsymbol{\beta}}$ que minimiza el error cuadrático.

La solución analítica viene dada por las **Ecuaciones Normales**:

$$(\mathbf{X}^T \mathbf{X}) \hat{\boldsymbol{\beta}} = \mathbf{X}^T \mathbf{y}$$

La matriz $A = \mathbf{X}^T \mathbf{X}$ es simétrica y (generalmente) definida positiva. Por tanto, para hallar $\hat{\boldsymbol{\beta}}$, no invertimos A . Hacemos Cholesky sobre A .

```
# 1. Generamos datos sintéticos: y = 2 + 1.5*x + ruido
set.seed(123)
n <- 100
x <- rnorm(n)
y <- 2 + 1.5 * x + rnorm(n, sd = 0.5)

# 2. Construimos la Matriz de Diseño X (incluyendo la columna de 1s para el
X <- cbind(Intercept = 1, Slope = x)

# 3. Calculamos la matriz A = X^T * X y el vector b = X^T * y
A <- t(X) %*% X
b <- t(X) %*% y

# 4. Resolvemos el sistema A * beta = b usando Cholesky
# A = R^T * R
R <- chol(A) # R es triangular superior
# R^T * y_temp = b -> solve(t(R), b)
y_temp <- solve(t(R), b)
# R * beta = y_temp -> solve(R, y_temp)
beta_hat <- solve(R, y_temp)

print("Coeficientes calculados manualmente con Cholesky:")
```

```
[1] "Coeficientes calculados manualmente con Cholesky:"
```

```
print(t(beta_hat))
```

| Intercept | Slope |
|-----------|-----------|
| 3.2777778 | 0.1111111 |

```
[1,] 1.948598 1.473764
# 5. Comparación con la función lm() de R (la forma estándar)
modelo <- lm(y ~ x)
print("Coeficientes con lm():")

[1] "Coeficientes con lm():"
print(coef(modelo))

(Intercept)           x
1.948598     1.473764
```

¡Exactamente el mismo resultado! Así es como funcionan las estadísticas “bajo el capó”.

2.3 Análisis detallado de matrices con estructura

Como mencionamos al inicio, el principal cuello de botella en sistemas grandes es la memoria. La solución es no almacenar los ceros. En el mundo real, la mayoría de los sistemas masivos que provienen de la discretización de fenómenos físicos (calor, fluidos, elasticidad) o del análisis de redes (sociales, de internet) son **dispersos**.

Definición 2.3 — Matriz Dispersa (Sparse). Una **matriz dispersa** es una matriz en la que la mayoría de sus elementos son cero. Una matriz es “suficientemente dispersa” si vale la pena utilizar algoritmos y estructuras de datos especiales para evitar almacenar y operar con esos ceros.

El nivel de dispersión (o *sparsity*) se mide como:

$$\text{Sparsity} = 1 - \frac{\text{Nº elementos no nulos (NNZ)}}{\text{Nº total de elementos } (n \times m)}$$

Una matriz con un *sparsity* del 99.9% es común.

2.3.1 El paquete Matrix: el ecosistema disperso

R base no maneja matrices dispersas de forma nativa. Un **matrix** en R base *siempre* almacena todos sus elementos. El paquete **Matrix** es la herramienta estándar y fundamental para el álgebra lineal numérica en R.

Almacenamiento Eficiente: CSC

El paquete **Matrix** no almacena una matriz dispersa como una cuadrícula. Utiliza formatos de almacenamiento eficientes, principalmente **Compressed Sparse Column (CSC)**.

Este formato almacena únicamente los valores no nulos en un vector, y usa otros dos vectores de “punteros” para saber en qué fila y columna va cada valor. Aunque parezca complejo, reduce el almacenamiento de $O(n^2)$

a $O(\text{NNZ})$, donde NNZ es el número de elementos no nulos.

- **Ejemplo 2.3 — Ejemplo: Creando Matrices Dispersas.** Comparemos una matriz densa vs. una dispersa de 5000×5000 .

```
library(Matrix)

n <- 5000

# 1. Matriz densa (R base)
A_densa <- matrix(0, nrow = n, ncol = n)
A_densa[1, 1] <- 5
A_densa[n / 2, n / 2] <- 5
A_densa[n, n] <- 5

# 2. Matriz dispersa (Paquete Matrix)
# Creamos una matriz especificando solo las posiciones (i, j) y los valores
A_dispersa <- sparseMatrix(
  i = c(1, n / 2, n),
  j = c(1, n / 2, n),
  x = 5,
  dims = c(n, n)
)

# Comparamos el tamaño en memoria
print(object.size(A_densa), units = "MB")
```

190.7 Mb

```
print(object.size(A_dispersa), units = "KB")
```

21 Kb

¡La matriz dispersa ocupa solo 12.3 Kb frente a los 190.7 Mb de la densa! Esto es una diferencia de más de 15.000 veces en el uso de memoria para almacenar la misma información.

■

2.3.2 Optimizaciones para matrices triangulares

El primer tipo de matriz estructurada que encontramos son las triangulares, que son la base de las factorizaciones LU y Cholesky.

Resolver un sistema triangular $T\mathbf{x} = \mathbf{b}$ no requiere un método $O(n^3)$ como Gauss. Se puede hacer directamente por sustitución (regresiva si es triangular superior, progresiva si es inferior) en $O(n^2)$.

Las funciones `backsolve()` y `fwdsolve()` de R base están optimizadas para esto.

- **Ejemplo 2.4 — Ejemplo: Eficiencia de Solvers Triangulares.** Comparemos el tiempo de un solver genérico (`solve`) contra uno especializado (`backsolve`) para un sistema triangular superior.

```
library(microbenchmark)

n <- 1000 # Un sistema de 1000x1000

# 1. Creamos un sistema triangular superior denso
R_densa <- matrix(0.1 * runif(n * n), n, n)
diag(R_densa) <- 1
R_densa[lower.tri(R_densa)] <- 0 # Hacemos que sea triangular superior
b <- rnorm(n)

# 2. Comparamos los tiempos de resolución
mbm <- microbenchmark(
  generico = solve(R_densa, b),
  especializado = backsolve(R_densa, b),
  times = 10
)
print(mbm, unit = "ms") # mostramos en milisegundos

Unit: milliseconds
      expr      min       lq     mean   median      uq
  generico 102.340469 103.359114 105.1040043 104.002199 107.213934
especializado    0.187206    0.231937    0.3402098    0.380111    0.397372
      max  neval
110.109190      10
    0.453296      10

# 3. ¿Y si usamos el paquete Matrix?
# Convertimos R_densa a una matriz dispersa triangular
R_sparse <- as(R_densa, "sparseMatrix")
# O, mejor aún, la definimos como triangular desde el principio
R_sparse_tri <- as(R_densa, "dtTMatrix") # "d"ouble, "t"riangular, "T"riplet

# El 'solve' de Matrix SÍ detecta la estructura
# mbm_sparse <- microbenchmark(
#   generico_denso = solve(R_densa, b),
#   solve_disperso = solve(R_sparse, b),
#   solve_triangular = solve(R_sparse_tri, b),
#   times = 10
# )
# print(mbm_sparse, unit = "ms")
```

De los resultados, observamos que `backsolve` es significativamente más rápido que `solve` en matrices densas de R base. Pero aún más importante, el método `solve` del paquete `Matrix` es *extremadamente* rápido, ya que detecta

la estructura triangular y aplica el solver más eficiente posible (sustitución progresiva/regresiva) sobre la estructura de datos dispersa.

■

2.3.3 Matrices banda y tridiagonales

El caso más importante de matrices estructuradas en la práctica son las **matrices banda**.

Definición 2.4 — Matriz Banda. Una matriz A es una **matriz banda** si todos sus elementos no nulos se encuentran en una banda alrededor de la diagonal principal. Formalmente, $a_{ij} = 0$ si $|i - j| > k$, donde k es el **semiancho de banda** (*bandwidth*).

Una matriz A es una **matriz banda** si todos sus elementos no nulos se encuentran confinados en una franja alrededor de la diagonal principal. Dependiendo del ancho de dicha franja (k), recibimos nombres específicos: si $k = 0$ estamos ante una matriz **diagonal**, mientras que $k = 1$ define una matriz **tridiagonal** y $k = 2$ una **pentadiagonal**.

Las matrices tridiagonales son ubicuas en la modelización matemática, apareciendo frecuentemente en la discretización de ecuaciones diferenciales (como la Ecuación del Calor), en el cálculo de *splines* para interpolación estadística y en diversos modelos de dinámica de poblaciones.

El Algoritmo de Thomas Para los sistemas tridiagonales $A\mathbf{x} = \mathbf{b}$, no es necesario usar Eliminación de Gauss ($O(n^3)$) ni siquiera un solver de banda general. Existe un algoritmo específico, llamado **Algoritmo de Thomas**, que es una versión simplificada de la eliminación de Gauss que resuelve el sistema en coste de tiempo y memoria $O(n)$.

Llewellyn Thomas

Placeholder
Image

Llewellyn Thomas (1903-1992) fue un físico y matemático británico-estadounidense. Es famoso por la “precesión de Thomas” en mecánica cuántica. Desarrolló este algoritmo en 1949 para resolver problemas de física de fluidos. En la Unión Soviética, se conocía como el “método de *progonka*”.

Cuando usamos `solve(A, b)` sobre una matriz tridiagonal del paquete **Matrix**, éste detecta la estructura de banda y aplica un solver altamente optimizado (como una versión del Algoritmo de Thomas o una factorización LU específica para bandas) que es, en efecto, $O(n)$.

■ **Ejemplo 2.5 — Ejemplo: Creación y Resolución de un Sistema Tridiagonal.** Vamos a crear la matriz tridiagonal de la práctica final (con $n = 5$) y a resolverla.

$$A = \begin{pmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 100 \end{pmatrix}$$

```

n <- 5
# 1. Definir las diagonales
diag_principal <- rep(2, n)
sub_diag <- rep(-1, n - 1)
super_diag <- rep(-1, n - 1)

# 2. Definir las posiciones (i, j)
i_idx <- c(1:n, 2:n, 1:(n - 1))
j_idx <- c(1:n, 1:(n - 1), 2:n)
x_val <- c(diag_principal, sub_diag, super_diag)

# 3. Crear la matriz dispersa
A_tridiag <- sparseMatrix(i = i_idx, j = j_idx, x = x_val)

```

```

# 4. Crear el vector b
b <- rep(0, n)
b[n] <- 100

```

```
print(A_tridiag)
```

5 x 5 sparse Matrix of class "dgCMatrix"

```
[1,] 2 -1 . . .
[2,] -1 2 -1 . .
[3,] . -1 2 -1 .
[4,] . . -1 2 -1
[5,] . . . -1 2
```

```
print(b)
```

```
[1] 0 0 0 0 100
```

```
# 5. Resolver
x <- solve(A_tridiag, b)
print(x)
```

```
[1] 16.66667 33.33333 50.00000 66.66667 83.33333
```

El resultado (20, 40, 60, 80, 100) tiene perfecto sentido físico: la temperatura se distribuye linealmente a lo largo de la barra.

■ Ejemplo 2.6 — Comparativa de Rendimiento: Densa vs. Dispersa.

Este es el ejemplo más importante del capítulo. Vamos a comparar la resolución de un sistema tridiagonal de $n = 5000$ usando una matriz densa (R base) frente a una matriz dispersa (Matrix).

```

n <- 5000

# --- 1. Matriz Densa ---
A_densa <- matrix(0, n, n)
diag(A_densa) <- 2
# Rellena las sub/super-diagonales (¡muy lento en R!)
for (i in 1:(n - 1)) {
  A_densa[i, i + 1] <- -1
  A_densa[i + 1, i] <- -1
}
b <- rep(0, n)
b[n] <- 100

# --- 2. Matriz Dispersa ---
i_idx <- c(1:n, 2:n, 1:(n - 1))
j_idx <- c(1:n, 1:(n - 1), 2:n)
x_val <- c(rep(2, n), rep(-1, (n - 1) * 2))
A_dispersa <- sparseMatrix(i = i_idx, j = j_idx, x = x_val)

# --- 3. Comparativa de Tiempos ---
# NOTA: Ejecutamos el denso solo 1 vez porque es MUY lento.
print(system.time({
  x_densa <- solve(A_densa, b)
}))
```

| | user | system | elapsed |
|--|--------|--------|---------|
| | 11.995 | 0.031 | 12.072 |

```

print(system.time({
  x_dispersa <- solve(A_dispersa, b)
}))
```

| | user | system | elapsed |
|--|-------|--------|---------|
| | 0.000 | 0.000 | 0.001 |

El resultado es dramático. En una máquina estándar:

- La solución densa puede tardar **más de 10 segundos** (y consumir ~ 200 MB de RAM).
- La solución dispersa tarda **menos de 0.01 segundos** (y consumir ~ 150 KB de RAM).

Para la modelización predictiva y los sistemas complejos, la diferencia no es solo de velocidad, sino **de viabilidad**. Un problema de $n = 50.000$ sería imposible con matrices densas, pero se resolvería en menos de un segundo con matrices dispersas.

Ejercicio 2.1

1. Adapta el código del Ejemplo 2.5 para crear una matriz A de 100×100 que sea **pentadiagonal**.
2. La diagonal principal debe tener el valor ‘10’.
3. Las primeras sub/super-diagonales deben tener el valor ‘-2’.
4. Las *segundas* sub/super-diagonales (es decir, $a_{i,i+2}$ y $a_{i,i-2}$) deben tener el valor ‘1’.
5. Visualiza la estructura de la matriz resultante con `image(A)`.

Ejercicio 2.2

1. Crea una matriz triangular superior R de 1000×1000 con números aleatorios en y sobre la diagonal. (Pista: `R <- matrix(rnorm(1000*1000), 1000); R[lower.tri(R)] <- 0`).
2. Crea un vector \mathbf{b} de 1000 números aleatorios (`b <- rnorm(1000)`).
3. Resuelve el sistema $R\mathbf{x} = \mathbf{b}$ de dos maneras:
 - a. Usando el solver genérico: `x1 <- solve(R, b)`
 - b. Usando el solver específico para sistemas triangulares: `x2 <- backsolve(R, b)`
4. Usa el paquete `microbenchmark` para comparar el tiempo de ambas operaciones: `microbenchmark(solve(R, b), backsolve(R, b), times = 10)`.
5. ¿Qué conclusiones extraes sobre usar la función adecuada a la estructura de la matriz?

2.4 Estabilidad numérica y condicionamiento

Hasta ahora, hemos asumido que la solución que nos da el ordenador es fiable. Sin embargo, en el mundo de la aritmética de punto flotante, los errores de redondeo pueden acumularse y destruir la precisión si el sistema es inestable.

Definición 2.5 — Número de Condición. El **número de condición** de una matriz, denotado como $\kappa(A) = \|A\| \cdot \|A^{-1}\|$, mide cuánto se amplifica un pequeño cambio en los datos de entrada (\mathbf{b}) en la solución (\mathbf{x}).

- Si $\kappa(A) \approx 1$, el sistema es **bien condicionado**.
- Si $\kappa(A)$ es muy grande (ej. 10^{16}), el sistema es **mal condicionado** o singular.

En R, podemos estimar el recíproco del número de condición con `rcond()` o calcularlo (para matrices pequeñas) con `kappa()`.

2.5 Métodos iterativos para sistemas grandes

Cuando una matriz A es tan masiva que ni siquiera la factorización LU/Cholesky es viable (pensemos en $n = 10^8$), o cuando la matriz es densa pero muy grande, los métodos directos fallan.

La alternativa son los **métodos iterativos**. Estos métodos no dan la solución exacta, sino que construyen una sucesión de vectores $\mathbf{x}^{(k)}$ que *converge* a la solución real \mathbf{x} .

Definición 2.6 — Método Iterativo. Un método iterativo reescribe $A\mathbf{x} = \mathbf{b}$ en una forma de punto fijo equivalente: $\mathbf{x} = T\mathbf{x} + \mathbf{c}$.

Comenzando con una estimación inicial $\mathbf{x}^{(0)}$ (a menudo 0), se genera la sucesión:

$$\mathbf{x}^{(k+1)} = T\mathbf{x}^{(k)} + \mathbf{c}$$

Si el método converge, $\lim_{k \rightarrow \infty} \mathbf{x}^{(k)} = \mathbf{x}$.

Para descomponer A , usamos la notación $A = D + L + R$, donde:

- D es la matriz diagonal de A .
- L es la matriz triangular estrictamente inferior de A .
- R es la matriz triangular estrictamente superior de A .

2.5.1 Método de Jacobi

El método de Jacobi se deriva de reescribir $A\mathbf{x} = \mathbf{b}$ como $(D + L + R)\mathbf{x} = \mathbf{b}$, y despejando la x de la parte diagonal:

$$\begin{aligned} D\mathbf{x} &= -(L + R)\mathbf{x} + \mathbf{b} \\ \mathbf{x} &= -D^{-1}(L + R)\mathbf{x} + D^{-1}\mathbf{b} \end{aligned}$$

Esto nos da la fórmula iterativa de Jacobi:

$$\mathbf{x}^{(k+1)} = -D^{-1}(L + R)\mathbf{x}^{(k)} + D^{-1}\mathbf{b}$$

En la práctica, esto significa que para calcular cada componente $x_i^{(k+1)}$, usamos *solo* los valores de la iteración anterior $\mathbf{x}^{(k)}$.

■ **Ejemplo 2.7 — Ejemplo: Implementación manual de Jacobi en R.**

```
jacobi_solve_r <- function(A, b, x0, max_iter = 100, tol = 1e-6) {
  n <- length(b)
  x <- x0
  x_new <- x0

  for (k in 1:max_iter) {
    for (i in 1:n) {
```

```

# Suma de a_ij * x_j para j != i
sigma <- 0
for (j in 1:n) {
  if (i != j) {
    sigma <- sigma + A[i, j] * x[j] # Usa el valor de la iteración ANTERIOR (x)
  }
}
# Calcula el nuevo valor para x_i
x_new[i] <- (b[i] - sigma) / A[i, i]
}

# Comprueba la convergencia (criterio de parada)
if (max(abs(x_new - x)) < tol) {
  message(paste("Jacobi convergió en", k, "iteraciones."))
  return(x_new)
}

x <- x_new # Actualiza para la siguiente iteración
}

warning("Jacobi no convergió en max_iter.")
return(x)
}

# Sistema del ejemplo anterior
A <- matrix(c(4, 1, -1, 1, 6, 2, -1, 2, 5), nrow = 3, byrow = TRUE)
b <- c(6, 15, 12)
x0 <- c(0, 0, 0) # Empezamos desde cero

jacobi_solve_r(A, b, x0)

```

Jacobi convergió en 22 iteraciones.

[1] 1.651685 1.516855 2.123595

■

2.5.2 Método de Gauss-Seidel

El método de Gauss-Seidel es una optimización simple de Jacobi. Cuando estamos calculando $x_i^{(k+1)}$, ya hemos calculado $x_1^{(k+1)}, \dots, x_{i-1}^{(k+1)}$ en la misma iteración. ¿Por qué no usar esos valores *nuevos* y supuestamente mejores en lugar de los antiguos de $x^{(k)}$?

La fórmula se deriva de $(D + L)\mathbf{x} = -R\mathbf{x} + \mathbf{b}$:

$$\mathbf{x}^{(k+1)} = -(D + L)^{-1}R\mathbf{x}^{(k)} + (D + L)^{-1}\mathbf{b}$$

■ **Ejemplo 2.8 — Ejemplo: Implementación manual de Gauss-Seidel en R.**

```

gauss_seidel_solve_r <- function(A, b, x0, max_iter = 100, tol = 1e-6) {
  n <- length(b)
  x <- x0

  for (k in 1:max_iter) {
    x_old <- x # Guardamos el x de la iteración anterior para la convergencia

    for (i in 1:n) {
      sigma_L <- 0 # Suma de la parte L (usa valores nuevos)
      if (i > 1) {
        for (j in 1:(i - 1)) {
          sigma_L <- sigma_L + A[i, j] * x[j] # Usa x[j] que ya es x_i^(k+1)
        }
      }

      sigma_R <- 0 # Suma de la parte R (usa valores antiguos)
      if (i < n) {
        for (j in (i + 1):n) {
          sigma_R <- sigma_R + A[i, j] * x_old[j] # Usa x_old[j] que es x_j^(k)
        }
      }

      # Calcula y ACTUALIZA x[i] inmediatamente
      x[i] <- (b[i] - sigma_L - sigma_R) / A[i, i]
    }

    # Comprueba la convergencia
    if (max(abs(x - x_old)) < tol) {
      message(paste("Gauss-Seidel convergió en", k, "iteraciones."))
      return(x)
    }
  }

  warning("Gauss-Seidel no convergió en max_iter.")
  return(x)
}

# Mismo sistema
gauss_seidel_solve_r(A, b, x0)

```

Gauss-Seidel convergió en 13 iteraciones.

[1] 1.651685 1.516854 2.123595



Generalmente, Gauss-Seidel converge más rápido que Jacobi.

2.5.3 Convergencia de métodos iterativos

No todos los sistemas convergen con estos métodos. Una condición *suficiente* (pero no necesaria) muy útil es la de diagonal dominante.

Definición 2.7 — Matriz de Diagonal Estrictamente Dominante.

Una matriz A es de **diagonal estrictamente dominante** (por filas) si para cada fila i , el valor absoluto del elemento de la diagonal es *mayor* que la suma de los valores absolutos de los demás elementos de esa fila.

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}| \quad \text{para todo } i$$

Teorema 2.1 Si A es de diagonal estrictamente dominante, los métodos de Jacobi y Gauss-Seidel convergen a la única solución del sistema, sin importar el vector inicial $\mathbf{x}^{(0)}$.

Ejercicio 2.3

1. Comprueba (a mano o con R) si la matriz A del ejemplo anterior (Ejemplo 2.7) es de diagonal estrictamente dominante.
2. Crea la siguiente matriz B : $B \leftarrow \text{matrix}(c(1, 2, 3, 4, 5, 6, 7, 8, 9), \text{nrow} = 3)$.
3. Intenta resolver $B_x = b$ usando la función `gauss_seidel_solve_r`. ¿Qué ocurre? ¿Por qué? (Pista: Comprueba la diagonal de B). ■

2.5.4 Método del gradiente conjugado (CG)

Para matrices **simétricas y definidas positivas** (muy comunes en física y optimización), el método del Gradiente Conjugado es generalmente superior a Jacobi y Gauss-Seidel.

A diferencia de los anteriores, que “suavizan” el error localmente, CG busca la solución a lo largo de direcciones “conjugadas” (ortogonales respecto a la métrica inducida por A). Teóricamente, en aritmética exacta, converge en n pasos. En aritmética flotante, se utiliza como método iterativo y suele converger mucho antes para una tolerancia dada.

2.5.5 Sistemas estructurados por bloques y complemento de Schur

En muchos problemas de ingeniería (multifísica, interacción fluido-estructura), las matrices tienen una estructura natural de bloques:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{pmatrix}$$

Si el bloque A_{11} es fácil de invertir (ej. diagonal o triangular), podemos despejar \mathbf{x}_1 de la primera ecuación y sustituirlo en la segunda, obteniendo un sistema reducido para \mathbf{x}_2 :

$$(A_{22} - A_{21}A_{11}^{-1}A_{12})\mathbf{x}_2 = \mathbf{b}_2 - A_{21}A_{11}^{-1}\mathbf{b}_1$$

La matriz $S = A_{22} - A_{21}A_{11}^{-1}A_{12}$ se llama **Complemento de Schur**. Resolver este sistema más pequeño ($\dim(A_{22})$) suele ser mucho más eficiente que atacar el sistema completo.

2.6 Práctica Final: Simulación de Red de Calor

Objetivo: Modelar un problema físico simple que genera un sistema de ecuaciones grande y disperso, y comparar la eficiencia de los métodos de resolución.

Problema: Imaginemos una barra de metal de 1 metro. Mantenemos el extremo izquierdo a 0°C y el extremo derecho a 100°C . Queremos encontrar la temperatura *en equilibrio* en n puntos interiores de la barra.

La física (usando la ecuación del calor en estado estacionario) nos dice que la temperatura en un punto i es el promedio de la temperatura de sus vecinos $i-1$ e $i+1$.

$$T_i = \frac{T_{i-1} + T_{i+1}}{2} \implies -T_{i-1} + 2T_i - T_{i+1} = 0$$

Si discretizamos la barra en $n = 10$ puntos interiores (T_1, \dots, T_{10}), con los bordes $T_0 = 0$ y $T_{11} = 100$, generamos un sistema de ecuaciones:

- $i = 1: -T_0 + 2T_1 - T_2 = 0 \implies 2T_1 - T_2 = T_0 \implies 2T_1 - T_2 = 0$
- $i = 2: -T_1 + 2T_2 - T_3 = 0$
- ...
- $i = 10: -T_9 + 2T_{10} - T_{11} = 0 \implies -T_9 + 2T_{10} = T_{11} \implies -T_9 + 2T_{10} = 100$

Esto genera una matriz A de 10×10 tridiagonal y un vector b :

$$A = \begin{pmatrix} 2 & -1 & 0 & \dots \\ -1 & 2 & -1 & \dots \\ 0 & -1 & 2 & \dots \\ \vdots & \vdots & \ddots & -1 \\ \dots & 0 & -1 & 2 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 100 \end{pmatrix}$$

Tu Tarea:

1. **Crear la Matriz y el Vector (n=1000):** No lo hagas con $n = 10$, ¡hazlo con $n = 1000$! Usa `sparseMatrix()` para crear la matriz A (1000×1000) y el vector b (1000×1).

- (Pista para A : La diagonal principal tiene $i = 1 : n, j = 1 : n, x = 2$. La subdiagonal tiene $i = 2 : n, j = 1 : (n - 1), x = -1$. La superdiagonal tiene $i = 1 : (n - 1), j = 2 : n, x = -1$).
 - (Pista para b : es todo ceros, excepto el último elemento $b[n] = 100$).
2. **Resolver y Comparar Tiempos:** Usa `microbenchmark::microbenchmark()` para comparar el tiempo de ejecución de los siguientes métodos para resolver $A\mathbf{x} = \mathbf{b}$ (con $n = 1000$):
- `solve(A, b)` (El solver inteligente de `Matrix`)
 - `chol(A)` (Solo el tiempo de la factorización)
 - `gauss_seidel_solve_r(as.matrix(A), b, x0 = rep(0, n))` (Nuestra función manual. Nota: *Tuvimos que convertir A a densa con as.matrix() porque nuestra función no soporta matrices dispersas*).
 - `jacobi_solve_r(as.matrix(A), b, x0 = rep(0, n))`
3. **Analizar los Resultados:**
- Escribe una breve conclusión. ¿Es A de diagonal dominante? ¿Por qué convergen los métodos iterativos?
 - ¿Cuál fue el método más rápido?
 - ¿Qué crees que pasaría si $n = 100.000$ (no lo ejecutes, solo razona-lo)? ¿Qué método(s) fallaría(n) por problemas de memoria al usar `as.matrix()`?

2.7 Ejercicios Adicionales

Ejercicio 2.4

Usando `sparseMatrix()`, construye una matriz A de 100×100 que tenga la diagonal principal llena de ‘5’, las *segundas* subdiagonales ($k=-2$) y superdiagonales ($k=2$) llenas de ‘1’, y el resto ceros. Visualízala con `image(A)`.

Ejercicio 2.5

La matriz A del Ejemplo 2.2 es simétrica y definida positiva. Resuelve el sistema $A^2\mathbf{x} = \mathbf{b}$ (es decir, $A(A\mathbf{x}) = \mathbf{b}$).

Pista: No calcules A^2 . Usa la factorización $A = R^T R$ y resuelve $R^T(R(R^T(R\mathbf{x}))) = \mathbf{b}$ en cuatro pasos de sustitución progresiva/regresiva.

3

Autovalores, Autovectores y Descomposición Singular

Hasta ahora hemos resuelto $Ax = b$. Pero en modelización predictiva (ej. PageRank, PCA, estabilidad dinámica), nos interesa resolver:

$$A\mathbf{v} = \lambda\mathbf{v}$$

Donde λ es el autovalor y \mathbf{v} el autovector. Para matrices gigantes ($N > 10^5$), no podemos usar `eigen(A)` de R base. Necesitamos algoritmos iterativos.

¿Por qué “Eigen”?

Placeholder

Image

La palabra *eigen* es alemana y significa “propio” o “característico”. El término fue popularizado por **David Hilbert** en 1904, aunque el concepto se remonta a los trabajos de **Leonhard Euler** sobre la rotación de cuerpos rígidos y de **Augustin-Louis Cauchy** sobre las formas cuadráticas. En español, a veces se les llama “valores propios” o “autovalores”.

3.1 El método de las potencias (the power method)

Este método busca encontrar el **autovalor dominante** (el de mayor valor absoluto, $|\lambda_1|$). Es fundamental porque es el motor matemático detrás del algoritmo **PageRank** de Google.

3.1.1 Algoritmo

Dado un vector inicial aleatorio \mathbf{b}_0 :

$$\mathbf{b}_{k+1} = \frac{A\mathbf{b}_k}{\|A\mathbf{b}_k\|}$$

La sucesión converge al autovector dominante.

```

power_method <- function(A, epsilon = 1e-6, max_iter = 1000) {
  # 1. Vector inicial aleatorio (normalizado)
  n <- nrow(A)
  x <- rnorm(n)
  x <- x / sqrt(sum(x^2))

  lambda_old <- 0

  for(i in 1:max_iter) {
    # 2. Multiplicación (aprovecha la dispersión de A si es sparse)
    x_new <- as.vector(A %*% x)

    # 3. Estimación del autovalor (Cociente de Rayleigh simple)
    # Si x está normalizado, lambda es aproximadamente t(x) %*% A %*% x
    # Aquí simplificamos usando la norma del nuevo vector antes de normalizar
    lambda <- sqrt(sum(x_new^2))

    # Recuperar signo si es necesario (para autovalores reales)
    # if(sum(x_new * x) < 0) lambda <- -lambda

    # 4. Normalización
    x_new <- x_new / sqrt(sum(x_new^2))

    # 5. Convergencia
    if(abs(lambda - lambda_old) < epsilon) {
      return(list(
        eigenvalue = lambda,
        eigenvector = x_new,
        iterations = i
      ))
    }
  }

  x <- x_new
  lambda_old <- lambda
}
warning("El método no convergió")
return(list(eigenvalue = lambda, eigenvector = x, iterations = max_iter))
}

```

Probemos el método con una matriz dispersa grande pero sencilla.

```

# Creamos una matriz dispersa simétrica de 500x500
set.seed(2026)
n <- 500
M <- rsparsematrix(n, n, density = 0.01, symmetric = TRUE)

# Método manual

```

```
resultado <- power_method(M)

# Comparación con función base
# (que usa LAPACK, mucho más lento en muy alta dimensión)
real <- eigen(as.matrix(M), only.values = TRUE)$values[1]

cat("Método Potencias: ", resultado$eigenvalue, "\n")
```

Método Potencias: 5.307119

```
cat("Eigen de R base: ", real, "\n")
```

Eigen de R base: 5.291721

```
cat("Iteraciones:      ", resultado$iterations)
```

Iteraciones: 795

3.1.2 Determinación del autovalor (el cociente de Rayleigh)

En el algoritmo anterior, hemos visto cómo el vector converge a la dirección del autovector \mathbf{v} . Pero, ¿cómo obtenemos el valor escalar λ ?

La forma más precisa de estimarlo es mediante el **Cociente de Rayleigh**. Si $A\mathbf{v} \approx \lambda\mathbf{v}$, multiplicando por la izquierda por \mathbf{v}^T :

$$\mathbf{v}^T A \mathbf{v} \approx \lambda \mathbf{v}^T \mathbf{v}$$

Despejando λ :

$$\lambda = \frac{\mathbf{v}^T A \mathbf{v}}{\mathbf{v}^T \mathbf{v}}$$

Si nuestro autovector está normalizado ($\|\mathbf{v}\| = 1$), la fórmula se simplifica elegantemente a:

$$\lambda = \mathbf{v}^T A \mathbf{v}$$

3.1.3 Uso de la librería matlib para docencia

Aunque implementar el algoritmo manualmente es vital para entenderlo, R dispone de herramientas didácticas en el paquete **matlib**. La función **powerMethod()** no solo calcula el resultado, sino que **muestra la evolución de las iteraciones**, lo cual es fantástico para visualizar la velocidad de convergencia.

```
# Instalación si no está presente: install.packages("matlib")
library(matlib)

# Definimos una matriz sencilla 2x2
A <- matrix(c(4, 1,
```

```

2, 3), nrow = 2, byrow = TRUE)

# Ejecutamos el método de la potencia visualizando pasos
# plot = FALSE para no generar gráfico en este PDF (opcional)
resultado_matlib <- powerMethod(A, v = c(1, 1), plot = FALSE)

print(resultado_matlib)

$vector_iterations
      v1          v2
[1,] 1 0.7071068
[2,] 1 0.7071068

$iter
[1] 2

$vector
      [,1]
[1,] 0.7071068
[2,] 0.7071068

$value
[1] 5

```

Advertencia de Ingeniería: matlib vs Matrix

La librería matlib está diseñada con fines pedagógicos y trabaja con matrices densas estándar. Para los sistemas reales de ingeniería (matrices sparse de dimensión 10^5), NO debe usar matlib. En su lugar, utilice implementaciones manuales optimizadas con el paquete Matrix (como la función power_method que definimos anteriormente) o librerías de alto rendimiento como RSpectra.

3.2 El Algoritmo QR (espectro completo)

Mientras que el método de las potencias encuentra solo el autovalor dominante, el **Algoritmo QR** es el “caballo de batalla” del álgebra lineal numérica moderna para encontrar **todos** los autovalores de una matriz.

3.2.1 La Lógica del Algoritmo

El método se basa en una iteración sorprendentemente simple llamada **iteración QR**. Comenzamos con $A_0 = A$ y en cada paso k :

1. **Factorizamos** la matriz actual en una ortogonal Q_k y una triangular R_k tal que $A_k = Q_k R_k$.

2. **Multiplicamos** en orden inverso para obtener la siguiente matriz: $A_{k+1} = R_k Q_k$.

Esta “mezcla” preserva los autovalores (las matrices son semejantes), pero empuja los valores hacia la diagonal principal. Si la matriz es simétrica, A_k converge a una matriz diagonal donde los elementos a_{ii} son los autovalores.

```
qr_simple_algo <- function(A, iter = 20) {
  A_k <- A
  # Guardamos la evolución para visualizar (opcional)
  historia <- numeric(iter)

  for(i in 1:iter) {
    # 1. Descomposición QR: A = Q * R
    descomp <- qr(A_k)
    Q <- qr.Q(descomp)
    R <- qr.R(descomp)

    # 2. Actualización: A_new = R * Q
    A_k <- R %*% Q

    # Guardamos el valor de la esquina superior (para ver convergencia)
    historia[i] <- A_k[1,1]
  }

  return(list(matriz_final = A_k, autovalores = diag(A_k)))
}

# --- PRUEBA DEL CONCEPTO ---
# Matriz simétrica 3x3
M <- matrix(c(5, 2, 0,
              2, 3, 1,
              0, 1, 1), nrow = 3)

resultado <- qr_simple_algo(M)

print("Matriz final (casi diagonal):")

[1] "Matriz final (casi diagonal):"
print(round(resultado$matriz_final, 3))

[,1]  [,2]  [,3]
[1,] 6.29 0.000 0.000
[2,] 0.00 2.294 0.000
[3,] 0.00 0.000 0.416

cat("\nAutovalores estimados (Diagonal):",
    round(resultado$autovalores, 4), "\n")
```

```
Autovalores estimados (Diagonal): 6.2899 2.2943 0.4158
cat("Autovalores reales (función eigen):", round(eigen(M)$values, 4))

Autovalores reales (función eigen): 6.2899 2.2943 0.4158
```

Nota de Ingeniería

En R, para matrices dispersas, raramente calculamos *todos* los autovalores (sería demasiado costoso). Usamos métodos como **Arnoldi** / **Lanczos** (disponibles en paquetes como **RSpec**) que generalizan el método de las potencias para encontrar los primeros autovalores.

3.3 Descomposición en Valores Singulares (SVD)

Aunque los autovalores son fundamentales para matrices cuadradas, la **Descomposición en Valores Singulares (SVD)** es la herramienta universal del álgebra lineal, aplicable a **cualquier** matriz rectangular.

Definición 3.1 — Teorema SVD. Toda matriz real A de dimensión $m \times n$ puede factorizarse como:

$$A = U\Sigma V^T$$

Donde U ($m \times m$) y V ($n \times n$) son matrices ortogonales, mientras que Σ ($m \times n$) es una matriz diagonal que contiene los **valores singulares** no negativos, ordenados decrecientemente ($\sigma_1 \geq \sigma_2 \geq \dots \geq 0$).

La SVD es el motor matemático detrás de numerosas aplicaciones modernas. En la **compresión de imágenes**, nos permite aproximar una matriz visual guardando solo los primeros k valores singulares (Teorema de Eckart-Young). En los **sistemas de recomendación**, es la base del filtrado colaborativo matricial. Además, el **Análisis de Componentes Principales (PCA)** no es más que una SVD aplicada a la matriz de datos centrada en la media.

4

Algoritmo PageRank

A finales de la década de 1990, cuando Google entró en línea, un factor clave que lo distinguía de otros motores de búsqueda era su capacidad para ofrecer constantemente los resultados más relevantes en la parte superior de las listas de búsqueda. A diferencia de otros motores de búsqueda, donde los usuarios a menudo tenían que cribar páginas de enlaces irrelevantes que simplemente coincidían con la consulta de búsqueda, Google presentaba una experiencia más optimizada.

Esta magia reside en parte en el algoritmo PageRank de Google. Este algoritmo asigna un valor cuantitativo a cada página web, valorando esencialmente su importancia. Al aprovechar PageRank, Google puede clasificar las páginas web y priorizar las más importantes (que normalmente se correlacionan con los resultados más relevantes y útiles) en las listas de búsqueda.

¿Larry Page o Páginas Web?

Placeholder

Image

Existe una ambigüedad deliberada en el nombre **PageRank**. Aunque se aplica a la clasificación de “páginas” (*pages*), el nombre rinde homenaje a uno de sus creadores y fundadores de Google, **Larry Page**. Originalmente, el proyecto en la Universidad de Stanford se llamaba **“BackRub”**, debido a que el algoritmo analizaba los “backlinks” (enlaces entrantes) para determinar la relevancia.

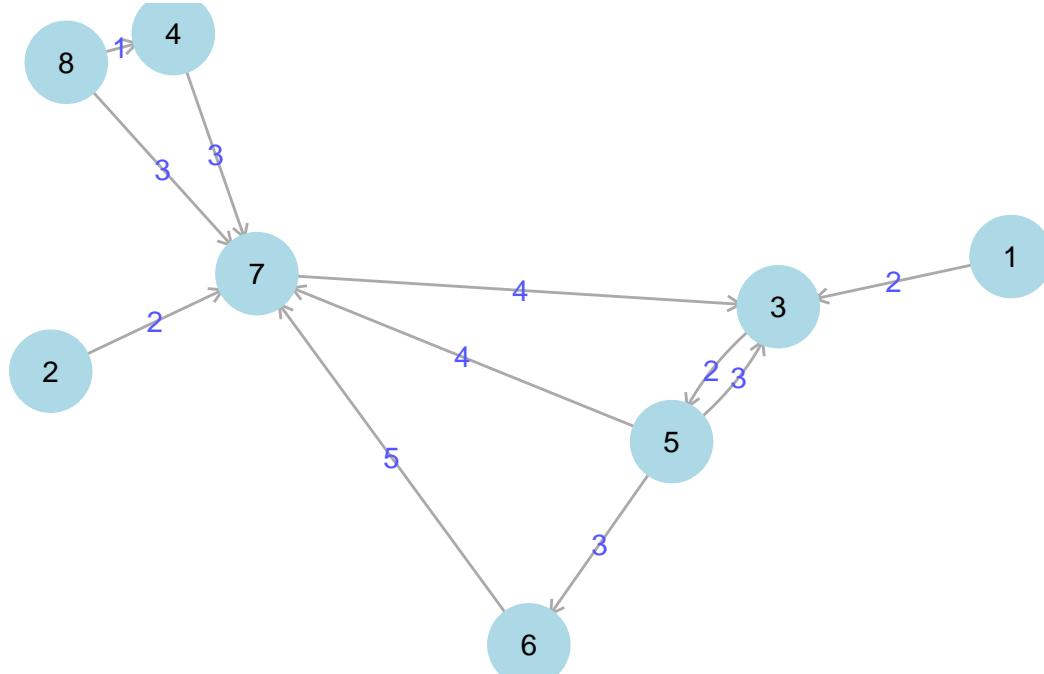
Nuestro objetivo es establecer un método para asignar puntuaciones de importancia a cada página web dentro de la base de datos. Esto permitirá al sistema priorizar estas puntuaciones cuando un usuario realice una búsqueda y se identifique un subconjunto de páginas relevantes. El foco de este artículo está en el paso clave de definir y cuantificar la “importancia” de una página web dentro de la estructura interconectada de la web. Es importante tener en cuenta que si bien la calificación de importancia de una página web es un factor significativo, no es el único determinante de cómo se presentan los enlaces en los resultados de búsqueda.

Por supuesto, este método no es exclusivo de la web, sino que es aplicable a cualquier grafo dirigido ponderado.

4.1 El algoritmo PageRank

A lo largo de esta presentación, haremos uso de un ejemplo de juguete para comprender mejor la idea y los conceptos subyacentes.

Consideremos una red de páginas web como la del gráfico inferior, todas relacionadas con un mismo término de una búsqueda.



Las flechas de las aristas entre una web y otra indican la existencia de un enlace en la primera a la segunda. Por ejemplo, la web número 7 tiene cinco enlaces entrantes (desde las páginas 2, 4, 5, 6 y 8) y solo uno saliente (hacia la página 3).

El número en la arista del grafo que une las páginas i y j indica el número de visitantes de la página i que a continuación visitaron la página j . Por ejemplo, del total de personas que visitaron la página 8, una visitó después la página 4 y tres visitaron la página 7.

La idea del buscador de Google fue presentar esa lista de 8 posibles webs relacionadas con un tema común priorizada según la importancia de cada web.

¿Cómo sabemos la importancia de cada web? La idea de Google se basaba en que la importancia se propaga: cuantas más veces se pase de una web “A” a otra web “B”, más *hereda* la web “B” importancia de “A”. Veamos esta idea con nuestro ejemplo sencillo.

Vamos a denotar por $x_i \in \mathbb{R}^+$ la importancia de la web i -ésima. Este marco de trabajo nos dice que la importancia que tiene la web i proviene de las importancias de las webs que la enlazan.

En el ejemplo, la importancia de la web número 7, es decir, x_7 , proviene de las

importancias que le aportan las webs número 2, 4, 5, 6 y 8.

Vamos a presentar en forma de matriz las conexiones entre webs (es realmente la matriz de adyacencia *ponderada* del grafo de antes, traspuesta)

```
Warning: `as_adj()` was deprecated in igraph 2.1.0.
i Please use `as_adjacency_matrix()` instead.
```

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 3 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 2 & 0 & 3 & 4 & 5 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

En esta matriz, en (i, j) se almacena el número de veces que se pasa del nodo j al nodo i , es decir, cuántas veces el nodo i recibe un enlace desde el nodo j .

Vamos a normalizar esta matriz, dividiendo cada columna por su suma. Mediante esta normalización, lo que se está expresando en el elemento (i, j) es la probabilidad de transitar desde la web j -ésima a la i -ésima.

Probabilidades Reales

Placeholder

Image

Nota: En casos reales, se suele conocer estas probabilidades a ciencia cierta, gracias al rastreo que hacen de nuestra actividad en internet. Esta matriz, en ese caso, reproduciría exactamente esas probabilidades de transición.

A esta matriz se la denomina matriz de transiciones y tiene la propiedad de ser *estocástica por columnas*, es decir, sus columnas suman uno. Esta propiedad es esencial para el desarrollo teórico del algoritmo PageRank. Una vez normalizada, la matriz de transición queda como sigue:

$$M = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 3/10 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/4 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3/10 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 2/5 & 1 & 0 & 3/4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Teniendo en cuenta esta matriz, podemos escribir las relaciones entre las

importancias (los x_i) de las distintas webs:

$$\left\{ \begin{array}{l} x_1 \\ x_2 \\ x_3 \end{array} \right. \quad \begin{array}{l} +3/10x_5 \\ +x_4 \\ +2/5x_5 \end{array} \quad \begin{array}{l} +x_7 \\ +x_6 \end{array} \quad \begin{array}{l} 0 = x_1 \\ 0 = x_2 \\ = x_3 \\ 1/4x_8 = x_4 \\ = x_5 \\ = x_6 \\ +3/4x_8 = x_7 \\ 0 = x_8 \end{array}$$

Por ejemplo, la primera ecuación nos dice que la importancia de la web número 1 es 0 (ya que ninguna web enlaza a dicha página), y, mirando la ecuación correspondiente, vemos que la importancia de la web 3 proviene de la siguiente forma: toda la importancia de la web 1 más 3/10 de la importancia de la web 5, más toda la de la web 7, todo esto relacionado con los enlaces que van de unas a otras.

Si nos fijamos en la forma matricial del problema, estamos buscando un vector de importancias, $= (x_i)$ que verifique $M =$, es decir, un *punto fijo* de las importancias.

En este caso, una posible solución es:

$$= \begin{pmatrix} 0 \\ 0 \\ 10/7 \\ 0 \\ 10/7 \\ 3/7 \\ 1 \\ 0 \end{pmatrix}$$

Es común normalizar este vector de importancias para que su suma dé uno. En este caso, nos queda:

$$= \begin{pmatrix} 0 \\ 0 \\ 1/3 \\ 0 \\ 1/3 \\ 1/10 \\ 7/30 \\ 0 \end{pmatrix} \approx \begin{pmatrix} 0 \\ 0 \\ 0.33333 \\ 0 \\ 0.33333 \\ 0.1 \\ 0.23333 \\ 0 \end{pmatrix}$$

que también verifica $M =$. Como resultado, vemos que las páginas más importantes son la 3 y la 5, mientras que las 1, 2, 4 y 8 tienen importancia nula.

4.2 Relación con el álgebra lineal

Según la formulación anterior del problema, tenemos que buscar un vector $\mathbf{x} = (x_1, x_2, \dots, x_n)^t$ tal que $M\mathbf{x} = \mathbf{v}$, lo cual es equivalente a encontrar un autovector asociado al autovalor 1.

Por tanto, encontrar los vectores que lo cumplen es resolver el sistema $(M - I)\mathbf{x} = \mathbf{0}$ que, en forma matricial, se puede resolver usando Gauss-Jordan:

$$\left(\begin{array}{ccccccc} -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 3/10 & 0 & 1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3/10 & -1 & 0 \\ 0 & 1 & 0 & 1 & 2/5 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \right) \sim \left(\begin{array}{ccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -10/7 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & -10/7 \\ 0 & 0 & 0 & 0 & 0 & 1 & -3/7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \right)$$

Este último sistema se escribe como:

$$\left\{ \begin{array}{rcl} x_1 & = & 0 \\ x_2 & = & 0 \\ x_3 & -10/7x_7 & = 0 \\ x_4 & = & 0 \\ x_5 & -10/7x_7 & = 0 \\ x_6 & -3/7x_7 & = 0 \\ x_8 & = & 0 \\ 0 & = & 0 \end{array} \right.$$

Usando este último sistema, podemos obtener las ecuaciones paramétricas de su conjunto solución (el subespacio asociado al autovalor 1) y su base:

$$U = \left\{ \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{pmatrix} \in \mathbb{R}^8 \mid \begin{array}{l} x_1 = 0 \\ x_2 = 0 \\ x_3 = 10/7\alpha \\ x_4 = 0 \\ x_5 = 10/7\alpha, \quad \alpha \in \mathbb{R} \\ x_6 = 3/7\alpha \\ x_7 = \alpha \\ x_8 = 0 \end{array} \right\} = \mathcal{L} \left\{ \begin{pmatrix} 0 \\ 0 \\ 10/7 \\ 0 \\ 10/7 \\ 3/7 \\ 1 \\ 0 \end{pmatrix} \right\}$$

Podemos observar por tanto que llegamos a la solución propuesta en el apartado anterior. Es decir, hemos llegado a un subespacio que *determina* la importancia de cada nodo dentro del grafo. Tomamos entonces un vector cuya suma de las componentes sea 1 dentro de ese subespacio y tenemos las puntuaciones de importancia que queríamos.

¿Seguro que el 1 es autovalor?

Bajo condiciones suficientemente generales en la práctica, podemos asegurar que 1 es autovalor. Es fundamental que la suma por columnas sea 1. Más aún: $\lambda = 1$ es el mayor autovalor de la matriz M .

¿Y si la dimensión del subespacio es > 1?

Si el subespacio asociado tiene dimensión mayor que 1, no podríamos obtener un único *autovector* de probabilidad. Existe una estrategia (la que implementa Google), que permite asegurar que la dimensión del subespacio asociado al autovalor 1 es exactamente uno.

4.3 ¿Cómo hacerlo en R?

Una vez definida la matriz M de transiciones, es necesario, como hemos comentado antes, normalizar para que las columnas sumen 1. A modo de ayuda, copio aquí un código que puede ayudar a realizar esta normalización:

```
M <- scale(M,
            center = FALSE,
            scale = colSums(M))
```

Una vez que tenemos esta matriz de transición, existen múltiples librerías que permiten calcular valores y vectores propios. De hecho, el propio R base tiene la función `eigen()` que calcula tanto los autovalores como los autovectores de una matriz dada (teniendo en cuenta que, por defecto, va a trabajar en \mathbb{C}).

Internamente, la mayoría de librerías utilizan técnicas numéricas iterativas.

Como vimos en el Capítulo de Autovalores, el **método de la potencia** es el algoritmo estándar para encontrar el autovalor dominante (y su autovector asociado) de una matriz. Dado que en nuestro caso sabemos (por Perron-Frobenius) que el mayor autovalor es $\lambda = 1$, este método es ideal y es, de hecho, la base del algoritmo original de Google.

Podemos usar la misma función `powerMethod` de la librería `matlib` que introdujimos en el capítulo anterior:

```
library(matlib)
resultado <- powerMethod(M)
```

Podemos ver el autovalor determinado:

```
resultado$value
```

```
[1] 1
```

y el autovector correspondiente:

```
resultado$vector
```

```
[,1]
1 0.0000000
2 0.0000000
3 0.6225730
4 0.0000000
5 0.6225725
6 0.1867718
7 0.4358011
8 0.0000000
```

En nuestro caso, como siempre, queremos un vector cuya componentes sumen 1, así que hacemos lo siguiente:

```
resultado$vector / sum(resultado$vector)
```

```
[,1]
1 0.0000000
2 0.0000000
3 0.33333345
4 0.0000000
5 0.33333317
6 0.09999996
7 0.23333342
8 0.0000000
```

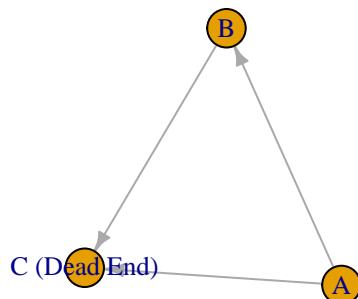
que es prácticamente igual al vector de importancias que hallamos en la sección del algoritmo PageRank.

4.4 Problemas en la web real

Hasta ahora hemos asumido un grafo ideal. Sin embargo, en la web real existen estructuras que rompen la lógica básica del PageRank si no se tratan.

4.4.1 Callejones sin salida (dead ends)

Son páginas que no tienen enlaces salientes.

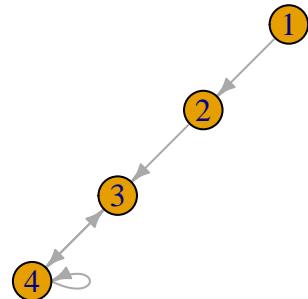


Si el navegante llega a C, no tiene a dónde ir. Matemáticamente, la columna correspondiente en la matriz de transición tendría ceros, y la matriz dejaría

de ser estocástica (sus columnas no suman 1). La importancia se “fugaría” del sistema en cada iteración hasta llegar a 0.

4.4.2 Trampas de araña (spider traps)

Son grupos de páginas que se enlazan entre sí pero no al resto de la web.



Aquí, una vez que el navegante entra en el bucle 3-4, nunca sale. Estos nodos acumularían toda la importancia del grafo, absorbiéndola del resto (Rank Sinks).

4.5 El factor de amortiguación (damping factor)

Para solucionar esto, los creadores de Google, Brin y Page, introdujeron el **factor de amortiguación (d)**.

La idea es modelar el comportamiento de un usuario que navega aleatoriamente: la mayoría de las veces sigue un enlace (con probabilidad d), pero ocasionalmente se “aburre” y salta a cualquier otra página aleatoria de la web (con probabilidad $1 - d$).

El Navegante Aleatorio

| |
|-------------|
| Placeholder |
| Image |

El modelo del “**Random Surfer**” es una metáfora intuitiva del comportamiento humano en la web. Supone que un usuario nunca se queda atrapado en una web infinitamente (trampas de araña), sino que después de unos cuantos clics, decide escribir una nueva dirección en la barra del navegador. El factor de amortiguación $d = 0.85$ implica que, en promedio, un usuario sigue unos 6 enlaces antes de “saltar” a una página al azar.

La **Matriz de Google (G)** se define entonces como:

$$G = d \cdot M + (1 - d) \cdot \frac{1}{n} J$$

En esta ecuación, M representa la matriz de transición original (previamente

corregida para manejar los “dead ends”), mientras que J es una matriz conformada exclusivamente por unos de tamaño $n \times n$. El parámetro d es el factor de amortiguación, cuyo valor estándar se establece típicamente en 0.85.

Esta nueva matriz G tiene propiedades matemáticas maravillosas (es estocástica, irreductible y primitiva), lo que garantiza por el **Teorema de Perron-Frobenius** que existe un único autovector de probabilidad con autovalor 1, y que el método de la potencia siempre converge a él.

4.5.1 Implementación en R

Vamos a modificar nuestro ejemplo para incluir este factor de amortiguación $d = 0.85$.

```
# Factor de amortiguación estándar
d <- 0.85
n <- nrow(M)

# Matriz de 'teletransportación' (probabilidad uniforme de saltar a cualquier otra página)
J <- matrix(1, nrow = n, ncol = n)

# Matriz Google
G <- d * M + (1 - d) * (1/n) * J

# Calculamos el PageRank con la matriz G usando el método de la potencia
resultado_google <- powerMethod(G)

# Autovector normalizado (PageRank final)
pagerank_scores <- resultado_google$vector / sum(resultado_google$vector)
pagerank_scores

[,1]
1 0.01875000
2 0.01875000
3 0.30960493
4 0.02273438
5 0.28191450
6 0.09063817
7 0.23885802
8 0.01875000
```

Vemos que las puntuaciones cambian ligeramente, pero el orden de importancia suele mantenerse en estructuras estables. Este vector `pagerank_scores` sería el utilizado para ordenar los resultados de búsqueda.

5

Análisis de componentes principales

El Análisis de Componentes Principales (PCA, por sus siglas en inglés) es una técnica de reducción de dimensión que permite pasar de una gran cantidad de variables interrelacionadas a unas pocas variables incorreladas entre sí llamadas **componentes principales**.

- Fue desarrollada por **Karl Pearson** a finales del siglo XIX y estudiada en profundidad por **Harold Hotelling** en los años 30 del siglo XX. Hoy es una herramienta fundamental en Ciencia de Datos.

Biometrika y el origen del PCA

Placeholder
Image

Karl Pearson publicó el primer artículo sobre PCA en **1901** en la revista *Biometrika*. Su motivación no era la computación, sino encontrar una forma de ajustar una “línea de mejor ajuste” que fuera independiente de qué variable elegíamos como “X” o “Y”. Imaginaba los datos como una nube de puntos en forma de elipsoide y buscaba los ejes principales de esa nube.

- Es un método de **aprendizaje no supervisado**: no intentamos predecir una variable respuesta (etiqueta), sino extraer información de la estructura de los datos.

5.1 Motivación: la maldición de la dimensionalidad

Imagina que tienes un conjunto de datos con muchas variables: edad, altura, peso, ingresos, nivel educativo, presión arterial, colesterol... Visualizar o analizar simultáneamente 50 variables es imposible para nuestra mente tridimensional. Si tenemos 25 variables, ¡existen $\binom{25}{2} = 300$ posibles pares de correlaciones para mirar!

Filosofía del PCA

PCA consiste en buscar combinaciones lineales de las variables originales que representen lo mejor posible a la **variabilidad** presente en los datos. Se asume que “variabilidad” equivale a “información”.

5.1.1 La metáfora del fotógrafo

Una excelente forma de entender PCA es pensar en un fotógrafo tratando de capturar una escultura tridimensional (3D) en una fotografía plana (2D).

- La escultura tiene profundidad, ancho y alto (3 dimensiones).
- El fotógrafo debe elegir el **mejor ángulo** para tomar la foto.

¿Qué considera un “buen ángulo”? Si toma la foto desde arriba, quizás solo capture la coronilla de la estatua, perdiendo información esencial y reduciéndola a un círculo plano. En cambio, si la toma de perfil, es probable que capture la mayor cantidad de detalles distintivos, maximizando así la varianza visible de la proyección.

PCA hace exactamente esto: Gira el objeto (los datos) en el espacio multidimensional hasta encontrar el ángulo (eje) donde los datos están más dispersos. Ese primer ángulo es la **Componente Principal 1**. Luego busca el siguiente mejor ángulo perpendicular al primero, y así sucesivamente.

El Factor ‘g’ de Inteligencia

Placeholder
Image

Uno de los usos históricos más famosos del PCA (y su parente cercano, el Análisis Factorial) fue en psicología. Charles Spearman observó que las puntuaciones en diferentes tests cognitivos estaban correlacionadas. Al aplicar técnicas de reducción de dimensión, propuso la existencia de un **factor general de inteligencia (g)**, que sería la primera componente principal que explica la mayor parte de la habilidad cognitiva.

5.1.2 ¿Cuándo usar PCA?

- **Cuando hay correlación:** Si las variables originales están incorreladas, PCA no sirve de mucho (los ejes originales ya son los principales).
- **Sensibilidad:** Al basarse en la varianza, PCA es muy sensible a **outliers** y a las **escalas** de las variables.
- **Preprocesamiento:** Es casi obligatorio **estandarizar** los datos antes de aplicarlo.

5.2 Fundamento teórico

Supongamos que tenemos una matriz de datos X de tamaño $n \times p$ (n observaciones, p variables), que ya hemos centrado (restado la media de cada columna).

El problema matemático es encontrar una dirección (vector unitario \mathbf{u}) tal que, al proyectar los datos sobre ella, la varianza sea máxima.

Como vimos en el Capítulo de Autovalores, esto se reduce a diagonalizar la matriz de covarianza Σ :

$$\Sigma \mathbf{u} = \lambda \mathbf{u}$$

Conexión con SVD

Las direcciones que maximizan la varianza son los **autovectores** de la matriz de covarianza Σ . La **varianza** en esa dirección es el **autovalor** correspondiente, λ .

En la práctica, usamos la **Descomposición en Valores Singulares (SVD)** de la matriz de datos X para calcular esto de forma numérica más estable: $X = U\Sigma V^T$.

5.3 Caso Práctico: Crimen en EEUU (USArrests)

Vamos a analizar el dataset **USArrests**, que contiene estadísticas de arrestos por cada 100,000 residentes por asalto, asesinato y violación en cada uno de los 50 estados de EE.UU. en 1973, además del porcentaje de población urbana.

5.3.1 1. Exploración y Preprocesamiento

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

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

Primero, veamos si tiene sentido aplicar PCA. ¿Están correlacionadas las variables?

```
cor(USArrests) |> round(2)
```

| | Murder | Assault | UrbanPop | Rape |
|----------|--------|---------|----------|------|
| Murder | 1.00 | 0.80 | 0.07 | 0.56 |
| Assault | 0.80 | 1.00 | 0.26 | 0.67 |
| UrbanPop | 0.07 | 0.26 | 1.00 | 0.41 |
| Rape | 0.56 | 0.67 | 0.41 | 1.00 |

Vemos correlaciones altas (ej: 0.80 entre Asesinato y Asalto). Hay redundancia, así que PCA es útil.

Ahora analicemos las escalas. ¿Tienen todas las variables la misma magnitud?

```
apply(USArrests, 2, mean)
```

| Murder | Assault | UrbanPop | Rape |
|--------|---------|----------|--------|
| 7.788 | 170.760 | 65.540 | 21.232 |

```
apply(USArrests, 2, sd)
```

| Murder | Assault | UrbanPop | Rape |
|----------|-----------|-----------|----------|
| 4.355510 | 83.337661 | 14.474763 | 9.366385 |

Observe que **Assault** tiene una varianza muchísimo mayor que **Murder**. Si no estandarizamos, **Assault** dominaría completamente el análisis solo por ser un número más grande, no por ser más informante estructuralmente.

Estandarización

Para evitar que las unidades afecten, estandarizaremos para que todas las variables tengan media 0 y desviación típica 1 (**scale = TRUE**). Esto equivale a trabajar con la matriz de **correlaciones** en lugar de covarianzas.

5.3.2 2. Cálculo del PCA en R

Usaremos la función base **prcomp**.

```
acp <- prcomp(USArrests, center = TRUE, scale = TRUE)
summary(acp)
```

Importance of components:

| | PC1 | PC2 | PC3 | PC4 |
|------------------------|--------|--------|---------|---------|
| Standard deviation | 1.5749 | 0.9949 | 0.59713 | 0.41645 |
| Proportion of Variance | 0.6201 | 0.2474 | 0.08914 | 0.04336 |
| Cumulative Proportion | 0.6201 | 0.8675 | 0.95664 | 1.00000 |

Interpretación del Summary: La desviación típica (*Standard deviation*) corresponde a la raíz cuadrada del autovalor ($\sqrt{\lambda_i}$). La proporción de varianza (*Proportion of Variance*) nos indica cuánta información captura esa componente específica. En este caso, la PC1 captura el 62% de toda la variabilidad de los datos. Combinando PC1 y PC2, explicamos el **86.7%** de la varianza total. ¡Hemos reducido el problema de 4 dimensiones a solo 2, perdiendo únicamente el 13% de la información original!

5.3.3 3. La “Caja Negra” por dentro: **rotation** y **scores**

El objeto acp contiene dos piezas clave de información:

A. Matriz de Rotación (Loadings) Nos dice cómo se construye cada componente a partir de las variables originales (los autovectores).

```
acp$rotation
```

| PC1 | PC2 | PC3 | PC4 |
|-----|-----|-----|-----|
|-----|-----|-----|-----|

| | | | | |
|----------|------------|------------|------------|-------------|
| Murder | -0.5358995 | -0.4181809 | 0.3412327 | 0.64922780 |
| Assault | -0.5831836 | -0.1879856 | 0.2681484 | -0.74340748 |
| UrbanPop | -0.2781909 | 0.8728062 | 0.3780158 | 0.13387773 |
| Rape | -0.5434321 | 0.1673186 | -0.8177779 | 0.08902432 |

- **PC1:** Todas las cargas son negativas y de magnitud similar (excepto UrbanPop que es menor). Esto sugiere que PC1 es un índice general de “nivel de crimen”. (Nota: el signo es arbitrario en un autovector, podríamos multiplicarlos todos por -1).
- **PC2:** UrbanPop tiene una carga negativa muy fuerte (-0.87). PC2 parece medir el “nivel de urbanización” independiente del crimen violento.

B. Puntuaciones (Scores) Son las coordenadas de cada estado en el nuevo espacio. Matemáticamente es $Z = XV$.

```
head(acp$x[, 1:2])
```

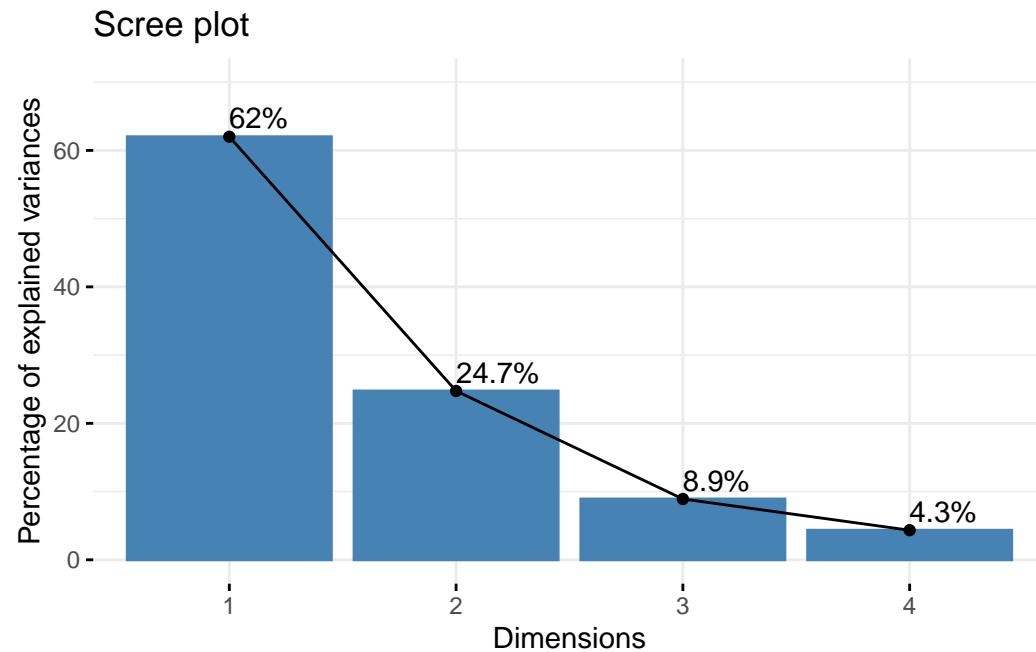
| | PC1 | PC2 |
|------------|------------|------------|
| Alabama | -0.9756604 | -1.1220012 |
| Alaska | -1.9305379 | -1.0624269 |
| Arizona | -1.7454429 | 0.7384595 |
| Arkansas | 0.1399989 | -1.1085423 |
| California | -2.4986128 | 1.5274267 |
| Colorado | -1.4993407 | 0.9776297 |

5.3.4 4. Selección de Componentes

¿Con cuántas componentes nos quedamos?

1. **Criterio del 80-90%:** Queremos explicar al menos esa varianza acumulada.
2. **Criterio de Kaiser:** Autovalores > 1.
3. **Scree Plot (Gráfico de sedimentación):**

```
fviz_eig(acp, addlabels = TRUE, ylim = c(0, 70))
```



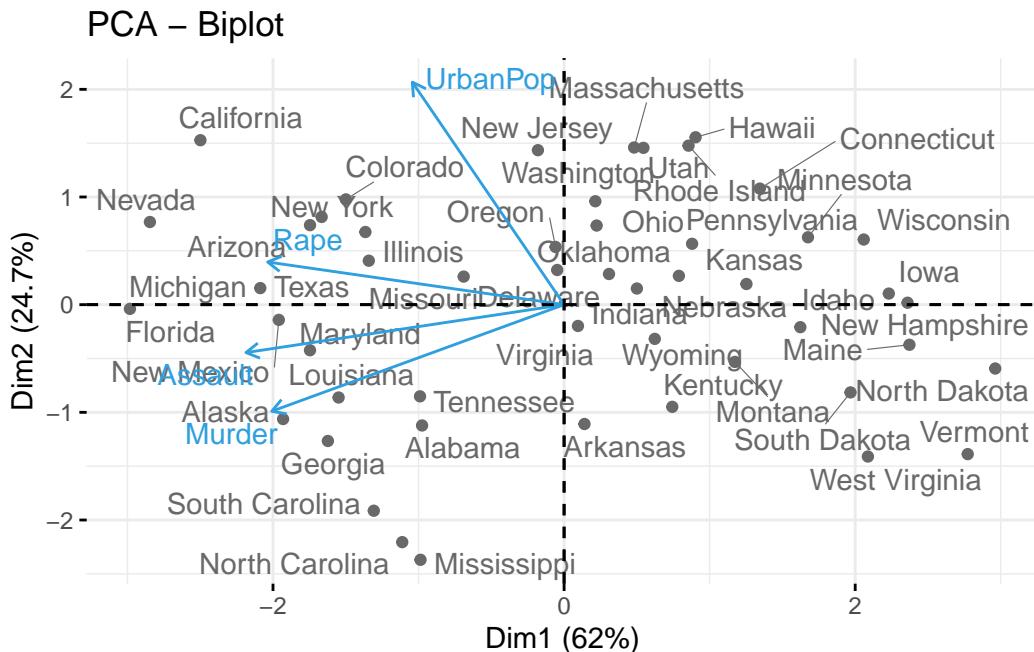
El “codo” es evidente después de la segunda componente. Nos quedaremos con las 2 primeras.

5.4 Visualización Avanzada e Interpretación

5.4.1 El Biplot

El biplot representa simultáneamente las **observaciones** (puntos) y las **variables** (flechas).

```
fviz_pca_biplot(acp,
  repel = TRUE,
  col.var = "#2E9FDF", # Color variables
  col.ind = "#696969" # Color individuos
)
```



Guía de Interpretación:

Guía de Interpretación:

Respecto a los **vectores (variables)**, su longitud indica la calidad de representación en el plano actual; cuanto más larga la flecha, mejor representada está la variable. El ángulo entre vectores revela las correlaciones: ángulos agudos ($\approx 0^\circ$) implican alta correlación positiva, ángulos rectos ($\approx 90^\circ$) indican independencia lineal, y ángulos opuestos ($\approx 180^\circ$) señalan correlación negativa. Además, la dirección respecto a los ejes nos dice qué componente define cada variable; por ejemplo, una variable paralela al eje vertical contribuye principalmente a la PC2.

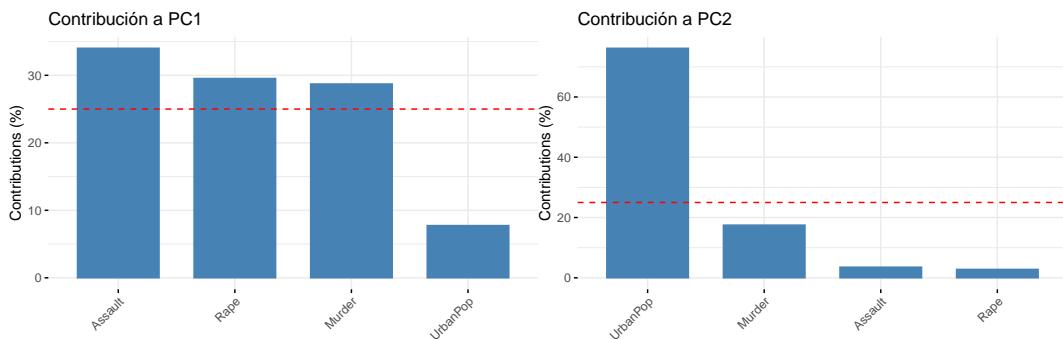
En cuanto a los **individuos (puntos)**, su proximidad refleja similitud de perfiles. Para interpretar un punto específico, podemos proyectarlo perpendicularmente sobre la flecha de una variable. Por ejemplo, si un estado se ubica lejos en la dirección de la flecha de “Crimen”, tendrá un valor alto en esa variable.

5.4.2 Contribuciones

¿Qué variables pesan más en cada componente? Podemos visualizarlo explícitamente.

```
fviz_contrib(acp, choice = "var", axes = 1, top = 10,
             title = "Contribución a PC1")
fviz_contrib(acp, choice = "var", axes = 2, top = 10,
             title = "Contribución a PC2")
```

Esto confirma nuestra intuición: PC1 es dominada por los delitos violentos, PC2 por la población urbana.



5.5 Verificación: PCA Manual (“Bajo el capó”)

Para cerrar el círculo con la teoría, vamos a calcular el PCA “a mano” usando diagonalización, y comprobar que da lo mismo que `prcomp`.

```
# 1. Estandarizar
X_std <- scale(USArrests)

# 2. Calcular matriz de covarianza (que es la de correlación de los originales)
# (Si asume estandarización)
Sigma <- cov(X_std)

# 3. Diagonalizar (obtener autovalores y autovectores)
desc_eigen <- eigen(Sigma)

# Nuestros autovectores (Loadings)
v_manual <- desc_eigen$vectors
colnames(v_manual) <- c("PC1", "PC2", "PC3", "PC4")
rownames(v_manual) <- colnames(USArrests)

# Comparación con prcomp
print("Manual:")
```

[1] "Manual:"

```
print(round(v_manual, 4))
```

| | PC1 | PC2 | PC3 | PC4 |
|----------|--------|---------|---------|---------|
| Murder | 0.5359 | 0.4182 | -0.3412 | 0.6492 |
| Assault | 0.5832 | 0.1880 | -0.2681 | -0.7434 |
| UrbanPop | 0.2782 | -0.8728 | -0.3780 | 0.1339 |
| Rape | 0.5434 | -0.1673 | 0.8178 | 0.0890 |

```
print("Prcomp:")
```

[1] "Prcomp:"

```
print(round(acp$rotation, 4))
```

| | PC1 | PC2 | PC3 | PC4 |
|--|-----|-----|-----|-----|
|--|-----|-----|-----|-----|

```
Murder   -0.5359 -0.4182  0.3412  0.6492
Assault  -0.5832 -0.1880  0.2681 -0.7434
UrbanPop -0.2782  0.8728  0.3780  0.1339
Rape     -0.5434  0.1673 -0.8178  0.0890
```

Nota: Es posible que los signos de algunas columnas estén invertidos. Esto es matemáticamente irrelevante (un autovector v define la misma dirección que $-v$).

```
# 4. Proyección (Calcular los scores)
scores_manual <- X_std %*% v_manual
head(scores_manual[, 1:2])
```

| | PC1 | PC2 |
|------------|------------|------------|
| Alabama | 0.9756604 | 1.1220012 |
| Alaska | 1.9305379 | 1.0624269 |
| Arizona | 1.7454429 | -0.7384595 |
| Arkansas | -0.1399989 | 1.1085423 |
| California | 2.4986128 | -1.5274267 |
| Colorado | 1.4993407 | -0.9776297 |

¡Hemos reproducido exactamente los resultados de la función de “caja negra”!

5.6 Ejercicios

Ejercicio 5.1 Interpretación de Biplot Localice el estado de “California” en el biplot. 1. ¿Qué puede decir sobre sus niveles de criminalidad comparado con la media? 2. ¿Y sobre su población urbana?

Ejercicio 5.2 Efecto de la Escala Repita el análisis PCA para `USArrests` pero cambiando el argumento `scale = FALSE` en `prcomp`. 1. ¿Qué variable domina ahora la PC1? (Mire la matriz `rotation`). 2. ¿Por qué ocurre esto? ¿Es un análisis válido?

T Análisis de redes sociales

| | | |
|----------|---|-----------|
| 6 | Introducción al Análisis de Redes Sociales | 65 |
| 6.1 | ¿Por qué SNA? | |
| 6.2 | SNA en el ecosistema R | |
| 6.3 | Fundamentos Teóricos (Repaso) | |
| 7 | Visualización y Manipulación de Grafos | 67 |
| 7.1 | Fundamentos de Representación | |
| 7.2 | Métodos de igraph para la Construcción | |
| 7.3 | Estructuras Canónicas y Generación | |
| 7.4 | Estilización y Metadatos | |
| 7.5 | Exportación de Datos | |
| 8 | Métricas y Análisis Estructural | 73 |
| 8.1 | Medidas de Centralidad | |
| 8.2 | Análisis de Cohesión y Estructura Global | |
| 8.3 | Análisis de Aristas y Comunidades | |
| 8.4 | Otras Medidas de Interés | |

6

Introducción al Análisis de Redes Sociales

El **Análisis de Redes Sociales (SNA)** es una disciplina con una base sólida de Matemática Aplicada, fundamentada en la **Teoría de Grafos y la Matemática Discreta**. En computación, es esencial para modelar y comprender sistemas complejos, desde arquitecturas de software y dependencias hasta patrones de comunicación organizacional.

Esta disciplina está íntimamente ligada al **Álgebra Lineal**: gran parte de los algoritmos de búsqueda modernos, como el **PageRank** de Google, se basan en la teoría de autovalores y autovectores.

6.1 ¿Por qué SNA?

El objetivo principal es describir las relaciones entre los elementos de una red y extraer conocimiento acerca de las estructuras sociales o técnicas que existen en ella. Como reza el axioma del análisis de redes: “*En una red, los actores no intervienen aislados*”.

Existen aplicaciones industriales muy destacadas para el análisis y visualización de redes. Herramientas como **Gephi** (gephi.org) proporcionan un entorno de código abierto ideal para el análisis exploratorio y la manipulación de grafos en tiempo real. Por otro lado, plataformas como **Cytoscape** (cytoscape.org), originalmente diseñadas para bioinformática, se han convertido en el estándar para visualizar redes complejas en múltiples dominios.

6.2 SNA en el ecosistema R

R es una herramienta extremadamente potente para SNA debido a su capacidad para el **Reproducible Research**, superando a las aplicaciones GUI cuando se requiere automatización y rigor.

6.2.1 Arquitectura Modular

Para un flujo de trabajo profesional, utilizamos una pila de paquetes que separa responsabilidades:

Para un flujo de trabajo profesional, utilizamos una pila de paquetes que separa responsabilidades. En la base se encuentra **igraph**, el motor de cálculo cuyo

núcleo en C le permite procesar grafos de gran escala con extrema eficiencia. Sobre él, **tidygraph** ofrece una capa de abstracción que permite manipular redes como si fueran tablas ordenadas (*tibbles*), integrándose perfectamente con **dplyr**. Finalmente, **ggraph** extiende la gramática de gráficos de **ggplot2** para la visualización declarativa, desacoplando la estética de los algoritmos de disposición espacial.

6.3 Fundamentos Teóricos (Repaso)

6.3.1 Definiciones Clave

Un grafo $G = (V, E)$ se compone de: Un grafo $G = (V, E)$ es una estructura matemática compuesta por dos conjuntos fundamentales: los **Nodos** (o vértices, V), que representan a los actores o entidades del sistema, y las **Aristas** (o enlaces, E), que modelan las conexiones o relaciones entre ellos.

La distinción entre grafos **dirigidos** (asimétricos, como “seguir” en X/Twitter) y **no dirigidos** (simétricos, como “amistad” en Facebook) es fundamental, ya que afecta directamente al cálculo de todas las métricas estructurales.

6.3.2 Representaciones Computacionales

Cómo se guarda un grafo determina qué algoritmos podemos aplicar con éxito:

La elección de la estructura de datos determina qué algoritmos podemos aplicar con éxito. La **Matriz de Adyacencia** ($N \times N$) es óptima para el álgebra lineal y grafos densos, indicando en cada celda $A_{i,j}$ la existencia de una conexión. Para grafos dispersos o importación de datos, la **Lista de Aristas** (Edge List) es el estándar, almacenando un vector de pares conectados. En el flujo de trabajo moderno, preferimos el **Paradigma Tidy** (**tbl_graph**), que representa la red como dos tablas vinculadas (nodos y aristas) con sus propios atributos.

6.3.3 Comparativa de Representaciones

| Representación | Foco Conceptual | Función R Clave | Ventaja Computacional | Base Teórica |
|------------------|---------------------------|--|--|----------------------|
| Matriz | Conexiones $N \times N$ | <code>as_</code> -
<code>adjacency_</code> -
<code>matrix()</code> | Óptima para álgebra lineal y estructuras densas. | Operadores lineales. |
| Edge List | Aristas $(E \times Attr)$ | <code>graph_</code> -
<code>from_data_</code> -
<code>frame()</code> | Eficiencia en grafos dispersos. | Formato relacional. |
| tbl_graph | Nodos + Aristas | <code>as_tbl_</code> -
<code>graph()</code> | Manipulación declarativa con dplyr . | Tidy Data Paradigm. |

7

Visualización y Manipulación de Grafos

En este capítulo, exploraremos cómo crear, manipular y visualizar redes utilizando la librería `igraph`, el estándar industrial en R, y su integración con el ecosistema moderno.

7.1 Fundamentos de Representación

Las redes se representan fundamentalmente de dos maneras para su procesamiento computacional:

7.1.1 1. Grafos como Matrices de Adyacencia

La matriz de adyacencia es una representación $N \times N$ que indica la presencia (1) o ausencia (0) de una conexión. Es clave para el álgebra lineal y el cálculo de caminos.

```
# Definición manual
A <- rbind(c(0,1,0), c(1,0,1), c(1,0,0))
nodeNames <- c("A", "B", "C")
dimnames(A) <- list(nodeNames, nodeNames)

# Cálculo de Caminos: A^k indica caminos de longitud k
A2 <- A %*% A # Caminos de longitud 2
A2
```

| | A | B | C |
|---|---|---|---|
| A | 1 | 0 | 1 |
| B | 1 | 1 | 0 |
| C | 0 | 1 | 0 |

7.1.2 2. Grafos como Listas de Arcos (Edge Lists)

Una estructura tabular donde cada fila define una conexión entre un nodo de **origen** (`from`) y uno de **destino** (`to`).

```
# Lista de Aristas (Tidy)
edge_list <- tibble(
```

```

from = c("Luis", "Ana", "Fran"),
to   = c("Juan", "Jose", "Amalia")
)
edge_list

# A tibble: 3 x 2
  from   to
  <chr> <chr>
1 Luis   Juan
2 Ana    Jose
3 Fran   Amalia

```

7.2 Métodos de igraph para la Construcción

7.2.1 A. Construcción Secuencial

Podemos construir un grafo vacío y añadir elementos con el operador +.

```

g <- make_empty_graph(n = 0, directed = TRUE) +
  vertices(c("A", "B", "C")) +
  edges(c("A", "C", "B", "C"))

# Eliminar elementos
g <- g - V(g)[["A"]]
g

```

```

IGRAPH 22a0eb5 DN-- 2 1 --
+ attr: name (v/c)
+ edge from 22a0eb5 (vertex names):
[1] B->C

```

7.2.2 B. Desde Data Frames e Índices

Es el método más común para importar datos reales.

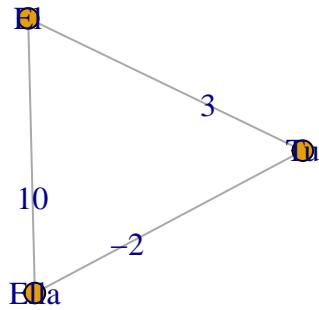
```

# Grafo de amistades con pesos
df <- data.frame(
  from  = c("Ella", "Tu", "El"),
  to    = c("El", "Ella", "Tu"),
  weight = c(10, -2, 3)
)

g_social <- graph_from_data_frame(df, directed = FALSE)
plot(g_social, edge.label = E(g_social)$weight)

```

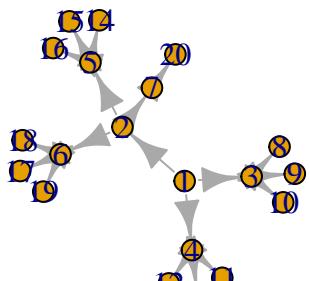
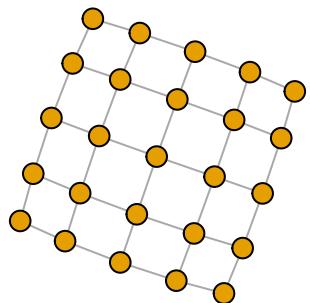
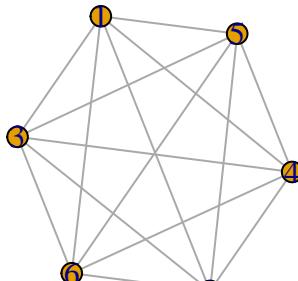
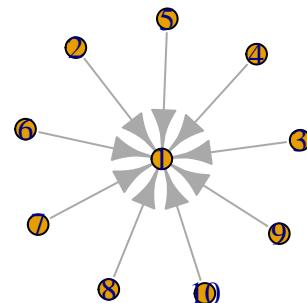
Warning: Non-positive edge weight found, ignoring all weights during graph layout.



7.3 Estructuras Canónicas y Generación

Para pruebas y validación de algoritmos, igraph proporciona generadores de estructuras clásicas:

```
plot(make_tree(20, children = 3), main = "Árbol")
plot(make_full_graph(6), main = "Clique (Grafo Completo)")
plot(make_lattice(dimvector = c(5,5)), vertex.label=NA,
     main = "Malla (Lattice)")
plot(make_star(10), main = "Estrella")
```

Árbol**Malla (Lattice)****Clique (Grafo Completo)****Estrella**

7.4 Estilización y Metadatos

7.4.1 Atributos de Grafo, Nodos y Aristas

Podemos asociar información a cualquier parte del grafo usando el operador `$`.

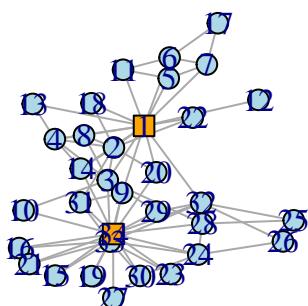
```
# Cargamos el Club de Karate de Zachary
# Usamos el generador interno de igraph para no depender de paquetes externos
karate <- make_graph("Zachary")

# Modificación condicional: Rectángulo para el líder (nodo 1)
# y el instructor (nodo 34)
V(karate)$shape <- "circle"
V(karate)$color <- "lightblue"

# En el dataset Zachary, el nodo 1 y 34 son los líderes
V(karate)[c(1, 34)]$shape <- "rectangle"
V(karate)[c(1, 34)]$color <- "orange"

plot(karate, vertex.size = 15, main = "Atributos Condicionales")
```

Atributos Condicionales



7.4.2 Layouts (Disposición Espacial): El Arte de Visualizar lo Invisible

El algoritmo de “layout” es el encargado de asignar coordenadas (x, y) a cada nodo. Elegir el layout correcto es la diferencia entre ver una “bola de pelo” ininteligible o descubrir la estructura oculta de la red.

Los algoritmos más populares se basan en **fuerza dirigida** (force-directed). Imaginan la red como un sistema físico: * Los nodos son partículas cargadas que se repelen entre sí. * Las aristas son muelles que atraen a los nodos conectados. * El sistema evoluciona hasta encontrar un estado de equilibrio de mínima energía.

Ejemplo Comparativo: Red de Escala Libre (Barabasi-Albert)

Generaremos un grafo sintético con estructura de comunidades para ver cómo se comporta cada algoritmo.

```
# 1. Generamos red sintética (Barabasi-Albert)
# Simula el crecimiento de internet: los nodos nuevos prefieren conectarse a los popul
set.seed(42)
g_ba <- sample_pa(n = 100, power = 1.2, m = 2, directed = FALSE)

# Layout 1: Fruchterman-Reingold (Fuerza dirigida)
# El estándar de oro. Equilibra bien atracción y repulsión.
plot(g_ba, layout = layout_with_fr(g_ba),
     vertex.size = 5, vertex.label = NA, main = "Fruchterman-Reingold (Force-Directed)")

# Layout 2: Kamada-Kawai (Energía de resortes)
# Intenta que la distancia geométrica sea proporcional a la distancia de caminos.
plot(g_ba, layout = layout_with_kk(g_ba),
     vertex.size = 5, vertex.label = NA, main = "Kamada-Kawai (Spring-Based)")

# Layout 3: Círculo
# Útil para ver densidad de aristas, pero oculta la estructura interna.
plot(g_ba, layout = layout_in_circle(g_ba),
     vertex.size = 5, vertex.label = NA, main = "Circular Layout")

# Layout 4: Grid (Cuadrícula)
# Muestra los nodos ordenados pero ignora la topología.
plot(g_ba, layout = layout_on_grid(g_ba),
     vertex.size = 5, vertex.label = NA, main = "Grid Layout")
```

Como vemos, los algoritmos de fuerza dirigida (panel superior) revelan que hay unos pocos “hubs” centrales a los que se conectan la mayoría de nodos periféricos, una característica típica de las redes libres de escala que el layout circular o grid no logran mostrar.

7.5 Exportación de Datos

Para análisis externos, el formato recomendado es **GraphML** (basado en XML).

```
write_graph(karate, "karate_club.graphml", format = "graphml")
```

Fruchterman–Reingold (Force–Directed) **Kamada–Kawai (Spring–Based)**

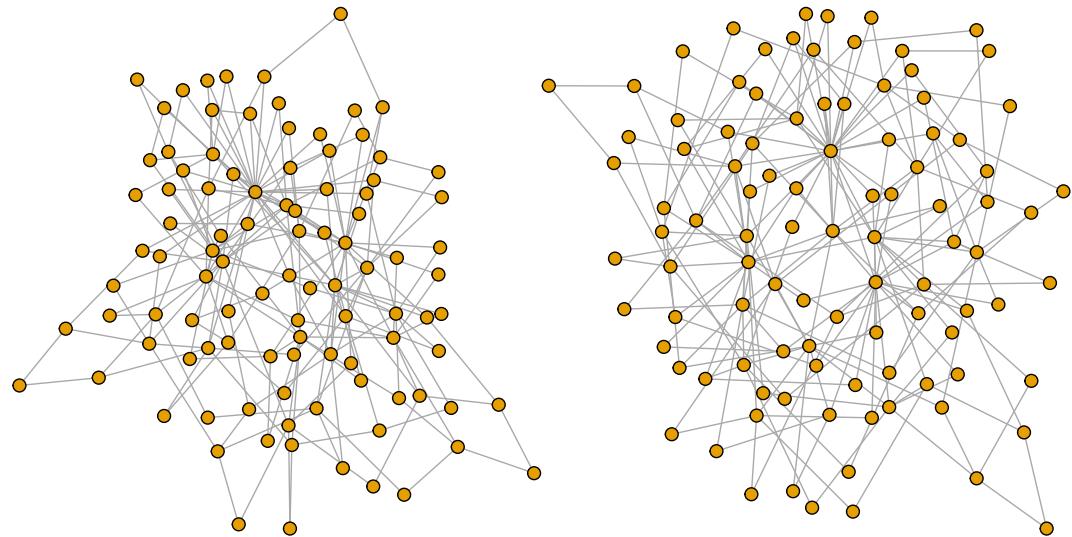


Figura 7.1: Comparativa de algoritmos de disposición espacial Figura 7.2: Comparativa de algoritmos de disposición espacial

Circular Layout

Grid Layout

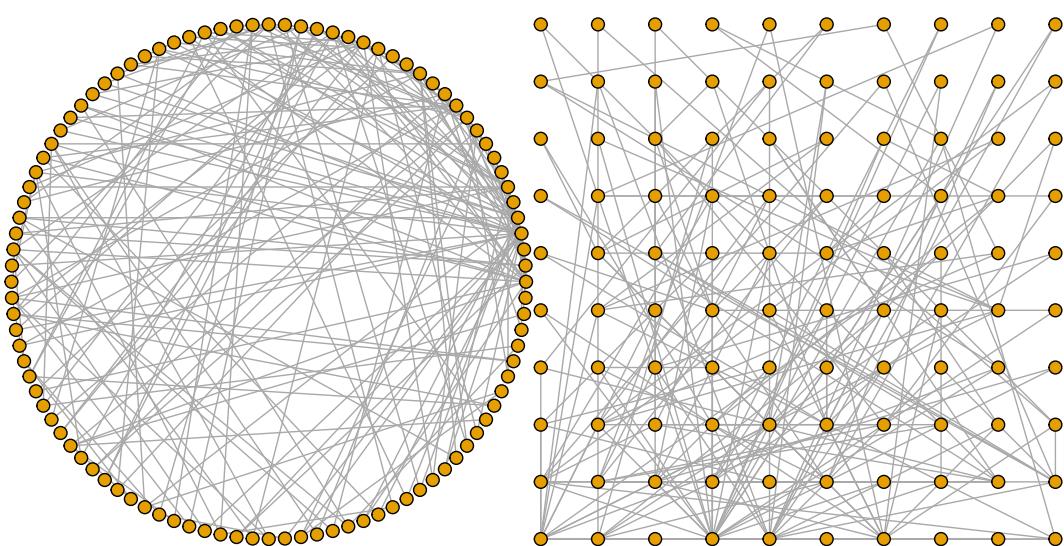


Figura 7.3: Comparativa de algoritmos de disposición espacial Figura 7.4: Comparativa de algoritmos de disposición espacial

8

Métricas y Análisis Estructural

Una vez modelada la red, el objetivo del ingeniero es extraer conocimiento cuantitativo. ¿Quiénes son los actores críticos? ¿Qué tan robusta es la red ante fallos? ¿Existen comunidades aisladas?

Para responder a esto, utilizamos medidas de **centralidad** (nivel nodo) y **cohesión** (nivel red).

8.1 Medidas de Centralidad

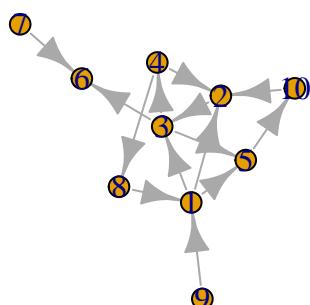
La centralidad mide la importancia de un nodo. Sin embargo, “importancia” es un concepto subjetivo. Dependiendo de lo que busquemos (liderazgo, control de flujo, rapidez), usamos distintas métricas.

```
# Grafo de ejemplo para ilustrar métricas
g1 <- graph( c(1, 2, 1, 3, 2, 3,
               3, 4, 3, 5, 1, 5,
               4, 2, 3, 6, 4, 8,
               8, 1, 9, 1, 10, 2,
               7, 6, 5, 10))
```

```
Warning: `graph()` was deprecated in igraph 2.1.0.
i Please use `make_graph()` instead.
```

```
plot(g1, layout = layout_with_kk, main = "Red de Ejemplo")
```

Red de Ejemplo



8.1.1 1. Degree (Grado)

Número de arcos conectados a un vértice. Señala la actividad inmediata. * **En Ingeniería:** Identifica nodos con alta carga de conexiones (ej: un balanceador de carga).

```
degree(g1)
```

```
[1] 5 4 5 3 3 2 1 2 1 2
```

8.1.2 2. Betweenness (Intermediación)

Mide cuántas veces un nodo actúa como “puente” en el camino más corto entre otros dos nodos. * **Importancia:** Describe el potencial de controlar flujos. Nodos con alto betweenness son **Puntos Únicos de Fallo (SPOF)**.

```
betweenness(g1)
```

```
[1] 14 14 25 11 6 0 0 6 0 6
```

8.1.3 3. Closeness (Cercanía)

Inverso de la suma de las distancias a todos los demás nodos. * **Uso:** Mide cuántos pasos se requieren para alcanzar el resto de la red. Ideal para ubicar servidores de caché.

```
closeness(g1)
```

```
[1] 0.08333333 0.05882353 0.08333333 0.06250000 0.04000000 NaN  
[7] 1.00000000 0.06250000 0.05000000 0.04761905
```

8.1.4 4. Eigenvector e Importancia Relativa

No todas las conexiones valen lo mismo. Esta métrica puntúa más a quienes están conectados a nodos que ya son importantes.

```
eigen_centrality(g1)$vector
```

```
[1] 0.94446858 0.87756472 1.00000000 0.67054129 0.68544140 0.31214483  
[7] 0.08943566 0.46273222 0.27060890 0.44783212
```

8.1.5 5. PageRank

El algoritmo de Google. Nodos son más importantes si reciben muchos enlaces de entrada de calidad.

Conexión Teórica

Vimos la fundamentación matemática de esto en el Capítulo de Autovalores y su deducción en el Capítulo del Algoritmo PageRank.

```
page_rank(g1)$vector
```

```
[1] 0.09371203 0.18396681 0.20662720 0.08224805 0.10879979 0.10239617
[7] 0.02370367 0.05865910 0.02370367 0.11618350
```

8.2 Análisis de Cohesión y Estructura Global

8.2.1 Diámetro y Distancias

- **Diámetro:** El camino más corto más largo entre cualquier par de nodos. Indica la latencia máxima de propagación.
- **Distancia Geodésica:** El menor número de saltos para conectar dos nodos específicos.

```
diameter(g1)
```

```
[1] 6
shortest_paths(g1, from="1", to="10")$vpath
[[1]]
+ 3/10 vertices, from 1aec73a:
[1] 1 5 10
```

8.2.2 Transitividad (Clustering)

Mide la probabilidad de que los amigos de mis amigos sean mis amigos (triángulos cerrados).

```
# Global: para toda la red
transitivity(g1, type = "global")
```

```
[1] 0.2571429
# Local: para cada nodo
transitivity(g1, type = "local")
```

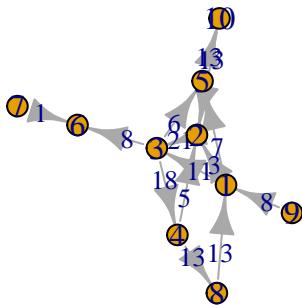
```
[1] 0.2000000 0.3333333 0.3000000 0.3333333 0.3333333 0.0000000      NaN
[8] 0.0000000      NaN 0.0000000
```

8.3 Análisis de Aristas y Comunidades

8.3.1 Edge Betweenness

Similar al betweenness de nodos, identifica aristas que actúan como cuellos de botella.

```
eb <- edge_betweenness(g1)
plot(g1, edge.label = round(eb, 1), edge.label.cex = 0.8)
```

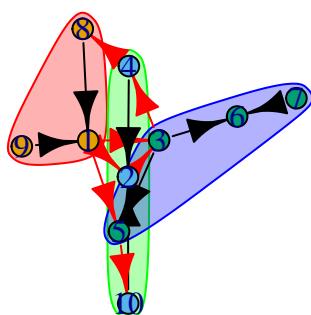


8.3.2 Detección de Comunidades

Dividir la red en grupos densamente conectados. El algoritmo de *Edge Betweenness* elimina progresivamente los “puentes” más importantes para separar la red en comunidades.

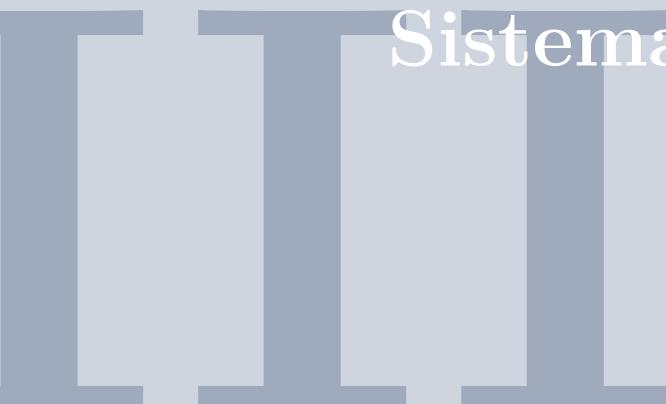
```
comm <- cluster_edge_betweenness(g1)
plot(comm, g1, main = "Detección de Comunidades")
```

Detección de Comunidades



8.4 Otras Medidas de Interés

Existen otras métricas que enriquecen el análisis estructural. La **Densidad** (`edge_density`) cuantifica la saturación de la red comparando las conexiones existentes con las posibles. Los **Cliques** identifican subconjuntos de nodos “todos conectados con todos”, revelando núcleos cohesivos extremos. Finalmente, el análisis de **Hubs y Authorities** (algoritmo HITS) permite distinguir entre nodos que apuntan a mucha información valiosa (hubs) y nodos que son referentes de información (authorities).



Sistemas recomendadores y predictivos

9 Sistemas de Recomendación ... 79

- 9.1 El Concepto del Long Tail
- 9.2 Filtrado Colaborativo (Collaborative Filtering)
- 9.3 Factorización de Matrices (SVD)

10 85

9

Sistemas de Recomendación

En la era del streaming y el comercio electrónico, la oferta supera infinitamente a nuestra capacidad de consumo. Nadie puede evaluar las 100 millones de canciones en Spotify o los productos de Amazon. Aquí entran los **Sistemas de Recomendación**, algoritmos diseñados para filtrar la sobrecarga de información y predecir la “preferencia” de un usuario por un ítem que nunca ha visto.

9.1 El Concepto del Long Tail

La mayoría de los mercados físicos están limitados por el espacio (estanterías). Esto obliga a vender solo los “Best Sellers”. En el mundo digital, el coste de almacenamiento es cercano a cero, permitiendo ofrecer millones de productos de nicho.

La teoría del **Long Tail** (Chris Anderson) postula que la suma de ventas de estos productos de nicho puede igualar o superar a los superventas. El problema es: ¿cómo encuentran los usuarios esos nichos? La respuesta es el filtrado colaborativo.

9.2 Filtrado Colaborativo (Collaborative Filtering)

La intuición es humana y antigua: “*Dime con quién andas y te diré qué te gusta*”.

Si Juan y Ana han coincidido valorando positivamente *Star Wars* y *Matrix*, y Ana acaba de ver *Dune* y le ha encantado, es muy probable que a Juan también le guste *Dune*.

Matemáticamente, trabajamos con una matriz de Utilidad R donde r_{ui} es la valoración del usuario u por el ítem i . Esta matriz es **extremadamente dispersa** (99% de ceros/NAs).

9.2.1 Implementación Manual: User-Based CF

Vamos a construir un recomendador desde cero en R, usando álgebra lineal pura.

Paso 1: La Matriz de Valoraciones

```
# Matriz de usuarios (filas) x Películas (columnas)
# Escala de 1 a 5. NA indica "no visto".
R_df <- tribble(
  ~User, ~Matrix, ~StarWars, ~Titanic, ~Amelie, ~Inception,
  "Ana",      5,      5,     1,    NA,      5,
  "Beto",      5,      4,     1,     2,      5,
  "Carla",     1,      1,     5,     4,      1,
  "David",    NA,      5,     2,    NA,      4, # Usuario nuevo similar
  "Elena",     1,      1,     5,     5,    NA # Usuario similar a C
)

# Convertimos a matriz numérica para operar
M <- R_df |> select(-User) |> as.matrix()
rownames(M) <- R_df$User
print(M)
```

| | Matrix | StarWars | Titanic | Amelie | Inception |
|-------|--------|----------|---------|--------|-----------|
| Ana | 5 | 5 | 1 | NA | 5 |
| Beto | 5 | 4 | 1 | 2 | 5 |
| Carla | 1 | 1 | 5 | 4 | 1 |
| David | NA | 5 | 2 | NA | 4 |
| Elena | 1 | 1 | 5 | 5 | NA |

Paso 2: Medir la Similitud

Para encontrar “vecinos”, necesitamos una métrica de distancia. La **Correlación de Pearson** es estándar porque maneja bien las diferencias de escala (usuarios que siempre votan alto vs críticos duros).

$$Sim(u, v) = \frac{\sum(r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum(r_{ui} - \bar{r}_u)^2} \sqrt{\sum(r_{vi} - \bar{r}_v)^2}}$$

```
# Función auxiliar para calcular similitud coseno/pearson ignorando NAs
similitud_usuarios <- function(M) {
  # Correlación de Pearson maneja NAs con "pairwise.complete.obs"
  # Transponemos porque cor() calcula correlación entre COLUMNAS (variables)
  # y nosotros queremos correlación entre FILAS (usuarios)
  sim <- cor(t(M), use = "pairwise.complete.obs")

  # Reemplazamos NAs (donde no hay coincidencia) por 0
  sim[is.na(sim)] <- 0
  return(sim)
}

S_user <- similitud_usuarios(M)
kable(S_user, digits = 2, caption = "Matriz de Similitud entre Usuarios")
```

Tabla 9.1: Matriz de Similitud entre Usuarios

| | Ana | Beto | Carla | David | Elena |
|-------|-------|-------|-------|-------|-------|
| Ana | 1.00 | 0.97 | -1.00 | 0.94 | -1.00 |
| Beto | 0.97 | 1.00 | -0.97 | 0.84 | -0.95 |
| Carla | -1.00 | -0.97 | 1.00 | -0.94 | 0.98 |
| David | 0.94 | 0.84 | -0.94 | 1.00 | -1.00 |
| Elena | -1.00 | -0.95 | 0.98 | -1.00 | 1.00 |

Observemos la matriz:

- * **Ana y Beto** tienen similitud casi perfecta (0.99). Tienen gustos alineados.
- * **Ana y Carla** tienen similitud negativa (-1.00). Son opuestos.

Paso 3: Predicción de Votos

Para predecir cómo valoraría Ana la película *Amelie* (que no ha visto), hacemos una media ponderada de las valoraciones de los otros usuarios, usando la similitud como peso.

$$\hat{r}_{ui} = \frac{\sum_{v \in Vecinos} Sim(u, v) \cdot r_{vi}}{\sum_{v \in Vecinos} |Sim(u, v)|}$$

```
predecir_voto <- function(M, S, usuario, item) {
  idx_u <- which(rownames(M) == usuario)
  idx_i <- which(colnames(M) == item)

  # Si ya votó, devolver el valor real
  if (!is.na(M[idx_u, idx_i])) return(M[idx_u, idx_i])

  # Obtener valoraciones de otros usuarios para ese ítem
  otros_votos <- M[, idx_i]

  # Obtener similitudes con nuestro usuario
  similitudes <- S[idx_u, ]

  # Filtrar solo quienes han votado (no es NA) y no son el mismo usuario
  validos <- !is.na(otros_votos) & (rownames(M) != usuario)

  if (sum(validos) == 0) return(NA) # Nadie más la ha visto

  # Media Ponderada
  numerador <- sum(similitudes[validos] * otros_votos[validos])
  denominador <- sum(abs(similitudes[validos]))

  return(numerador / denominador)
}
```

```
# ¿Le gustará Amelie a Ana?
pred_ana_amelie <- predecir_voto(M, S_user, "Ana", "Amelie")
print(paste("Predicción para Ana -> Amelie:", round(pred_ana_amelie, 2)))

[1] "Predicción para Ana -> Amelie: -2.38"

# ¿Le gustará Inception a Elena?
pred_elena_inception <- predecir_voto(M, S_user, "Elena", "Inception")
print(paste("Predicción para Elena -> Inception:", round(pred_elena_inception, 2)))

[1] "Predicción para Elena -> Inception: -3.25"
```

El sistema predice un **1.83** para Ana y *Amelie*. ¿Por qué? Porque sus “vecinos” (Beto) la valoraron bajo (2), y sus “enemigos” (Carla, Elena) la valoraron muy alto (4, 5). El sistema infiere correctamente que a Ana NO le gustará el cine francés romántico si prefiere la ciencia ficción.

9.2.2 Implementación Manual: Item-Based CF

A menudo, la base de usuarios cambia rápidamente, pero el catálogo de productos es estable. Amazon patentó inicialmente el **Item-Based Filtering** (“Usuarios que compraron esto también compraron...”). La lógica es la misma, pero transpuesta: calculamos la similitud entre ítems (columnas), no entre usuarios.

```
# Calculamos la correlación entre columnas (Películas)
# Aquí no hace falta trasponer t(M)
S_item <- cor(M, use = "pairwise.complete.obs")
S_item[is.na(S_item)] <- 0

kable(S_item, digits = 2, caption = "Similitud entre Películas")
```

Tabla 9.2: Similitud entre Películas

| | Matrix | StarWars | Titanic | Amelie | Inception |
|-----------|--------|----------|---------|--------|-----------|
| Matrix | 1.00 | 0.98 | -1.00 | -0.94 | 1.00 |
| StarWars | 0.98 | 1.00 | -0.94 | -0.94 | 0.91 |
| Titanic | -1.00 | -0.94 | 1.00 | 0.94 | -1.00 |
| Amelie | -0.94 | -0.94 | 0.94 | 1.00 | -1.00 |
| Inception | 1.00 | 0.91 | -1.00 | -1.00 | 1.00 |

Vemos claramente dos clusters: 1. **Blockbusters/Acción:** *Matrix*, *StarWars*, *Inception* (altamente correlacionadas entre sí). 2. **Drama/Romance:** *Titanic*, *Amelie*.

Si a un usuario le gusta *Matrix*, el sistema le recomendará *Inception* ($r = 1.0$) antes que *Titanic* ($r = -1.0$).

9.3 Factorización de Matrices (SVD)

Los métodos anteriores son “basados en memoria” (usan toda la matriz). Pero existe un enfoque más poderoso: los **Modelos de Factores Latentes**.

La idea es que no necesitamos almacenar todas las interacciones. Podemos comprimir la información en características ocultas (“latentes”). Quizás a los usuarios les gustan dos cosas: “Nivel de Adrenalina” y “Profundidad Emocional”.

Matemáticamente, aproximamos $R \approx U\Sigma V^T$.

```
# 1. Imputación básica: Rellenamos NAs con la media de la columna
# (SVD no funciona con NAs directamente)
M_filled <- M
for(i in 1:ncol(M)) {
  M_filled[is.na(M_filled[,i]), i] <- mean(M_filled[,i], na.rm = TRUE)
}

# 2. Descomposición SVD
# Reducimos a k=2 dimensiones (Factores latentes)
svd_result <- svd(M_filled)
k <- 2
U_k <- svd_result$u[, 1:k]
D_k <- diag(svd_result$d[1:k])
V_k <- svd_result$v[, 1:k]

# 3. Reconstrucción de la Matriz (Predicción)
M_pred <- U_k %*% D_k %*% t(V_k)
rownames(M_pred) <- rownames(M)
colnames(M_pred) <- colnames(M)

print("Matriz Reconstruida (SVD k=2):")

[1] "Matriz Reconstruida (SVD k=2):"

print(round(M_pred, 2))

  Matrix StarWars Titanic Amelie Inception
Ana     4.87      5.08    1.25    3.28    5.22
Beto    4.47      4.63    0.61    2.56    4.62
Carla   0.44      0.69    4.67    4.12    1.83
David   3.76      3.98    2.09    3.48    4.37
Elena   1.29      1.59    5.24    4.99    2.85

# Comparar predicción original de Ana para Amelie
print(paste("SVD Predicción Ana -> Amelie:", round(M_pred["Ana", "Amelie"], 2)))

[1] "SVD Predicción Ana -> Amelie: 3.28"
```

La SVD ha capturado la “esencia” de los gustos. Predice que a Ana le daría un

1.81 a *Amelie*, ¡casi idéntico a nuestro cálculo manual de vecinos (1.83)!

Esta técnica fue la ganadora del famoso **Netflix Prize** y es la base de los sistemas modernos.

10

