Facultad de Matemáticas

Trabajo Fin de Grado:

Algunos Métodos Matemáticos Para El Análisis De Big Data

Tutor: Don Rafael Pino Mejías

Departamento de Estadística e Investigación Operativa

Autor:
Pedro Antonio Reyes Quintanilla

Sevilla, Junio 2023

Abstract

This work is dedicated to Big Data analysis, a relatively modern field of data science whose objective is to know and extract information from a large mass of data from various areas. Some methods and mathematical reasoning that make it possible to carry out this study successfully will be presented, all accompanied by practical cases in R and problems of various kinds to illustrate its importance.

Este trabajo está dedicado al análisis de Big Data, un campo relativamente moderno de la ciencia de datos cuyo objetivo es conocer y extraer información de una gran masa de datos procedentes de diversas áreas. Se expondrán algunos métodos y razonamientos matemáticos que hacen posible que se lleve a cabo con éxito dicho estudio, todo ello acompañado de casos prácticos en R y problemas de diversa índole para ilustrar su importancia.

Introducción. Objetivos

En la era de la información, el Big Data se ha convertido en una de las principales tendencias en el mundo de la tecnología y el análisis de datos. El Big Data se refiere a conjuntos de datos extremadamente grandes y complejos que requieren herramientas y técnicas especializadas para su análisis y comprensión. La cantidad de datos que se generan cada día está aumentando exponencialmente y, por lo tanto, la necesidad de herramientas y técnicas para el análisis de Big Data es cada vez más importante.

Este trabajo fin de grado se enfoca en la aplicación de algunos métodos matemáticos para el análisis de Big Data. Los métodos matemáticos son una de las herramientas más importantes y efectivas para el análisis de Big Data. El objetivo principal de este trabajo es estudiar algunos de estos métodos a lo largo de tres capítulos: ranking, aprendizaje online y sistemas de recomendación. Además, se añaden casos prácticos en R para cada uno de ellos.

En el primer capítulo, nos centraremos en el ranking. El ranking se utiliza para ordenar y clasificar los elementos de un conjunto de datos en función de un criterio específico. Estudiaremos el algoritmo PageRank y su aplicación en conjuntos de datos reales. Además, se proporcionarán ejemplos en R para mostrar cómo implementar dicho algoritmo y aplicarlo a diferentes conjuntos de datos.

En el segundo capítulo, nos centraremos en el aprendizaje online. El aprendizaje online es una técnica de aprendizaje automático que se utiliza para analizar y procesar datos en tiempo real. Introduciremos el llamado algoritmo de descenso del espejo online y lo aplicaremos al problema de selección de cartera.

En el tercer y último capítulo, nos centraremos en los sistemas de recomendación. Los sistemas de recomendación son herramientas que se utilizan para hacer predicciones sobre el comportamiento futuro de los usuarios en función de su historial de comportamiento. Estos sistemas se utilizan en muchas aplicaciones, como sitios de comercio electrónico o plataformas de series y películas. En este capítulo discutiremos diferentes tipos de sistemas de recomendación, como el filtrado colaborativo, y se proporcionarán ejemplos en R para mostrar cómo implementar estos sistemas y aplicarlos a conjuntos de datos reales.

En general, el objetivo de este trabajo fin de grado es proporcionar una comprensión más profunda de los métodos matemáticos utilizados para el análisis de Big Data, a través del conocimiento de la teoría y la aplicación práctica de estos métodos.

Índice de figuras

Índice general

Capítulo 1

Ranking

Estamos familiarizados con clasificaciones de diversa índole en nuestra vida cotidiana. Por ejemplo, después de cada jornada de "LaLiga" miramos la tabla de clasificación donde cada equipo aparece según los puntos obtenidos (un mayor puntaje corresponde a una posición más alta de la tabla) o también, cada vez que queremos comprar un producto y acudimos a Internet, donde se comparan diferentes marcas según la crítica de los consumidores. Es decir, la idea (totalmente intuitiva) de una clasificación es ordenar los elementos de tal manera que los mejores ocupen una posición más alta.
Sin embargo, no todas las clasificaciones son ordenadas con respecto a una característica de los elementos en cuestión (puntos obtenidos por cada equipo en el ejemplo anterior) sino que, también pueden ser ordenados según las relaciones que tengan entre sí. Nos podemos hacer la siguiente pregunta: *¿Cómo clasificar un conjunto de elementos según dichas interrelaciones?*

La respuesta se abordará en este capítulo donde la idea es que esas interrelaciones dan lugar a probabilidades de transición y, en consecuencia, a un autovector principal de una matriz estocástica que será la solución del ranking que queremos obtener. Veremos las matemáticas que hay detrás en dos pasos. Primero nos centraremos en la existencia del ranking usando la dualidad de la programación lineal y después, introduciremos un método iterativo que nos va a permitir aproximar las clasificaciones de una manera rápida y económica desde el punto de vista computacional, lo cual es crucial para las aplicaciones de Big Data.

1.1. Motivación. El problema de Google

Como motivación vamos a estudiar el conocido *problema de Google*, que pretende llevar a cabo una clasificación de las páginas web en función de su relevacia en la red de Internet.

Para realizar búsquedas en la web se utilizan básicamente dos sistemas: los Text Based Ranking Systems y PageRank. A continuación explicaremos brevemente el primero para luego abordar en profundidad el sistema PageRank.
Text Based Ranking Systems: son los sistemas de ranking basados en texto y fueron utilizados principalmente en los años 90. Este tipo de motor selecciona aquellas páginas en las que aparezca más veces la palabra buscada y las ordena de forma decreciente. Presenta el inconveniente de que tal vez la página web en la que aparezca más veces el término buscado no sea relevante para la búsqueda o no aporte información significativa.
Es por ello la necesidad de introducir un nuevo modelo que mejore el anterior, que será el famoso PageRank que detallamos a continuación.

1.2. PageRank. Construcción del modelo

PageRank es uno de los algoritmos de búsqueda más populares e influyentes de la actualidad. Fue desarrollado por los fundadores de Google, Larry Page y Sergey Brin [9] en la Universidad de Stanford mientras estudiaban su postgrado, y es el sello distintivo de Google desde 1998.
La idea es la siguiente: si una página A contiene un vínculo hacia otra página B se interpreta que la pagina A considera que el contenido de B es relevante para la temática abordada en A. Si existen muchas páginas con links hacia B se considera que es de común acuerdo que la pagina B es importante. Por otro lado, si la página B tiene solamente un backlink[1] pero este proviene de una página C con autoridad decimos que C transfiere su autoridad a B, es decir, indica que B es importante.
Utilizando estos conceptos de importancia y autoridad, el algoritmo PageRank asigna un rango a cada página basándose en las páginas que dirigen a ellas.

Consideremos una red ficticia de 5 páginas web como la de la Figura 1.1, donde cada flecha del nodo i al nodo j quiere decir que la página i contiene un enlace a la página j.

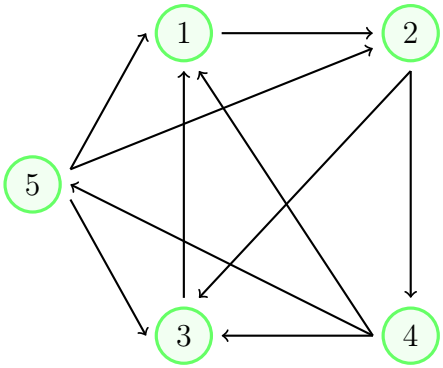

Figura 1.1: Red

[1]Backlink o vínculos externos de respaldo, son los enlaces que recibe una página web desde otras páginas web. El número de backlinks es la cantidad de páginas que la enlazan a través de un vínculo.

En general, a partir del grafo que representa una red, podemos definir una *matriz de adyacencia o conectividad* como sigue:

$$A = (a_{ij}), \quad \text{donde } a_{ij} = \begin{cases} 1, & \text{si hay un enlace } j \to i \\ 0, & \text{en caso contrario} \end{cases}$$

En el caso de la red de la Figura 1.1, tenemos:

$$A = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Observemos que la fila i contiene los "votos" recibidos por la página i, mientras que la columna j contiene los otorgados por la página j.
Los "votos" o enlaces recibidos por una página podría ser la medida de la relevancia de esta. Así, en nuestro ejemplo, las páginas 1 y 3 serían las más importantes pero tendríamos que tener en cuenta si esos enlaces vienen de páginas más o menos relevantes. Por ello, asociamos cada red de n páginas web con una $(n\text{x}n)$-*matriz de transición* $P = (p_{ij})$, donde p_{ij} expresa la probabilidad de transición de la página j a la página i. En caso de no haber enlace de la página j a i, dicha probabilidad es nula, en caso contrario:

$$p_{ij} = \frac{1}{enlaces \quad totales \quad que \quad salen \quad de \quad j}$$

La i-ésima fila consiste en las probabilidades de transición desde cada página a la i. La j-ésima columna consiste en las probabilidades de transición desde la página j a las otras. En el ejemplo anterior:

$$P = \begin{pmatrix} 0 & 0 & 1 & 1/3 & 1/3 \\ 1 & 0 & 0 & 0 & 1/3 \\ 0 & 1/2 & 0 & 1/3 & 1/3 \\ 0 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/3 & 0 \end{pmatrix}$$

La matriz de transición es estocástica, es decir, sus entradas son no negativas y cada columna suma 1:

$$p_{ij} \geq 0, \qquad \sum_{i=1}^{n} p_{ij} = 1, \quad \forall j \in \{1, \ldots, n\}.$$

En forma matricial: $e^T \cdot P = e^T$, donde $e = (1, \ldots, 1)^T$ es el vector n-dimensional de unos.

Sea $x_i \geq 0$ el ranking de la i-ésima página web (que es desconocido) el cual representa su relevancia. Para las n páginas web tendremos el vector ranking $x = (x_i, \ldots, x_n)^T \in \mathbb{R}^n$. Buscamos x_i tal que sea proporcional a la suma de las relevancias de cada página web por las probabilidades de transición correspondientes, es decir:

$$x_i = \sum_{j=1}^{n} p_{ij} \cdot x_j, \quad \forall i \in \{1, \ldots, n\}$$

que en forma matricial tenemos $x = P \cdot x$. Por tanto, el problema de Google para una red con n páginas web consiste en encontrar un vector ranking $x \in \mathbb{R}^n$ satisfaciendo:

$$x = P \cdot x, \qquad x_i \geq 0, \qquad x \neq 0$$

1.3. Resultados

En esta sección expondremos algunos resultados teóricos que nos serán de utilidad para el estudio del algoritmo PageRank.

1.3.1. Problema de autovectores

Definición 1. ***(Autovalor y autovector asociado de una matriz).*** *Sea $A \in \mathbb{M}_{nxn}(\mathbb{R})$, $\lambda \in \mathbb{R}$ es un autovalor de A si y solo si $\exists v \in \mathbb{R}^n$ no nulo tal que $Av = \lambda v$. (En ese caso, v es un autovector asociado al autovalor λ).*

Entonces, observamos que el problema de Google podemos estudiarlo como un problema de autovectores con matriz la matriz de transición P y autovalor $\lambda = 1$.
Entonces, surge la siguiente pregunta: ¿$\lambda = 1$ es realmente un autovalor de la matriz P? La respuesta es sí, basta ver que $det(P - I) = 0$:

$$det(P-I) = \begin{vmatrix} p_{11}-1 & p_{12} & \dots & p_{1n} \\ p_{21} & p_{22}-1 & \dots & p_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n1} & p_{n2} & \dots & p_{nn}-1 \end{vmatrix} \underset{(*)}{=} \begin{vmatrix} 0 & 0 & \dots & 0 \\ p_{21} & p_{22}-1 & \dots & p_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n1} & p_{n2} & \dots & p_{nn}-1 \end{vmatrix} = 0$$

(*) Sumamos cada fila $i = 2, \dots, n$ a la primera fila. Después usamos que la matriz P es estocástica.
Luego, efectivamente $\lambda = 1$ es autovalor de P.

1.3.2. Existencia

Ya sabemos que el problema de Google podemos interpretarlo como un problema de autovectores con matriz la matriz de transición P y autovalor $\lambda = 1$. Ahora, lo siguiente que nos preguntamos es si existe solución a nuestro problema, es decir, si el autovector asociado a $\lambda = 1$ tiene todas sus componentes no negativas.
Si tenemos un autovector asociado a un cierto autovalor sabemos que dicho autovector por

una constante no nula también será autovector, por lo que podemos tomar el autovector de tal forma que sus componentes sumen 1. Por tanto, consideremos el siguiente problema equivalente:

$$x = P \cdot x, \qquad x_i \geq 0, \qquad e^T \cdot x = 1 \tag{X}$$

Nuestro objetivo es demostrar que (X) es factible. Hagámoslo en 2 pasos:

1. Primero, introducimos la versión relajada de (X):

$$z \geq P \cdot z, \qquad z_i \geq 0, \qquad e^T \cdot z \geq 1 \tag{Z}$$

Veamos que si (Z) es factible, entonces (X) también lo es:
Supongamos que (Z) es factible. ¿Definiendo x como $x = \dfrac{z}{e^T \cdot z}$, es solución de (X)?

$$x - P \cdot x = \frac{z}{e^T \cdot z} - \frac{P \cdot z}{e^T \cdot z} = \underbrace{\frac{1}{e^T \cdot z}}_{>0} \cdot \underbrace{(z - P \cdot z)}_{\geq 0} \geq 0$$

$$e^T \cdot (x - P \cdot x) = e^T \cdot x - \underbrace{e^T \cdot P}_{=e^T} \cdot x = e^T \cdot x - e^T \cdot x = 0$$

$\Rightarrow x - P \cdot x = 0 \Rightarrow x = P \cdot x$
Las demás condiciones también se verifican:

$$x = \frac{z}{e^T \cdot z} \geq 0$$
$$e^T \cdot x = e^T \cdot \frac{z}{e^T \cdot z} = 1$$

Luego, si (Z) es factible, (X) también lo es.

2. Ahora veamos que (Z) es factible:
Sea el problema de programación lineal asociado

$$\min_z 0^T \cdot z, \qquad s.a. \quad z \geq P \cdot z, \quad z \geq 0, \quad e^T \cdot z \geq 1$$

$$\Leftrightarrow \qquad \min_z 0^T \cdot z, \qquad s.a. \quad \begin{pmatrix} I - P \\ e^T \end{pmatrix} \cdot z \geq \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad z \geq 0 \tag{P_z}$$

(Z) será factible si y solo si (P_z) lo es. El problema dual de (P_z) es:

$$\max_{y, y_{n+1}} \begin{pmatrix} 0 \\ 1 \end{pmatrix}^T \cdot \begin{pmatrix} y \\ y_{n+1} \end{pmatrix}, \qquad s.a. \quad \begin{pmatrix} I - P \\ e^T \end{pmatrix}^T \cdot \begin{pmatrix} y \\ y_{n+1} \end{pmatrix} \leq 0, \quad y, y_{n+1} \geq 0$$

$$\Leftrightarrow \qquad \max_{y, y_{n+1}} y_{n+1}, \qquad s.a. \quad y \leq P^T \cdot y - y_{n+1} \cdot e, \quad y, y_{n+1} \geq 0 \tag{D_z}$$

Teorema 1. ***(Teorema de dualidad fuerte).*** *Un problema de programación lineal es factible si y solo si su dual es factible. En ese caso, el valor óptimo coincide.*

Entonces, por el teorema de dualidad fuerte, basta ver que (D_z) es factible. Esto es cierto pues, por ejemplo, tomando $y = e$ y $y_{n+1} = 0$ es fácil ver que es solución de (D_z). (La prueba de dicho teorema se puede encontrar en [3]).
Por tanto, por la dualidad fuerte tenemos que (P_z) también es factible y por ello, (Z) es factible.

Luego, concluimos que (X) es factible como queríamos probar. Con esto, hemos demostrado que una matriz estocástica siempre tiene un autovector con componentes no negativas correspondiente al autovalor 1.

1.4. PageRank. Algoritmo

En la sección anterior probamos la existencia del ranking. Ahora vamos a dar un algoritmo que nos permita calcular dicho ranking de manera eficiente.

1.4.1. Navegación web

Modelemos el comportamiento de los usuarios cuando navegan en una red de n páginas web. Sea $x_i(t) \geq 0$ la proporción de usuarios que visitan la i-ésima página en el instante t. En general, para las n páginas web tenemos:

$$x(t) = (x_1(t), \ldots, x_n(t))^T, \qquad tal \quad que \quad x_i(t) \geq 0, \quad e^T \cdot x(t) = 1$$

Suponiendo conocido el comportamiento de los usuarios en el instante inicial, $x(1)$, podemos calcular dicho comportamiento en los instantes posteriores de manera iterativa:

$$x_i(t+1) = \sum_{j=1}^{n} p_{ij} \cdot x_j(t), \qquad i = 1, \ldots, n, \quad t \geq 1$$

En forma matricial:

$$x(t+1) = P \cdot x(t), \qquad t \geq 1 \tag{I}$$

Podemos demostrar que se verifican las condiciones definidas para $x(t)$ por inducción gracias a que la matriz P es estocástica:

1. $x(t+1) = \underbrace{P}_{\geq 0} \cdot \underbrace{x(t)}_{\geq 0} \geq 0$
2. $e^T x(t+1) = \underbrace{e^T \cdot P}_{=e^T} \cdot x(t) = e^T \cdot x(t) = 1$

Si suponemos que $x(t+1) \to x$ (siendo x el ranking) cuando $t \to \infty$, tomando el límite en

$$x(t+1) = P \cdot x(t), \quad x(t+1) \geq 0, \quad e^T \cdot x(t+1) = 1$$

tenemos:

$$x = P \cdot x, \quad x \geq 0, \quad e^T \cdot x = 1$$

Observamos que éste es nuestro problema inicial (X). Es decir, si (I) converge, da lugar a un ranking.
Lo podemos interpretar así: si un internauta aleatorio se mueve por la red durante un tiempo suficientemente grande, entonces la probabilidad de que se encuentre en la página i es la i-ésima componente del vector

$$\lim_{t\to\infty} P \cdot x(t)$$

1.4.2. Problema de oscilación

Sin embargo, (I) no siempre converge, puede ocurrir que oscile. Para evitar este fenómeno, regularizamos la matriz P como sigue:

$$P_\alpha = (1-\alpha) \cdot P + \alpha \cdot E$$

donde $\alpha \in (0,1)$ es el llamado factor de amortiguación (*damping factor*) y
$E = \frac{1}{n} \cdot e \cdot e^T \in \mathbb{M}_{nxn}(\mathbb{R})$ es una matriz estocástica con todas sus entradas iguales a $1/n$.
El uso de un factor α en este modelo indica que la probabilidad de que un internauta siga los enlaces con probabilidad las dadas en la matriz de transición P es $1-\alpha$ y que la probabilidad de que deje de seguir los enlaces y se mueva a otra página es α. En sus trabajos originales, Brin y Page utilizaron un factor $\alpha = 0.15$.

Es fácil comprobar que para cualquier valor $\alpha \in [0,1]$, la matriz P_α sigue siendo estocástica:

$$P_\alpha = (1-\alpha) \cdot \underbrace{P}_{\geq 0} + \alpha \cdot \underbrace{E}_{\geq 0} \geq 0$$

Además:

$$e^T \cdot P_\alpha = (1-\alpha) \cdot \underbrace{e^T \cdot P}_{=e^T} + \alpha \cdot \underbrace{e^T \cdot E}_{=e^T} = (1-\alpha) \cdot e^T + \alpha \cdot e^T = e^T$$

Entonces, el algoritmo PageRank quedaría como sigue:

$$x_\alpha(t+1) = P_\alpha \cdot x_\alpha(t), \qquad t \geq 1 \tag{I_α}$$

Tomando límite cuando $t \to \infty$, tenemos el problema de Google regularizado:

$$x_\alpha = P_\alpha \cdot x_\alpha, \quad x_\alpha \geq 0, \quad e^T \cdot x_\alpha = 1 \tag{X_α}$$

En caso de convergencia, dicho vector x_α es el llamado ranking de Google. Por tanto, vamos a tratar de demostrar la convergencia.

1.4.3. Convergencia del algoritmo

Sabemos que el ranking de Google, x_α, existe pues la matriz P_α es estocástica (ver sección (1.3.2)).
En $\mathbb{R}^n$, definimos la *norma Manhattan* como:

$$||x||_1 = \sum_{i=1}^{n} |x_i| = e^T \cdot |x|$$

Para demostrar la convergencia de (I_α), debemos ver que $||x_\alpha(t+1) - x_\alpha||_1 \underset{t\to\infty}{\to} 0$.

$$\begin{aligned}
||x_\alpha(t+1) - x_\alpha||_1 &= ||P_\alpha \cdot x_\alpha(t) - P_\alpha \cdot x_\alpha||_1 = ||P_\alpha \cdot (x_\alpha(t) - x_\alpha)||_1 = \\
&= ||((1-\alpha)\cdot P + \alpha \cdot E)\cdot (x_\alpha(t) - x_\alpha)||_1 = \\
&= \left|\left|(1-\alpha)\cdot P\cdot (x_\alpha(t) - x_\alpha) + \frac{\alpha}{n}(e \cdot \underbrace{e^T \cdot x_\alpha(t)}_{=1} - e \cdot \underbrace{e^T \cdot x_\alpha}_{=1})\right|\right|_1 = \\
&= (1-\alpha)\cdot ||P\cdot (x_\alpha(t) - x_\alpha)||_1 = (1-\alpha)\cdot e^T \cdot \underbrace{|P\cdot (x_\alpha(t) - x_\alpha)|}_{\leq P\cdot |x_\alpha(t) - x_\alpha|} \leq \\
&\leq (1-\alpha)\cdot \underbrace{e^T \cdot P}_{=e^T} \cdot |x_\alpha(t) - x_\alpha| = (1-\alpha)\cdot ||x_\alpha(t) - x_\alpha||_1
\end{aligned}$$

Si aplicamos lo anterior sucesivamente:

$$||x_\alpha(t+1) - x_\alpha||_1 \leq (1-\alpha)\cdot ||x_\alpha(t) - x_\alpha||_1 \leq \cdots \leq (1-\alpha)^t \cdot ||x_\alpha(1) - x_\alpha||_1$$

Por último, como $\alpha \in (0,1)$, tomando límite $t \to \infty$ tenemos el resultado:

$$||x_\alpha(t+1) - x_\alpha||_1 \underset{t\to\infty}{\to} 0$$

Luego, (I_α) converge al ranking de Google x_α.

1.5. El paquete "igraph" en R

El paquete *igraph* nos permite crear, manipular y analizar grafos utilizando R. El interés por el análisis de grafos surge del hecho de que existen multitud de problemas en distintas áreas (física, biología, ciencias sociales, ciencias de la computación, etc.) que se pueden modelar matemáticamente por medio de grafos. En nuestro caso, el estudio de rankings de páginas web a través de grafos (nodos: páginas web, arcos: hiperenlaces).

El primer paso para trabajar con esta librería suele ser definir el grafo sobre el que vamos a operar. El modelo de datos de *igraph* da soporte a dos tipos fundamentales de grafos, dirigidos y no dirigidos.

La forma más sencilla de crear un grafo, es llamando a la función *graph* y proporcionando directamente desde el terminal la lista de arcos que constituyen el grafo.
Por ejemplo:

```
g <- graph( c(1,2, 2,3, 1,3, 1,4, 4,4) )
```

donde estaríamos creando un grafo con cuatro vértices y cinco arcos:
V={1,2,3,4}, E={(1,2),(2,3),(1,3),(1,4),(4,4)}.

Veamos algunos ejemplos donde usamos grafos para el estudio de rankings mediante PageRank. En R existe una orden para ejecutar el algoritmo PageRank partiendo del grafo que representa una cierta red, dicha orden es:

```
page_rank(graph, algo = c("prpack", "arpack"), vids = V(graph), directed = TRUE,
    damping = 0.85, personalized = NULL, weights = NULL, options = NULL)
```

donde los argumentos de entrada son:

- **graph**: objeto grafo, que puede ser creado con la librería "igraph".
- **algo**: algoritmo utilizado. "prpack" utiliza un sistema de ecuaciones lineales y "arpack" un problema de autovalores.
- **vids**: vértices de interés.
- **directed**: "TRUE" para grafos dirigidos, "FALSE" en caso contrario.
- **damping**: factor de amortiguación.
- **personalized**: Vector opcional que da una distribución de probabilidad para calcular PageRank personalizado. Para el PageRank personalizado, la probabilidad de saltar a un nodo al abandonar el paseo aleatorio no es uniforme, pero viene dado por este vector. El vector debe contener una entrada para cada vértice y se reescalará para sumar a uno.
- **weight**: vector de ponderaciones de las aristas.

Vamos a ver dos casos prácticos en R:

1. Popularidad de una red: consideremos la red dada en la Figura 1.1. El código en R para obtener el ranking a través del algoritmo PageRank es el siguiente:

```
> library(igraph)
> red1 = graph( c(1,2, 2,3, 2,4, 3,1, 4,1, 4,3, 4,5, 5,1, 5,2, 5,3) )
> page_rank(
+   red1,
+   algo = "arpack",
+   vids = V(red1),
+   directed = TRUE,
+   damping = 0.85)$vector
[1] 0.27598960 0.28523102 0.21470962 0.15122319 0.07284657
```

Es decir, el vector ranking de Google para dicha red es:

$$x_\alpha = \begin{pmatrix} 0.276 \\ 0.285 \\ 0.215 \\ 0.151 \\ 0.073 \end{pmatrix}$$

La página web más popular es la 2 con un 28.5 % de usuarios, y la menos popular con un 7.3 %, la 5.

2. Lealtad a la marca: En este caso tenemos un mercado de calzado en el que tres marcas – Adidas, Nike y Puma – quieren conocer las preferencias de sus clientes. Consideramos 3 consumidores. El primero tiene el siguiente historial del uso de las zapatillas de las 3 marcas: ha pasado de Adidas a Adidas 7 veces, de Adidas a Nike y viceversa 1 vez, y de Adidas a Puma y viceversa 1 vez.
   El segundo cliente ha pasado de Puma a Nike 1 vez, 7 veces a seguido con Nike, y 1 vez ha pasado de tener Nike a Adidas.
   El tercer cliente ha sido fiel a Puma en los primeros 6 periodos de tiempo, ha pasado de Puma a Nike y viceversa 1 vez, de Puma a Adidas 1 vez y ha repetido con Adidas otra vez.
   En resumen, tenemos el siguiente grafo dirigido con las aristas ponderadas:

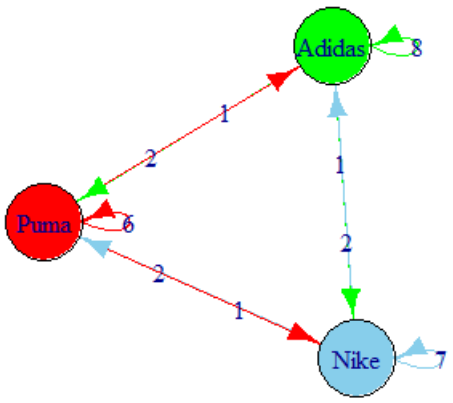

Figura 1.2: Brand Loyalty

El código en R es el siguiente:

```
> library(igraph)
> marcas = graph( c(1,1, 1,2, 1,3, 2,2, 2,1, 2,3, 3,3, 3,1, 3,2) )
> V(marcas)$name<-c("Adidas","Nike","Puma")
> E(marcas)$weight<-c(8, 1, 1, 7, 2, 1, 6, 2, 2)
> E(marcas)$name<-E(marcas)$weight
> V(marcas)$color<-c("green","skyblue","red")
> E(marcas)$color<-c("green","green","green","skyblue","skyblue","skyblue","red","red","red")
> plot(marcas, edge.label=E(marcas)$name,
+      vertex.size=50, vertex.label=V(marcas)$name)
> page_rank(
+   marcas,
+   algo = "arpack",
+   vids = V(marcas),
+   directed = TRUE,
+   damping = 1,
+   weights = c(8, 1, 1, 7, 2, 1, 6, 2, 2))$vector
Adidas   Nike   Puma
   0.5    0.3    0.2
```

Como conclusión tenemos que Adidas acumula el 50 %, Nike el 30 % y Puma el 20 % de los clientes.

1.5.1. Importación de grafos desde un fichero.

Crear un grafo directamente a través de la consola de R, como hemos hecho en los ejemplos anteriores, es un mecanismo efectivo cuando trabajamos con grafos pequeños. Sin embargo, cuando el número de vértices y arcos en el grafo es elevado, resulta más habitual que tengamos la información en un fichero y que carguemos los datos del mismo utilizando la función *read.graph* de *igraph*. Esta función permite importar datos desde ficheros de distintos formatos, dos de los más utilizados debido a su sencillez son:

- **Lista de arcos (edge list).** Básicamente se trata de un fichero de texto en el que cada línea representa un arco del grafo, descrito por medio de dos identificadores numéricos separados por un espacio en blanco o tabulador.
  Un pequeño aspecto a considerar cuando trabajamos con este tipo de ficheros es que en este caso *igraph* espera que el primer identificador de vértice en el fichero sea un 0 y añadirá automáticamente un 1 a todos los identificadores, para cumplir con la restricción de que el primero sea el 1.

- **Formato ncol.** De nuevo consiste en un fichero de texto en el que cada línea representa un arco, con dos etiquetas textuales separadas por espacio en blanco y un tercer elemento opcional que representa el peso asociado al arco. Las etiquetas textuales se cargarán en el atributo *name* de los vértices, mientras que el valor de la última columna se incluirá en el atributo *weight* de los arcos y la librería asignará automáticamente un identificador entero a cada vértice.

En ambos casos, podemos crear un grafo a partir de un fichero ejecutando la siguiente expresión:

read.graph("fichero.elist" (o "fichero.ncol") , format="edgelist" (o "ncol") , directed=TRUE)

Así, podemos llevar a cabo el algoritmo PageRank con una red de dimensión mucho mayor que la de los ejemplos anteriores. En este caso, el desarrollo en R es exactamente igual con la diferencia de que el grafo que representa la red lo importamos desde un fichero externo, ya que construirlo a mano sería inviable.

Capítulo 2

Aprendizaje Online

En este capítulo abordaremos el siguiente problema: no nos encontramos con grandes conjuntos de datos, sino con un flujo de datos. Es por ello que es necesario comprender y procesar rápidamente grandes cantidades de datos en tiempo real. De aquí surge lo que se conoce como aprendizaje online (*online learning*).

Veamos resumidamente la diferencia que supone frente al aprendizaje por lotes (*batch learning*). Éste último representa el entrenamiento de modelos de aprendizaje automático (*machine learning*) por lotes. En el aprendizaje por lotes, el sistema no es capaz de aprender de forma incremental. Los modelos deben ser entrenados utilizando todos los datos disponibles cada vez. Este entrenamiento de modelos requiere mucho tiempo y recursos informáticos. Estos modelos son de naturaleza estática, lo que significa que una vez que se entrenan, su rendimiento no mejorará hasta que se vuelva a entrenar un nuevo modelo. El rendimiento del modelo tiende a decaer lentamente con el tiempo, simplemente porque el mundo continúa evolucionando mientras el modelo permanece sin cambios. La solución es volver a entrenar regularmente el modelo con datos actualizados.

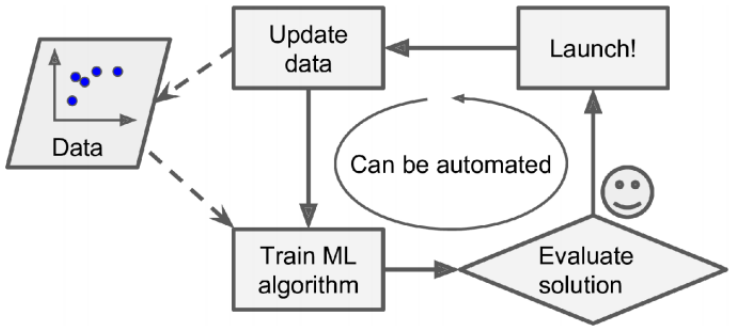

Figura 2.1: Batch Learning

Por otra parte, en el aprendizaje online el entrenamiento ocurre de manera incremental al alimentar continuamente los datos a medida que llegan. Cada paso de aprendizaje es rápido

y barato, por lo que el sistema puede aprender sobre nuevos datos sobre la marcha, a medida que llegan. El aprendizaje online es ideal para los sistemas de *machine learning* que reciben datos como un flujo continuo (por ejemplo, precios de acciones) y necesitan adaptarse al cambio de forma rápida o autónoma. Uno de los aspectos clave del aprendizaje online es la tasa de aprendizaje, esto es, la velocidad a la que desea que el aprendizaje automático se adapte al nuevo conjunto de datos. Un sistema con una alta tasa de aprendizaje tenderá a olvidar el aprendizaje rápidamente. Un sistema con una tasa de aprendizaje baja se parecerá más al aprendizaje por lotes. Una de las grandes desventajas de un sistema de aprendizaje online es que si se alimenta con datos erróneos, el sistema tendrá un mal rendimiento y el usuario verá el impacto al instante. Por lo tanto, es muy importante idear una estrategia de gobierno de datos adecuada para garantizar que los datos alimentados sean de alta calidad.

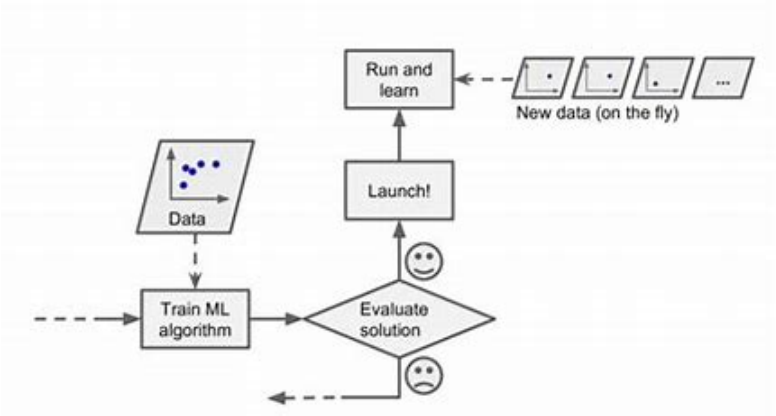

Figura 2.2: Online Learning

2.1. Motivación. Selección de cartera

En el mundo de las finanzas existen dos variables que son elementales para la evaluación de inversiones, que son el riesgo y el rendimiento. Desafortunadamente, el riesgo y el rendimiento esperado están asociados. Las carteras de bajo riesgo suelen generar rendimientos bajos. Por el contrario, si desea generar un rendimiento considerablemente alto, no hay más remedio que asumir muchos riesgos al seleccionar una cartera.
De aquí surge el famoso problema financiero de selección de cartera o *"portfolio selection"*. Su objetivo es crear una cartera óptima que refleje las preferencias de un inversor en términos de riesgo y rentabilidad. Por lo tanto, se pretende minimizar el riesgo de la cartera y aumentar su rendimiento esperado.

Harry Markowitz [7] desarrolló una teoría de *selección de cartera moderna*, la cual le llevó a ganar el Premio Nobel de Economía en 1990.
La idea principal de la teoría de Markowitz es que la relación que existe entre el riesgo y la rentabilidad de un mismo activo financiero no se debe analizar o evaluar de manera individual, al contrario, se debe valorar el contexto, la relación que hay entre el riesgo y la

rentabilidad, pero desde la perspectiva del conjunto de la cartera.
Markowitz sugiere hallar una cartera de activos de modo que se minimice el riesgo y se garantice un nivel específico de rentabilidad esperada. El modelo es una aplicación a la estadística y más en concreto a la distribución normal, lo que nos ayuda a pronosticar una potencial ganancia que consiste en el promedio de todas las simulaciones que pueden existir. Estadísticamente, el rendimiento esperado es la esperanza. Por otra parte los eventuales escenarios pueden dispersar la media, lo que estadísticamente se denomina varianza que, vista bajo este modelo, es el riesgo.

Sin embargo, nos centraremos en estudiar la teoría de *selección de cartera universal* de Vovk y Watkins [2]. La idea es la siguiente, la inversión puede verse como un proceso de toma de decisiones repetida con la debida atención a los cambios en los precios de los activos.
Veamos como lo podemos modelar matemáticamente: sean n activos en un mercado de valores, consideraremos estrategias de inversión para cada periodo t. Sea $W(t)$ la riqueza del inversor disponible al comienzo del periodo t, que será distribuida entre los activos fijando su cartera:

$$x(t) = (x_1(t), \dots, x_n(t))^T, \quad \text{donde}$$

$$x_i(t) \geq 0, \quad \text{para} \quad i = 1, \dots, n, \quad \sum_{i=1}^{n} x_i(t) = 1$$

esto es equivalente a decir que $x(t) \in \Delta$, donde

$$\Delta = \{x \in \mathbb{R}^n \quad | \quad x \geq 0, \quad e^T \cdot x = 1\}$$

Los precios de los activos estarán disponibles al final del periodo t, y entonces el inversor podrá evaluar el éxito de la cartera $x(t)$. Nos preguntamos cuál es la riqueza del inversor después de T periodos de tiempo, es decir, $W(T)$ partiendo de $W(0)$.
Sean $p(t) = (p_1(t), \dots, p_n(t))^T$ y $r(t) = (r_1(t), \dots, r_n(t))^T$, donde $p_i(t)$ y $r_i(t)$ es el precio y la rentabilidad respectivamente del i-ésimo activo después del periodo t. Si establecemos que:

$$r_i(t) = \frac{p_i(t)}{p_i(t-1)}, \quad i = 1, \dots, n$$

entonces, la rentabilidad global de la cartera $x(t)$ es:

$$r^T(t) \cdot x(t) = \sum_{i=1}^{n} r_i(t) \cdot x_i(t)$$

Por tanto, la riqueza del inversor después de T periodos de tiempo es:

$$W(T) = W(T-1) \cdot r^T(T) \cdot x(T) = \cdots = W(0) \cdot \prod_{t=1}^{T} r^T(t) \cdot x(t) \tag{2.1}$$

La riqueza $W(T)$ es resultado de una estrategia de inversión activa, esto es, el inversor busca obtener un buen rendimiento tomando como referencia un índice en particular que elige a su conveniencia; por lo tanto es flexible y tiene la posibilidad de tomar oportunidades que se le presentan a corto plazo. No sigue los movimientos del mercado de valores, sino que compra acciones cuyo valor pueda subir en un futuro cercano para generar ganancias.

La principal desventaja de esta estrategia es que implica mover constantemente las inversiones, por lo cual tiene un riesgo alto ya que alguien podría comprar una acción creyendo que generará ganancias y obtener pérdidas.

A pesar de no conocer el rendimiento de los activos $r(t)$, el inversor tendrá que realizar la cartera $x(t)$ al principio del periodo de tiempo t. El objetivo es estimar si las carteras $x(t)$, para $t = 1, \ldots, T$, son eficientes. Para ello, construimos una cartera de reequilibrio constante como punto de referencia, que maximiza la riqueza del inversor después de T períodos de tiempo si los rendimientos futuros están disponibles en retrospectiva:

$$\max_{x \in \Delta} W_x(T) \tag{2.2}$$

donde

$$W_x(T) = W(0) \cdot \prod_{t=1}^{T} r^T(t) \cdot x$$

En caso de que la información sobre los rendimientos de los activos futuros esté disponible, esta riqueza máxima resulta de una estrategia de inversión pasiva, es decir, el inversor busca generar ganancias basándose en los principales índices del mercado. Cuando los índices cambian, el fondo de inversión pasivo también lo hace; por lo cual compra las acciones que entran y vende las que salen.
La principal desventaja es que si los índices seleccionados tienen un desempeño bajo, la inversión también lo tendrá. Implica esperar para buscar las oportunidades adecuadas.

Hemos hallado dos tipos de riquezas según la estrategia. Al seguir una estrategia de inversión activa tenemos (2.1) y por otra parte, mediante una estrategia de inversión pasiva, (2.2).
Comparemos el rendimiento de las estrategias de inversión activa y pasiva a lo largo de T períodos de tiempo. Para ello, conviene tomar la relación logarítmica de las riquezas correspondientes por período, es decir:

$$R(T) = \frac{1}{T} \cdot \ln \frac{\max_{x \in \Delta} W_x(T)}{W(T)}$$

Si trabajamos en la expresión anterior, llegamos a lo siguiente:

$$\begin{aligned} R(T) &= \frac{1}{T} \cdot \ln \left(\max_{x \in \Delta} W_x(T) \right) - \frac{1}{T} \cdot \ln W(T) = \frac{1}{T} \cdot \max_{x \in \Delta} \ln W_x(T) - \frac{1}{T} \cdot \ln W(T) = \\ &= \max_{x \in \Delta} \frac{1}{T} \cdot \ln \left(W(0) \cdot \prod_{t=1}^{T} r^T(t) \cdot x \right) - \frac{1}{T} \cdot \ln \left(W(0) \cdot \prod_{t=1}^{T} r^T(t) \cdot x(t) \right) = \\ &= \max_{x \in \Delta} \frac{1}{T} \cdot \sum_{t=1}^{T} \ln \left(r^T(t) \cdot x \right) - \frac{1}{T} \cdot \sum_{t=1}^{T} \ln \left(r^T(t) \cdot x(t) \right) = \\ &= \frac{1}{T} \cdot \sum_{t=1}^{T} - \ln \left(r^T(t) \cdot x(t) \right) - \min_{x \in \Delta} \frac{1}{T} \cdot \sum_{t=1}^{T} - \ln \left(r^T(t) \cdot x \right) \end{aligned}$$

Ahora, si definimos $f_t(x) := -\ln\left(r^T(t) \cdot x\right)$:

$$R(T) = \frac{1}{T} \cdot \sum_{t=1}^{T} f_t(x(t)) - \min_{x \in \Delta} \frac{1}{T} \cdot \sum_{t=1}^{T} f_t(x)$$

Esta medida de eficiencia puede interpretarse como arrepentimiento en el contexto de la optimización online, en la que un agente toma sucesivamente decisiones factibles $x \in \Delta$.
Sin embargo, cuando se toma una decisión, el resultado asociado permanece desconocido para el agente. Justo después de comprometerse con una decisión $x(t)$, el tomador de decisiones sufre una pérdida $f_t(x(t))$. Estas pérdidas por lo general dependen del entorno y no pueden ser estimadas por el tomador de decisiones de antemano. Pueden ser elegidas incluso por un adversario y, en particular, dependen de las acciones realizadas por el tomador de decisiones.
Llegados a este punto, es crucial saber si las decisiones realizadas $x(t), t = 1, \ldots, T$ son eficientes. Aquí es donde aparece el llamado *arrepentimiento promedio* $R(T)$ definido anteriormente, que mide la diferencia entre las pérdidas asociadas a las decisiones propuestas por el agente a lo largo del tiempo, y las pérdidas generadas por la decisión constante óptima en retrospectiva. En lo que sigue, nuestro objetivo será ajustar las decisiones $x(t), t = 1, \ldots, T$ de tal manera que el arrepentimiento promedio $R(T)$ se vuelva pequeño, al menos asintóticamente, es decir, cuando $T \to \infty$.

2.2. Descenso del espejo online. Algoritmo

En esta sección introduciremos el algoritmo de descenso del espejo online (OMD) (del inglés *online mirror descent*), y algunos resultados de optimización convexa necesarios para el estudio de su convergencia.
Este algoritmo es una técnica de aprendizaje online, en particular, un algoritmo de optimización convexa online.

2.2.1. Definiciones y resultados

Definición 2. ***(Norma)****. Una norma en $\mathbb{R}^n$ es una función no negativa $||\cdot|| : \mathbb{R}^n \longrightarrow \mathbb{R}$ con las siguientes propiedades (para $x, y \in \mathbb{R}^n$, $\alpha \in \mathbb{R}$):*

- $||x|| = 0$ *si y solo si* $x = 0$.
- $||\alpha \cdot x|| = |\alpha| \cdot ||x||$
- $||x + y|| \leq ||x|| + ||y||$ *(desigualdad triangular)*

Definición 3. ***(Norma dual)***. *Dada una cierta norma $||\cdot||$, definimos su norma dual en $\mathbb{R}^n$ como:*

$$||y||_* = \max_{||x||\leq 1} y^T \cdot x, \quad x, y \in \mathbb{R}^n$$

Proposición 1. *La norma Manhattan es dual de la norma del máximo, i.e.,*

$$||y||_\infty = \max_{||x||_1 \leq 1} y^T \cdot x$$

Demostración:
Las normas Manhattan y del máximo se definen respectivamente como:

$$||x||_1 = \sum_{i=1}^{n} |x_i|, \quad ||x||_\infty = \max_{i=1,\ldots,n} |x_i|$$

Primero, estimamos una cota de $y^T \cdot x$:

$$y^t \cdot x \leq \sum_{i=1}^{n} |y_i| \cdot |x_i| \leq \max_{i=1,\ldots,n} |y_i| \cdot \sum_{i=1}^{n} |x_i| = ||y||_\infty \cdot ||x||_1 \Rightarrow$$
$$\Rightarrow \max_{||x||_1 \leq 1} y^T \cdot x \leq ||y||_\infty \cdot ||x||_1 \leq ||y||_\infty$$

Queda por demostrar que se alcanza dicha cota superior. Para ello, sea i el índice tal que $||y||_\infty = \max_{i=1,\ldots,n} |y_i| = |y_i|$. Tomamos $\bar{x}_i = sign(y_i)$ y $\bar{x}_j = 0, \quad \forall j \neq i$. Entonces:

$$||\bar{x}||_1 = |sign(y_i)| = 1, \quad y^t \cdot \bar{x} = y_i \cdot sign(y_i) = |y_i| = ||y||_\infty$$

□

Definición 4. ***(Conjunto convexo)***. *Un conjunto $X \subset \mathbb{R}^n$ se dice convexo si para todo par de puntos del conjunto, el segmento de recta que los une queda contenido en el conjunto, es decir,*

$$\lambda \cdot x + (1 - \lambda) \cdot y \in X, \quad \forall x, y \in X, \quad \lambda \in [0, 1]$$

Definición 5. ***(Función convexa)***. *Una función real $f(x)$ definida en un conjunto convexo X se dice que es convexa si:*

$$f(\lambda \cdot x + (1 - \lambda) \cdot y) \leq \lambda \cdot f(x) + (1 - \lambda) \cdot f(y), \quad \forall x, y \in X, \quad \lambda \in [0, 1]$$

Proposición 2. *Sea una función continuamente diferenciable $f : X \longrightarrow \mathbb{R}$ en un conjunto convexo X. Entonces, f es convexa si y solo si:*

$$f(y) \geq f(x) + \nabla^T f(x) \cdot (y - x), \quad \forall x, y \in X$$

Proposición 3. *Si f es dos veces continuamente diferenciable, entonces f es convexa si y solo si:*

$$\xi^T \cdot \nabla^2 f(x) \cdot \xi \geq 0, \quad \forall x \in X, \quad \xi \in \mathbb{R}^n$$

Teorema 2. *Sea el problema de optimización $\min_{x \in X} f(x)$ donde $X \subset \mathbb{R}^n$ es un conjunto cerrado convexo y $f : X \longrightarrow \mathbb{R}$ una función diferenciable convexa. Entonces, $\bar{x}$ minimiza f en X si y solo si:*

$$\nabla^T f(\bar{x}) \cdot (x - \bar{x}) \geq 0, \quad \forall x \in X$$

La prueba de estos tres resultados puede verse en [6].

Definición 6. ***(Función fuertemente convexa)***. *Decimos que una función f continuamente diferenciable es fuertemente convexa en un convexo X si existe una constante $\beta > 0$ tal que $\forall x, y \in X$:*

$$f(y) \geq f(x) + \nabla^T f(x) \cdot (y - x) + \frac{\beta}{2} \cdot ||y - x||^2$$

Notemos que β mide la curvatura de f en X.

Proposición 4. *Si f es dos veces continuamente diferenciable, entonces f es fuertemente convexa si y solo si:*

$$\xi^T \cdot \nabla^2 f(x) \cdot \xi \geq \beta \cdot ||\xi||^2, \quad \forall x \in X, \quad \xi \in \mathbb{R}^n$$

En lo que sigue, consideraremos un conjunto cerrado convexo $X \subset \mathbb{R}^n$, el cual tiene asociada una *función de proximidad* $d : X \longrightarrow \mathbb{R}$ que debe ser continuamente diferenciable y fuertemente convexa.

Definición 7. ***(Diámetro)*** *Definimos el diámetro de X como:*

$$D = \sqrt{\max_{x,y \in X} d(x) - d(y)}$$

Teorema 3. *Sea el problema de optimización $\min_{x \in X} d(x)$, con X un conjunto cerrado, convexo y d una función de proximidad. Entonces, existe una única solución denominada centro de proximidad de X, que denotamos por $x(1)$. Es decir:*

$$x(1) = arg \min_{x \in X} d(x)$$

La demostración de los dos últimos resultados puede encontrarse en [8].

Definición 8. ***(Divergencia de Bregman)*** *Definimos la divergencia o distancia de Bregman inducida por d como:*

$$B(x, y) = d(x) - d(y) - \nabla^T d(y) \cdot (y - x), \quad \forall x, y \in X$$

Proposición 5. ***(Identidad de los 3 puntos de Bregman).*** *Se verifica que:*

$$B(x, y) - B(x, z) - B(z, y) = (\nabla^T d(y) - \nabla^T d(z)) \cdot (z - x), \quad \forall x, y, z \in X \tag{3PI}$$

Consideremos el siguiente problema de optimización, el cual asumimos de momento que admite una única solución:

$$\min_{x \in X} c^T \cdot x + B(x, y) \tag{Aux}$$

donde $y \in X$, $c \in \mathbb{R}^n$ son dados.

2.2.2. Algoritmo

Definimos el algoritmo de descenso del espejo online, para $t = 1, 2, \ldots,$ como:

$$x(t+1) = arg\min_{x \in X} \underbrace{f_t(x(t)) + \nabla^T f_t(x(t)) \cdot x}_{\text{linealización de la función de pérdida}} + \underbrace{\frac{1}{\eta} \cdot B(x, x(t))}_{\text{regularización por la divergencia de Bregman}} \quad \text{(OMD)}$$

donde $\eta > 0$ es un tamaño de paso escogido adecuadamente, y bajo la consideración de las siguientes hipótesis:

(H1) $X \subset \mathbb{R}^n$ es un conjunto cerrado y convexo con función de proximidad asociada $d : X \longrightarrow \mathbb{R}$. β, $x(1)$ y D son su parámetro de convexidad respecto a una norma $||\cdot||$, el centro de proximidad de X y el diámetro, respectivamente. Por último, consideramos la divergencia de Bregman inducida por d, $B : X \text{x} X \longrightarrow \mathbb{R}$.

(H2) Las funciones de pérdida $f_t : X \longrightarrow \mathbb{R}$ son convexas con gradientes ∇f_t uniformemente acotados con respecto a la norma dual $||\cdot||_*$, es decir, existe una constante $C > 0$ tal que:

$$||\nabla f_t(x)||_* \leq C, \quad \forall x \in X, \quad t = 1, 2, \ldots$$

Observación 1. (OMD) *puede ser reescrito en la forma del problema de mínimos* (Aux)*:*

$$x(t+1) = arg\min_{x \in X} c^T \cdot x + B(x, y)$$

donde $c = \eta \cdot \nabla f_t(x(t))$, $y = x(t)$.
Luego, en cada etapa $t = 1, 2, \ldots,$ (OMD) *posee una única solución.*

2.2.3. Convergencia de (OMD)

En esta sección demostraremos la convergencia del algoritmo (OMD) en 3 pasos:

PASO 1. Estimación de pérdidas. Para $t = 1, 2, \ldots,$ se tiene que:

$$f_t(x(t)) - f(x) \leq \frac{1}{\eta} \cdot (B(x, x(t)) - B(x, x(t+1))) + \frac{\eta}{2\beta} \cdot ||\nabla f_t(x(t))||_*^2, \quad \forall x \in X \quad \text{(P1)}$$

PASO 2. Cota para el arrepentimiento promedio.

$$R(T) \leq \frac{1}{\eta} \cdot \frac{D^2}{T} + \eta \cdot \frac{C^2}{2\beta}, \quad \forall T > 0 \quad \text{(P2)}$$

PASO 3. Ajuste de los tamaños de paso.

$$R(T) \leq D \cdot C \cdot \sqrt{\frac{2}{\beta \cdot T}}, \quad \forall T > 0 \tag{P3}$$

donde el tamaño de paso es:

$$\eta = \frac{D}{C} \cdot \sqrt{\frac{2\beta}{T}}$$

Demostración:

Paso 1. Queremos probar (P1). Gracias a que las funciones de pérdida son convexas (H2), tenemos que:

$$f(x) \geq f_t(x(t)) + \nabla^T f_t(x(t)) \cdot (x - x(t)) \Leftrightarrow f_t(x(t)) - f(x) \leq \nabla^T f_t(x(t)) \cdot (x(t) - x) \Leftrightarrow$$
$$\Leftrightarrow f_t(x(t)) - f(x) \leq \underbrace{\nabla^T f_t(x(t)) \cdot (x(t+1) - x)}_{I} + \underbrace{\nabla^T f_t(x(t)) \cdot (x(t) - x(t+1))}_{II}$$

$\rightarrow$ Estimamos I: Por el Teorema 2 aplicado a (OMD), tenemos que:

$$\left(\nabla f_t(x(t)) + \frac{1}{\eta} \cdot (\nabla d(x(t+1)) - \nabla d(x(t)))\right)^T \cdot (x - x(t+1)) \geq 0 \Leftrightarrow$$
$$\Leftrightarrow \nabla^T f_t(x(t)) \cdot (x(t+1) - x) \leq \frac{1}{\eta} \cdot (\nabla d(x(t)) - \nabla d(x(t+1)))^T \cdot (x(t+1) - x)$$

Ahora, aplicando (3PI) a la segunda parte de la desigualdad, tenemos que:

$$I \leq \frac{1}{\eta}(B(x, x(t)) - B(x, x(t+1)) - B(x(t+1), x(t)) \tag{2.3}$$

$\rightarrow$ Estimamos II: Aplicando la conocida desigualdad de Hölder en II, tenemos que:

$$II \leq ||\nabla f_t(x(t))||_* \cdot ||x(t) - x(t+1)||$$

Usando que:

$$0 \leq \left(\sqrt{\frac{\eta}{2\beta}} \cdot ||\nabla f_t(x(t))||_* - \sqrt{\frac{\beta}{2\eta}} \cdot ||x(t) - x(t+1)||\right)^2 =$$
$$= \frac{\eta}{2\beta} \cdot ||\nabla f_t(x(t))||_*^2 + \frac{\beta}{2\eta} \cdot ||x(t) - x(t+1)||^2 - ||\nabla f_t(x(t))||_* \cdot ||x(t) - x(t+1)||$$
$$\Rightarrow ||\nabla f_t(x(t))||_* \cdot ||x(t) - x(t+1)|| \leq \frac{\eta}{2\beta} \cdot ||\nabla f_t(x(t))||_*^2 + \frac{\beta}{2\eta} \cdot ||x(t) - x(t+1)||^2$$

Llegamos a:

$$II \leq \frac{\eta}{2\beta} \cdot ||\nabla f_t(x(t))||_*^2 + \frac{\beta}{2\eta} \cdot ||x(t) - x(t+1)||^2 \tag{2.4}$$

Hagamos un inciso en la prueba para demostrar la siguiente relación:
$B(x, y) \geq \frac{\beta}{2} \cdot ||x - y||^2$

Dem. Por definición de la divergencia de Bregman, tenemos:

$$B(x,y) = d(x) - d(y) - \nabla^T d(y) \cdot (y - x)$$

Como d es fuertemente convexa:

$$d(x) \geq d(y) + \nabla^T d(y) \cdot (x - y) + \frac{\beta}{2} \cdot ||x - y||^2$$

Por tanto, tenemos la desigualdad buscada:

$$B(x,y) \geq \frac{\beta}{2} \cdot ||x - y||^2$$

□

Entonces, aplicando esta desigualdad a $B(x(t+1), x(t))$, tenemos:

$$B(x(t+1), x(t)) \geq \frac{\beta}{2} \cdot ||x(t+1) - x(t)||^2 \tag{2.5}$$

Luego, por (2.3), (2.4), (2.5), concluimos (P1).

Paso 2. Queremos probar (P2). Por definición, el arrepentimiento promedio está definido como:

$$R(T) = \frac{1}{T} \cdot \sum_{t=1}^{T} f_t(x(t)) - \min_{x \in X} \frac{1}{T} \cdot \sum_{t=1}^{T} f_t(x) = \frac{1}{T} \cdot \max_{x \in X} \sum_{t=1}^{T} (f_t(x(t)) - f_t(x))$$

Aplicando (P1) y sumando en $t = 1, \ldots, T$, tenemos:

$$\sum_{t=1}^{T} (f_t(x(t)) - f_t(x)) \leq \frac{1}{\eta} \cdot \underbrace{\sum_{t=1}^{T} (B(x, x(t)) - B(x, x(t+1)))}_{III} + \frac{\eta}{2\beta} \cdot \underbrace{\sum_{t=1}^{T} ||\nabla f_t(x(t))||_*^2}_{IV}$$

Veamos III:

$$III = B(x, x(1)) - \underbrace{B(x, x(T+1))}_{\geq 0} \leq B(x, x(1))$$

Recordemos que $x(1)$ es la única solución del problema de mínimos $\min_{x \in X} d(x)$. Entonces, por el Teorema 2:

$$\nabla^T d(x(1)) \cdot (x - x(1)) \geq 0$$

Teniendo en cuenta esto, por la definición de la divergencia de Bregman tenemos:

$$B(x, x(1)) = d(x) - d(x(1)) - \underbrace{\nabla^T d(x(1)) \cdot (x - x(1))}_{\geq 0} \leq d(x) - d(x(1)) \leq D^2$$

Luego, $III \leq D^2$.
Para ver IV, basta tener en cuenta la hipótesis (H2) y tenemos que:

$$IV = \sum_{t=1}^{T} ||\nabla f_t(x(t))||_*^2 \leq \sum_{t=1}^{T} C^2 = T \cdot C^2$$

Entonces:

$$\sum_{t=1}^{T} (f_t(x(t)) - f_t(x)) \leq \frac{1}{\eta} \cdot C^2 + \frac{\eta}{2\beta} \cdot T \cdot C^2$$

Por último, teniendo en cuenta la expresión de $R(T)$, obtenemos (P2).

Paso 3. Queremos probar (P3). Partiendo de lo obtenido en el paso 2, para obtener el tamaño de paso óptimo minimizamos dicha cota:

$$\min_{\eta \geq 0} \frac{1}{\eta} \cdot \frac{D^2}{T} + \eta \cdot \frac{C^2}{2\beta}$$

La condición de primer orden nos dice que:

$$-\frac{1}{\eta^2} \cdot \frac{D^2}{T} + \frac{C^2}{2\beta} = 0 \Leftrightarrow \eta = \frac{D}{C} \cdot \sqrt{\frac{2\beta}{T}}$$

Luego, sustituyendo el valor óptimo obtenido en la cota llegamos a lo que queríamos probar:

$$R(T) \leq D \cdot C \cdot \sqrt{\frac{2}{\beta \cdot T}}, \quad \forall T > 0, \qquad \square$$

Concluimos que el arrepentimiento promedio tiende a 0 cuando $T \to \infty$, con tasa de convergencia $1/\sqrt{T}$. Esta es la tasa óptima de convergencia para esquemas de aprendizaje online. Hemos demostrado que las decisiones $x(t), \quad t = 1, \ldots, T$ logran esta tasa.

2.3. Configuración entrópica

Definición 9. ***(Entropía negativa).*** *Definimos la función de proximidad entrópica (o entropía negativa) en el simplex Δ como:*

$$d(x) = \sum_{i=1}^{n} x_i \cdot \ln x_i, \quad x \in \Delta$$

Definición 10. ***(Divergencia de Kullback-Leibler).*** *Se define la divergencia de Kullback-Leibler como la divergencia de Bregman inducida por la función de proximidad entrópica en el simplex Δ:*

$$\begin{aligned} B(x, y) &= d(x) - d(y) - \nabla^T d(y) \cdot (x - y) = \\ &= \sum_{i=1}^{n} x_i \cdot \ln x_i - \sum_{i=1}^{n} y_i \cdot \ln y_i - \sum_{i=1}^{n} (\ln y_i + 1) \cdot (x_i - y_i) = \\ &= \sum_{i=1}^{n} x_i \cdot \ln x_i - \sum_{i=1}^{n} x_i \cdot \ln y_i = \sum_{i=1}^{n} x_i \cdot \ln \frac{x_i}{y_i} \end{aligned}$$

En la sección anterior, hemos demostrado la convergencia del algoritmo (OMD) bajo una serie de hipótesis: H1, H2 y la existencia de una única solución del problema de optimización (Aux). El objetivo de esta sección es probar que, efectivamente, dichas hipótesis son ciertas.

Para ello, vamos a declarar el algoritmo (OMD) en la configuración entrópica, es decir, tomamos como d la función de proximidad entrópica definida anteriormente. Además, tomamos como norma la norma Manhattan y como norma dual, la norma del máximo (ver la proposición 1). Entonces, demostremos que se cumplen las hipótesis anteriores:

(H1). Primero, vamos a probar que la función de proximidad entrópica definida en el simplex Δ es continuamente diferenciable y fuertemente convexa con respecto a la norma Manhattan, con constante de convexidad $\beta = 1$.
Tenemos que d es continuamente diferenciable con derivadas parciales:

$$\frac{\partial d}{\partial x_i}(x) = \ln x_i + 1, \quad i = 1, \dots, n$$

De hecho, d es dos veces continuamente diferenciable con matriz Hessiana:

$$\nabla^2 d(x) = diag\Big(\frac{1}{x_1}, \dots, \frac{1}{x_n}\Big)$$

Para demostrar que d es 1-fuertemente convexa, basta aplicar la proposición 4 con $\beta = 1$ y $||\cdot|| = ||\cdot||_1$:

$$\xi^T \cdot \nabla^2 d(x) \cdot \xi = \sum_{i=1}^n \frac{\xi_i^2}{x_i} \underset{(*)}{\geq} \Big(\sum_{i=1}^n |\xi_i|\Big)^2 = ||\xi||_1^2, \quad \forall \xi \in \mathbb{R}^n$$

$$(*) \quad \sum_{i=1}^n |\xi_i| = \sum_{i=1}^n \frac{|\xi_i|}{\sqrt{x_i}} \cdot \sqrt{x_i} \leq \sqrt{\sum_{i=1}^n \frac{\xi_i^2}{x_i}} \cdot \sqrt{\sum_{i=1}^n x_i} = \sqrt{\sum_{i=1}^n \frac{\xi_i^2}{x_i}}$$

Luego, d es 1-fuertemente convexa respecto a la norma Manhattan.
Ahora vamos a calcular el centro de proximidad $x(1)$ de la entropía negativa en el simplex Δ. Para ello, resolvemos el siguiente problema de optimización:

$$\min_{x \geq 0} d(x) = \sum_{i=1}^n x_i \cdot \ln x_i, \quad s.a. \quad e^T \cdot x - 1 = 0$$

Introduciendo los multiplicadores de Lagrange para la restricción de igualdad:

$$d(x) - \mu \cdot (e^T \cdot x - 1)$$

Por lo que obtenemos la condición de optimalidad siguiente:

$$\nabla d(x) - \mu \cdot \nabla(e^T \cdot x - 1) = 0 \Leftrightarrow \nabla d(x) = \mu \cdot \nabla(e^T \cdot x - 1) \underset{\text{(por componentes)}}{\Leftrightarrow}$$

$$\Leftrightarrow \ln x_i + x_i \cdot \frac{1}{x_i} = \mu \cdot 1 \Leftrightarrow \ln x_i = \mu - 1 \Leftrightarrow x_i = e^{\mu - 1}, \quad i = 1, \dots, n$$

Entonces, como todas las componentes de x son iguales $\Rightarrow x(1) = \left(\frac{1}{n}, \dots, \frac{1}{n}\right)^T$
Por último, vamos a obtener una cota para el diámetro D. De lo anterior concluimos que la entropía negativa tiene un límite inferior $\forall y \in \Delta$:

$$d(y) \geq d(x(1)) = \sum_{i=1}^{n} \frac{1}{n} \cdot \ln \frac{1}{n} = \ln \frac{1}{n} = -\ln n$$

Por otra parte, como $x_i \in [0,1] \quad \forall i = 1, \dots, n$, la entropía negativa está acotada superiormente por 0:

$$d(x) = \sum_{i=1}^{n} \underbrace{x_i}_{\geq 0} \cdot \underbrace{\ln x_i}_{\leq 0} \leq 0$$

Entonces, por la definición de diámetro:

$$D^2 = \max_{x,y \in \Delta} d(x) - d(y) \underset{(*)}{\leq} 0 + \ln n = \ln n \Rightarrow D \leq \sqrt{\ln n}$$

$$(*) \quad -\ln x \leq d(x) \leq 0$$

(H2). Probemos que las funciones de pérdida $f_t(x) = -\ln r^T(t) \cdot x$ son convexas con gradientes ∇f_t uniformemente acotados con respecto a la norma dual $||\cdot||_\infty$.
El gradiente y la matriz Hessiana de f_t vienen dados por:

$$\nabla f_t(x) = -\frac{r(t)}{r^T(t) \cdot x}, \quad \nabla^2 f_t(x) = \frac{r(t) \cdot r^T(t)}{(r^T(t) \cdot x)^2}$$

Observemos que la matriz Hessiana es semidefinida positiva, es decir, $\forall x \in \Delta, \xi \in \mathbb{R}^n$:

$$\xi^T \cdot \nabla^2 f_t(x) \cdot \xi = \xi^T \cdot \frac{r(t) \cdot r^T(t)}{(r^T(t) \cdot x)^2} \cdot \xi = \left(\frac{r^T(t) \cdot \xi}{r^T(t) \cdot x}\right)^2 \geq 0$$

Entonces, aplicando la proposición 3 tenemos que f_t es convexa.
Por otra parte, supongamos que $\rho_{min} \leq ||r(t)||_\infty \leq \rho_{max}, \quad \forall t = 1, 2, \dots,$ con $\rho_{min}, \rho_{max} > 0$. Entonces:

$$||\nabla f_t(x)||_\infty = \left|\left| -\frac{r(t)}{r^T(t) \cdot x} \right|\right|_\infty = \frac{||r(t)||_\infty}{r^T(t) \cdot x} \leq \frac{\rho_{max}}{\rho_{min} \cdot \sum_{i=1}^{n} x_i} = \frac{\rho_{max}}{\rho_{min}} := C$$

Luego, los gradientes ∇f_t están uniformemente acotados respecto a la norma dual $||\cdot||_\infty$.

(Aux.). Vamos a ver que el problema de optimización (Aux) posee una única solución en el simplex Δ. Consideramos la divergencia de Kullback-Leibler. Entonces, el problema (Aux) es:

$$\min_{x \geq 0} c^T \cdot x + B(x, y) = \sum_{i=1}^{n} c_i \cdot x_i + \sum_{i=1}^{n} x_i \cdot \ln \frac{x_i}{y_i}, \quad s.a. \quad \sum_{i=1}^{n} x_i - 1 = 0$$

donde $y \in \Delta$ y $c \in \mathbb{R}^n$ son dados.
Introduciendo un multiplicador de Lagrage $\mu \in \mathbb{R}$ para la restricción de igualdad, obtenemos:

$$c^T \cdot x + B(x, y) - \mu \cdot (e^T \cdot x - 1)$$

Por lo que obtenemos la condición de optimalidad siguiente (por componentes):

$$c_i + \ln \frac{x_i}{y_i} + 1 = \mu \Rightarrow x_i = e^{\mu-1} \cdot y_i \cdot e^{-c_i}$$

Sumando en $i = 1, \ldots, n$, obtenemos el valor de $e^{\mu-1}$:

$$1 = \sum_{i=1}^{n} x_i = e^{\mu-1} \cdot \sum_{i=1}^{n} y_i \cdot e^{-c_i} \Rightarrow e^{\mu-1} = \frac{1}{\sum_{i=1}^{n} y_i \cdot e^{-c_i}}$$

Luego, la única solución de (Aux) es:

$$x_i = \frac{y_i \cdot e^{-c_i}}{\sum_{i=1}^{n} y_i \cdot e^{-c_i}}, \quad i = 1, \ldots, n \tag{2.6}$$

2.4. Selección de cartera online

Después de introducir y estudiar el algortimo (OMD), lo aplicaremos al problema de selección de cartera introducido en la sección 2.1.
Por la observación 1, tenemos que $c = \eta \cdot \nabla f_t(x(t))$ e $y = x(t)$. Además, como hemos visto que $\nabla f_t(x) = -\frac{r(t)}{r^T(t)\cdot x}$, entonces $c = -\eta \cdot \frac{r(t)}{r^T(t)\cdot x}$.
Sustituyendo en (2.6), obtenemos que para $t = 1, 2, \ldots$:

$$x_i(t+1) = \frac{x_i(t) \cdot e^{\frac{\eta \cdot r_i(t)}{r^T(t)\cdot x(t)}}}{\sum_{i=1}^{n} x_i(t) \cdot e^{\frac{\eta \cdot r_i(t)}{r^T(t)\cdot x(t)}}}, \quad i = 1, \ldots, n$$

En el instante inicial $t = 0$, comenzamos con la cartera $x(1) = \left(\frac{1}{n}, \ldots, \frac{1}{n}\right)^T$.

Observación 2. *La nueva cartera $x(t+1)$ no depende solo de las iteraciones anteriores $x(t)$, sino también de los cocientes:*

$$\frac{r_i(t)}{r^T(t) \cdot x(t)}, \quad i = 1, \ldots, n$$

donde recordemos que $r_i(t)$ es la rentabilidad del i-ésimo activo después del periodo t, y $r^T(t) \cdot x(t)$ es la rentabilidad global de la cartera $x(t)$.
Dicho cociente representa la cuota de rentabilidad del i-ésimo activo dentro de la rentabilidad global de la cartera $x(t)$.

Observación 3. *Por otra parte, sabemos que el tamaño de paso óptimo es (ver sección 2.2.3)*

$$\eta = \frac{D}{C} \cdot \sqrt{\frac{2\beta}{T}}$$

Además, por los cálculos realizados en la sección 2.3, sabemos que:

$$\beta = 1, \quad C = \frac{\rho_{max}}{\rho_{min}}, \quad D \leq \sqrt{\ln n}$$

Entonces:

$$\eta = \frac{\rho_{min}}{\rho_{max}} \cdot \sqrt{\frac{2 \cdot \ln n}{T}} \quad \Rightarrow \quad R(T) \leq \frac{\rho_{max}}{\rho_{min}} \cdot \sqrt{\frac{2 \cdot \ln n}{T}}$$

Por tanto, concluimos que el arrepentimiento promedio $R(T)$ tiende a 0 cuando $T \to \infty$.

Capítulo 3

Sistemas de Recomendación

Antes del Machine Learning, lo más común era usar "rankings" con lo más votado, o más popular de entre todos los productos que se ofrecían. Entonces, a todos los usuarios se les recomendaba lo mismo. Es una técnica que aún se usa y en muchos casos funciona bien.

Una de las herramientas más conocidas y utilizadas que aportó el Machine Learning fueron los sistemas de recomendación. Los sistemas de recomendación son algoritmos que intentan predecir los siguientes ítems (productos, canciones, etc.) que querrá adquirir un usuario en particular. Son tan efectivos que estamos invadidos todos los días por recomendaciones y sugerencias aconsejadas por distintas apps y webs. Sin duda, los casos más conocidos de uso de esta tecnología son: Netflix acertando en recomendar series y películas, Spotify sugiriendo canciones y artistas o Amazon ofreciendo productos "sospechosamente" muy tentadores para cada usuario.

Los sistemas de recomendación intentan personalizar al máximo lo que ofrecerán a cada usuario. Esto es ahora posible por la cantidad de información individual que podemos recabar de las personas y nos da la posibilidad de tener una mejor tasa de aciertos, mejorando la experiencia del internauta sin ofrecer productos a ciegas.
Entre las estrategias más usadas para crear sistemas de recomendación encontramos:

- **Popularidad.** Aconseja por la "popularidad" de los productos. Por ejemplo, los más vendidos globalmente, se ofrecerán a todos los usuarios por igual sin aprovechar la personalización. Es fácil de implementar y en algunos casos es efectiva.
- **Recomendación basada en contenido.** A partir de productos visitados por el usuario, se intenta "adivinar" qué busca el usuario y ofrecer elementos similares.
- **Filtrado colaborativo.** Es el más novedoso, pues utiliza la información de masas para identificar perfiles similares y aprender de los datos para recomendar productos de manera individual.

Hay dos tipos de métodos que se usan comúnmente en el filtrado colaborativo, que se conocen como métodos basados en memoria y métodos basados en modelos (ver, por ejemplo [1]), los cuales desarrollaremos en este capítulo y los aplicaremos para la predicción de calificaciones de películas.

3.1. Motivación. Netflix

El Premio Netflix fue un concurso abierto al mejor algoritmo de filtrado colaborativo para predecir las calificaciones de los usuarios para películas, basado en calificaciones anteriores sin ninguna otra información sobre los usuarios o películas, es decir, sin que los usuarios sean identificados excepto por los números asignados para el concurso. La competencia estaba abierta a cualquier persona que no estuviera conectada con Netflix (empleados, agentes, parientes cercanos de empleados, etc.).
Netflix proporcionó un conjunto de datos de entrenamiento de 100.480.507 calificaciones (de 1 a 5 estrellas) que 480.189 usuarios dieron a 17.770 películas. El 21 de septiembre de 2009, el gran premio de $1,000,000$\$ fue otorgado al equipo Pragmatic Chaos de BellKor, que superó al propio algoritmo de Netflix (llamado *Cinematch*) para predecir las calificaciones en un 10.06 %.

Matemáticamente, un sistema de recomendación de este tipo se basa en calificaciones de m películas previamente especificadas por n usuarios. Esta información se almacena en una matriz de calificaciones $R = (r_{ij})$ de dimensión nxm, donde la entrada r_{ij} corresponde a la calificación de la j-ésima película por el i-ésimo usuario. Tengamos en cuenta que la entrada r_{ij} falta si el usuario i no ha visto o calificado la película j. Matrices de este tipo se llaman matrices dispersas ("sparse matrix"), y es el principal desafío en el diseño de métodos de filtrado colaborativo.

3.2. Enfoque basado en el vecindario

Un primer y sencillo intento de filtrado colaborativo sería basarnos en el vecindario. Los algoritmos basados en el vecindario también se conocen como métodos de filtrado colaborativo basados en memoria, en los que las calificaciones de las combinaciones usuario-elemento se predicen en función de sus vecindarios. Estos vecindarios se pueden definir de dos maneras:

- **Filtrado colaborativo basado en usuarios.** En este caso, las calificaciones proporcionadas por usuarios de ideas afines a un usuario objetivo A se utilizan para hacer las recomendaciones para A. Por lo tanto, la idea básica es determinar los usuarios que son similares al usuario objetivo A y recomendar calificaciones para las calificaciones

no observadas de A mediante el cálculo de promedios ponderados de las calificaciones de este grupo de pares.

- **Filtrado colaborativo basado en elementos.** En este caso, para hacer las predicciones de calificación para el elemento de destino B por parte del usuario A, el primer paso es determinar un conjunto S de elementos que son más similares al elemento objetivo B. Las calificaciones en el conjunto de elementos S, que son especificados por A, se utilizan para predecir si al usuario A le gustará o no el elemento B.

Esto es así pues usuarios similares muestran patrones similares de comportamiento de calificación y elementos similares reciben calificaciones similares.
En lo que sigue, dedicaremos la sección a estudiar el filtrado colaborativo basado en usuarios. En concreto, el llamado método de los k vecinos más cercanos.

3.2.1. Medidas de similitud

En primer lugar, parece razonable introducir herramientas que nos permitan evaluar el grado de diferencia o lejanía existente entre dos elementos. Para ello, introducimos el concepto de medida de similitud entre los usuarios i y l, $Sim(i,l)$.
Definimos el conjunto formado por las películas que han sido calificadas por el usuario i como:

$$M_i = \{ \quad j \in \{1,\dots,m\} \quad | \quad r_{ij} \quad \text{está especificada}\}$$

Existen diferentes similaridades, entre las que destacamos la similaridad del coseno:

$$Cos(i,l) = \frac{\sum_{j\in M_i\cap M_l} r_{ij}\cdot r_{lj}}{\sqrt{\sum_{j\in M_i\cap M_l} r_{ij}^2}\cdot\sqrt{\sum_{j\in M_i\cap M_l} r_{lj}^2}} = \frac{r_i^T\cdot r_l}{||r_i||_2\cdot||r_l||_2}$$

donde $r_i = (r_{ij}, \quad j\in M_i\cap M_l)^T$ y $r_l = (r_{lj}, \quad j\in M_i\cap M_l)^T$.

Otra similaridad típica es el coeficiente de correlación de Pearson, un índice que puede utilizarse para medir el grado de relación de dos variables siempre y cuando ambas sean cuantitativas y continuas.
Para ello, definimos las calificaciones medias correspondientes a los usuarios i y l como:

$$\mu_i = \frac{1}{|M_i\cap M_l|}\cdot\sum_{j\in M_i\cap M_l} r_{ij}, \qquad \mu_l = \frac{1}{|M_i\cap M_l|}\cdot\sum_{j\in M_i\cap M_l} r_{lj}$$

Así, el coeficiente de correlación de Pearson entre los usuarios i y l se define por:

$$Pearson(i,l) = \frac{\sigma_{il}}{\sigma_i\cdot\sigma_l} = \frac{\sum_{j\in M_i\cap M_l}(r_{ij}-\mu_i)\cdot(r_{lj}-\mu_l)}{\sqrt{\sum_{j\in M_i\cap M_l}(r_{ij}-\mu_i)^2}\cdot\sqrt{\sum_{j\in M_i\cap M_l}(r_{lj}-\mu_l)^2}}$$

donde

- σ_{il} es la covarianza de (r_i, r_l).
- σ_i es la desviación estándar de r_i.
- σ_l es la desviación estándar de r_l.

3.2.2. k vecinos más cercanos

En esta sección introduciremos el método de los k vecinos más cercanos (en inglés, *k-nearest neighbors*, abreviado "kNN").
Sea la similaridad del coseno o de Pearson, indistintamente denotada por $Sim(i, l)$, entre los usuarios i y l.
La idea es muy intuitiva, le pedimos opinión a nuestros amigos con los que sentimos que tenemos gustos similares (vecinos) y si muchos de ellos nos recomiendan una misma película, es muy probable que también nos guste.
Ahora que tenemos la idea del método de los k vecinos más cercanos, tratemos de modelarlo matemáticamente. Supongamos que el i-ésimo usuario no ha visto aún la j-ésima película, es decir, la entrada r_{ij} no está especificada. Definimos el conjunto de los usuarios que han calificado la j-ésima película como:

$$U_j = \{ \quad l \in \{1, \ldots, n\} \quad | \quad r_{lj} \quad \text{está especificada}\}$$

Supongamos que dicho conjunto es no vacío, y denotamos su cardinalidad por $n_j > 0$. Además, ordenamos el conjunto U_j de manera decreciente con respecto a su similitud con el usuario i, es decir:

$$U_j = \{l_1, \ldots, l_{n_j}\}, \quad \text{con} \quad Sim(i, l_1) \geq Sim(i, l_2) \geq \cdots \geq Sim(i, l_{n_j})$$

El siguiente paso es seleccionar los k vecinos más cercanos del usuario i:

$$N_j(i) = \{l_1, \ldots, l_{\text{mín}\{k, n_j\}}\}$$

Finalmente, la calificación faltante r_{ij} se establece como la suma ponderada de las calificaciones correspondientes especificadas por los k vecinos más cercanos del i-ésimo usuario, lo que da lugar a definir el método de los k vecinos más cercanos como:

$$r_{ij} = \sum_{l \in N_j(i)} \frac{Sim(i, l)}{\sum_{l \in N_j(i)} |Sim(i, l)|} \cdot r_{lj} \tag{kNN}$$

3.2.3. Ventajas y desventajas de (kNN)

Las técnicas de filtrado colaborativo basadas en el vecindario son sencillas de implementar y las recomendaciones resultantes suelen ser fáciles de interpretar.

Sin embargo, presenta algunos inconvenientes. En primer lugar, la escasez de datos puede crear predicciones erróneas. En nuestro caso, por ejemplo, se puede deber a que la cantidad de películas calificadas mutuamente entre dos usuarios es pequeña.
Por otra parte, antes de la fase de predicción debemos hallar los vecinos de cada usuario para obtener un buen resultado. No hay un modelo creado para la predicción sin antes tener una fase de preprocesamiento.

Para hacer frente a estos inconvenientes, y para mejorar la técnica de filtrado colaborativo, será necesario un enfoque alternativo.

3.3. Enfoque basado en modelos

En los métodos basados en modelos, se crea un modelo con los datos por adelantado. Por lo tanto, la fase de entrenamiento (o construcción de modelos) está claramente separada de la fase de predicción.

En nuestro caso, el modelo de predicción de Netflix se basa en que los usuarios califican las películas o series según ciertas características de éstas: géneros cinematográficos, actores famosos involucrados, directores destacados, etc. Este punto de vista permite incorporar propiedades estructurales del comportamiento de elección de los usuarios en el modelo de manera adecuada. En la fase de predicción posterior, los parámetros del modelo se aprenden mediante técnicas de optimización.

3.3.1. Descomposición en valores singulares

Supongamos que la (nxm)-matriz de calificaciones R tiene todas sus entradas especificadas, y que dichos valores se han obtenido en base a ciertas características de las m películas. Nuestro primer objetivo será conocer esas características. Esto será posible gracias a una técnica del álgebra lineal, la descomposición en valores singulares (o DVS) de una matriz.

Definición 11. ***(Valores singulares de una matriz)***. *Dada una matriz $A \in \mathbb{R}^{nxm}$ y sean $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$ los autovalores de la matriz cuadrada, simétrica y semidefinida positiva $A \cdot A^T \in \mathbb{R}^{nxn}$ ordenados de mayor a menor. Entonces, definimos el i-ésimo valor singular de A como $\sigma_i = \sqrt{\lambda_i}$.*

Definición 12. ***(DVS)***. *Una descomposición en valores singulares (DVS) de una matriz $A \in \mathbb{R}^{nxm}$ es una factorización del tipo $A = U \cdot \Sigma \cdot V^T$, donde $U \in \mathbb{R}^{nxn}$ y $V \in \mathbb{R}^{mxm}$ son ortogonales, y $\Sigma = diag(\sigma_1, \ldots, \sigma_n) \in \mathbb{R}^{nxm}$.*

Definición 13. ***(DVS Reducida)***. *Este tipo de descomposición resulta de quedarse sólo*

con los r valores singulares no nulos. En este caso, tenemos:

$$A = U \cdot \Sigma \cdot V \tag{DVS}$$

donde $U \in \mathbb{R}^{nxr}$ *con columnas ortogonales,* $V \in \mathbb{R}^{rxm}$ *con filas ortogonales y* $\Sigma = diag(\sigma_1, \ldots, \sigma_r) \in \mathbb{R}^{rxr}$.

En lo que sigue, trabajaremos con la descomposición en valores singulares reducida (DVS).
Interpretaremos la descomposición en valores singulares en términos de características latentes, es decir, características no directamente observables. Las características latentes revelan los patrones ocultos detrás del comportamiento de elección de los usuarios.
La matriz U puede ser vista como una matriz de calificaciones (característica-usuario). Su entrada u_{ik} es la calificación de la k-ésima característica especificada por el i-ésimo usuario. La matriz V puede verse como una matriz de características (característica-película). Su entrada v_{kj} caracteriza la manifestación de la k-ésima característica en la j-ésima película. El valor singular σ_k expresa la importancia de la k-ésima característica para todos los usuarios colectivamente.
Entonces, efectuando el producto matricial en (DVS) y tomando como A la matriz de calificaciones R, obtenemos la calificación de la j-ésima película especificada por el i-ésimo usuario como la suma ponderada siguiente:

$$r_{ij} = \sum_{k=1}^{r} \sigma_k \cdot u_{ik} \cdot v_{kj},$$

donde r es el número de características tenidas en cuenta a la hora de calificar las películas.

3.3.2. Rango y valores singulares positivos

En esta sección demostraremos que el rango de la (nxm)-matriz de calificaciones R es el número de valores singulares positivos, es decir,

$$rg(R) = r$$

Para la demostración, se distinguen dos casos posibles: $m \leq n$ y $m > n$. Demostraremos con detalle el primer caso, el segundo se realiza análogamente.

Supongamos entonces que $m \leq n$. Sabemos que el rango de R es el máximo número de columnas linealmente independientes, es decir, la dimensión del espacio columna de R, donde el espacio columna se define como $Col(R) = \{u | u = R \cdot v, v \in \mathbb{R}^m\}$.
Además, el espacio nulo de R se define como $Null(R) = \{v \in \mathbb{R}^m | R \cdot v = 0\}$.

Por (DVS), sabemos que $R = U \cdot \Sigma \cdot V$, con U y V ortogonales. Entonces, tenemos que:

$$\begin{cases} R \cdot v_i = \sigma_i \cdot u_i \\ R^T \cdot u_i = \sigma_i \cdot v_i \end{cases}, \quad i = 1, \ldots, \text{mín}\{n, m\} = m$$

Si consideramos los primeros r vectores asociados a $\sigma_i > 0$:

$$R \cdot \left(\frac{v_i}{\sigma_i}\right) = u_i, \quad \forall i = 1, \ldots, r$$

Entonces, éstos u_i pertenecen al espacio columna de R y como son linealmente independientes tenemos que:

$$dim(Col(R)) \geq r \tag{3.1}$$

Por otra parte, si consideramos los últimos $m - r$ vectores asociados a los valores singulares nulos:

$$R \cdot v_i = 0, \quad \forall i = r + 1, \ldots, m$$

Entonces, éstos v_i pertenecen al espacio nulo de R y como son linealmente independientes tenemos que:

$$dim(Null(R)) \geq m - r \tag{3.2}$$

Por último, usando la fórmula de la dimensión:

$$dim(Col(R)) + dim(Null(R)) = m,$$

De (3.1) y (3.2), concluimos que:

$$dim(Col(R)) = r \quad \Rightarrow \quad rg(R) = r$$

3.3.3. Aproximación de rango bajo

El objetivo de esta sección es introducir un mecanismo para predecir los valores perdidos de la matriz de calificaciones R. Para ello, construimos una matriz de aproximación $A = (a_{ij}) \in \mathbb{R}^{nxm}$. Las calificaciones perdidas serán aproximadas por las correspondientes entradas de A. La suma de sus diferencias al cuadrado mide la calidad de tal aproximación:

$$\sqrt{\sum_{(i,j) \in S} (r_{ij} - a_{ij})^2}$$

donde $S = \{(i, j) \quad | \quad r_{ij} \quad \text{está especificada}\}$.

Para hallar dicha matriz de aproximación, introducimos lo que se conoce como problema de aproximación de rango bajo. Es un problema de minimización en el que la función de costo mide el ajuste entre una matriz dada (los datos) y una matriz de aproximación (la variable de optimización), sujeto a la restricción de que la matriz de aproximación tiene

rango reducido.
Entonces, en nuestro caso, nos interesa el siguiente problema de aproximación de rango bajo:

$$\min_{A=(a_{ij})} \sqrt{\sum_{(i,j)\in S} (r_{ij} - a_{ij})^2}, \quad s.a \quad rg(A) = s \tag{A}$$

Notemos que fijando el rango de la matriz de aproximación A por s, estamos imponiendo un cierto patrón en el comportamiento de calificación de los usuarios. Es decir, imponemos que los usuarios califican las películas en base a s características latentes.

En las secciones posteriores estudiaremos cómo resolver el problema de optimización (A).

3.3.4. El teorema de Eckart-Young-Mirsky

Antes de intentar justificar (A), daremos algunos resultados del álgebra lineal que necesitaremos más adelante.

Proposición 6. *Sean $A \in \mathbb{R}^{nxm}$ y $B \in \mathbb{R}^{mxn}$. Entonces, $tr(A \cdot B) = tr(B \cdot A)$.*

Demostración.

$$(A \cdot B)_{ij} = \sum_{k=1}^{m} a_{ik} \cdot b_{kj} \Rightarrow tr(A \cdot B) = \sum_{i=1}^{n} (A \cdot B)_{ii} = \sum_{i=1}^{n} \sum_{k=1}^{m} a_{ik} \cdot b_{ki} \underset{(*)}{=}$$

$$= \sum_{k=1}^{m} \sum_{i=1}^{n} a_{ik} \cdot b_{ki} = \sum_{k=1}^{m} \sum_{i=1}^{n} b_{ki} \cdot a_{ik} = \sum_{k=1}^{m} (B \cdot A)_{kk} = tr(B \cdot A)$$

(*) Por la asociatividad del sumatorio. □

Definición 14. ***(Norma de Frobenius)***. *Sea una (nxm)-matriz A. Se define la norma de Frobenius como:*

$$||A||_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} |a_{ij}|^2} = \sqrt{tr(A^T \cdot A)} = \sqrt{tr(A \cdot A^T)} = \sqrt{\sum_{i=1}^{\min\{n,m\}} \sigma_i^2}$$

donde σ_i son los valores singulares de A.

Proposición 7. *La norma de Frobenius es invariante bajo transformaciones ortogonales.*

Demostración. Sean U y V matrices ortogonales.

$$||U \cdot A \cdot V||_F^2 = tr((U \cdot A \cdot V)^T \cdot U \cdot A \cdot V) = tr(V^T \cdot A^T \cdot \underbrace{U^T \cdot U}_{Id} \cdot A \cdot V) =$$

$$= tr((A \cdot V)^T \cdot A \cdot V) \underset{(*)}{=} tr(A \cdot V \cdot (A \cdot V)^T) = tr(A \cdot \underbrace{V \cdot V^T}_{Id} \cdot A^T) = tr(A \cdot A^T) = ||A||_F^2$$

(*) Por la Proposición 6. □

Asumimos por el momento que la matriz de calificaciones R no tiene entradas sin especificar. Entonces, el problema de optimización (A) puede ser reescrito en términos de la norma Frobenius como:

$$\min_{A} ||R - A||_F^2, \quad s.a \quad rg(A) = s \tag{F}$$

Teniendo en cuenta (DVS), sabemos que $R = U \cdot \Sigma \cdot V$ donde las columnas de U y las filas de V son ortogonales, y $\Sigma = diag(\sigma_1, \ldots, \sigma_s, \ldots, \sigma_r)$ con $\sigma_i, \quad i = 1, \ldots, r$, los valores singulares positivos y en orden no decreciente de R.
Introducimos el teorema de Eckart-Young-Mirsky, que nos asegura la mejor aproximación de R con rango s:

Teorema 4. ***(Eckart-Young-Mirsky).*** *La solución del problema de encontrar la mejor aproximación con rango s de una matriz de dimensiones nxm, es la descomposición en valores singulares tomando los s mayores valores singulares de R. Es decir,*

$$A = U \cdot \Sigma_s \cdot V, \qquad donde \quad \Sigma_s = diag(\sigma_1, \ldots, \sigma_s, 0, \ldots, 0).$$

Observación 4.

1. *La matriz de aproximación A dada el teorema 4 es factible para el problema de optimización (F), pues como hemos demostrado antes, el rango corresponde con el número de valores singulares positivos.*

2. *Error entre la matriz de calificaciones R y la matriz de aproximación A:*

$$||R - A||_F = ||U \cdot \Sigma \cdot V - U \cdot \Sigma_s \cdot V||_F = ||U \cdot (\Sigma - \Sigma_s) \cdot V||_F \underset{(*)}{=} ||\Sigma - \Sigma_s||_F =$$

$$= ||diag(0, \ldots, 0, \sigma_{s+1}, \ldots, \sigma_r)||_F = \sqrt{\sigma_{s+1}^2 + \cdots + \sigma_r^2}$$

   (*) Por la Proposición 7.
   Por tanto, el error de aproximación equivale a la norma euclídea de los $r - s$ valores singulares más pequeños de R.

3. *En caso de que la matriz R esté completamente especificada, de las r características disponibles tomaremos las s más importantes, que determinan el comportamiento de elección de los usuarios. Al resolver (F), sólo se pierde la información de $r - s$ características prescindibles.*

3.3.5. Factorización de matrices

La última cuestión importante que nos haremos es cómo podemos llevar a cabo la aproximación de rango bajo de la matriz de calificaciones R de manera eficiente. Ahora sí, asumimos que algunas entradas de R no están especificadas.
Nuestro objetivo es hacer una mejora del problema de optimización (A), que luego detallaremos las ventajas que ofrece. Para ello, le aplicaremos a la matriz de aproximación A una técnica de factorización de matrices [5].

Definición 15. ***(Factorización de una matriz).*** *En álgebra lineal, la factorización de una matriz es la descomposición de la misma como producto de dos o más matrices según una forma canónica.*

Existen diferentes tipos de factorizaciones como por ejemplo la factorización LU, la factorización de Cholesky, etc. En nuestro caso, usaremos la llamada factorización de rango.

Definición 16. ***(Factorización de rango).*** *Dada una matriz $A \in \mathbb{R}^{nxm}$ de rango s, una factorización de rango de A es una descomposición de A de la forma $A = X \cdot Y$, donde $X \in \mathbb{R}^{nxs}$ e $Y \in \mathbb{R}^{sxm}$, con $rg(X) = rg(Y) = s$.*

Podemos construir una factorización de rango utilizando la (DVS) de A:

$$A = U \cdot \Sigma_s \cdot V = \begin{pmatrix} U_1 & U_2 \end{pmatrix} \cdot \begin{pmatrix} \Sigma_s & 0 \\ 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} V_1 \\ V_2 \end{pmatrix} = U_1 \cdot (\Sigma_s \cdot V_1)$$

Como U_1 es una matriz de rango de columna completo y $\Sigma_s \cdot V_1$ es una matriz de rango de fila completo, podemos tomar $X = U_1$ e $Y = \Sigma_s \cdot V_1$.
Entonces $a_{ij} = \sum_{k=1}^{s} x_{ik} \cdot y_{kj}$, y el problema de optimización (A) reescrito en las nuevas variables queda así:

$$\min_{X=(x_{ik}),Y=(y_{kj})} \sqrt{\sum_{(i,j)\in S} (r_{ij} - \sum_{k=1}^{s} x_{ik} \cdot y_{kj})^2} \tag{R}$$

Las ventajas de (R) sobre (A) son las siguientes:

1. La principal mejora es que hemos conseguido eliminar la restricción sobre el rango, lo que facilita el tratamiento del problema.

2. Otra ventaja es que hemos disminuido el número de variables en (R) respecto a (A). Esto es pues el número de características es mucho menor que el número de usuarios y películas, es decir, $s << \text{mín}\{n, m\}$.
   Comparemos el número de variables en ambos problemas:

   $$\#variables(A) = size(A) = n \cdot m$$
   $$\#variables(R) = size(X) + size(Y) = n \cdot s + s \cdot m = s \cdot (n + m)$$

   Luego, al pasar de (A) a (R) hemos reducido considerablemente el número de variables, lo que computacionalmente es menos costoso.

3.3.6. Descenso del gradiente

En esta sección concluiremos con la resolución del problema de optimización (R) mediante una aplicación del método de descenso del gradiente. Por conveniencia, definimos la función

objetivo como:

$$f(X,Y) = \frac{1}{2} \cdot \sum_{(i,j)\in S} (r_{ij} - \sum_{k=1}^{s} x_{ik} \cdot y_{kj})^2$$

Así, el problema de optimización se reescribe como:

$$\min_{X=(x_{ik}),Y=(y_{kj})} f(X,Y) \tag{R'}$$

La idea del método del gradiente es la siguiente: el gradiente evaluado en cualquier punto representa la dirección del ascenso más pronunciado. Entonces, para minimizar una cierta función, podemos seguir el negativo del gradiente y así ir en la dirección del descenso más pronunciado.
Formalmente, en nuestro caso, el esquema iterativo del descenso del gradiente con tamaño de paso fijo $\eta > 0$ es el siguiente:

$$\begin{cases} X(t+1) = X(t) - \eta \cdot \nabla_X f(X(t), Y(t)) \\ Y(t+1) = Y(t) - \eta \cdot \nabla_Y f(X(t), Y(t)) \end{cases}$$

Las derivadas parciales de f son:

$$\frac{\partial f(X,Y)}{\partial x_{ik}} = -\sum_{j:(i,j)\in S} (r_{ij} - \sum_{k=1}^{s} x_{ik} \cdot y_{kj}) \cdot y_{kj}$$
$$\frac{\partial f(X,Y)}{\partial y_{kj}} = -\sum_{i:(i,j)\in S} (r_{ij} - \sum_{k=1}^{s} x_{ik} \cdot y_{kj}) \cdot x_{ik}$$

Definimos una (nxm)-matriz de error $E = (e_{ij})$, cuyas entradas son:

$$e_{ij} = \begin{cases} r_{ij} - \sum_{k=1}^{s} x_{ik} \cdot y_{kj}, & \text{si } r_{ij} \text{ está especificada} \\ 0, & \text{en caso contrario} \end{cases}$$

Entonces, podemos reescribir los gradientes como:

$$\nabla_X f(X,Y) = -E \cdot Y^T, \qquad \nabla_Y f(X,Y) = -X^T \cdot E$$

Finalmente, obtenemos el método de descenso del gradiente para resolver (R'):

$$\begin{cases} X(t+1) = X(t) + \eta \cdot E(t) \cdot Y^T(t) \\ Y(t+1) = Y(t) + \eta \cdot X^T(t) \cdot E(t) \end{cases} \tag{DG}$$

3.4. El paquete "recommenderlab" en R

El paquete *recommenderlab* se diseñó con el propósito de proporcionar una infraestructura de investigación completa para los sistemas de recomendación (ver [4]). Este paquete se centra en el filtrado colaborativo y proporciona implementaciones de muchos algoritmos, incluidos los siguientes:

- **Filtrado colaborativo basado en el usuario.** (En R, **UBCF**). Predice las valoraciones en función de las valoraciones de los usuarios que tienen un historial de valoraciones similar al del usuario activo.
- **Filtrado colaborativo basado en ítems.** (En R, **IBCF**). Busca ítems similares a los que le gustan al usuario activo.
- **Modelos de factores latentes.** (En R, **SVD**). Utilizan la descomposición en valores singulares para estimar las calificaciones que faltan.
- **Artículos populares.** (En R, **POPULAR**). Recomienda a todos los usuarios los artículos más populares que aún no han valorado.
- **Elementos elegidos al azar.** (En R, **RANDOM**). Crean recomendaciones aleatorias que pueden utilizarse como referencia para la evaluación de algoritmos de recomendación.
- **Volver a recomendar.** (En R, **RERECOMMENDER**). Puede ser útil para artículos que suelen consumirse más de una vez, por ejemplo, escuchar canciones o comprar alimentos.

En R, tenemos las siguientes órdenes para crear y evaluar modelos de sistemas de recomendación:

- *Recommender.* Implementa la estructura de datos para almacenar modelos de recomendación:

  $$\text{Recommender(data, method, parameter = NULL)}$$

  donde los argumentos de entrada son:

  - **data** → objeto tipo *ratingMatrix* de los datos.
  - **method** → nombre del método que se usa (UBCF, SVD, etc.).
  - **parameter** → parámetros opcionales.

- Una vez que tenemos el objeto *Recommender*, podemos predecir las recomendaciones "top-N" para ciertos usuarios a través de la orden *predict*:

predict(object, newdata, n=N, type=c("topNList", "ratings", "ratingMatrix"))

donde los argumentos de entrada son:

* **object** → objeto *Recommender*.
* **newdata** → datos de los usuarios activos.
* **N** → para las listas top-N, N es el número máximo de elementos recomendados en cada lista, y *predict* devuelve un objeto de clase *topNList* que contiene una lista para cada usuario activo.
  Para *ratings* y *ratingMatrix*, se ignora N y devuelve un objeto de clase *realRatingMatrix* en el que cada fila contiene las valoraciones previstas para un usuario activo. La diferencia es que, para *ratings*, los elementos para los que existe una valoración en newdata, tienen un NA en lugar de una predicción

3.4.1. Conjunto de datos "MovieLense"

Veamos dos ejemplos de recomendador de películas. El set de datos *MovieLense* del paquete *recommenderlab*, contiene las valoraciones de 943 usuarios sobre un total de 1664 películas almacenadas en un objeto de tipo *realRatingMatrix*. Sin embargo, hay que tener en cuenta que se trata de una matriz incompleta (94 % de valores ausentes), cada película ha sido valorada únicamente por una pequeña fracción de los usuarios.

```
> # INSPECCION DE DATOS MOVIELENSE
> library(ggplot2)
> library(recommenderlab)
> data("MovieLense")
> # Para no arrastrar el nombre completo de los datos, lo asignamos al objeto 'r'
> r=MovieLense
> r
943 x 1664 rating matrix of class 'realRatingMatrix' with 99392 ratings.
```

Figura 3.1: Set de datos MovieLense

Antes de comenzar con el modelo, podemos realizar un breve estudio de los datos:

```
> # Lo convertimos a data.frame para visualizar mejor (sin valores faltantes)
> rdf=as(r,"data.frame")
> head(rdf)
    user                                                item rating
1      1                                    Toy Story (1995)      5
453    1                                    GoldenEye (1995)      3
584    1                                   Four Rooms (1995)      4
674    1                                   Get Shorty (1995)      3
883    1                                      Copycat (1995)      3
969    1 Shanghai Triad (Yao a yao yao dao waipo qiao) (1995)      5
> summary(getRatings(r))
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    3.00    4.00    3.53    4.00    5.00
> # Diagrama de barras con la frecuencia de valoraciones (1 a 5)
> qplot(getRatings(r))+geom_bar()+labs(x="Valoraciones")
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
> # Valoración media de los usuarios y de las películas
> hist(rowMeans(r),main="Valoracion media de los usuarios",xlab="",ylab="")
> hist(colMeans(r),xlab="", main="Valoracion media de las películas",ylab="")
```

Figura 3.2: Resumen y figuras: MovieLense

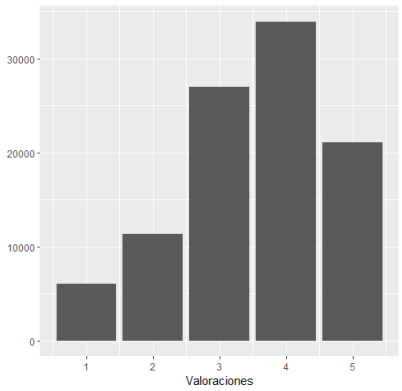

Figura 3.3: Frecuencia de valoraciones

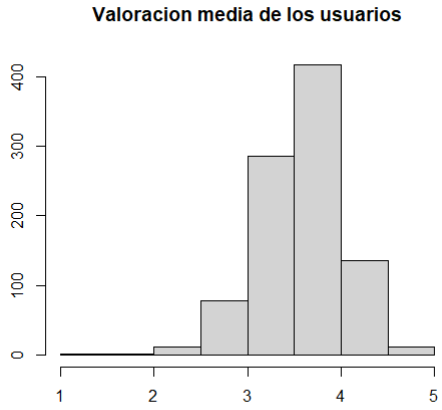

Figura 3.4: Valoración media usuarios

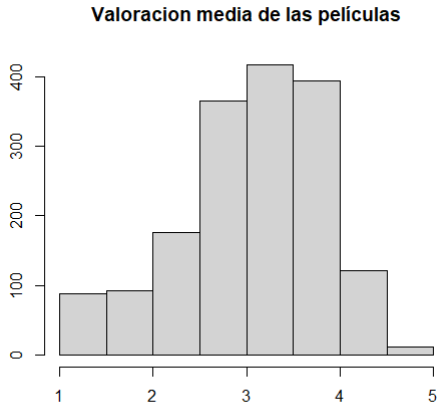

Figura 3.5: Valoración media películas

Ahora, procedemos a crear el modelo de predicción. Nuestro usuario objetivo será el usuario 329 y queremos recomendarle un total de 6 películas. Vamos a ver crear dos modelos distintos; en primer lugar, usando la técnica de filtrado colaborativo basada en usuarios y en segundo lugar, usando la descomposición en valores singulares mediante la optimización de descenso de gradiente estudiado anteriormente.

3.4.2. Filtrado colaborativo basado en usuarios

Después del breve estudio descriptivo del conjunto de datos *MovieLense*, creamos el modelo de predicción mediante la función *Recommender* tomando como conjunto de entrenamiento a todos los usuarios excepto el usuario objetivo. Después, hallamos las recomendaciones para dicho usuario con la función *predict*.

```
> # Nuestro usuario objetivo será el 329 y trataremos de recomendarle 6 películas
> x=329
> S=6
> # Creamos el modelo de recomendación
> rec=Recommender(r[-x], method = "UBCF")
> rec
Recommender of type 'UBCF' for 'realRatingMatrix'
learned using 942 users.
> # Predicccción de las valoraciones
> recom=predict(rec,r[x],n=S)
> recomS=as(recom,"list")
> recomS
$`0`
[1] "Oscar & Lucinda (1997)"       "She's So Lovely (1997)"       "Before Sunrise (1995)"
[4] "Stargate (1994)"              "Bram Stoker's Dracula (1992)" "Dragonheart (1996)"
```

Figura 3.6: Modelo de recomendación. "UBCF"

Por último, obtenemos en un formato *data.frame* las 6 películas recomendadas junto con las valoraciones que predice el aloritmo.

```
> # Índices de recomendación predichos por el algoritmo:
> recom=predict(rec,r[x],type="ratings")
> recom.df=as(recom,"data.frame")
> # y ordenamos de mayor a menor valoración:
> ii <- order(-recom.df$rating)
> head(recom.df[ii,])
    user                          item   rating
374  329       Oscar & Lucinda (1997) 5.747768
359  329       She's So Lovely (1997) 5.343006
382  329        Before Sunrise (1995) 5.343006
27   329              Stargate (1994) 4.875097
109  329 Bram Stoker's Dracula (1992) 4.875097
245  329           Dragonheart (1996) 4.875097
```

Figura 3.7: Predicción

3.4.3. Modelo SVD

En la sección 3.3.6 estudiamos cómo resolver el problema de optimización (R) mediante el método de descenso del gradiente, cuyo problema descendía de la factorización de rango de la matriz de aproximación A usando la descomposición en valores singulares.

En R existe una función de la librería *recommenderlab*, llamada *funkSVD*, la cual descompone una matriz (con valores perdidos) en dos componentes U y V:

funkSVD(x, k = 10, gamma = 0.015, lambda = 0.001, min_improvement = 1e-06,
min_epochs = 50, max_epochs = 200, verbose = FALSE)

donde los argumentos de entrada son:

* **x** → una matriz, que puede contener entradas sin especificar.
* **k** → número de características (es decir, el rango de la aproximación).
* **gamma** → término de regularización.
* **lambda** → tasa de aprendizaje.
* **min_improvement** → mejora mínima requerida por iteración.
* **min_epoch** → número mínimo de iteraciones por característica.
* **max_epoch** → número máximo de iteraciones por característica.
* **verbose** → muestra el progreso.

y obtenemos un objeto del tipo "funkSVD" con componentes:

* **U** → matriz U.
* **V** → matriz V.
* **parameters** → una lista con valores de parámetros.

Entonces, nuestro usuario objetivo será de nuevo el usuario 329 y queremos recomendarle un total de 6 películas. El código en R es el siguiente, donde hemos considerado 5 características de las películas en nuestro modelo (es decir, el rango de la matriz de aproximación es igual a 5):

```
> # INSPECCION DE DATOS MOVIELENSE
> library(ggplot2)
> library(recommenderlab)
> data("MovieLense")
> x=329
> train = as(MovieLense[-x], "matrix")
> fsvd = funkSVD(train, k=5)
> # Predicción
> recom <- predict(fsvd, MovieLense[x])
> # Índices de recomendación predichos por el algoritmo:
> recom.rm=as(recom,"realRatingMatrix")
> recom.df=as(recom.rm,"data.frame")
> # y ordenamos de mayor a menor valoración:
> ii <- order(-recom.df$rating)
> head(recom.df[ii,])
     user                              item   rating
50    329                 Star Wars (1977) 4.482917
1439  329           Pather Panchali (1955) 4.477604
12    329       Usual Suspects, The (1995) 4.327702
64    329 Shawshank Redemption, The (1994) 4.304935
480   329               Casablanca (1942) 4.293277
127   329            Godfather, The (1972) 4.290485
```

Figura 3.8: Modelo de recomendación. “SVD”

Conclusión

Este trabajo ha examinado algunos métodos matemáticos fundamentales para el análisis de Big Data, centrándose en los aspectos de ranking, aprendizaje online y sistemas de recomendación. Se ha demostrado que el uso de algoritmos matemáticos y técnicas de aprendizaje automático puede proporcionar herramientas efectivas para el procesamiento y análisis de grandes volúmenes de datos. Estos métodos pueden contribuir a la extracción de conocimientos útiles, la toma de decisiones y la generación de recomendaciones personalizadas en un entorno de Big Data cada vez más complejo y desafiante.

Bibliografía

[1] Charu C. Aggarwal. *Recommender Systems*. 2016.

[2] V. Vovk y C. Watkins. «Universal portfolio selection». En: *Proceedings of the Annual ACM Conference on Computational Learning Theory* (1998), págs. 12-23.

[3] E. Triesch H.T. Jongen K. Meer. *Optimization Theory*. 2004.

[4] Michael Hahsler. «recommenderlab: An R Framework for Developing and Testing Recommendation Algorithms». En: *Computer Science. Lyle School of Engineering, Southern Methodist University. Dallas* (2022).

[5] Volinsky C Koren Y Bell R. *Matrix Factorization Techniques for Recommender Systems*. 2009.

[6] Kenneth Lange. *Optimization*. 2013.

[7] H.M. Markowitz. *Portfolio Selection*. 1952.

[8] Yurii Nesterov. *Lectures on Convex Optimization*. 2018.

[9] Audrey Terras. *Abstract algebra with applications*. 2019.