Instituto Tecnológico Autónomo de México

Un sistema de recomendación basado en factorización de matrices y descenso en gradiente estocástico

TESIS

QUE PARA OBTENER EL TÍTULO DE

Licenciado en Matemáticas Aplicadas

PRESENTA

MARIO HUMBERTO BECERRA CONTRERAS

ASESOR

Fernando Esponda Darlington

CIUDAD DE MÉXICO 2017

"Con fundamento en los artículos 21 y 27 de la Ley Federal de Derecho de Autor y como titular de los derechos moral y patrimonial de la obra titulada "**Un sistema de recomendación basado en factorización de matrices y descenso en gradiente estocástico**", otorgo de manera gratuita y permanente al Instituto Tecnológico Autónomo de México y a la biblioteca Raúl Baillères Jr., autorización para que fijen la obra en cualquier medio, incluido el electrónico, y la divulguen entre sus usuarios, profesores, estudiantes o terceras personas, sin que pueda percibir por tal divulgación una prestación"

Mario Humberto Becerra Contreras

Fecha

Firma

Resumen

En este documento se presenta un método para crear un sistema de recomendación de productos que tienen calificaciones explícitas. Esto se hace utilizando un modelo de factorización de matrices y descenso en gradiente estocástico como método de optimización. Se prueba el modelo con dos conjuntos de datos y se compara su desempeño con un modelo base sencillo. Se ve que el modelo implementado es superior al modelo base para ambos conjuntos de datos.

Índice general

1. Introducción **1**

2. Marco teórico **4**

2.1. Aprendizaje estadístico 4

2.1.1. Aprendizaje supervisado 5

2.1.2. Aprendizaje no supervisado 7

2.1.3. Sobreajuste y regularización 7

2.2. Optimización de funciones de pérdida 9

2.2.1. Descenso en gradiente 10

2.2.2. Descenso en gradiente estocástico 17

2.3. Sistemas de recomendación 22

2.3.1. Basados en contenido 23

2.3.2. Colaborativos 25

2.3.3. Evaluación de modelos 34

3. Resultados **36**

3.0.1. Análisis exploratorio de datos 36

3.0.2. Comparación de modelos 41

4. Conclusiones y trabajo futuro **57**

Bibliografía **59**

Índice de figuras

2.1. Datos generados para una regresión lineal simple. 13

2.2. Iteraciones del algoritmo de descenso en gradiente para regresión lineal. 13

2.3. Datos generados para una regresión logística. 16

2.4. Iteraciones del algoritmo de descenso en gradiente para regresión logística. 16

2.5. Iteraciones del algoritmo de descenso en gradiente estocástico para regresión lineal. 19

2.6. Iteraciones del algoritmo de descenso en gradiente para regresión lineal. Cada flecha apunta al valor de los parámetros después de cada *epoch*. 20

2.7. Iteraciones del algoritmo de descenso en gradiente para regresión logística. 21

2.8. Iteraciones del algoritmo de descenso en gradiente para regresión logística. Cada flecha apunta al valor de los parámetros después de cada *epoch*. 22

3.1. Frecuencia de calificaciones del conjunto de datos *MovieLens*. . . . 37

3.2. Histograma del promedio de calificaciones por película del conjunto de datos *MovieLens*. 37

3.3. Cola larga del conjunto de datos *MovieLens*. 38

3.4. Calificación promedio semanal de todas las películas del conjunto de datos *MovieLens*. 38

3.5. Frecuencia de calificaciones del conjunto de datos *BookCrossing*. . 39

3.6. Histograma del promedio de calificaciones por libro del conjunto de datos *BookCrossing*. 40

3.7. Cola larga del conjunto de datos *BookCrossing*. 41

3.8. Errores del modelo base en el conjunto de prueba de *MovieLens*, de acuerdo al valor del parámetro de regularización γ. 43

3.9. Errores del modelo de factorización en el conjunto de prueba de *MovieLens*. LR es la tasa de aprendizaje, DL es el número de dimensiones latentes y L es el parámetro de regularización λ. 44

3.10. Errores de entrenamiento y validación del modelo de factorización en cada iteración del algoritmo de optimización para *MovieLens*. . 45

3.11. Recall para cada valor de N en la tarea de las mejores N recomendaciones para *MovieLens*. 46

3.12. Precisión para cada valor de N en la tarea de las mejores N recomendaciones para *MovieLens*. 47

3.13. Errores del modelo base en el conjunto de prueba de *BookCrossing*, de acuerdo al valor del parámetro de regularización γ. 52

3.14. Errores de entrenamiento y validación del modelo de factorización en cada iteración del algoritmo de optimización para *BookCrossing*. 53

3.15. Recall para cada valor de N en la tarea de las mejores N recomendaciones para *BookCrossing*. 54

3.16. Precisión para cada valor de N en la tarea de las mejores N recomendaciones para *BookCrossing*. 55

Índice de tablas

3.1. Número de usuarios, calificaciones y artículos en los conjuntos de *MovieLens*. 42

3.2. Algunas películas y sus vecinos más cercanos. 49

3.3. Algunas películas y sus vecinos más cercanos. 50

3.4. Número de usuarios, calificaciones y artículos en los conjuntos de *BookCrossing*. 51

3.5. Algunos libros y sus vecinos más cercanos. 56

Capítulo 1

Introducción

Este primer capítulo enuncia la motivación y los objetivos de este trabajo, además de explicar los contenidos de los demás capítulos.

El objetivo de este trabajo es implementar un sistema de recomendación utilizando factorización de matrices utilizando como método de optimización descenso en gradiente estocástico. Todo se implementa desde cero, es decir, no se usa ninguna paquetería especializada en el tema. El modelo de factorización de matrices que se implementa en este trabajo fue muy utilizado entre todos los concursantes del concurso de Netflix [4], aunque con distintas pequeñas variaciones. De hecho, el modelo ganador del concurso era un ensamble de muchos modelos predictores [3]. El modelo explicado asume que se tienen calificaciones explícitas por usuario y por ende, no puede ser extendido más allá. El alcance de este trabajo es explicar e ilustrar desde un principio los conceptos necesarios para construir e implementar un sistema de recomendación con las características ya mencionadas.

Los sistemas de recomendación se han vuelto más populares últimamente debido a un concepto: cola larga. Este concepto rompe la creencia del principio de Pareto que afirma que el 80 % de las consecuencias (ganancias) provienen del 20 % de las causas (productos). Este principio se ha usado como regla de dedo en distintas disciplinas, pero en ciertos productos no se cumple, por ejemplo, las rentas de

películas en *Netflix*, las cuales tienen una cola larga. En una distribución de cola larga, es menos del 80 % de las ganancias el que proviene del 20 % de los productos, esto significa que se pueden tener ganancias a partir de productos no tan populares [1].

Con la cantidad de información que se genera hoy en día y la cantidad de servicios *online* que se tienen, la cola larga (*long tail*) se presenta muy a menudo. Esto porque la principal limitante de la cola larga es la escasez. Por ejemplo, en un supermercado no se puede tener la misma cantidad de productos ofrecidos como en *Ebay* porque el primero está limitado a lo que cabe en el supermercado. En servicios como *Netflix* o *YouTube* no se tiene esta limitante, por lo que hay muchísimos productos que se ofrecen, y para todos los productos hay algún consumidor, y en estos productos que poca gente consume se crean pequeños nichos de mercado que pueden ser explotados. De ahí nace la importancia y la necesidad de personalizar el contenido que se ofrece a los usuarios, para que puedan encontrar productos que les son de interés y que no pueden encontrar de forma sencilla debido a la enorme cantidad de opciones que se tienen. Esto le funcionó bastante bien a *Netflix*, pues en un principio, el 30 % de sus rentas provenía de estrenos, comparado con el 70 % de *Blockbuster*; y esto era en parte gracias al sistema de recomendación, y a que *Netflix*, a diferencia de *Blockbuster*, no tenía el espacio limitado para guardar películas físicamente [18].

Un ejemplo extremo de la aplicación de los sistemas de recomendación y de la cola larga es el libro *Touching the Void*. Este libro no fue muy popular cuando acababa de salir, pero años después un libro parecido llamado *Into Thin Air* fue publicado. Ambos eran vendidos por *Amazon*, y su sistema de recomendación encontró algunas personas que compraron ambos libros y empezó a recomendar *Touching the Void* a usuarios que habían comprado o considerado comprar *Into Thin Air*, y eventualmente *Touching the Void* se volvió popular [17].

La investigación en el área de los sistemas de recomendación creció mucho a partir de 2006 cuando en octubre Netflix Inc. publicó un conjunto de datos que contenía 100 millones de calificaciones anónimas de más o menos 18 mil películas producidas por cerca de 480 mil usuarios, esto con el objetivo de retar a la comunidad

científica para que crearan algoritmos que superaran el sistema que utilizaban en ese entonces, denominado *Cinematch*, con una recompensa de 1 millón de dólares al primero que lo mejorara en un 10 % [3] [5] [22].

En el capítulo 2 se presenta la teoría matemática y estadística, los conceptos de los sistemas de recomendación, el modelo base y el modelo de factorización de matrices. En el capítulo 3 se muestra el desempeño del sistema implementado comparándolo con el modelo base con dos conjuntos de datos distintos. En el capítulo 4 se presentan las conclusiones y el trabajo futuro.

Capítulo 2

Marco teórico

2.1. Aprendizaje estadístico

Las definiciones enunciadas en esta sección provienen de [6], [11] y [14].

El aprendizaje estadístico se refiere a un conjunto de herramientas para modelar y entender conjuntos de datos complejos. Es un área relativamente moderna, y tiene componentes de computación. Con la enorme cantidad de datos a los que se tiene acceso ahora, el aprendizaje estadístico se ha vuelto un campo muy demandado y fructífero en muchas áreas. El progreso que se ha logrado en los últimos años se ha debido en gran parte al desarrollo del poder de cómputo, sin el cual muchas de las técnicas modernas sería imposible tener un resultado.

Usualmente se divide al aprendizaje estadístico en dos vertientes: aprendizaje supervisado y aprendizaje no supervisado. El primero involucra construir un modelo estadístico para predecir o estimar una variable llamada *de respuesta* basado en una o más variables llamadas *de entrada*; mientras que en el aprendizaje no supervisado se tienen variables de entrada pero no de salida, se desea aprender relaciones y estructura de los datos.

Un ejemplo de aprendizaje supervisado es estimar el ingreso de un hogar a partir

de variables acerca de la zona en donde se habita, tipo de casa en la que se vive, número de carros, etc. Y un ejemplo de aprendizaje no supervisado sería encontrar grupos de los hogares que surjan naturalmente a partir de los datos, de tal forma que dentro de cada grupo exista una gran similitud (definida *a priori*) y entre los grupos haya diferencias. Gran parte del problema aquí radica en definir la medida de similitud.

2.1.1. Aprendizaje supervisado

Como se mencionó, en el aprendizaje supervisado, se tiene una variable respuesta, la cual será denotada como el vector Y de dimensión n, y cada variable de entrada se denomina X_i, con i desde 0 hasta p. Cada X_i, al igual que Y, es un vector de dimensión n, por lo que el conjunto de datos $\{X_1, ..., X_p\}$ se puede escribir como una matriz $X = [X_1, ..., X_p]$ de dimensiones $n \times p$. Se asume que existe una relación entre Y y X que se puede escribir de forma general como

$$Y = f(X) + \varepsilon. \tag{2.1}$$

Aquí, f es una función de X fija pero desconocida, la cual tiene información sistemática de X acerca de Y, y ε es un término de error aleatorio, independiente de X y con media cero. El objetivo esencial del aprendizaje supervisado es poder encontrar o aprender f a partir del *conjunto de entrenamiento* denominado $\mathcal{L}$, tal que $\mathcal{L} = \{(x_1, y_1), ..., (x_n, y_n)\}$. En este caso, cada y_i es un escalar y cada x_i es un vector de dimensión p. Con $\mathcal{L}$ se construye un predictor $\hat{f}$, que es una estimación de f, de tal forma que se tiene una estimación $\hat{Y}$ de Y aplicando X a $\hat{f}$, es decir

$$\hat{Y} = \hat{f}(X).$$

La estimación $\hat{f}$ se hace seleccionando f de una familia de funciones $\mathcal{F}$ de tal forma que $y_i \approx \hat{f}(x_i)$ para cada $(x_i, y_i) \in \mathcal{L}$. Por ejemplo, en regresión lineal se resuelve el problema de mínimos cuadrados

$$\hat{f} = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2,$$

donde $\mathcal{F}$ es la familia de funciones de la forma $f(X) = X\beta$ con $\beta \in \mathbb{R}^p$. Así, $\hat{f}(X) = X\hat{\beta}$, donde $\hat{\beta}$ resuelve el problema de mínimos cuadrados. En este caso, se quiere encontrar el vector de parámetros β tal que la suma de residuales al cuadrado sea mínima.

Si la familia de funciones $\mathcal{F}$ es muy inflexible (como regresión lineal), entonces las predicciones tenderán a ser malas pues no se capta bien la relación entre X y Y, sin embargo, esto puede ser deseable en algunos casos, como cuando se pretende interpretar la relación entre X y Y.

Es prudente introducir el concepto de *función de pérdida* o de error. Sea $\hat{Y} = \hat{f}(X)$, la predicción de Y; en el caso de regresión lineal, $\hat{Y} = X\hat{\beta}$. En este mismo caso la función de pérdida fue la de pérdida cuadrática, definida como $L_{\mathcal{L}}(\hat{Y}, Y) = (\hat{f}(X) - Y)^2$, o sea, la diferencia entre la estimación de Y y la verdadera Y al cuadrado, con los datos del conjunto $\mathcal{L}$. Existen distintos tipos de funciones de pérdida, pero muchas veces se utiliza la pérdida cuadrática debido a las propiedades diferenciables que tiene.

En general, cuando uno entrena un modelo para predecir, no se desea minimizar el error de entrenamiento $L_{\mathcal{L}}$, sino el error de predicción, esto es, el error de cualquier observación futura (X^0, Y^0). Esto quiere decir que no se quiere minimizar $L_{\mathcal{L}}(f, \hat{f})$, sino el error esperado de predicción, definido como $\mathbb{E}\left[L(\hat{f}(X^0), Y^0)\right]$.

Debido a que se quiere minimizar el error esperado de predicción, es común separar el conjunto de datos (X, Y) en dos, el conjunto de entrenamiento, presentado anteriormente como $\mathcal{L}$, y un *conjunto de validación* o *conjunto de prueba* denotado como $\mathcal{T}$. Para hacer esto, del conjunto original de datos, se toma una muestra aleatoria para entrenar (este es el conjunto de entrenamiento $\mathcal{L}$) y el resto es el conjunto de validación $\mathcal{T}$, con el cual se prueba el poder predictivo del modelo. A $L_{\mathcal{T}}$ se le conoce como el error de validación o error de prueba, y es una estimación del error de predicción. Si $\mathcal{T}$ tiene m elementos, y m es grande, entonces el error de prueba se aproxima al error esperado de predicción.

Notar que el error de entrenamiento no aproxima el error de predicción porque el error de entrenamiento depende de $\mathcal{L}$. De hecho la predicción del error de predicción utilizando el error de entrenamiento está sesgada hacia abajo, especialmente para modelos complejos.

I. Regresión y clasificación

El aprendizaje supervisado, a su vez, puede ser dividido en dos tipos de problemas dependiendo del tipo de variable respuesta. Cuando la variable respuesta es categórica, se dice que es un problema de clasificación, en otro caso, se dice que es un problema de regresión. Un ejemplo de problema de regresión es modelar el precio de una casa, mientras que un ejemplo de problema de clasificación es modelar el género de una persona.

2.1.2. Aprendizaje no supervisado

El aprendizaje no supervisado describe un problema más retador en cuanto a que para cada $i = 1, .., n$, se tiene un vector de observaciones x_i para el cual no se tiene una respuesta asociada y_i. Es por esto que se le llama *no supervisado*, pues no hay una variable respuesta en el análisis. Uno de los principales análisis que se hace en el aprendizaje no supervisado es *análisis de conglomerados*, en el cual a cada una de las observaciones $x_1, ..., x_n$ se clasifica en un grupo. Existen muchos métodos de análisis de conglomerados, sin embargo, no es el objetivo de este trabajo estudiarlas, por lo que no se mencionará más al respecto.

2.1.3. Sobreajuste y regularización

Para explicar el concepto de sobreajuste (*overfitting*, considérese el caso de regresión de una variable Y con el modelo propuesto en 2.1. Si la función f es muy flexible, se tiene que estimar un mayor número de parámetros, lo cual lleva a que parte del error sea capturado por estos parámetros. A este fenómeno de le conoce como sobreajuste, y el problema con él es que un modelo sobreajustado va a tender

a dar malas predicciones con datos no observados previamente. Es por esto que los modelos sobreajustados tienen un error de entrenamiento pequeño, pero el error de prueba es mucho más grande.

Una forma de controlar el problema de sobreajuste en un modelo es con regularización, la cual es una técnica que agrega un término de penalización a la función de error para evitar que los parámetros alcancen valores muy grandes. La forma más simple y de las más utilizadas es la de la suma de los parámetros al cuadrado.

Para ilustrar esto, se toma el caso de regresión lineal múltiple con pérdida cuadrática y penalización cuadrática, en el cual se tiene una función f de la forma $f(X) = X\beta$. Sin regularización, la función de pérdida que se quiere minimizar es

$$\sum_{i=1}^{n} \left(y_i - \sum_{k=0}^{p} (\beta_k x_k) \right)^2, \tag{2.2}$$

pero con el término de penalización de la regularización, 2.2 se modifica para obtener

$$\sum_{i=1}^{n} \left(y_i - \sum_{k=0}^{p} (\beta_k x_k) \right)^2 + \lambda \left(\sum_{k=1}^{p} \beta_k^2 \right), \tag{2.3}$$

donde $\lambda > 0$ es llamado el parámetro de regularización, y controla la importancia relativa entre los dos términos principales de la suma. Este caso particular de regularización en regresión lineal es llamado regresión cordillera (o *ridge*). Nótese que la suma de los cuadrados de los coeficientes puede ser expresado como la norma ℓ_2 al cuadrado del vector de coeficientes $(\beta_1, \beta_2, \ldots, \beta_p)^T$. Al ser una función cuadrática, es diferenciable y se puede encontrar una forma cerrada al problema de minimización.

El valor del parámetro de regularización λ puede tener mucho impacto en la reducción de los parámetros. Si es muy grande, entonces se va a penalizar demasiado y la mayoría de los parámetros estarán cerca de 0; por el contrario, si λ es muy chico, entonces no se tiene el efecto de la regularización y es como si no se penalizara nada. Es común en la práctica elegir distintos valores de λ y probar con un

conjunto de validación cuál de los valores da un menor error.

Otro tipo de regularización involucra la norma ℓ_1 del vector de parámetros y, en el caso de regresión, es llamada regresión LASSO, pero su estudio va más allá del objetivo de este trabajo.

2.2. Optimización de funciones de pérdida

Ya que se han dado a conocer los principales conceptos del aprendizaje estadístico, incluyendo el de función de pérdida y minimización del error, resta conocer cómo encontrar los parámetros que minimizan las funciones de pérdida. En algunos casos, como en el de regresión lineal y regresión *ridge*, se tiene una fórmula cerrada obtenida a partir de las condiciones de primer y segundo orden. Sin embargo, hay muchos otros casos en los que no se puede llegar a una forma cerrada de los parámetros. En estos casos es útil conocer el área de optimización numérica. Las definiciones de esta sección fueron obtenidas de [7], [15], [19] y [21].

El problema general de optimización sin restricciones se expresa de la siguiente forma

$$\min_{\theta \in \mathbb{R}^p} L(\theta). \tag{2.4}$$

En el caso de este trabajo, la función L en 2.4 es una función de pérdida. En muchos casos del aprendizaje estadístico, la función L es convexa (como en regresión lineal), por lo que encontrar un mínimo local asegura que se tiene un mínimo global; pero en muchos otros casos no lo es, y el encontrar un óptimo local no asegura tener un óptimo global, por lo que se tienen que tomar precauciones.

En general, los algoritmos de optimización numérica son iterativos: empiezan con ciertos valores iniciales y en cada iteración van mejorando el valor de la función objetivo hasta que terminan. Cuando han terminado, se espera que hayan obtenido una solución. Una solución es un vector θ^* llamado minimizador local, el cual hace que la función L sea mínima en una vecindad. Formalmente, un vector x^* es un

minimizador local si existe un vecindario $\mathcal{N}$ de θ^* tal que $L(\theta^*) \leq L(\theta)$ para todo $\theta \in \mathcal{N}$.

Es evidente que para saber si un punto θ^* es un minimizador local, no es necesario revisar todos los puntos en su vecindad, sino que se tienen herramientas matemáticas para esto. En particular, se tienen las condiciones suficientes de segundo orden para una función continuamente diferenciable L, definidas a continuación.

Supóngase que la Hessiana $\nabla^2 L$ es continua en una vecindad abierta de θ^*, que el gradiente $\nabla L(\theta^*) = 0$ y que $\nabla^2 L(\theta^*)$ es positiva definida; entonces θ^* es un minimizador local de L.

Este resultado se basa en cálculo elemental, pero provee la base de los algoritmos de optimización numérica. En general, todos los algoritmos buscan un punto θ^* tal que $\nabla L(\theta^*) = 0$.

2.2.1. Descenso en gradiente

El descenso en gradiente pertenece a un tipo de algoritmos llamados de **búsqueda de línea**. Estos algoritmos buscan en cada iteración una dirección y actualizan el valor actual de acuerdo a esa dirección. Es decir, en la k-ésima iteración, se tiene un valor θ_k, y se busca una dirección p_k para actualizar al valor $\theta_{k+1} = \theta_k + \alpha_k p_k$, donde $\alpha_k > 0$ es la 'distancia' que se recorre en la dirección p_k, y es llamada la **longitud de paso**. Una vez que se actualizó el valor de la iteración, se encuentra una nueva dirección de avance y se actualiza el valor. Esto se hace hasta que se cumpla cierto criterio de paro, siendo este usualmente que la norma del gradiente sea menor a un escalar pequeño y positivo.

En el caso del descenso en gradiente, la dirección de avance p_k es la del máximo descenso, es decir, el negativo del gradiente en la iteración actual $-\nabla L(\theta_k)$, por lo que en cada iteración se hace

$$\theta_{k+1} = \theta_k - \alpha_k \nabla L(\theta_k). \tag{2.5}$$

Al calcular la longitud de paso α_k se tiene un problema: se quiere un valor tal que la

función objetivo L se reduzca lo más posible, pero tampoco se quiere desperdiciar mucho tiempo escogiendo el valor. La mejor opción es el minimizador global de la función $\phi(\alpha_k) = L(\theta_k + \alpha_k p_k)$, pero es muy caro de calcular. Generalmente se utilizan heurísticas que consisten en elegir una sucesión de valores para α_k y probar cuál de todos satisface ciertas condiciones.

Una de estas condiciones es llamada la condición de Armijo, y estipula que α_k debe generar un descenso suficiente en la función objetivo L, medido como

$$L(\theta_k + \alpha_k p_k) \leq L(\theta_k) + c_1 \alpha_k \nabla L(\theta_k)^T p_k,$$

para alguna constante $c_1 \in (0, 1)$. Usualmente c_1 es pequeño, como 10^{-4}. Esta condición puede no ser suficiente, pues valores muy pequeños de α_k pueden hacer que se cumpla, y longitudes de paso muy pequeñas no son deseables. Una forma de arreglar esto es usar *backtracking* en el algoritmo. Este consiste en elegir un valor relativamente grande de α_k (algunos métodos inician con $\alpha_k = 1$), y se inicia un subalgoritmo iterativo el cual reduce el valor de α_k hasta que se cumple la condición de Armijo.

Hay muchas otras formas de escoger la longitud de paso α_k, las cuales van más allá del objetivo de este trabajo. Si se quisiera estudiar más a fondo este tema, se puede consultar [19] para más detalles.

Para explicar un poco mejor el concepto de descenso en gradiente, se presentan dos ejemplos, uno para regresión lineal y otro para regresión logística. Ambos ejemplos fueron implementados en el lenguaje para cómputo estadístico `R` [20].

En el caso de regresión lineal, en el caso general, se busca minimizar la función de pérdida cuadrática

$$L(x, \beta) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_{i1} - \ldots \beta_p x_{ip})^2 \tag{2.6}$$

Tomando las derivadas parciales de 2.6, se tiene que para cada $j \in \{0, \ldots, p\}$

$$\frac{\partial L}{\partial \beta_j} = -\frac{2}{n}\sum_{i=1}^{n}\left(x_{ij}\ell_i(x,\beta)\right) \tag{2.7}$$

donde $x_{i1} = 1$ para toda $i \in \{1,\ldots,n\}$ y

$$\ell_i(x,\beta) = \left(y_i - \beta_0 - \beta_1 x_{i1} - \ldots \beta_p x_{ip}\right). \tag{2.8}$$

De aquí, se llega a que la dirección de descenso en gradiente en cada iteración es

$$\nabla_\beta L(x,\beta) = \left(\frac{\partial L}{\partial \beta_0},\ldots,\frac{\partial L}{\partial \beta_p}\right)^T.$$

En el ejemplo implementado, se generó un vector $x \in \mathbb{R}^n$ con $n = 1000$ tal que $x_i \sim N(0,1)$ para cada $i \in \{1,\ldots,n\}$ y se construyó la variable respuesta $y = \beta_0 + \beta_1 x + \varepsilon$ con $\varepsilon \sim N(0,1)$, $\beta_0 = 2$ y $\beta_1 = 1$. Los datos generados se pueden ver en la figura 2.1.

Entonces, para minimizar la pérdida cuadrática 2.6, se empieza con un vector $\beta^0 \in \mathbb{R}^2$, y en cada iteración, se actualiza

$$\beta^{k+1} = \beta^k - \alpha_k \nabla_\beta L(x,\beta)$$

hasta un criterio de paro. En este caso, el criterio de paro era que la norma del gradiente $\nabla_\beta L(x,\beta)$ fuera menor a 0.000001 o que se excediera de 100 iteraciones. El vector de parámetros inicial fue $\beta^0 = (0,0)^T$.

Los valores del vector de parámetros en cada iteración se pueden ver en la figura 2.2. El punto grande que se ve en la figura es el valor real de los parámetros (i.e. 2 y 1), y el tache que se ve es el valor obtenido utilizando el paquete `lm` de `R`. Se puede ver que el algoritmo implementado converge a estos valores. De hecho, el valor obtenido con el método implementado es exactamente igual al que regresa `lm`.

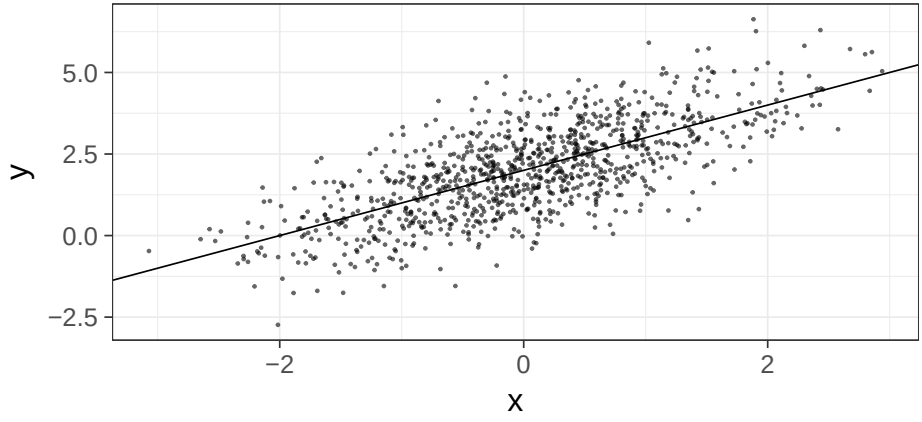

Figura 2.1: Datos generados para una regresión lineal simple.

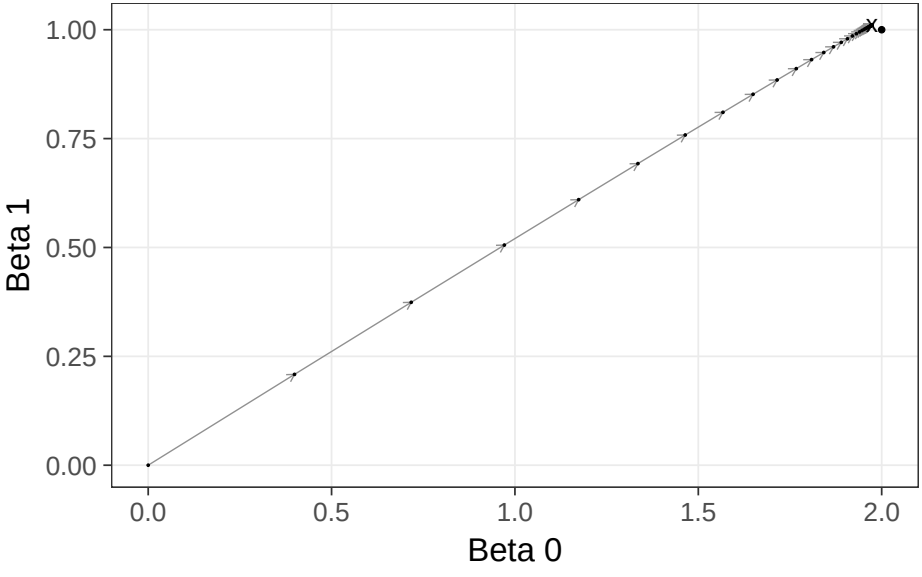

Figura 2.2: Iteraciones del algoritmo de descenso en gradiente para regresión lineal.

En el caso de regresión logística, en el caso general, se busca minimizar la función

de pérdida llamada **devianza**, definida como

$$L(x,\beta) = -\frac{2}{n}\sum_{i=1}^{n}\left[y_i \log(h(\beta^T x_i)) + (1-y_i)\log(1-h(\beta^T x_i))\right] = -\frac{2}{n}\sum_{i=1}^{n}\ell_i(\beta) \tag{2.9}$$

donde

$$\beta^T x_i = \sum_{j=0}^{p}\beta_j x_{ij},$$

$$h(w) = \frac{e^w}{1+e^w}$$

y

$$\ell_i(x) = y_i \log(h(\beta^T x_i)) + (1-y_i)\log(1-h(\beta^T x_i)).$$

Se tiene que

$$\frac{\partial L}{\partial \beta_j} = -\frac{2}{n}\sum_{i=1}^{n}\frac{\partial \ell_i}{\partial \beta_j}$$

y además, usando el hecho de que $h'(w) = h(w)(1-h(w))$, se tiene

$$\begin{aligned}
\frac{\partial \ell_i}{\partial \beta_j} &= \frac{y_i h'(\beta^T x_i)x_{ij}}{h(\beta^T x_i)} + \frac{(1-y_i)(-1)h'(\beta^T x_i)x_{ij}}{1-h(\beta^T x_i)} \\
&= \frac{h'(\beta^T x_i)x_{ij}y_i}{h(\beta^T x_i)} - \frac{(1-y_i)h'(\beta^T x_i)x_{ij}}{1-h((\beta^T x_i))} \\
&= h'(\beta^T x_i)x_{ij}\left(\frac{y_i}{h(\beta^T x_i)} - \frac{1-y_i}{1-h(\beta^T x_i)}\right) \\
&= h'(\beta^T x_i)x_{ij}\left(\frac{y_i - y_i h(\beta^T x_i) - h(\beta^T x_i) + y_i h(\beta^T x_i)}{h(\beta^T x_i)(1-h(\beta^T x_i))}\right) \\
&= x_{ij}(y_i - h(\beta^T x_i)).
\end{aligned} \tag{2.10}$$

Por lo tanto, se tiene que

$$\frac{\partial L}{\partial \beta_j} = -\frac{2}{n}\sum_{i=1}^{n} x_{ij}(y_i - h(\sum_{j=0}^{p}\beta_j x_{ij})),$$

donde nuevamente $x_{i1} = 1$ para toda $i \in \{1, \ldots, n\}$.

En el ejemplo implementado, se generó un vector $x \in \mathbb{R}^n$ con $n = 1000$ tal que $x_i \sim N(0, 1)$ para cada $i \in \{1, \ldots, n\}$ y se construyó la variable auxiliar $p_i = \frac{1}{\exp(-\beta_0 - \beta_1 x_i)}$ con $\beta_0 = 1$ y $\beta_1 = 4$. Finalmente, la variable respuesta y se construyó simulando una variables aleatorias Bernoulli, de tal forma que $y_i \sim Bern(p_i)$. Los datos generados se pueden ver en la figura 2.3.

Entonces, para minimizar la devianza 2.9, exactamente igual que en el otro caso, se empieza con un vector $\beta^0 \in \mathbb{R}^2$, y en cada iteración, se actualiza

$$\beta^{k+1} = \beta^k - \alpha_k \nabla_\beta L(x, \beta)$$

hasta un criterio de paro. En este caso, el criterio de paro fue un poco más estricto: la norma del gradiente $\nabla_\beta L(x, \beta)$ debía ser menor a 0.0001, o el cociente de normas entre cada iteración fuera mayor a 0.8, o que se excediera de 500 iteraciones. El vector de parámetros inicial fue $\beta^0 = (-10, 10)^T$.

Los valores del vector de parámetros en cada iteración se pueden ver en la figura 2.4. El punto grande que se ve en la figura es el valor real de los parámetros (i.e. 1 y 4), y el tache que se ve es el valor obtenido utilizando el paquete `glm` de `R`. Se puede ver que el algoritmo implementado converge a estos valores y, nuevamente, el valor obtenido con el método implementado es exactamente igual al que regresa el paquete utilizado (`glm`).

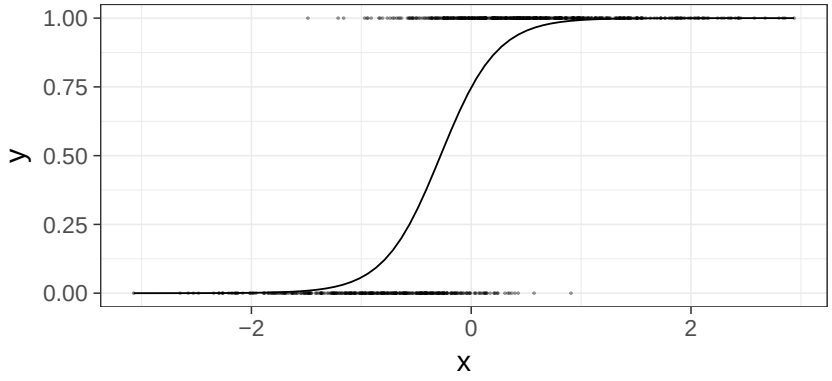

Figura 2.3: Datos generados para una regresión logística.

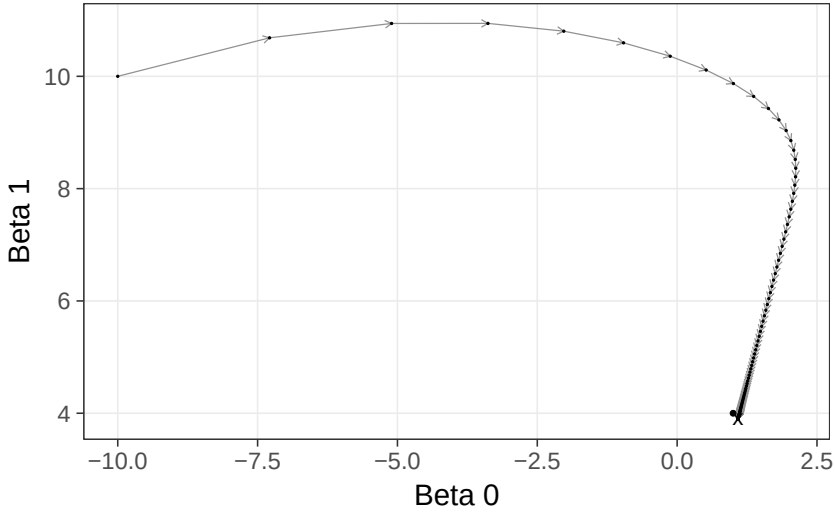

Figura 2.4: Iteraciones del algoritmo de descenso en gradiente para regresión logística.

2.2.2. Descenso en gradiente estocástico

En el aprendizaje estadístico es muy común encontrar problemas de optimización de la forma

$$\min_{\theta \in \mathbb{R}^p} L(x, \theta), \quad \text{donde } L(\theta) = \frac{1}{n} \sum_{i=1}^{n} \psi_i(x, \theta). \tag{2.11}$$

En los ejemplos de la subsección 2.2.1 se puede ver esto con claridad: las funciones de pérdida a minimizar son de la forma 2.11. El método de descenso en gradiente presentado en 2.2.1 utiliza iteraciones de la forma de la ecuación 2.5, es decir,

$$\theta_{k+1} = \theta_k - \alpha_k \nabla L(\theta_k) := \theta_k - \frac{\alpha_k}{n} \sum_{i=1}^{n} \nabla \psi_i(\theta_k),$$

lo cual involucra la evaluación de n gradientes para después promediarlos. En los casos de aprendizaje automático de gran escala, el número de observaciones n es grande, por lo que calcular estos gradientes puede resultar costoso. Debido a esto, surgen los métodos como descenso en gradiente estocástico, en los cuales el número de gradientes a evaluar no depende de n, sino que es constante. Este método utiliza iteraciones de la forma

$$\theta_{k+1} = \theta_k - \frac{\alpha_k}{n} \nabla \psi_{i_k}(\theta_k),$$

donde $i_k \in \{1, 2, \ldots, n\}$ es escogido aleatoriamente. El gradiente $\nabla \psi_{i_k}(\theta_k)$ es un estimador insesgado de $\nabla L(\theta_k)$. De esta forma, cada iteración es muy barata pues involucra la evaluación de solamente un gradiente. Puede suceder que alguna $\nabla \psi_{i_k}(\theta_k)$ particular no otorgue una dirección de descenso partiendo de θ_k, pero, intuitivamente, en promedio sí se dan direcciones de descenso, de tal forma que le sucesión $\{\theta_0, \theta_1, \ldots\}$ puede ser guiada a un minimizador θ^*.

Para ilustrar este método, se implementaron los dos mismos ejemplos que en la sección 2.2.1, es decir, la regresión lineal y la regresión logística.

En el caso de la regresión lineal, se desea minimizar la pérdida cuadrática definida

en la ecuación 2.6. Aquí, si se utiliza la notación utilizada en la ecuación 2.11, cada $\psi_i(x, \theta)$ es

$$\psi_i(x, \theta) = (y_i - \beta_0 - \beta_1 x_{i1} - \dots \beta_p x_{ip})^2 = \ell_i^2(x, \beta),$$

con $\ell_i(x, \beta)$ definido en la ecuación 2.8. Se puede ver que la derivada parcial de $\psi_i(x, \beta)$ respecto a cada β_j con $j \in \{0, \dots, p\}$ es

$$\frac{\partial \psi_i}{\partial \beta_j} = \frac{\partial \ell_i^2}{\partial \beta_j} = -2x_{ij}\ell_i(x, \beta),$$

por lo que la dirección de avance en cada iteración del algoritmo es

$$\nabla_\beta \psi_i(x, \beta) = \left(\frac{\partial \psi_i}{\partial \beta_0}, \dots, \frac{\partial \psi_i}{\partial \beta_p} \right).$$

Nuevamente, el vector de parámetros iniciales fue $\beta^0 = (0, 0)^T$. El criterio de paro era que la norma del gradiente $\nabla_\beta L(x, \beta)$ fuera menor a 0.000001, que se excediera de 300 iteraciones o que la norma dos al cuadrado de las diferencias del vector de parámetros entre una iteración y otra (i.e. $\left\|\beta^{k+1} - \beta^k\right\|_2^2$) fuera menor a 10^{-15}.

Los valores del vector de parámetros en cada iteración se pueden ver en las figuras 2.5 y 2.6. Para entender mejor estas imágenes, es pertinente explicar qué es un *epoch*. Un *epoch* es un conjunto de n accesos al conjunto de datos. Es decir, en cada *epoch*, se evalúan los gradientes de todos los datos en el conjunto de entrenamiento.

En la figura 2.5 se muestran los valores de los parámetros para todos los gradientes en todos los *epochs*. En la figura 2.6 se muestran solo los valores al final de cada *epoch* para tener mayor claridad en la visualización. Se puede apreciar en la figura 2.5 que, a diferencia de la figura 2.2 donde las direcciones de descenso se ven más uniformes hacia una misma dirección, las direcciones van hacia todos lados en una especie de *zig-zag* que eventualmente llega a la solución.

En ambas figuras se muestra un punto grande que representa el valor real de los parámetros ($\beta_0 = 2$ y $\beta_1 = 1$) y el tache que representa el valor obtenido utilizando el paquete `lm` de `R`, pero en la figura 2.5 no se alcanza a distinguir tan bien, pero

en ambas imágenes se puede ver que el algoritmo implementado converge a los valores deseados.

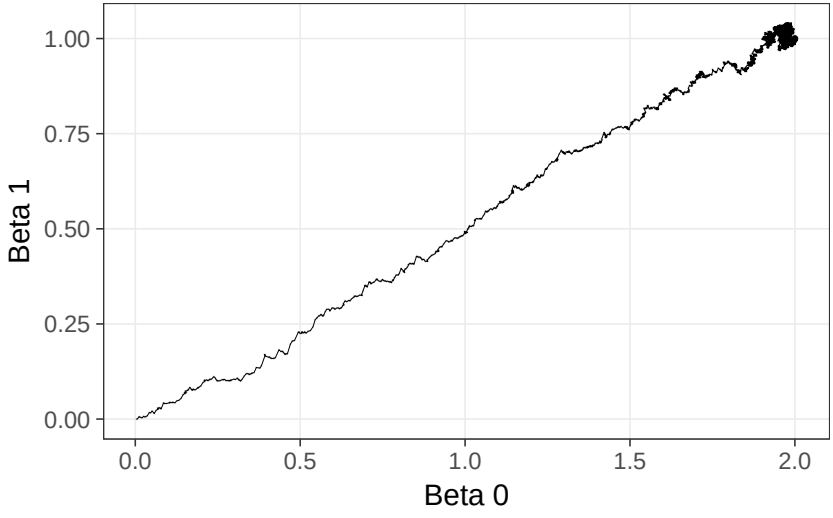

Figura 2.5: Iteraciones del algoritmo de descenso en gradiente estocástico para regresión lineal.

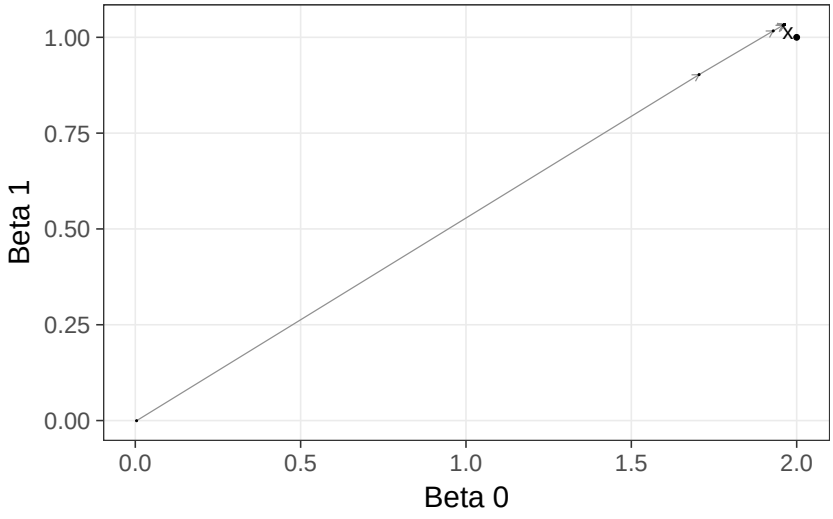

Figura 2.6: Iteraciones del algoritmo de descenso en gradiente para regresión lineal. Cada flecha apunta al valor de los parámetros después de cada *epoch*.

Para la regresión logística ya se tienen hechos casi todos los cálculos necesarios en la subsección 2.2.1. Con la notación utilizada en la ecuación 2.11, cada $\psi_i(x, \theta)$ es

$$\psi_i(x, \theta) = y_i \log(h(\beta^T x_i)) + (1 - y_i) \log(1 - h(\beta^T x_i)) = \ell_i(x, \beta).$$

Además, en las ecuaciones 2.10 se llega a que

$$\frac{\partial \ell_i}{\partial \beta_j} = \frac{\partial \psi_i}{\partial \beta_j} = x_{ij}(y_i - h(\beta^T x_i)).$$

Por lo que ya se tiene todo lo necesario. En la implementación, el criterio de paro fue el mismo que en descenso en gradiente estocástico para regresión lineal: que la norma del gradiente $\nabla_\beta L(x, \beta)$ fuera menor a 0.000001, que se excediera de 300 iteraciones o que la norma dos al cuadrado de las diferencias del vector de parámetros entre una iteración y otra fuera menor a 10^{-15}.

Los valores del vector de parámetros en cada iteración se pueden ver en las figuras

2.7 y 2.8. Las explicaciones de estas dos imágenes son análogas a las de regresión lineal. Nuevamente, el valor obtenido con el método implementado converge al valor real y al que se obtuvo con el paquete `glm`.

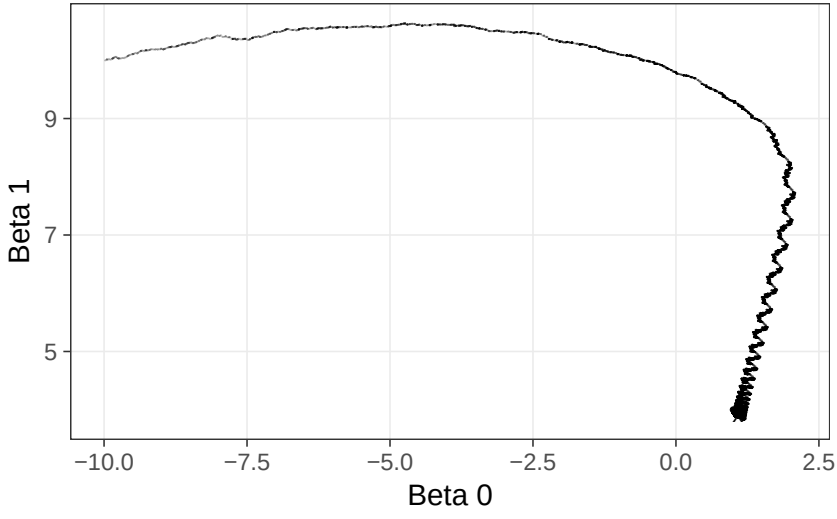

Figura 2.7: Iteraciones del algoritmo de descenso en gradiente para regresión logística.

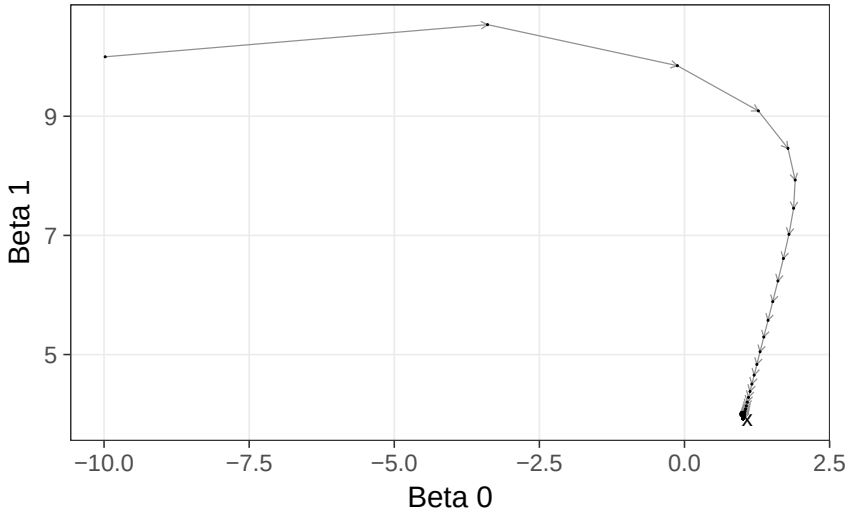

Figura 2.8: Iteraciones del algoritmo de descenso en gradiente para regresión logística. Cada flecha apunta al valor de los parámetros después de cada *epoch*.

2.3. Sistemas de recomendación

Las definiciones de esta sección fueron obtenidas de [12], [16] y [17].

El objetivo de los sistemas de recomendación es predecir las respuestas de los usuarios a distintas opciones. Dos clasificaciones muy amplias de los sistemas de recomendación son: basados en contenido y filtrado colaborativo. Existen dos grandes tipos de sistemas de recomendación de acuerdo a las técnicas que usan.

- Basados en contenido: A partir de características y propiedades de los productos (por ejemplo, género, actores, país de origen, año, etc.) intentamos predecir el gusto por el producto construyendo variables derivadas del contenido de los artículos (como qué actores salen, año, etc.).
- Colaborativos: A partir de usuarios y productos se construyen medidas de similitud en el sentido de que les han gustado los mismos productos o que

les gustaron a las mismas personas. Así, los productos recomendados a un usuario son los que le gustaron a otros usuarios.

En este trabajo se supone que se tiene una matriz de calificaciones R, tal que las filas representan usuarios que calificaron a los productos representados en las columnas de la siguiente forma:

	P_1	P_2	P_3	$\cdots$	P_n
U_1	1	2	3	$\cdots$	-
U_2	4	-	4	$\cdots$	-
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
U_m	-	-	1	$\cdots$	3

donde cada P_i es el producto i-ésimo que se ofrece y U_i es el usuario i-ésimo que ha calificado el producto correspondiente. Los espacios donde hay guiones representan datos faltantes. Es usual que este tipo de matrices tengan la mayor parte de las entradas faltantes, pues no todos los usuarios han visto todas las películas, o calificado todos los productos. El objetivo del sistema de recomendación es llenar los espacios faltantes mediante una predicción y recomendar los productos que tienen una predicción alta. Notar que aquí se considera que ya se tiene la matriz de calificaciones, lo cual a veces, en la práctica, puede ser un trabajo difícil y haya que pedir a usuarios que califiquen productos.

2.3.1. Basados en contenido

Para este tipo de recomendaciones se construye un **perfil** para cada producto. Como se había mencionado, este perfil puede contener el género, los actores, país de origen, etc. Esto puede ser más sencillo en el caso de películas pues existen fuentes de información sobre las películas como *Internet Movie Database (IMDb)*; pero con documentos o imágenes, por ejemplo, no siempre se tiene esta información, así que es necesario la construcción de variables a través de *tags*, palabras clave y, en el caso de texto, tal vez un modelo de lenguaje.

Una vez construidos los perfiles, el proceso que queda es de alguna forma emparejar los gustos del usuario obtenidos de la matriz de utilidad con los perfiles de los productos.

Un ejemplo muy sencillo es el siguiente. Supóngase dos películas con 8 actores en total (A_1, ..., A_8), cinco actores cada una y que dos actores están en las dos películas, y que además las calificaciones promedio de cada película son 3 y 4. Entonces los perfiles de dos películas se ven como:

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	Calif
P_1	0	1	1	0	1	1	0	1	3
P_2	1	1	0	1	0	1	1	0	4

Con esta información se puede obtener una medida de similitud entre las películas. Una medida muy utilizada es la similitud coseno, definida como

$$sim(P_1, P_2) = \frac{P_1^T P_2}{\|P_1\| \, \|P_2\|}.$$

Supóngase ahora que los usuarios U_k y U_j solo han calificado dos películas. Si el usuario U_k calificó con 2 y 4 las películas P_1 y P_2 respectivamente y si el usuario U_j calificó con 4 y 4 las películas P_1 y P_2 respectivamente, entonces se puede ver esto en forma matricial:

	P_1	P_2
U_k	2	4
U_j	4	4

Tiene sentido centrar las calificaciones por usuario (restarle la media) para reducir un poco la heterogeneidad de la escala, entonces se tiene:

	P_1	P_2
U_k	-1	1
U_j	0	0

Así, para el usuario U_k se tiene que los perfiles para cada película se ven como:

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	Calif
P_1	0	-1	-1	0	-1	-1	0	-1	2
P_2	1	1	0	1	0	1	1	0	4

Y para el usuario U_j se ven como:

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	Calif
P_1	0	4	4	0	4	4	0	4	4
P_2	4	4	0	4	0	4	4	0	4

Entonces, para el perfil general de cada usuario, se promedian por película los perfiles de cada uno y se obtiene

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	Calif
U_k	0.5	0	-0.5	0.5	-0.5	0	0.5	-0.5	3
U_j	2	4	2	2	2	4	2	2	4

Con esto se puede estimar qué tanto le va a gustar a un usuario una película calculando la similitud coseno entre el perfil del usuario y el perfil de la película. Esta forma es muy rudimentaria y sencilla, pero puede servir como base para otros modelos.

2.3.2. Colaborativos

I. Modelo base

En esta sección se construye un modelo base muy sencillo para hacer predicciones, y el cual sirve como comparación con otros modelos más complejos. Para este modelo se supone que los artículos y los usuarios tienen sesgos, o sea, hay usuarios que tienden a calificar más alto, o a ser más estrictos y calificar más bajo; al mismo tiempo, puede haber artículos que tiendan a recibir calificaciones más altas; y que gran parte de la calificación observada se debe a efectos asociados a estos sesgos.

Sea μ la media general de todos los artículos y todos los usuarios, entonces la predicción del modelo base es

$$\hat{r_{ij}} = \mu + a_i + b_j$$

donde a_i es la desviación del usuario i respecto a la media general y b_j es la desviación del artículo j respecto a la media general.

Una forma de estimar los sesgos es calculando

$$a_i = \frac{1}{M_i} \sum_t r_{it} - \mu,$$

y

$$b_j = \frac{1}{N_j} \sum_s r_{sj} - \mu,$$

donde donde M_i es el número de artículos calificados por el usuario i y N_j es el número de calificaciones que tiene el artículo j.

Otra forma un poco más precisa es resolver el problema de mínimos cuadrados regularizado

$$\min_b \sum_{(i,j)\in A} \left(r_{ij} - \mu - a_i - b_j\right)^2 + \lambda \left(\sum_i a_i^2 + \sum_j b_j^2 \right)$$

donde r_{ij} es la calificación del usuario i del artículo j, A es el conjunto de usuarios y artículos para los cuales se conoce la calificación r_{ij}, $|A|$ es la cardinalidad de A (o sea, el total de artículos calificados por todos los usuarios) y λ es un término de regularización que evita el sobreajuste.

A este sencillo modelo se le pueden agregar otros términos de regularización que ayuden a disminuir el ruido causado por artículos y usuarios con pocas calificaciones, al encoger las calificaciones a la media. Una vez que se calcularon a_i y b_j usando cualquiera de los métodos mencionados, se hace la predicción

$$r_{ij} = \mu + \frac{M_i}{\gamma + M_i} a_i + \frac{N_j}{\gamma + N_j} b_j \tag{2.12}$$

donde M_i es el número de artículos que ha calificado el usuario i, N_j es el número de calificaciones que tiene el artículo j, y γ es el parámetro de regularización.

II. Métodos de vecindario

Este tipo de modelos se centran en la similitud de los usuarios y los artículos. Los usuarios son similares si los vectores que los representan (es decir, las filas en la matriz de calificaciones) están cercanos de acuerdo a alguna similitud definida. Una recomendación para un usuario i se hace al ver a los usuarios más parecidos a i y recomendando artículos que hayan sido del agrado de estos usuarios parecidos.

El primer problema con estos métodos surge al definir una medida de similitud entre artículos o usuarios. Dos medidas muy utilizadas son la similitud de Jaccard y, nuevamente, la similitud coseno. En el caso de una matriz de calificaciones binaria, la similitud Jaccard tendría más sentido, pero si son calificaciones explícitas conviene más utilizar similitud coseno. Una desventaja de utilizar la similitud coseno es que los elementos faltantes son tratados como 0, lo cual tiene el efecto de tratar la falta de calificación como disgusto hacia el artículo. Una forma de solucionar esto es centrando las calificaciones por usuario al restar la media, de esta forma el faltante al ser tratado como cero tiene el efecto de que al usuario ni le gusta ni le disgusta el artículo.

Una forma de hacer predicciones de calificaciones para un usuario i y un artículo j es encontrar los un vecindario de los n usuarios más parecidos a i y tomar el promedio de las calificaciones del artículo j. O de forma dual, se podría encontrar un vecindario con los n artículos más parecidos a j y tomar el promedio de las calificaciones que el usuario i ha dado a esos artículos. Aquí se hace una vertiente en los métodos de vecindario, dependiendo de si se centran en las similitudes de los usuarios o de los artículos.

Basados en usuarios

Este tipo de algoritmos pretenden imitar las recomendaciones de boca en boca al asumir que los usuarios con preferencias similares calificarán los artículos de forma

similar. De esta manera, para hacer una predicción para el usuario i y el artículo j habría que encontrar un vecindario de i, denotado como $N(i)$, y tomar el promedio de las calificaciones en el vecindario:

$$\hat{r_{ij}} = \frac{\sum_{a \in N(i)} r_{aj}}{|N(i)|}. \tag{2.13}$$

La ecuación 2.13 está ignorando el hecho de que algunos usuarios en el vecindario son más similares a otros, por lo que es una buena idea tener un ponderador de similitud. Así, en lugar de 2.13, la predicción es

$$\hat{r_{ij}} = \frac{\sum_{a \in N(i)} s_{ia} r_{aj}}{\sum_{a \in N(i)} s_{ia}}, \tag{2.14}$$

donde s_{ia} es la similitud entre el usuario i y el usuario a perteneciente al vecindario de i.

Basados en artículos

Este tipo de algoritmos asumen que los usuarios prefieren artículos que son similares a otros artículos que les gustaron. Muchas veces este tipo de sistemas producen información más confiable porque es más sencillo encontrar artículos del mismo género que encontrar usuarios a los que solo les gustan artículos de un solo género. En este caso, para hacer una predicción para el usuario i y el artículo j habría que encontrar un vecindario de j, denotado como $N(j)$, y tomar el promedio (ponderado por nivel de similitud para mejor estimación) de las calificaciones que el usuario ha hecho a los artículos en $N(j)$, es decir

$$\hat{r_{ij}} = \frac{\sum_{a \in N(j)} s_{ja} r_{ia}}{\sum_{a \in N(j)} s_{ja}}, \tag{2.15}$$

donde s_{ja} es la similitud entre el artículo j y el artículo a perteneciente al vecindario de j.

III. Métodos de factorización de matrices

Modelo básico de factorización de matrices

La idea de la factorización de matrices proviene de los modelos de factores latentes, en los cuales se supone que hay factores ocultos o latentes que caracterizan a los usuarios y a los productos. Por ejemplo, se podría suponer que las películas se pueden resumir en dos factores: una dimensión de seriedad-comedia y una dimensión de fantasía-realidad, ambas numéricas, de tal forma que en la primera dimensión un valor más alto significa que una película es más seria y más bajo significa que es más inclinada a comedia. Para la segunda dimensión, similarmente, un valor más alto significa que la película está más inclinada a fantasía y un menor valor significa que está más apegada a la realidad. Estos factores latentes también se pueden pensar para los usuarios, en el sentido de que un usuario tiene un valor más alto en la primera dimensión si le gustan más las películas serias y un valor más alto en la segunda dimensión si le gustan las películas de fantasía.

Teniendo esto en cuenta, los usuarios y las películas se pueden representar en vectores. El vector correspondiente a un usuario que le gustan mucho las películas serias de fantasía se podría ver así:

$$u_1 = \begin{bmatrix} 3 & 4 \end{bmatrix}^T.$$

Y un usuario que le gustan las comedias pero es indiferente entre si es de fantasía o realidad se podría ver así:

$$u_2 = \begin{bmatrix} -2 & 0 \end{bmatrix}^T.$$

De la misma forma se podría pensar en dos películas, una seria con fantasía, como *El Señor de los Anillos*, y una apegada a la realidad de comedia, como *Mean Girls*. Sus vectores correspondientes se podrían ver así:

$$p_1 = \begin{bmatrix} 2 & 4 \end{bmatrix}^T$$

$$p_2 = \begin{bmatrix} -3 & -3 \end{bmatrix}^T.$$

Entonces una forma de modelar la calificación que daría cada usuario a cada película sería pensando en la combinación lineal de los usuarios y las películas. Es decir, la calificación que daría el usuario u_1 a la película p_1 sería el producto punto de los vectores: $u_1^T p_1 = 22$. Si se hace esto para todos los usuarios y películas, se tiene el siguiente sistema lineal:

$$R = UP^T,$$

donde

$$U = \begin{bmatrix} u_1^T \\ u_2^T \end{bmatrix} = \begin{bmatrix} 3 & 4 \\ -2 & 0 \end{bmatrix}, \quad P = \begin{bmatrix} p_1^T \\ p_2^T \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ -3 & -3 \end{bmatrix}$$

y

$$R = \begin{bmatrix} u_1^T p_1 & u_1^T p_2 \\ u_2^T p_1 & u_2^T p_2 \end{bmatrix} = \begin{bmatrix} 22 & -21 \\ -4 & 6. \end{bmatrix}$$

Aquí se puede ver que el usuario u_1, a quien le gustan las películas serias de fantasía califica más alto *El Señor de los Anillos* que *Mean Girls*, mientras que el usuario u_2 hace lo contrario, pues le gustan las comedias.

Ahora, en la vida real no se conocen los valores de las matrices U y P, sino que se tiene acceso a ciertas entradas de la matriz R, y el reto está en tener una buena estimación de las matrices U y P. Este problema es una instancia de la descomposición en valores singulares, con la peculiaridad de que no se tienen todas las entradas de la matriz R, sino solo una pequeña proporción de ellas.

Entonces, si se tiene acceso a la matriz R, se busca encontrar dos matrices U y P de rango k tales que $R \approx UP^T$. De esta forma, la aproximación de la calificación que da el usuario i al artículo j sería

$$r_{ij} = u_i^T p_j. \tag{2.16}$$

Hay distintas formas de elegir qué función de pérdida utilizar para saber qué tan cerca el producto de matrices está de R, pero muchas veces se utiliza la raíz del error cuadrático medio (RMSE por *root-mean-squared-error*), el cual se calcula de la siguiente forma:

$$RMSE = \left(\frac{1}{|A|} \sum_{(i,j)\in A} (r_{ij} - u_i^T p_j)^2 \right)^{\frac{1}{2}}, \tag{2.17}$$

donde r_{ij} es la calificación del usuario i del artículo j, A es el conjunto de usuarios y artículos para los cuales se conoce la calificación r_{ij}, $|A|$ es la cardinalidad de A (o sea, el total de artículos calificados por todos los usuarios), u_i es la i-ésima fila de la matriz U y p_j es la j-ésima fila de la matriz P. Es decir, el RMSE es la raíz cuadrada del promedio de las desviaciones de la aproximación de R a la R verdadera.

Otra opción muy utilizada es el error absoluto medio (MAE por *mean-absolute-error*), definido como la suma de las desviaciones absolutas de la aproximación de R a R, esto es

$$MAE = \frac{1}{|A|} \sum_{(i,j)\in A} |r_{ij} - u_i^T p_j|.$$

El RMSE puede ser preferible en situaciones en las que pequeños errores de predicción no son importantes, pues el RMSE penaliza errores más fuertemente que el MAE.

Por lo tanto, cuando se quiere minimizar el RMSE, el problema a resolver es

$$\min_{p,u} \sum_{(i,j)\in A} \left[(r_{ij} - u_i^T p_j)^2 + \lambda(\|p_j\|^2 + \|u_i\|^2) \right] \tag{2.18}$$

donde λ es un término de regularización que evita el sobreajuste.

Agregando sesgos

Al modelo simple enunciado en la sección anterior se le puede agregar la idea del modelo base: que existen sesgos en los artículos y en los usuarios. Entonces al modelo 2.16 se le agregan los sesgos, de tal forma que se tiene el nuevo modelo con sesgos

$$r_{ij} = \mu + a_i + b_j + u_i^T p_j, \tag{2.19}$$

con μ, a_i y b_j definidos anteriormente. Estos parámetros se pueden estimar resolviendo el problema

$$\min_{p,u,b,a} \sum_{(i,j)\in A} \left[\left(r_{ij} - \mu - a_i - b_j - u_i^T p_j \right)^2 + \lambda \left(\|p_j\|^2 + \|u_i\|^2 + a_i^2 + b_j^2 \right) \right] \tag{2.20}$$

Modelo de optimización

Sea L la función a optimizar en el problema 2.20, es decir,

$$\begin{aligned} L &= \sum_{(i,j)\in A} \left[\left(r_{ij} - \mu - a_i - b_j - u_i^T p_j \right)^2 + \lambda \left(\|p_j\|^2 + \|u_i\|^2 + a_i^2 + b_j^2 \right) \right] \\ &= \sum_{(i,j)\in A} \left[\left(x_{ij} - a_i - b_j - u_i^T p_j \right)^2 + \lambda \left(\|p_j\|^2 + \|u_i\|^2 + a_i^2 + b_j^2 \right) \right] \\ &= \sum_{(i,j)\in A} \left(x_{ij} - a_i - b_j - u_i^T p_j \right)^2 \\ &+ \lambda \left(\sum_j \sum_{m=1}^{k} p_{jm}^2 + \sum_i \sum_{m=1}^{k} u_{im}^2 + \sum_i a_i^2 + \sum_j b_j^2 \right). \end{aligned}$$

Como ya se había mencionado en al subsección 2.2.2, el descenso en gradiente estocástico utiliza únicamente una observación para calcular la dirección de descenso, por lo que se define la función de pérdida ℓ_{ij} como

$$\ell_{ij} = \left(x_{ij} - a_i - b_j - u_i^T p_j\right)^2 + \lambda \left(\sum_{m=1}^{k} p_{jm}^2 + \sum_{m=1}^{k} u_{im}^2 + a_i^2 + b_j^2 \right).$$

Notar que $L = \sum_{(i,j)\in A)} \ell_{ij}$.

Las derivadas parciales de ℓ_{ij} son

$$\frac{\partial \ell_{ij}}{\partial u_{im}} = -2e_{ij}p_{jm} + 2\lambda u_{im} \qquad \frac{\partial \ell_{ij}}{\partial p_{jm}} = -2e_{ij}u_{im} + 2\lambda v_{jm}$$

$$\frac{\partial \ell_{ij}}{\partial a_i} = -2e_{ij} + 2\lambda a_i \qquad \frac{\partial \ell_{ij}}{\partial b_j} = -2e_{ij} + 2\lambda b_j$$

donde $e_{ij} = x_{ij} - a_i - b_j - u_i^T p_j$ es el residual de la calificación del usuario i del producto j.

Entonces, para resolver 2.20, se inicia con un unas matrices P y Q, y los vectores a_i y b_j para todos los usuarios i y artículos j, con valores arbitrarios. Y se prosigue a ajustar U, P, a y b iterativamente usando la dirección contraria al gradiente en cada punto hasta cierto criterio de paro.

Algoritmo 1: Algoritmo de descenso en gradiente estocástico para 2.20

Iniciar vectores a, b y matrices P y Q
mientras *no se cumpla el criterio de paro* **hacer**
  para todo $(i,j) \in A$ **hacer**
    para todo $m \in 1, \ldots, k$ **hacer**
      $u_{im} \leftarrow u_{im} - \gamma\left(-2e_{ij}p_{jm} + 2\lambda u_{im}\right)$
      $p_{jm} \leftarrow p_{jm} - \gamma\left(-2e_{ij}u_{im} + 2\lambda v_{jm}\right)$
    fin
    $a_i \leftarrow a_i - \gamma\left(-2e_{ij} + 2\lambda a_i\right)$
    $b_j \leftarrow b_j - \gamma\left(-2e_{ij} + 2\lambda b_j\right)$
  fin
fin

2.3.3. Evaluación de modelos

Una primera medida para evaluar un sistema de recomendación que ya fue presentada anteriormente es el RMSE. Esta medida da una idea de el desempeño del modelo para todos los artículos ya conocidos, sin embargo, no es suficiente para ofrecer recomendaciones que le vayan a gustar al usuario. La mayoría de las veces se busca recomendar cierto número de artículos a un usuario, digamos N, y se desea que estos N artículos sean de agrado del usuario, o a veces los usuarios están interesados en descubrir cosas nuevas. Es por esto que también se debe de evaluar cómo los sistemas se desempeñan en este tipo de situaciones.

Una forma de evaluar este tipo de recomendaciones es tomar esto como un problema de clasificación binario, en el cual se tiene un acierto si a un usuario se le recomendó un artículo en los mejores N (top-N) y este usuario lo vio/compró, y no se acierta en otro caso. Con este tipo de problemas se pueden obtener las medidas típicas en un problema de clasificación binario (precisión, tasa de verdaderos positivos, tasa de falsos negativos, etc).

En [8] se muestra una forma de evaluar de esta forma un sistema de recomendación, la cual consiste en seleccionar cada elemento i calificado con la calificación máxima por cada usuario u en el conjunto de prueba:

1. Seleccionar 1000 artículos adicionales que no fueron calificados por el usuario u. Se asume que la mayoría de ellos no van a ser de interés para el usuario u.

2. Calcular la calificación de acuerdo al modelo para el artículo i y cada uno de los 1000 artículos adicionales.

3. Se forma una lista ordenada de acuerdo a las calificaciones predichas por el modelo para los 1001 artículos. Sea p el lugar que ocupa el artículo de prueba i entre todos los elementos de la lista. El mejor resultado corresponde al caso en que el artículo de prueba i está antes de todos los artículos aleatorios (esto es, $p = 1$).

4. Se forma una lista de las mejores N recomendaciones al seleccionar los N artículos mejor calificados en la lista. Si $p \leq N$ entonces se tiene un acierto, en otro caso se tiene un desacierto. La probabilidad de obtener aciertos aumenta si N aumenta, y si $N = 1001$, siempre se tiene un acierto.

Después de hacer eso, se puede calcular la precisión y el *recall*. Estos están definidos como:

$$\text{recall}(N) = \frac{\#hits}{|T|}$$

$$\text{prec}(N) = \frac{\#hits}{N \cdot |T|} = \frac{\text{recall}(N)}{N}$$

donde $|T|$ es el número de calificaciones en el conjunto de prueba.

Capítulo 3

Resultados

Los algoritmos presentados en este trabajo fueron implementados en el lenguaje para cómputo estadístico `R` [20] y fueron probados con dos conjuntos de datos: *MovieLens*, una página que ofrece recomendaciones de películas [13]; y *BookCrossing*, una plataforma similar, pero enfocada a libros [25]. Ambos conjuntos de datos están abiertos al público. Algunas de las funciones utilizadas fueron implementadas en el lenguaje `C++` y compiladas en `R` con el paquete `Rcpp` [9] [10]. Las gráficas fueron hechas con `ggplot2` [23] y la manipulación de los datos fue mayoritariamente con los paquetes incluidos en `tidyverse` [24].

3.0.1. Análisis exploratorio de datos

I. *MovieLens*

El conjunto de datos de *MovieLens* consiste en 20,000,263 calificaciones de 26,744 películas hechas por 138,493 usuarios. Todos los usuarios habían calificado al menos 20 películas. Las calificaciones están en una escala de 0.5 a 5, con intervalos de medio punto, es decir, se puede poner $0.5, 1, \ldots, 4.5, 5$ como calificación. La distribución de las calificaciones se puede ver en la figura 3.1; mientras que en la figura 3.2 se puede ver el histograma de los promedios por película.

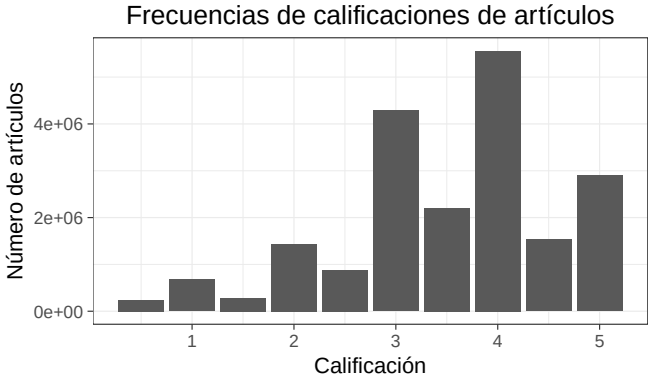

Figura 3.1: Frecuencia de calificaciones del conjunto de datos *MovieLens*.

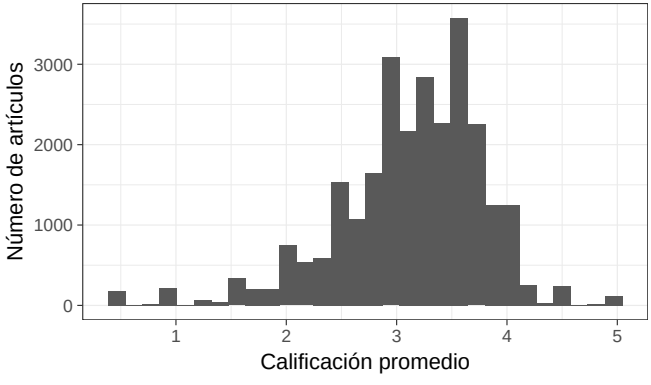

Figura 3.2: Histograma del promedio de calificaciones por película del conjunto de datos *MovieLens*.

En el capítulo 1 se introduce el concepto de cola larga. En la figura 3.3 se puede ver cómo este fenómeno ocurre en los datos de *MovieLens*, donde es claro que muy pocas películas tienen una gran cantidad de calificaciones, mientras que muchos artículos tienen pocas calificaciones.

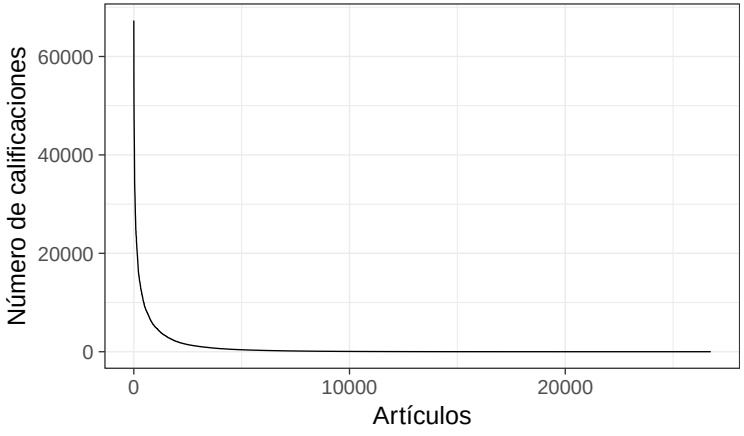

Figura 3.3: Cola larga del conjunto de datos *MovieLens*.

En el conjunto de datos de *MovieLens*, se tiene disponible la fecha en que se hizo la calificación. Como se mencionó antes, esta temporalidad podría ser utilizada en el modelo, pero en este trabajo se decidió no usarlo. En la figura 3.4 se muestra la calificación promedio de todas las películas en cada semana. Se puede ver que la calificación promedio no ha cambiado drásticamente, sin embargo se aprecia que tienen forma de U, siendo el mínimo cerca de 2005.

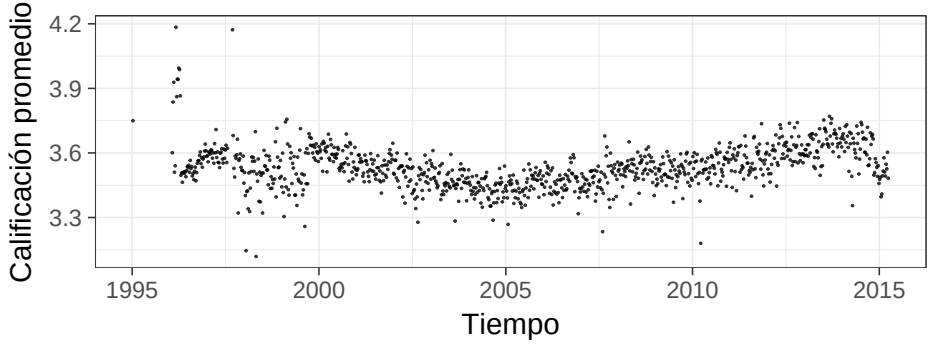

Figura 3.4: Calificación promedio semanal de todas las películas del conjunto de datos *MovieLens*.

II. *BookCrossing*

El conjunto de datos de *BookCrossing* consiste en 1,149,780 calificaciones de 271,379 libros hechas por 278,858 usuarios. Los datos se obtuvieron durante cuatro semanas de agosto a septiembre de 2004. Las calificaciones están en una escala de 1 a 10, con intervalos de un punto, es decir, se puede poner $1, 2, \ldots, 9, 10$ como calificación. Se tiene como calificación especial el 0, el cual es solamente una calificación implícita, es decir, si leyó o no el libro. Debido a que para usar el método presentado en este trabajo se necesitan calificaciones explícitas, se filtraron estas del conjunto de datos original. Con esta operación, quedaron 383,852 calificaciones de 153,683 libros hechas por 68,092 usuarios. La distribución de las calificaciones filtradas se puede ver en la figura 3.5; mientras que en la figura 3.6 se puede ver el histograma de los promedios por libro.

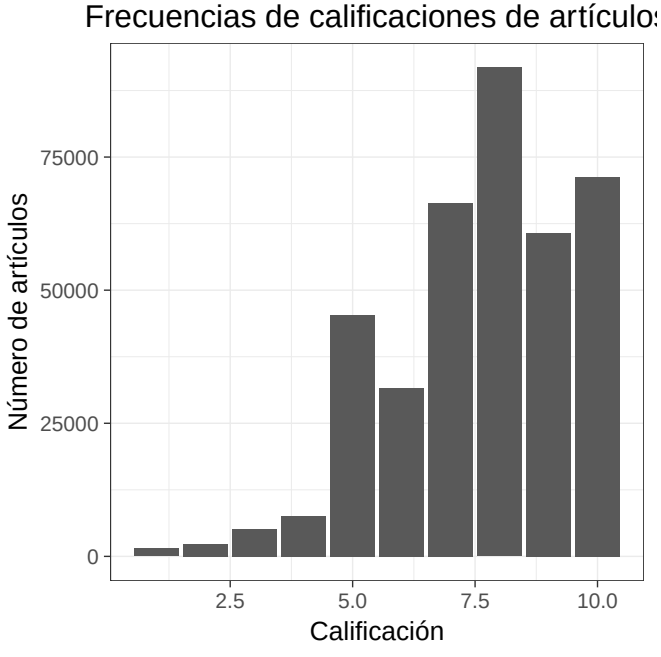

Figura 3.5: Frecuencia de calificaciones del conjunto de datos *BookCrossing*.

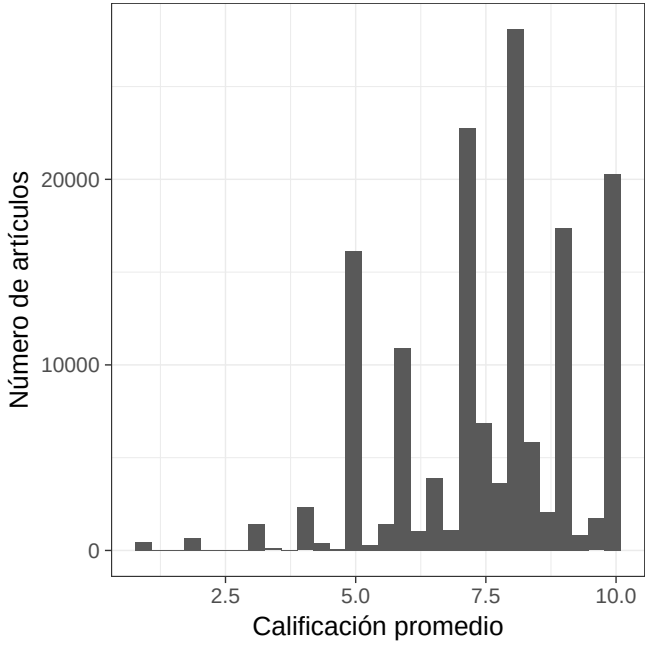

Figura 3.6: Histograma del promedio de calificaciones por libro del conjunto de datos *BookCrossing*.

En la figura 3.7 se puede ver la cola larga en los datos de *BookCrossing*, donde nuevamente es claro que muy pocas películas tienen una gran cantidad de calificaciones, mientras que muchos artículos tienen pocas calificaciones.

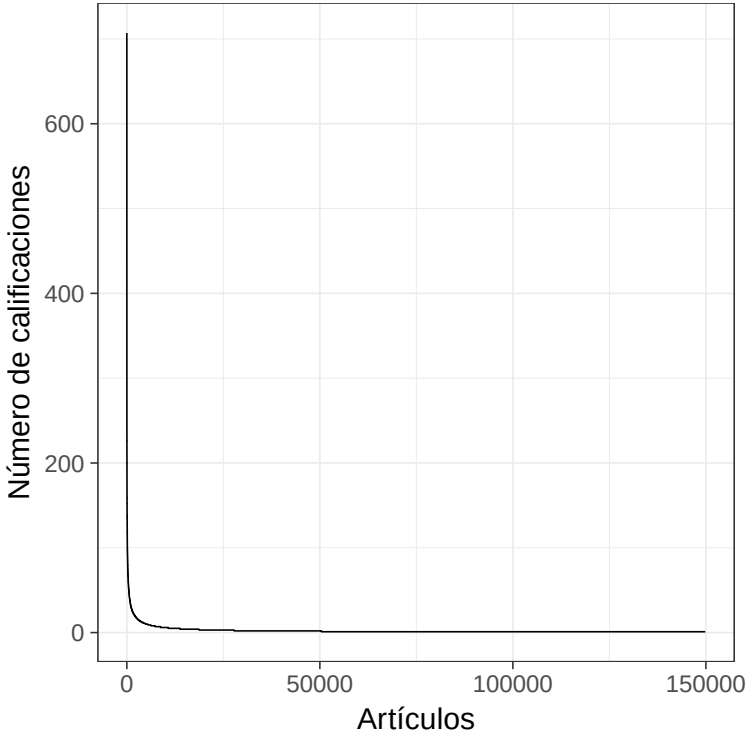

Figura 3.7: Cola larga del conjunto de datos *BookCrossing*.

3.0.2. Comparación de modelos

Para cada conjunto de datos se calculó un modelo base como el descrito en la sección 2.3.2.I en la ecuación 2.12 y un modelo de factorización como descrito en la sección 2.3.2.III. Cada uno de los conjuntos de datos se dividió en tres subconjuntos: un conjunto de entrenamiento, un conjunto de prueba y un conjunto de validación. Los parámetros de los modelos fueron estimados usando el conjunto de entrenamiento. En el caso del modelo de factorización, el conjunto de validación fue utilizado para estimar el error de predicción y poder utilizarlo como criterio de paro en el algoritmo de optimización y así evitar el sobreajuste. Para el modelo base, el conjunto de validación no fue utilizado, los errores que se muestran en

relación con este conjunto son con el conjunto de prueba. La medida de error utilizada fue la raíz del error cuadrático medio (RMSE), definido en la ecuación 2.17. De aquí en adelante, cuando se hable de error, es con referencia a la raíz del error cuadrático medio.

I. *MovieLens*

Para *MovieLens*, el número de usuarios, artículos y calificaciones en cada uno de los conjuntos se puede ver en la tabla 3.1.

Tabla 3.1: Número de usuarios, calificaciones y artículos en los conjuntos de *MovieLens*.

Conjunto	Número de artículos	Número de usuarios	Número de calificaciones
Entrenamiento	26,247	138,493	18,029,206
Validación	6,256	41,483	1,469,158
Prueba	2,895	20,676	501,899

En la figura 3.8 se pueden ver los errores del modelo base en el conjunto de prueba, de acuerdo al valor del parámetro de regularización γ, definido en la ecuación 2.12. Se puede ver que con $\gamma \approx 19$ se minimiza el valor de la raíz del error cuadrático medio en el conjunto de prueba, siendo este de aproximadamente 0.8707.

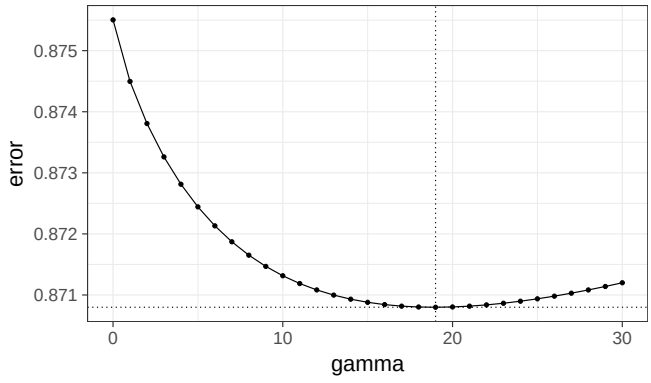

Figura 3.8: Errores del modelo base en el conjunto de prueba de *MovieLens*, de acuerdo al valor del parámetro de regularización γ.

En la figura 3.9 se pueden ver los errores de entrenamiento y de validación del modelo de factorización con distintos valores en el número de dimensiones latentes, tasa de aprendizaje y parámetro de regularización λ. Como es de esperarse, el error de entrenamiento es menor en cada uno de los casos. También se podría esperar que mientras más dimensiones latentes haya, el error de entrenamiento sea menor, debido a que hay más dimensiones para aproximar la matriz que se quiere aproximar; y es justo lo que sucede: los modelos con más dimensiones latentes tienen menor error de entrenamiento. Sin embargo, se sabe que menor error de entrenamiento no implica necesariamente menor error de validación; por lo que se busca el modelo con menor error de validación. En este caso, todos los modelos tienen un error de validación relativamente parecido, siendo el de 500 dimensiones latentes, tasa de aprendizaje de 0.001 y $\lambda = 0.01$ el modelo con menor error (0.763325), siguiendo el de 200 dimensiones latentes, tasa de aprendizaje de 0.001 y $\lambda = 0.01$ con un error de 0.765166.

Aquí se podría tomar una decisión en cuanto a qué modelo utilizar: el modelo con 500 dimensiones latentes tiene menor error, pero tiene más del doble de dimensiones latentes que el de 200, el cual tiene un desempeño muy parecido, por lo que tal vez se preferiría utilizar el modelo con 200 dimensiones latentes debido a que

es más simple y tiene casi el mismo desempeño. El modelo que se utilizó como final fue el de 200 dimensiones latentes, con un error en el conjunto de prueba de 0.7651675 (muy parecido al del conjunto de validación). Se hace notar que en todos los casos, el algoritmo de factorización presenta un error menor que el modelo base.

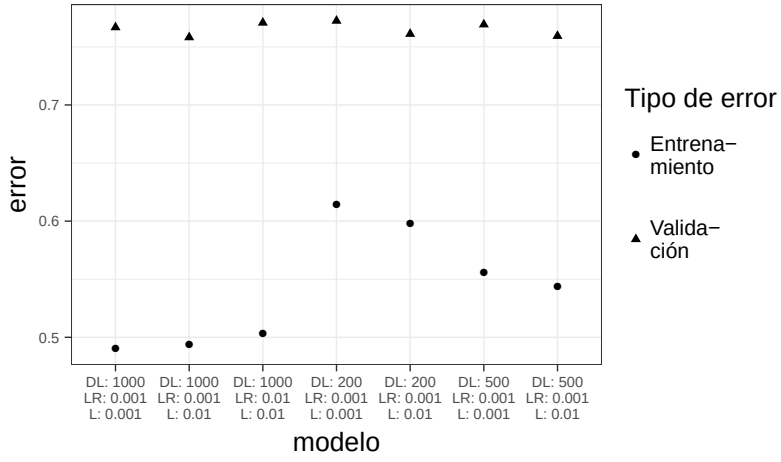

Figura 3.9: Errores del modelo de factorización en el conjunto de prueba de *MovieLens*. LR es la tasa de aprendizaje, DL es el número de dimensiones latentes y L es el parámetro de regularización λ.

En la figura 3.10 se pueden ver los errores de entrenamiento y de validación del modelo de factorización final (i.e., 200 dimensiones latentes, tasa de aprendizaje de 0.001 y $\lambda = 0.01$) en cada iteración del algoritmo de descenso en gradiente estocástico. Se puede ver cómo ambos errores disminuyen con cada iteración, sin embargo, el error de entrenamiento disminuye mucho más rápido. Si no se hubiera tenido como criterio de paro el error de validación, posiblemente el error de entrenamiento hubiera continuado disminuyendo, pero el de validación hubiera empezado a aumentar, debido al sobreajuste.

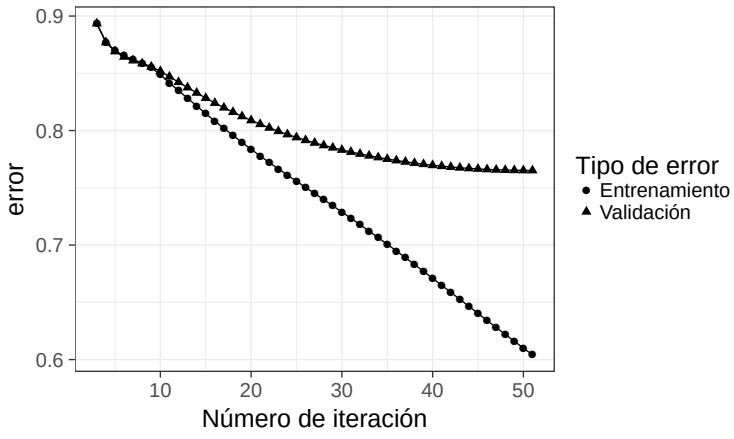

Figura 3.10: Errores de entrenamiento y validación del modelo de factorización en cada iteración del algoritmo de optimización para *MovieLens*.

Como se menciona en 2.3.3, además del RMSE, una forma de evaluar un sistema de recomendación es como un problema de clasificación con la tarea de recomendar los mejores N artículos. En las figuras 3.11 y 3.12 se puede ver el resultado de esta tarea comparando el modelo base y el modelo de factorización como función de N, el número de artículos recomendados. Se puede observar que el modelo de factorización es superior que el modelo base para todas las N para las que se calcularon las medidas. Por ejemplo, para $N = 10$, el recall del modelo de factorización es de 0.429, lo cual significa que el modelo tiene una probabilidad de 0.429 de poner en el top-10 de recomendaciones una película que atraiga a un usuario; mientras que el recall del modelo base en $N = 10$ es de 0.059, lo cual significa que la probabilidad mencionada es de 0.059.

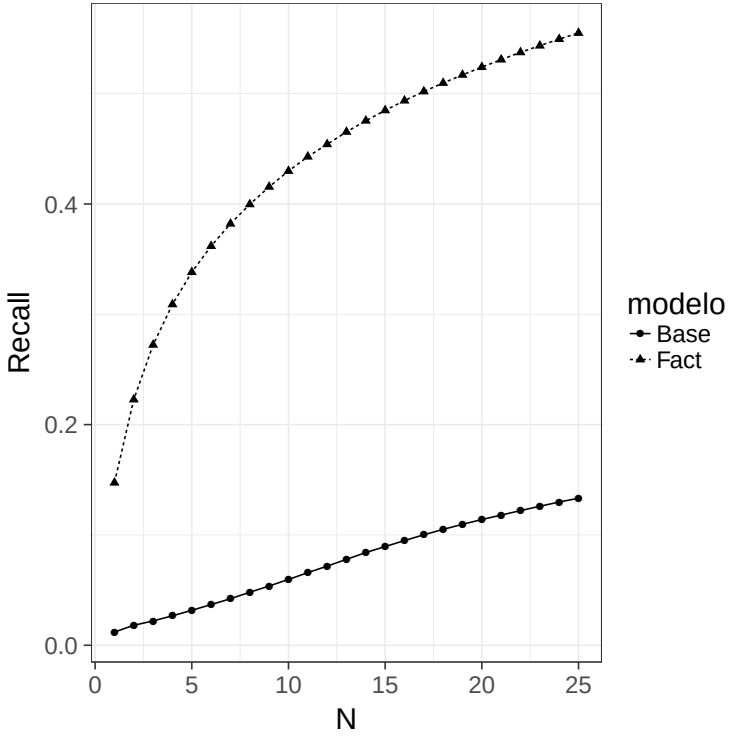

Figura 3.11: Recall para cada valor de N en la tarea de las mejores N recomendaciones para *MovieLens*.

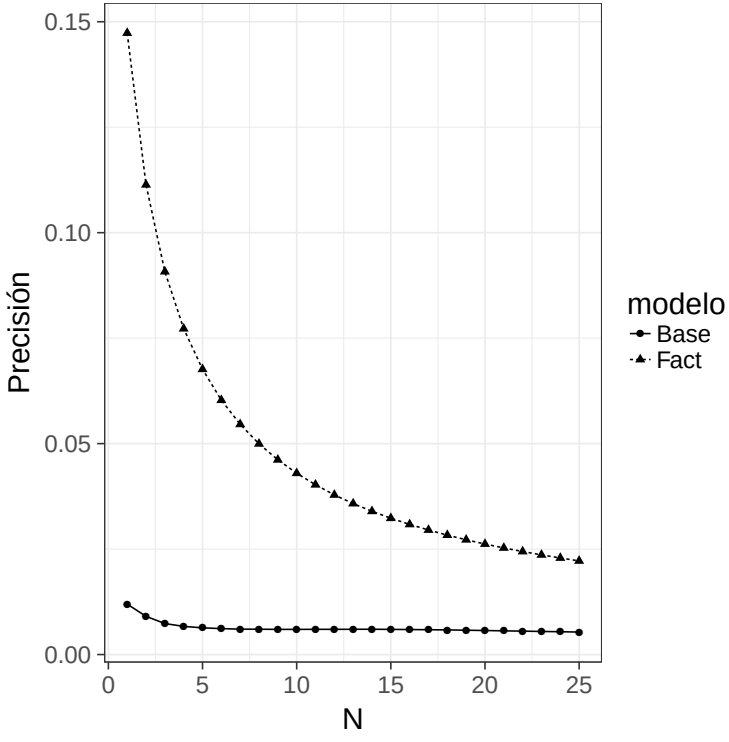

Figura 3.12: Precisión para cada valor de N en la tarea de las mejores N recomendaciones para *MovieLens*.

Como se mencionó antes, cada artículo es expresado como una combinación lineal de distintos factores latentes, por lo que artículos y usuarios son mapeados a un mismo espacio latente. Por ende, cada artículo tiene un vector representante en un espacio euclidiano, por lo que se pueden calcular distancias y encontrar vecinos con estas distancias. Con esta idea, además de recomendar directamente a un usuario artículos personalizados, se pueden encontrar los artículos más parecidos a un determinado artículo mediante la norma euclidiana de la diferencia o la similitud coseno.

Para el conjunto de datos de *MovieLens* se calculan los vecinos más cercanos para

algunas películas selectas. Se utilizó la distancia euclidiana y se calculó con el paquete `RANN` [2] de `R`. Estos vecinos se pueden ver en las tablas 3.2 y 3.3, y se puede apreciar que en cierta forma, los vecinos más cercanos de cada película sí son parecidos.

Tabla 3.2: Algunas películas y sus vecinos más cercanos.

Película	**Película más cercana**	**Distancia**
Aladdin (1992)	Lion King, The (1994)	1.74
	Beauty and the Beast (1991)	1.79
	Little Mermaid, The (1989)	2.09
	Princess and the Frog, The (2009)	2.33
	Tarzan (1999)	2.42
	Mulan (1998)	2.43
	Oliver & Company (1988)	2.47
	Great Mouse Detective, The (1986)	2.47
	How to Train Your Dragon 2 (2014)	2.49
	Anastasia (1997)	2.49
Hannibal (2001)	Red Dragon (2002)	2.58
	Hitcher, The (2007)	2.88
	Venom (2005)	2.89
	Quiet, The (2005)	2.89
	Hannibal Rising (2007)	2.89
	Mindhunters (2004)	2.9
	187 (One Eight Seven) (1997)	2.91
	Raven, The (2012)	2.91
	Taking Lives (2004)	2.93
	Don't Say a Word (2001)	2.93
Harry Potter and the Sorcerer's Stone (2001)	HP and the Chamber of Secrets (2002)	0.47
	HP and the Prisoner of Azkaban (2004)	1.11
	HP and the Goblet of Fire (2005)	1.13
	HP and the Order of the Phoenix (2007)	1.33
	HP and the Half-Blood Prince (2009)	1.59
	HP and the Deathly Hallows: Part 2 (2011)	1.83
	HP and the Deathly Hallows: Part 1 (2010)	1.87
	Narnia: The Voyage of the Dawn Treader (2010)	2.76
	Narnia: Prince Caspian (2008)	2.88
	The Lion King 1 1/2 (2004)	2.91

Tabla 3.3: Algunas películas y sus vecinos más cercanos.

Película	**Película más cercana**	**Distancia**
Lion King, The (1994)	Aladdin (1992)	1.74
	Beauty and the Beast (1991)	2.1
	Mulan (1998)	2.4
	Little Mermaid, The (1989)	2.42
	Tarzan (1999)	2.44
	Oliver & Company (1988)	2.44
	Princess and the Frog, The (2009)	2.48
	Frozen (2013)	2.49
	Land Before Time, The (1988)	2.49
	How to Train Your Dragon 2 (2014)	2.55
Pulp Fiction (1994)	Reservoir Dogs (1992)	2.26
	Inglorious Bastards (1978)	3.03
	Goodfellas (1990)	3.15
	An Evening with Kevin Smith 2 (2006)	3.19
	Black Mirror (2011)	3.2
	Ricky Gervais Live 3: Fame (2007)	3.23
	Nightcrawler (2014)	3.24
	Inside Llewyn Davis (2013)	3.25
	Flirting (1991)	3.25
	Generation Kill (2008)	3.25
Toy Story (1995)	Toy Story 2 (1999)	1.43
	Monsters, Inc. (2001)	1.91
	Toy Story 3 (2010)	2
	Bug's Life, A (1998)	2.02
	Finding Nemo (2003)	2.03
	Incredibles, The (2004)	2.26
	Up (2009)	2.47
	For the Birds (2000)	2.5
	Wreck-It Ralph (2012)	2.5
	Partly Cloudy (2009)	2.5

II. *BookCrossing*

En esta subsección se presentan los mismos resultados en el mismo orden que en la sección anterior para *BookCrossing*. El número de usuarios, artículos y calificaciones en cada uno de los conjuntos se puede ver en la tabla 3.4.

Tabla 3.4: Número de usuarios, calificaciones y artículos en los conjuntos de *BookCrossing*.

Conjunto	Número de artículos	Número de usuarios	Número de calificaciones
Entrenamiento	140,807	64,459	351,217
Validación	13,072	5,332	21,224
Prueba	5,065	2,286	7,435

En la figura 3.13 se pueden ver los errores del modelo base en el conjunto de prueba, de acuerdo al valor del parámetro de regularización γ, definido en la ecuación 2.12. Se puede ver que con $\gamma \approx 6$ se minimiza el valor de la raíz del error cuadrático medio en el conjunto de prueba, siendo este de aproximadamente 1.7022.

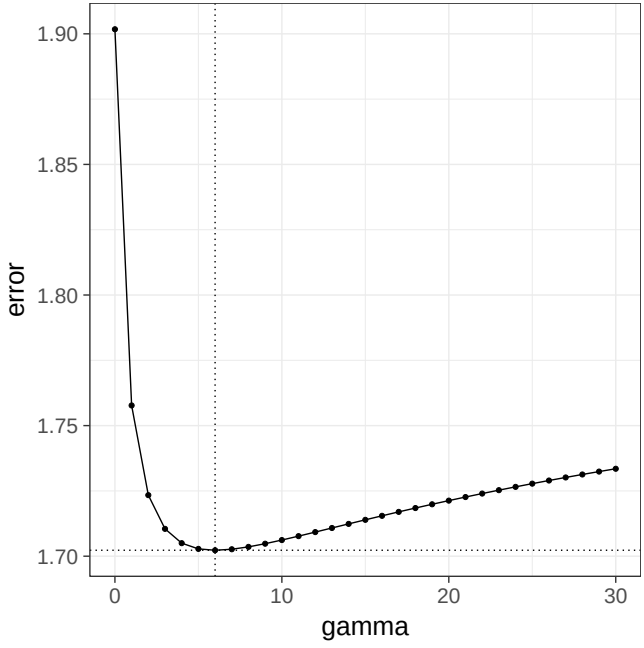

Figura 3.13: Errores del modelo base en el conjunto de prueba de *BookCrossing*, de acuerdo al valor del parámetro de regularización γ.

En el caso de *BookCrossing*, el modelo de factorización utilizado y para el cual se presentan resultados es, al igual que con *MovieLens*, uno con 200 dimensiones latentes, tasa de aprendizaje de 0.001 y $\lambda = 0.01$. Este modelo presentó con un error en el conjunto de prueba de 1.586566. Nuevamente, este error es menor al error del modelo base.

En la figura 3.14 se puede ver, como en la subsección anterior, los errores de entrenamiento y de validación del modelo de factorización final en cada iteración del algoritmo de optimización.

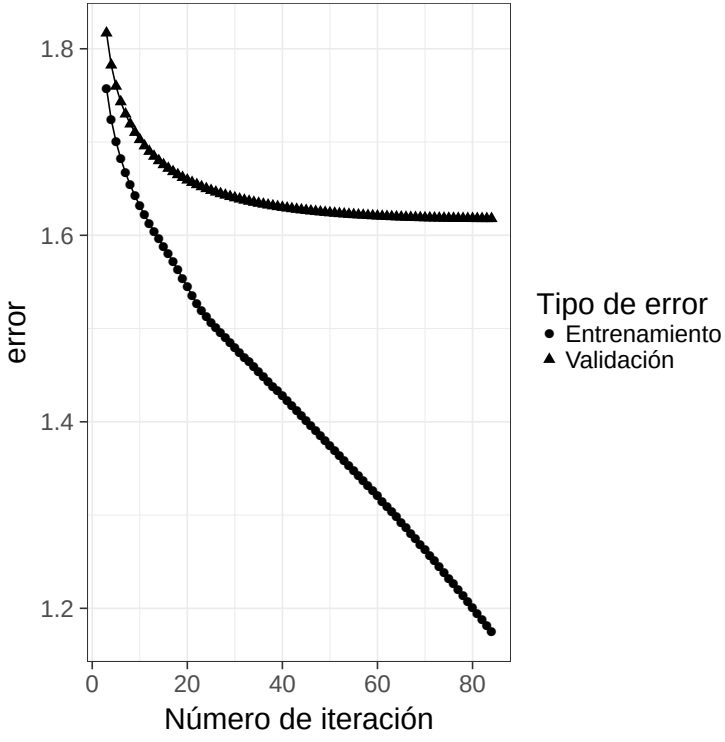

Figura 3.14: Errores de entrenamiento y validación del modelo de factorización en cada iteración del algoritmo de optimización para *BookCrossing*.

En las figuras 3.15 y 3.16 se puede ver el desempeño del modelo base y del modelo de factorización como problema de clasificación y como función de N (el número de artículos recomendados). Nuevamente, el modelo de factorización es superior que el modelo base para todas las N para las que se calcularon las medidas. Si se compara con el desempeño del conjunto de datos de *MovieLens*, los resultados son mucho peores en este caso. Por ejemplo, para $N = 10$, el recall del modelo de factorización de *MovieLens* es de 0.429, mientras que el del modelo de factorización de *BookCrossing* es de 0.1144. Aún así, en la tarea de recomendar artículos, el algoritmo de factorización supera al modelo base. En el caso de $N = 10$, el modelo base presenta un recall de aproximadamente 0.008, menos de una décima parte

que el de factorización.

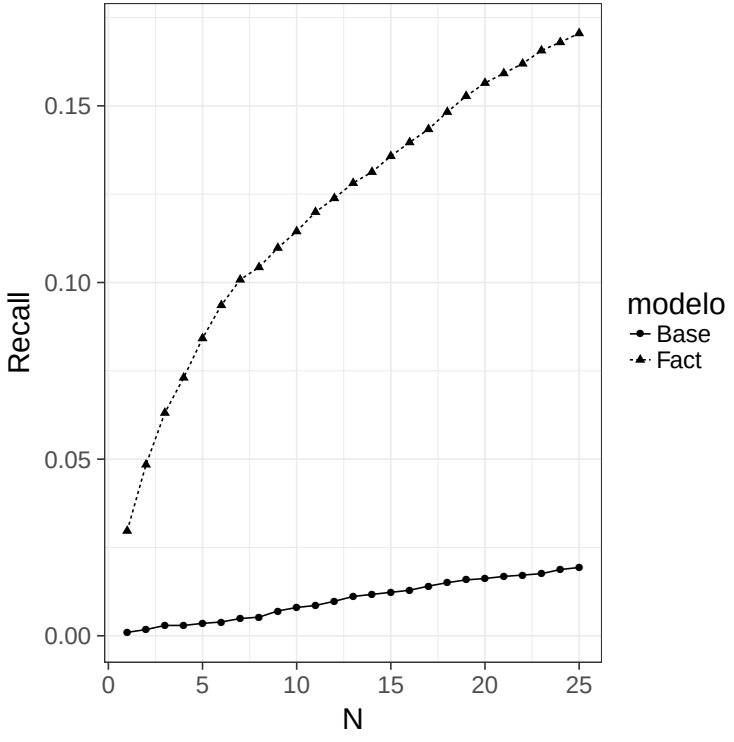

Figura 3.15: Recall para cada valor de N en la tarea de las mejores N recomendaciones para *BookCrossing*.

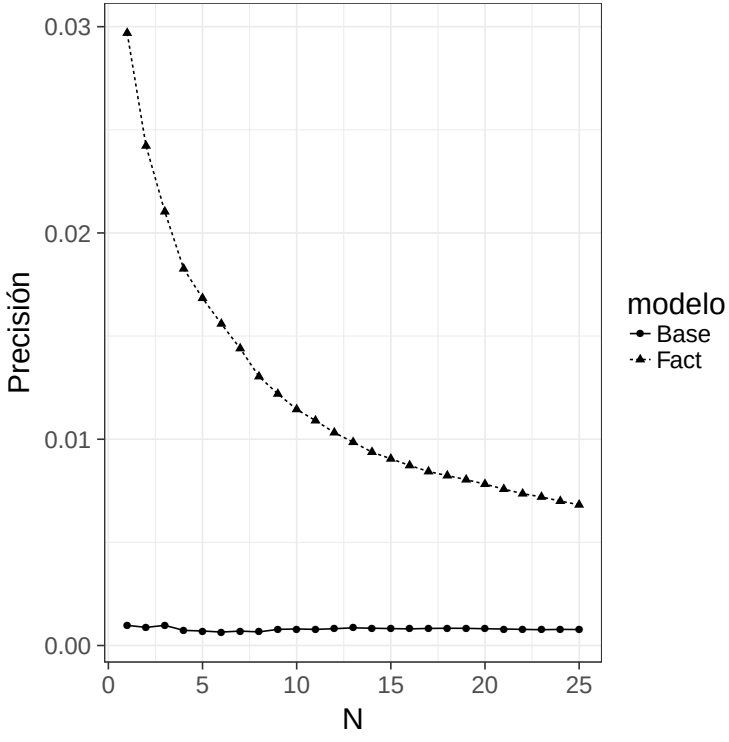

Figura 3.16: Precisión para cada valor de N en la tarea de las mejores N recomendaciones para *BookCrossing*.

Para el conjunto de datos de *BookCrossing* también se calculan los vecinos más cercanos para algunos libros mediante distancia euclidiana y con el paquete `RANN`. Los resultados se pueden ver en la tabla 3.5. Nótese que en este caso, aunque sí se puede apreciar similitud, no es tanta como en el caso de *MovieLens*. Esto se debe a que el conjunto de datos de *BookCrossing* es mucho más ralo.

Tabla 3.5: Algunos libros y sus vecinos más cercanos.

Libro	**Libro más cercano**	**Distancia**
Harry Potter and the Sorcerer's Stone	Who in Hell Is Wanda Fuca?	1.71
	A Ranking of the Most Influential Persons in History	1.75
	Harry Potter and the Goblet of Fire (Book 4)	1.77
	Surviving Production: The Art of Production ...	1.77
	Just Tell Me When We're Dead	1.79
	The Conquest	1.8
	The Ghosts of Cougar Island	1.8
	Dr. Death (Alex Delaware Novels (Paperback))	1.81
	Familiar Obsession (Fear Familiar) (Intrigue, 570)	1.81
	My Gal Sunday	1.81
The Catcher in the Rye	Soul Stories	4.18
	Enchanting Baby The Birth Place	4.18
	The Road To Echo Point	4.29
	The Rosewood Casket	4.3
	The Real Allie Newman	4.32
	Holman Concise Topical Concordance	4.35
	East of Eden	4.36
	No Lesser Plea	4.36
	Juxtaposition (Apprentice Adept)	4.37
	Jane Eyre (Puffin Classics)	4.37
The Fellowship of the Ring	I Hate Texas: 303 Reasons Why You Should, Too	0.69
	King Lear	0.73
	When Your Marriage Needs Repair	0.73
	365 Things Every Couple Should Know	0.73
	So You Want to Be a Witch	0.74
	Keeping of Customers: A Treasury of Facts, Tips & ...	0.74
	Zin! Zin! Zin! A Violin	0.74
	Midsummer Nights Dream	0.75
	Pinky And Rex And The Mean Old Witch	0.75
	Miss Nelson Is Missing!	0.75

Capítulo 4

Conclusiones y trabajo futuro

Se cumplió el objetivo de implementar desde cero un sistema de recomendación basado en factorización de matrices y el de probar el desempeño con datos reales. El modelo implementado es una buena opción para un conjunto de datos de calificaciones explícitas. Mejora sustancialmente el modelo base planteado, tanto en RMSE como en la tarea de *top-N recommendations*. El método de descenso en gradiente estocástico permite optimizar la función de pérdida de una manera rápida y barata sin la necesidad de *hardware* especializado: todo se puede implementar y ejecutar desde una computadora personal.

Este método tiene sus limitantes: el tener acceso a conjuntos de datos con calificaciones explícitas no es algo tan común, de hecho, muchos de los datos que se generan en áreas en las que se podría utilizar la tecnología aquí descrita tienen más comúnmente datos con calificaciones implícitas. Sin embargo, este trabajo puede servir como base, y la teoría de aplicaciones que usan datos implícitos no difiere mucho de la que es presentada aquí.

Como trabajo futuro, se plantea el mejoramiento del modelo con una modificación

para que tome en cuenta la dinámica temporal. El modelo aquí presentado es estático, i.e., no toma en cuenta el cambio en el tiempo de la percepción y de la popularidad de los productos. Un modelo dinámico tomaría en cuenta estos conceptos y probablemente mejoraría el desempeño en RMSE y en la tarea de *top-N recommendations*.

Se puede encontrar una copia de este trabajo y del código en:

`https://github.com/mariobecerra/Tesis_LMA`.

Bibliografía

[1] Chris Anderson. *The long tail: Why the future of business is selling less of more*. Hachette Books, 2006.

[2] Sunil Arya y col. *RANN: Fast Nearest Neighbour Search (Wraps Arya and Mount's ANN Library)*. R package version 2.5. 2015. URL: `https://CRAN.R-project.org/package=RANN`.

[3] Robert M Bell y Yehuda Koren. «Lessons from the Netflix prize challenge». En: *Acm Sigkdd Explorations Newsletter* 9.2 (2007), págs. 75-79.

[4] Robert M Bell, Yehuda Koren y Chris Volinsky. «The bellkor 2008 solution to the netflix prize». En: *Statistics Research Department at AT&T Research* (2008).

[5] James Bennett, Stan Lanning y col. «The netflix prize». En: *Proceedings of KDD cup and workshop*. Vol. 2007. New York, NY, USA. 2007, pág. 35.

[6] Christopher M.. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[7] Léon Bottou, Frank E. Curtis y Jorge Nocedal. *Optimization Methods for Large-Scale Machine Learning*. 2016. eprint: `arXiv:1606.04838`.

[8] Paolo Cremonesi, Yehuda Koren y Roberto Turrin. «Performance of Recommender Algorithms on Top-n Recommendation Tasks». En: *Proceedings of the Fourth ACM Conference on Recommender Systems*. RecSys '10. New York, NY, USA: ACM, 2010, págs. 39-46. ISBN: 978-1-60558-906-0. DOI: `10.`

1145/1864708.1864721. URL: http://doi.acm.org/10.1145/1864708.1864721.

[9] Dirk Eddelbuettel. *Seamless R and C++ Integration with Rcpp*. ISBN 978-1-4614-6867-7. New York: Springer, 2013.

[10] Dirk Eddelbuettel y Romain François. «Rcpp: Seamless R and C++ Integration». En: *Journal of Statistical Software* 40.8 (2011), págs. 1-18. URL: http://www.jstatsoft.org/v40/i08/.

[11] Jerome Friedman, Trevor Hastie y Robert Tibshirani. *The elements of statistical learning*. Vol. 1. Springer series in statistics Springer, Berlin, 2001.

[12] Michael Hahsler. *recommenderlab: Lab for Developing and Testing Recommender Algorithms*. R package version 0.2-1. 2016. URL: https://CRAN.R-project.org/package=recommenderlab.

[13] F Maxwell Harper y Joseph A Konstan. «The movielens datasets: History and context». En: *ACM Transactions on Interactive Intelligent Systems (TiiS)* 5.4 (2016), pág. 19.

[14] Gareth James y col. *An introduction to statistical learning*. Vol. 6. Springer, 2013.

[15] Rie Johnson y Tong Zhang. «Accelerating stochastic gradient descent using predictive variance reduction». En: *Advances in Neural Information Processing Systems*. 2013, págs. 315-323.

[16] Yehuda Koren, Robert Bell y Chris Volinsky. «Matrix factorization techniques for recommender systems». En: *Computer* 42.8 (2009).

[17] Jure Leskovec, Anand Rajaraman y Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2014. (Visitado 27-05-2015).

[18] Timothy J. Mullaney. *Netflix*. https://www.bloomberg.com/news/articles/2006-05-24/netflix.

[19] Jorge Nocedal y Stephen J Wright. *Numerical Optimization, Second Edition*. Springer New York, 2006.

[20] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2016. URL: https://www.R-project.org/.

[21] Nicolas L Roux, Mark Schmidt y Francis R Bach. «A stochastic gradient method with an exponential convergence _rate for finite training sets». En: *Advances in Neural Information Processing Systems*. 2012, págs. 2663-2671.

[22] *The Netflix Prize*. http://www.netflixprize.com/.

[23] Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2009. ISBN: 978-0-387-98140-6. URL: http://ggplot2.org.

[24] Hadley Wickham. *tidyverse: Easily Install and Load 'Tidyverse' Packages*. R package version 1.0.0. 2016. URL: https://CRAN.R-project.org/package=tidyverse.

[25] Cai-Nicolas Ziegler y col. «Improving recommendation lists through topic diversification». En: *Proceedings of the 14th international conference on World Wide Web*. ACM. 2005, págs. 22-32.