UNIVERSIDAD DE BUENOS AIRES
Facultad de Ciencias Exactas y Naturales
Departamento de Matemática

Tesis de Licenciatura

Sistemas de Recomendación: Las máquinas de factorización y una aplicación al problema de arranque en frío

Gastón Elián Bujía

Director: Leandro Lombardi

Marzo 2018

Agradecimientos

En primer lugar quiero agradecerle a Leandro, por su confianza y consejos que me brindo a lo largo de esta tesis y la paciencia que tuvo para ayudarme en todo lo necesite. Así también quiero agradecerles a los jurados por tomarse el tiempo de leer esta tesis y corregirla.

A Exactas por todas las oportunidades brindadas y grandes experiencias vividas a lo largo de estos años y los que están por venir.

A todos mis compañeros de la carrera con los que compartí tanto tiempo de cursadas, estudio, charlas y debates.

A Luz por sus consejos y opiniones en el último tramo de la carrera, por su frontalidad ante todo y su constante buen humor.

A Lucho, compañero de finales incansable, viajando al oeste cuando se lo necesitaba, por el aguante constante y sus consejos, siempre presente ya sea estudiando, cursando o saliendo a tomar algo.

A los pibes de ecuaciones, Juan, Juan Z, Pepe y Javi, por las noches de estudio con mucha cerveza y siempre algo a la parrilla.

A mis compañeros de docencia, en especial a Eugenio y a Flor, mis compañeros en el 2017, por todo el aguante en los tramos finales de la carrera.

A Mauricio y Dario, MAUDAGA, por su apoyo constante en los tiempos más adversos, por ser compañeros de aventuras y noches de cerveza. Por una amistar que dure toda la vida porque son las personas con la que constantemente nos ayudamos a ser mejor cada uno de nosotros. Sin ellos, todo el camino recorrido hubiese sido mucho más difícil.

A Agostina, Natalia y Matias, por los excelentes momentos vividos durante todo este tiempo juntos en Tecnopolis, las largas charlas de política y de la vida.

A mis compañeros de Tecnopolis, sobretodo a Seba, Ema, Chris, Aye, Nacho, Nico, Axel, por darme siempre más ganas de trabajar en Tecnopolis y hacer que lo era un trabajo se convirtiera en una casa de la que no me quería ir.

A Camila, Yamila y Sol, por ayudarme a preocuparme menos por todo y sacar las mejores carcajadas, siempre haciéndome el aguante con unos mates o unas cervezas.

A Lucas, por su amistad incondicional y su eterna compañía en la vida, en el colegio, en viajes, en nuestras casas, por las risas y los llantos, por los mates y las charlas. Por siempre tratar de sacar lo mejor de mí aún en los peores momentos.

A Facundo, Nicolás, Leandro, Damián y Sergio, por las juntadas espontáneas y los vicios compartidos.

A Beto, Ale, Ara, Mica y Juan, por los mates de domingo, las juntadas nocturnas, los debates, las charlas, las películas, siempre haciéndome sentir uno más de la familia.

A Nacho y Gabi, por las tardes en la empresa vendiendo humo y las posteriores salidas a Finisterre.

A mis abuelas Irma e Isabel, que aunque ya no están conmigo, siempre me acompañarán a donde sea que vaya.

A mi hermano Mariano, porque en nuestras diferencias siempre supo hacerme reír, porque siempre creyó en mí y nunca me dejo dudar de mi. Y a Luciana, por ser mi hermana, por acompañarme siempre en las buenas y en las malas, por ser la hermana perfecta que cualquiera querría tener.

Por último, a mis viejos, sostén total de todo lo que pude hacer en mi vida. Gracias por el amor y la paciencia que me ayudaron a alcanzar mis metas, sin su apoyo nada de esto hubiese sido posible. Gracias por acompañarme en este increíble viaje. Gracias siempre.

Índice general

Introducción

Los sistemas de recomendación se han vuelto muy populares en los últimos años debido a la gran cantidad de información que existe en la época actual. Los mismos se encuentran ampliamente extendidos en los ámbitos de las tiendas en línea, sitios de música, películas, libros, artículos, inmuebles o hasta aplicaciones sociales. Estos sistemas, tratan de recomendar a los usuarios diferentes artículos según sus gustos o necesidades, valiéndose de técnicas propias del aprendizaje automático.

El objetivo de esta tesis es la de realizar una introducción al tema, enfocándonos en una rama particular conocida como filtros colaborativos y las problemáticas relacionadas al área. En el primer capitulo realizaremos un resumen de las definiciones más importantes, describiremos los algoritmos de recomendación clásicos de los filtros colaborativos y analizaremos diferentes formas de evaluar su rendimiento.

En el segundo capítulo, introduciremos las máquinas de factorización y veremos como aplicarlas en el ámbito de recomendación, comparando su rendimiento con las técnicas clásicas.

En el tercer capítulo, nos centraremos en uno de los problemas más importantes de los filtros colaborativos, conocido como el problema del arranque en frío. Describiremos algunas técnicas para mitigar este problema, y propondremos un acercamiento diferente.

En los últimos dos capítulos, detallaremos las simulaciones y experimentos llevadas a cabo para comparar el rendimiento de las máquinas de factorización con el resto de las técnicas y veremos como se comporta nuestra propuesta para el escenario de arranque en frío.

Capítulo 1

Sistemas de recomendación

En este capítulo introduciremos los sistemas de recomendación y haremos particular hincapié en un tipo específico de sistema conocidos como *filtros colaborativos*. Describiremos los principales conceptos y definiciones, su clasificación así como las métricas de evaluación relacionadas y las técnicas clásicas en el área.

1.1. Preliminares

Los sistemas de recomendación son técnicas y herramientas que se engloban dentro del aprendizaje automático y la teoría de recuperación de la información. Su objetivo principal es predecir la valoración de un usuario sobre un objeto y recomendar a un conjunto de usuarios uno o más elementos de una lista predeterminada que pueden resultar de su interés. Dichos elementos que llamaremos *items*, suelen pertenecer a alguna categoría específica de objetos como ser películas, libros, música, papers, documentos, noticias, etc. [43]. Para desarrollar esta tarea, los sistemas de recomendación utilizarán la información existente sobre la interacción entre los usuarios y los items.

Para poder medir el interés de un usuario u sobre un ítem j llamaremos r_{uj} al *rating* o valoración que tiene dicho usuario sobre ese ítem. Los posibles ratings $r_{uj} \in \mathcal{R}$, donde $\mathcal{R}$ será el conjunto de ratings, por ejemplo $\mathcal{R} = \{1, 2, 3, 4, 5\}$ o puede ser un rating binario donde $\mathcal{R} =$\{"Me gusta", "No me gusta"\}, o incluso venir de forma implícita como cantidad de clicks hechos sobre un ítem o si se compró el ítem o no.

Definición 1.1. Sea $R \in \mathbb{R}^{n \times m}$ la matriz de ratings usuarios-items, donde las filas representan a los n usuarios y las columnas a los m items:

$$R = \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1m} \\ r_{21} & r_{22} & \cdots & r_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ r_{n1} & r_{n2} & \cdots & r_{nm} \end{pmatrix}$$

Las entradas (u, i) representan el rating r_{ui} del usuario $u \in U$ sobre el ítem $i \in I$, donde U es el conjunto de usuarios e I el de items.

La matriz usuario-ítem R del sistema de recomendación es una matriz con una gran cantidad de valores faltantes, esto se debe en general a la gran cantidad de items que suelen contar los sistemas de recomendación en comparación a los pocos que evalúan los usuarios, lo que representa una de las principales dificultades que enfrentan estas técnicas. Por ejemplo, uno de los sistemas de recomendación más conocidos es el brindado por la empresa Netflix, la cual ha contribuido significativamente al área con una competencia que se conoció como *The Netflix Prize* [11]. El conjunto de datos de entrenamiento para esta competencia contenía 100,480,507 ratings que 480,189 usuarios dieron a 17,770 películas, con lo cual la densidad de la matriz de datos es aproximadamente 1.2 %.

Ejemplo 1.1. Supongamos que tenemos un sistema de recomendación de películas que utiliza un rating del 1 al 5, donde tenemos seis usuarios $U = \{u_1, \cdots, u_6\}$ y seis películas $I = \{i_1, \cdots, i_6\}$, y las evaluaciones observadas sobre las películas vistas vienen dadas por la matriz:

	i_1	i_2	i_3	i_4	i_5	i_6
u_1	2		5		4	
u_2		3	1			4
u_3	1		4	2		
u_4			3		5	3
u_5	4	5		3		1
u_6		2		4	4	

Los espacios en blanco representan las películas que no fueron puntuadas los usuarios, por ejemplo el usuario u_1 vio la película i_3 y la evaluó con cinco puntos, entonces $r_{1,3} = 5$. Mientras que u_1 no vio las películas i_2, i_4 e i_6. □

El objetivo principal de un sistema de recomendación, en este contexto, es predecir las entradas desconocidas de la matriz de ratings R, es decir la valoración $\hat{r}_{uj}$ del usuario u sobre el ítem j, sin embargo pueden establecerse otros objetivos como hablaremos más adelante. Para realizar dicha tarea hay varias estrategias en general que se suelen dividir en tres grandes grupos [43]:

Filtros Colaborativos Los filtros colaborativos utilizan los ratings aportados por múltiples usuarios para poder estimar un rating desconocido de un ítem para un cierto usuario. Este tipo de técnicas son las más estudiadas y utilizadas en los sistemas de recomendación. Describiremos este tipo de sistemas con mayor detalle más adelante.

Basados en Contenido En los sistemas de recomendación basados en el contenido, los atributos descriptivos de los elementos se utilizan para hacer recomendaciones. El término *contenido* se refiere a estas descripciones referidas

a cualidades de los items. En los métodos basados en el contenido, las calificaciones y el comportamiento de compra de los usuarios se combinan con la información de contenido disponible en los artículos. Por ejemplo, supongamos que sabemos que un usuario valoró altamente una película dada, pero no conocemos el rating de otros usuarios respecto a esta película ni al usuario. Sin embargo, las características de la película pueden ser usadas para hacer una recomendación, en este caso el contenido que podríamos utilizar son el género de la película, los actores, el director, el año de filmación, etc.

Basados en Conocimiento Los sistemas de recomendación basados en el conocimiento son particularmente útiles en el contexto de elementos que no se compran muy a menudo. Los ejemplos incluyen elementos tales como bienes inmobiliarios, automóviles, solicitudes de turismo, servicios financieros o bienes de lujo caros. En estos casos, como los artículos se compran raramente o simplemente el interés de un usuario sobre los mismos tiene mucha variabilidad en el tiempo, entonces no tenemos acceso al comportamiento del usuario para poder hacer una recomendación. Tales casos se pueden abordar con sistemas de recomendación basados en el conocimiento, en los que las calificaciones no se utilizan para el propósito de las recomendaciones, sino que el proceso de recomendación es realizado sobre la base de las similitudes entre los requisitos del usuario y las descripciones de los elementos. Para esto los sistemas de recomendación recurren a la interacción con el usuario activo, es decir con el usuario al cual le queremos hacer una recomendación.

Cada uno de estos tipos de sistemas tienen sus puntos fuertes y sus debilidades dependiendo del problema considerado. Debido a esto, es usual en la práctica la utilización de sistemas de recomendación *híbridos* o *basados en ensambles*, un sistema que combina varias técnicas recién mencionadas para crear un sistema de recomendación más robusto.

1.2. Filtros Colaborativos

Las técnicas denominadas como *filtros colaborativos* utilizan las valoraciones que se poseen sobre los items dadas por los usuarios, basándose en el supuesto de que existe una alta correlación entre los ratings de los items y los usuarios [43], para poder predecir los valores desconocidos de la matriz de ratings R. Estas técnicas son de las más estudiadas y utilizadas del área y en las que nos centraremos en esta tesis.

Para poder realizar las recomendaciones, los filtros colaborativos tratan de establecer relaciones entre los usuarios y los items, y para eso existen dos principales tipos de acercamiento conocidas como *Basados en la Memoria* y *Basados en el Modelo* [43] [22]:

Métodos basados en memoria Estos métodos se basan en utilizar los ratings conocidos en la matriz R, estableciendo similitudes entre los items entre sí

y/o entre los usuarios entre sí. La intuición detrás de estos métodos es que en general un usuario al que le gusta un determinado ítem preferirá otros items similares a éste, o preferirá los mismos items que prefirió algún otro usuario que tenga gustos similares al mismo. Estos métodos suelen ser conocidos también como *basados en vecindades*, y dependiendo de donde se utilice la similitud pueden ser métodos *basados en usuarios* o *basados en items*. La principal ventaja de estas técnicas es que son fáciles de implementar pero por otro lado no suelen funcionar bien con matrices ralas.

Métodos basados en el modelo Estos métodos utilizan herramientas de aprendizaje automático para encontrar modelos predictivos para el sistema. El acercamiento más utilizado es el conocido como *Modelo de Factores Latentes* que utiliza técnicas de factorización de matrices para encontrar descomposiciones de la matriz R que permitan generar predicciones. La ventaja es que en general estos métodos funcionan mejor que los basados en memoria con matrices ralas pero a expensas de mayores tiempos computacionales por su mayor complejidad.

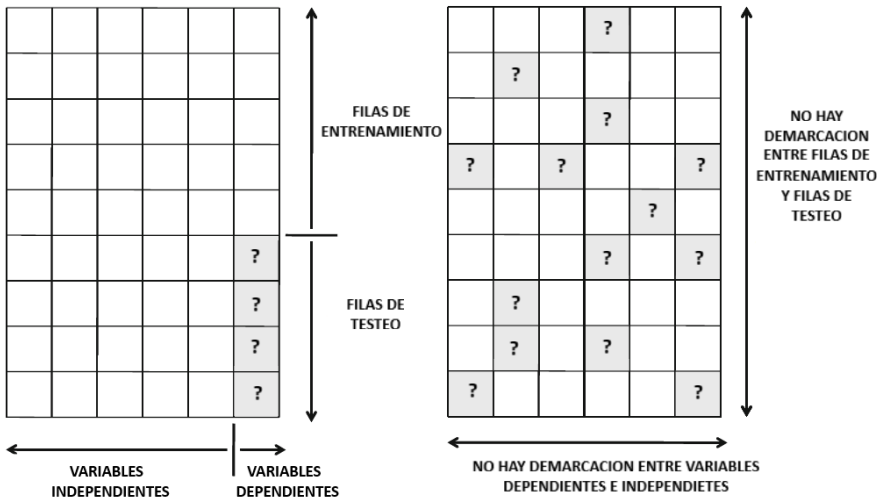


Figura 1.1: A la izquierda una matriz usual en regresión y a la derecha la matriz del problema de filtrado colaborativo

El problema del filtrado colaborativo se puede ver como una generalización de un problema de clasificación o regresión [43] [22] [9] o como un caso especial del problema de valores faltantes (*missing data*). En un problema de clasificación en general podemos pensar que la variable dependiente o clase a determinar, es en realidad un atributo con valores faltantes que queremos calcular, representados en una columna y todas las otras columnas son las variables independientes. En el caso de los filtros colaborativos permitimos que los valores faltantes a predecir no sean sólo un atributo, es decir solo una columna, sino que son varias diferentes

y no pertenecen necesariamente a las mismas columnas, por lo cual no es posible identificar las variables dependientes de las independientes. En estos casos tiene más sentido pensar en entradas de entrenamiento y observaciones de testeo que en filas de entrenamiento y testeo como es en el caso de la clasificación/regresión. En la figura 1.1 podemos ver la comparación entre la clasificación y los filtros colaborativos, donde las posiciones con signos de interrogación representan los valores desconocidos.

A la hora de elegir algún método de estos debemos tener claro cual es el objetivo del algoritmo así como su diseño, para que una mejora en la métrica refleje verdaderamente una mejora en la efectividad del mismo. En lineas generales, los dos principales tareas que se plantean los sistemas de recomendación son los siguientes:

- **Predicción** Este es el más simple de los objetivos y la formulación primitiva de un sistema de recomendación, predecir la valoración desconocida r_{uj} del usuario u sobre el ítem j como $\hat{r}_{uj}$.
- **Recomendación** En la mayoría de las configuraciones prácticas, el sistema no necesariamente busca las valoraciones o ratings específicos de los pares desconocidos usuario-ítem. En vez de esto, se busca hallar los k items más relevantes para cada usuario, o los k usuarios más relevantes para cada ítem. Esta tarea es denominada como la recomendación *top-k* de items o de usuarios. En general es más común el escenario de recomendación *top-k* items que de usuarios, el cual será el paradigma al que nos referiremos en esta tesis.

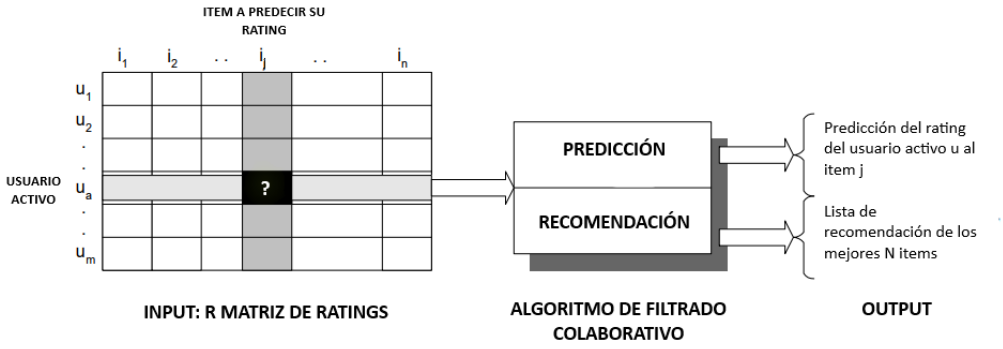


Figura 1.2: Esquema del proceso del filtrado colaborativo, dependiendo de la tarea que se quiera resolver.

Los dos problemas antes mencionados están estrechamente relacionados [43]. Por ejemplo, para determinar los elementos top-k para un usuario en particular, uno puede predecir los ratings de cada ítem para ese usuario y tomar los k elementos con las mejores valoraciones predichas. Sin embargo, los dos problemas no son necesariamente el mismo, de esto hablaremos más adelante cuando discutamos las formas de medir el rendimiento de un sistema.

Existen otros objetivos secundarios que suelen complementar a las tareas recién descritas, algunas de ellas son la novedad de los items, la diversidad, la cobertura y la serendipia entre otros [34]. Aun así, estos últimos objetivos son difíciles de cuantificar y su evaluación suele ser muy subjetiva a cada sistema, por lo cual en el alcance de esta tesis nos limitaremos a utilizar las métricas más generales que midan la precisión en la predicción de los ratings y rankings.

1.3. Métricas de evaluación

Dentro de la literatura de los sistemas de recomendación existen múltiples métricas utilizadas para evaluar su calidad [43] [34] [7]. La evaluación de un sistema se puede realizar en dos instancias diferentes, una *online* y otra *offline*. La primera de estas requiere necesariamente de interacción continua con los usuarios para obtener los resultados y aplicar técnicas del estilo de *A/B testing* para comparar los mismos. Es decir que los datos que utilizaríamos para estimar el rendimiento de nuestro modelo, serían provistos por el usuario a medida que el mismo interactúa con el sistema. Por otro lado, la evaluación offline utiliza datos históricos en general ya estandarizados, que contienen pares de usuario-items de los cuales se conocen los ratings. La evaluación offline es la más utilizada para comparar técnicas entre sí [43] debido a la estandarización de los conjuntos de datos públicos existentes para testeo como son *Jester*, *MovieLens* y *Netflix* entre otros [26]. Para ampliar en técnicas de evaluación online se puede ver [34] [17].

La evaluación offline de la precisión de un sistema la podemos dividir en dos tipos de tareas diferentes según el objetivo de nuestro sistema, medir la precisión de los ratings que fueron predichos o la precisión del ranking de los items recomendados. Esta última se basa en el hecho de que podemos mirar solo el orden en el que se recomiendan los items sin tener en cuenta el rating de los mismos de forma explícita. A continuación describiremos brevemente las medidas consideradas con sus ventajas y desventajas.

1.3.1. Medidas de predicción de ratings

Las medidas de predicción de ratings son las más populares y más usadas debido a su simplicidad matemática y algorítmica. La idea de fondo detrás de las mismas es la suposición de que un usuario preferirá un algoritmo que pueda predecir sus ratings en general con más precisión. Para definir las métricas de evaluación, consideraremos que contamos con un conjunto de ratings observados S:

$$S = \{r_{ui} \in R : u \in U, i \in I / \text{ el usuario } u \text{ evaluó el ítem } i\}$$

Sean E, T una partición disjunta de S, donde E es el conjunto de datos que utilizaremos para estimar los ratings(conjunto de entrenamiento) y T es el conjunto que utilizaremos para evaluar el algoritmo(conjunto de test). La elección de estos conjunto puede ser aleatoria dada una proporción fija o utilizando una validación

cruzada de 5 o más iteraciones (*K-fold cross-validation*) como describiremos en más detalle en el capítulo 4.

Dado $r_{ui} \in T$ haremos un pequeño abuso de notación al utilizar $(u, i) \in T$ en su lugar, definimos el error $e_{ui} = r_{ui} - \hat{r}_{ui}$ donde $\hat{r}_{ui}$ representa el rating predicho. Ahora podemos definir el error cuadrático medio o MSE (*Mean Squared Error*) por sus siglas en inglés y respectivamente la raíz del error cuadrático medio (RMSE)

Definición 1.2. Definimos el MSE y el RMSE como:

$$MSE = \frac{\sum_{(u,i)\in T} e_{ui}^2}{|T|} \tag{1.1}$$

$$RMSE = \sqrt{\frac{\sum_{(u,i)\in T} e_{ui}^2}{|T|}} \tag{1.2}$$

En esta tesis, utilizaremos el término error cuadrático medio para referirnos al RMSE. Además otra métrica muy utilizada que definiremos es el error medio absoluto o MAE(*Mean Absolute Value*).

Definición 1.3. Definimos el MAE como:

$$MAE = \frac{\sum_{(u,i)\in T} |e_{ui}|}{|T|} \tag{1.3}$$

Claramente valores más pequeños del RMSE y MAE significan errores más pequeños en los ratings estimados y por lo tanto mejor predicción de los ratings. También se suele utilizar la versión normalizada de estas métricas, NRMSE y NMAE las cuales resultan de dividir las mismas por la longitud del rango de ratings (r_{min}, r_{max}) que nos permiten garantizar que nuestras métricas estén dentro del intervalo $(0, 1)$.

¿Cuál de estas métricas es mejor? En general no hay una respuesta definitiva a esta pregunta y dependemos de la aplicación del algoritmo de recomendación. El RMSE tiende a penalizar fuertemente errores muy grandes debido a que los eleva al cuadrado en comparación al MAE, en la competencia *The Netflix Prize* se utilizo el RMSE [11]. Se puede encontrar una discusión más profunda acerca de las ventajas y desventajas entre ambas métricas en Willmott et al. [10] y Chai et al. [28].

1.3.2. Medidas de predicción de rankings

Tanto el RMSE como el MAE, tienen en cuenta los errores realizados sobre los ratings predichos, pero en muchas aplicaciones de los sistemas de recomendación no tienen como objetivo la estimación de los ratings en sí, sino en una recomendación de una lista de k elementos de mayor interés para el usuario, lo que denominamos como la recomendación *top-k*. Bajo este contexto, dentro de la teoría de la recuperación de información (*Information Retrieval*), existen múltiples métricas para evaluar este tipo de problemas que dependen del tipo de rating de los items, ya sea un sistema de evaluación binaria (por ejemplo compras, clicks, etc.) o de múltiples valores (puntuación, rating, etc.).

Precision-Recall y MAP

Llamaremos *usuario activo* al usuario u al cual se le realizará la recomendación y sea $M_u(k)$ el *top-k* de items recomendados por nuestro sistema para ese usuario. Sea I_u el conjunto de items evaluados por el usuario activo, cada uno de esos items, podemos clasificarlos como relevantes para este usuario o no a partir de algún valor de corte. Por ejemplo en un caso de un rating del 1 al 5, podría depender de la media de los ratings del usuario o ser un valor fijo. A la vez, dados los items $j \in M_u(k)$, diremos que esos items son recomendados si el rating estimado $\hat{r}_{uj}$ también es mayor al corte determinado. Resumiendo, si elegimos un valor de corte para los ratings $\delta > 0$:

$$\begin{aligned} i \in I \text{ es relevante para } u &\iff r_{ui} \geq \delta \\ i \in I \text{ es recomendado para } u &\iff \hat{r}_{ui} \geq \delta \text{ e } i \in M_u(k) \end{aligned}$$

Y definimos $G_u \subset I_u$ como el conjunto de todos los items relevantes para este usuario:

$$G_u = \{i \in I : r_{ui} \geq \delta\}$$

Además definiremos una indicadora para el usuario activo y un ítem $i \in M_u(k)$, que marcará cuándo efectivamente recuperamos un ítem relevante en la recomendación:

$$\mathrm{Rel}_u(i) = \begin{cases} 1, & \text{si } r_{ui} \geq \delta \wedge \hat{r}_{ui} \geq \delta \\ 0, & \text{si no} \end{cases}$$

Definición 1.4. Definimos la precisión ($P_u(k)$ o **P@k**) y la exhaustividad (o sensibilidad) (*Recall*, $R_u(k)$ o **R@k**) en k como:

$$P_u(k) = \frac{|M_u(k) \cap G_u|}{|S_u(k)|} \tag{1.4}$$

$$R_u(k) = \frac{|M_u(k) \cap G_u|}{|G_u|} \tag{1.5}$$

□

La precisión $P(k)$[1] para un usuario u, mide la proporción de items relevantes que hay en $M_u(k)$ y la exhaustividad, $R(k)$, mide la proporción del total de los items relevantes que han sido recomendados.

Una manera fácil de interpretar estas medidas es pensar que los items relevantes y que son recomendados a la vez, son los resultados verdaderos positivos

[1]En general haremos un abuso de notación, utilizando $P(k)$ en lugar de $P_u(k)$ cuando se entiende que está fijo el usuario. Respectivamente, haremos lo mismo con $R_u(k)$ y $S_u(k)$

(True-Positive) conseguidos en $M_u(k)$ como en el cuadro 1.1, entonces la precisión debería ser el resultado de dividir estos resultados verdaderos positivos sobre todos los recomendados en $M_u(k)$, es decir los Verdaderos-Positivos (TP) más los Falsos-Positivos(FP). Análogamente podemos definir el recall, resultando entonces:

$$P(k) = \frac{TP}{TP + FP} \tag{1.6}$$

$$R(k) = \frac{TP}{TP + FN} \tag{1.7}$$

Generalmente buscamos un equilibrio entre estas cantidades ya que, por ejemplo, recomendar más items puede llevar a una pérdida de precisión, pues ésta no es monótona, pero a la vez puede capturar más elementos relevantes que antes, lo que mejoraría la exhaustividad. Para poder analizar este intercambio podemos graficar lo que se conoce como la curva *precision-recall*, de la cual hablaremos más adelante.

	Recomendado	No recomendado
Relevante	True-Positive (TP)	False-Negative (FN)
No Relevante	False-Positive (FP)	True-Negative (TN)

Cuadro 1.1: Representación de los posibles resultados de las recomendaciones/relevancia.

La precisión es una medida muy utilizada debido a su clara interpretación, pero posee algunas desventajas, por ejemplo, si el usuario elegido no poseía items relevantes, el valor de $P(k)$ no será informativo. Además la precisión no valora el orden en el cual los items son recomendados, dentro del conjunto de items recomendado solo evalúa si son relevantes o no. Por este motivo introducimos la *Media de la Precisión Promedio* o **MAP** por sus siglas en ingles (*Mean Average Precision*) que calcula la precisión de la lista de longitud l donde $l = 1, \ldots, k$ y los promedia y luego nuevamente promedia pero sobre todos los usuarios. Para esto, primero introduciremos la precisión promedio:

Definición 1.5. Fijado el usuario activo u, sea $g_u = |G_u|$ como la cantidad de items valorados que son relevantes, llamaremos *precisión promedio* en k ($AP_u(k)$ o AP@k) como:

$$AP_u(k) = \frac{1}{\text{mín}(k, g_u)} \sum_{l=1}^{k} P_u(l) \cdot \text{Rel}(l) \tag{1.8}$$

Donde el término $\text{mín}(k, g_u)$ normaliza sobre la cantidad de items relevantes, previniendo que sea fuertemente penalizada la métrica en caso de que la cantidad de items a recomendar fuera mucho menor a los items relevantes.

A diferencia de las métricas de precisión de rating vistas(RMSE y MAE), para las cuales un valor menor en la métrica significa una mejor predicción, en el caso

de la precisión promedio, la precisión y la exhaustividad, un mayor valor de las mismas significa una mejor predicción del ranking. En el caso de la precisión y la exhaustividad en k, el valor máximo que pueden alcanzar es 1, aunque no siempre lo alcanzarán.

Definición 1.6. Sea n la cantidad total de usuarios, definimos el promedio de la precisión promedio(MAP) en k como:

$$\mathbf{MAP@k} = \frac{1}{n}\sum_{u=1}^{n} AP_u(k) \tag{1.9}$$

Una de las ventajas de MAP es que no solo genera una medida global sobre todos los usuarios, sino que le otorga mayor importancia al orden en el cual los items fueron recomendados al estar basado en la precisión promedio.

Ejemplo 1.2. Supongamos que tenemos un sistema de recomendación de libros donde aplicaremos dos algoritmos diferentes, A y B, y un tercer algoritmo perfecto C para contrastar los anteriores. Queremos evaluar su rendimiento para recomendar ocho libros a un usuario activo y para eso compararemos su precision, recall y MAP. Supongamos que hay solo tres items relevantes entre los disponibles para el usuario m_u y los resultados obtenidos son los siguientes:

	Algoritmo A			Algoritmo B			Algoritmo C		
Ranking(i)	Rel	P(i)	R(i)	Rel	P(i)	R(i)	Rel	P(i)	R(i)
1	1	1/1	1/3	1	1/1	1/3	1	1/1	1/3
2	1	2/2	2/3	0	1/2	1/3	1	2/2	2/3
3	0	2/3	2/3	0	1/3	1/3	1	3/3	3/3
4	1	3/4	3/3	1	2/4	2/3	0	3/4	3/3
5	0	3/5	3/3	0	2/5	2/3	0	3/5	3/3
6	0	3/6	3/3	0	2/6	2/3	0	3/6	3/3
7	0	3/7	3/3	0	2/7	2/3	0	3/7	3/3
8	0	3/8	3/3	1	3/8	3/3	0	3/8	3/3

Notemos que si solo tomasemos en cuenta $P(8)$ y $R(8)$, la última fila del recuadro, tendíamos que el rendimiento de los tres algoritmos es el mismo, pero es claro que el algoritmo C, el perfecto, tiene mejor rendimiento en conseguir recomendar de manera óptima los libros, ya que son los tres primeros que recomendo. Por ejemplo si tomásemos $P(3)$ y $R(3)$, se ve claramente como el algoritmo C es superior al A y al B. Calculemos ahora la precisión promedio para cada algoritmo, donde si $Rel(i) = 0$ no aportará a la métrica:

$$\begin{aligned} AP^{(A)}(8) = &\quad (1\cdot 1 + \tfrac{2}{2}\cdot 1 + \tfrac{3}{4}\cdot 1)\times \tfrac{1}{3} \approx 0.92 \\ AP^{(B)}(8) = &\quad (1\cdot 1 + \tfrac{2}{4}\cdot 1 + \tfrac{3}{8}\cdot 1)\times \tfrac{1}{3} \approx 0.62 \\ AP^{(A)}(8) = &\quad (1\cdot 1 + \tfrac{2}{2}\cdot 1 + \tfrac{3}{3}\cdot 1)\times \tfrac{1}{3} = 1 \end{aligned}$$

Ahora queda claro que el mejor rendimiento entre los dos algoritmos A y B, es el del A, lo cual era evidente ya que conseguía en las primeras 4 recomendaciones pegarle a los tres libros relevantes, mientas que el B solo conseguía recomendar dos de ellos y recién en la octava recomendación conseguía el libro relevante restante.

Métricas basadas en utilidad

Las métricas basadas en utilidad hacen uso tanto de los ratings conocidos de los usuarios, como del ranking de recomendación del sistema. En general estas métricas utilizaran alguna función de ganancia que involucra ambas cosas, rating y ranking. Definiremos tres métricas muy utilizadas en este aspecto, *R-Score*, $NDCG$ y el $ARHR$.

Consideremos el conjunto I_u de los items evaluados por el usuario u, para el ítem i definimos la utilidad basada en rating como el máx$\{r_{ui} - C_u, 0\}$ donde C_u es un punto de corte para el usuario u que puede ser definido como la media de los items que evaluó y sea v_i el ranking del ítem i en la lista de recomendación de nuestro sistema para nuestro usuario activo, esto es, la posición en la que aparece dicho ítem al ordenar la recomendación del más relevante al menos relevante. Dado α un parámetro de ajuste, definimos entonces la utilidad del par (u, i) como:

$$F(u,i) = \frac{\text{máx}\{r_{ui} - C_u, 0\}}{2^{(v_i-1)/\alpha}} \tag{1.10}$$

Y definimos entonces para cada usuario lo que se conoce como *R-score*(u) y el *R-score* global como:

$$\text{R-score(u)} = \sum_{i \in I_u} F(u,i) \tag{1.11}$$

$$\text{R-score} = \sum_{u} \text{R-score}(u) \tag{1.12}$$

El término $2^{-(v_i-1)}$ generalmente conocido como la utilidad basada en el ranking, asegura una caída exponencial en el mismo bajo la lógica de que un usuario estará más interesado en los items rankeados más alto que en los últimos recomendados. Sin embargo, esto depende mucho del contexto del sistema de recomendación y la cantidad de elementos recomendados en la lista.

A continuación, introduciremos una métrica que penaliza menos el ranking que el R-score. Sea f_{ui} una función de utilidad del usuario u sobre el ítem i, típicamente se define como una función exponencial del ranking o relevancia como por ejemplo:

$$f_{ui} = 2^{rel_{ui}} - 1 \tag{1.13}$$

Aquí la relevancia (rel_{ui}) del ítem i para el usuario u esta representada por el valor verdadero de su rating [16]. También se suele utilizar como función de ganancia lineal a la misma relevancia $f_{ui} = rel_{ui}$, que a diferencia de la función exponencial, no le da un peso tan grande a los items de mayor relevancia. Consideremos ahora un

término de descuento o penalización sobre el orden en el que aparecen los items, en general se utiliza un función creciente logarítmica que penalice esa ganancia como $\log_2(i+1)$. De esta forma, penalizamos items relevantes que aparezcan bajos en el orden de recomendación.

Definición 1.7. Definimos la Ganancia Acumulada Descontada o Discounted Cumulative Gain (DCG) sobre todos los usuarios y su versión normalizada ($NDCG$) como:

$$DCG = \frac{1}{n}\sum_{u=1}^{n}\sum_{i\in I_u}\frac{g_{ui}}{\log_2(i+1)} \tag{1.14}$$

$$NDCG = \frac{DCG}{IDCG} \tag{1.15}$$

Donde el IDCG es el Ideal Discounted Cumulative Gain, que se calcula con el DCG pero en vez de utilizar la relevancia de los rankings predichos, utilizamos el orden de la recomendación óptima para el usuario. El NDCG se encuentra entre $[0,1]$, siendo 1 el valor óptimo de esta medida, y al igual que con la precisión, mientas mayor sea esta métrica, mejor será el rendimiento. En general utilizaremos la versión normalizada pero calculada sólo sobre los items en la recomendación top-k y promediaremos sobre todos los usuarios:

$$DCG(k) = \frac{1}{n}\sum_{u=1}^{n}\sum_{i\in I_u, i\leq k}\frac{2^{rel_{ui}}-1}{\log_2(i+1)} \tag{1.16}$$

$$NDCG(k) = \frac{DCG(k)}{IDCG(k)} \tag{1.17}$$

Por último definimos el Average Reciprocal Hit-Rate ($ARHR$) o también conocido como Mean Reciprocal Rank (MRR) como:

$$ARHR(u) = \sum_{i\in I_u}\frac{rel_{ui}}{v_i} \tag{1.18}$$

En este caso el $rel_{ui} \in \{0,1\}$ representa si el ítem es relevante o no, nuevamente dependiendo de algún punto de corte. Entonces el ARHR global es:

$$ARHR = \frac{1}{n}\sum_{u=1}^{n} ARHR(u) \tag{1.19}$$

Ejemplo 1.3. Supongamos que tenemos un sistema de recomendación de películas en el cual las mismas son evaluadas del 0 al 3. Supongamos que queremos evaluar el rendimiento en el top-6 del sistema con la métrica NDCG de ganancia exponencial. Supongamos que el sistema recomienda en orden descendente de preferencia las películas $F_1, \cdots, F_6$. Primero calculemos el DCG, para esto necesitamos los verdaderos ratings r_{ui} que el usuario había dado a los items que el sistema recomendó, y sobre eso calculamos la ganancia y el descuento:

Ranking	Película	Rating	Ganancia	Descuento
1	F_1	3	7	1
2	F_2	2	3	1.585
3	F_3	3	7	2
4	F_4	0	0	2.322
5	F_5	1	1	2.585
6	F_6	2	3	2.807

Por lo tanto el DCG(6) $= 7 + (\frac{3}{1.585}) + (\frac{7}{2}) + (\frac{0}{2.322}) + (\frac{1}{2.585}) + (\frac{3}{2.807}) \approx 13.848$ El valor del DCG solamente no parece ser muy informativo sin una referencia, para lo cual calculamos el IDCG para poder normalizar el valor de la métrica. Asumamos que el orden real de las 6 películas preferidas del usuario eran $\{F_1, F_3, F_7, F_2, F_6, F_9\}$, donde aparecen películas que el sistema no había recomendado como la F_7 y la F_9 con $r_{u,7} = 3$ y $r_{u,9} = 1$ respectivamente:

Ranking	Película	Rating	Ganancia	Descuento
1	F_1	3	7	1
2	F_3	3	7	1.585
3	F_7	3	7	2
4	F_2	2	3	2.322
5	F_6	2	3	2.585
6	F_9	1	1	2.807

Entonces IDCG(6) $= \frac{7}{1} + \frac{7}{1.585} + \frac{7}{2} + \frac{3}{2.322} + \frac{3}{2.585} + \frac{1}{2.807} \approx 17.725$, y por lo tanto:

$$NDCG(6) = \frac{DCG(6)}{IDCG(6)} = \frac{13.848}{17.725} \approx 0.718 \tag{1.20}$$

Obtenemos así una métrica estandarizada, que nos permite tener una mejor idea de qué tan cerca estuvo nuestro sistema de recomendar de manera óptima las películas. □

1.4. Técnicas clásicas de filtros colaborativos

En esta sección desarrollaremos algunas de las técnicas más populares dentro de los filtros colaborativos como SVD y *Slope One*, y otras más actuales como son las Máquinas de Factorización, las cuales describiremos con más detalle en el capítulo 2. Describiremos como se establecen las predicciones de los ratings y cuales son sus ventajas y desventajas.

Recordemos algunos aspectos de la notación y definiciones que utilizaremos en general:

- Sea $U = \{u_1, u_2, \cdots, u_n\}$ la lista de los n usuarios.
- Sea $I = \{i_1, i_2, \cdots, i_m\}$ el conjunto de los m items.

- Sea R la matriz de ratings usuarios-items con los ratings observados. Separaremos en un principio los datos observados S en dos conjuntos disjuntos E y T, donde E es el conjunto de datos de entrenamiento para el algoritmo y T es el conjunto de datos para el testeo.

1.4.1. $k-$Vecinos más cercanos

Los algoritmos basados en vecindades tienen su fundamento en la hipótesis de que si dos usuarios tienen un comportamiento similar a la hora de evaluar los mismos items, entonces probablemente podamos usar esa información para predecir el comportamiento de un usuario sobre un ítem no evaluado a partir de los usuarios parecidos. Este razonamiento se puede hacer de forma análoga para los items que son similares entre sí, entonces podemos definir dos acercamientos diferentes [30]:

- *Modelos basados en usuarios*: usuarios similares tendrán similares gustos por el mismo ítem. Por ejemplo, si dos usuarios A y B evaluaron los mismos items de manera muy similar excepto por un solo ítem i que solo evaluó A, entonces tendría sentido utilizar ese rating para estimar el rating de B sobre j.

- *Modelos basados en items*: items similares serán evaluados de forma similar por el mismo usuario.

Utilizaremos un modelo basado en items, supondremos que queremos estimar el rating de un usuario u sobre el ítem activo j, el algoritmo lo podemos desglosar en tres pasos [3]:

1. Calcular la similitud entre el ítem j activo y el resto de los items.

2. Seleccionar un subconjunto de esos items como *vecindad* del ítem activo.

3. Utilizando los ratings de la vecindad del ítem activo, predecir el rating del usuario activo para el ítem j.

Para definir la vecindad necesitaremos fijar un parámetro $k \in \mathbb{N}$, entonces la vecindad será $N_j(u) = \{i_1, i_2, \ldots, i_k\}$ el conjunto de los k items que más se parecen a j, que han sido evaluados por el usuario u. Esta forma de definir la vecindad es la que caracteriza a esta técnica, conocida por sus siglas en inglés como k-NN por *k-Nearest Neighbours*.

Cálculo de similitud

Existen múltiples formas de evaluar la similitud entre dos items o entre dos usuarios [22] [30], nosotros describiremos dos en particular, el *índice de correlación de Pearson* y la *similitud coseno*, ambas medidas de las más utilizadas [43] [3] [4]. Sea u el usuario activo y supongamos que queremos predecir el rating $\hat{r}_{uj}$ para un ítem j, para esto tenemos que establecer la similitud entre el ítem activo y otro

ítem i. Consideremos U_i, U_j los conjuntos de los usuarios que evaluaron cada ítem respectivamente en el conjunto de entrenamiento E, las dos medidas de similitud consideradas se pueden definir de la siguiente manera:

Definición 1.8. Dados dos items i y j, definimos la similitud coseno como:

$$\text{Sim}^{(C)}(i,j) = Cos(i,j) = \frac{\sum_{v \in U_i \cap U_j} r_{vi} \cdot r_{vj}}{\sqrt{\sum_{v \in U_i \cap U_j} r_{vi}^2} \sqrt{\sum_{v \in U_i \cap U_j} r_{vj}^2}} \tag{1.21}$$

Antes de definir la correlación de Pearson, definimos μ_i como el promedio de los ratings sobre i:

$$\mu_j = \frac{1}{|U_j|} \sum_{v \in U_j} r_{vj} \tag{1.22}$$

Definición 1.9. Consideremos μ_i el promedio de los ratings que recibió el ítem i, definimos la similitud entre dos items como el índice de *correlación de Pearson*:

$$\text{Sim}^{(P)}(i,j) = Pearson(i,j) = \frac{\sum_{v \in U_i \cap U_j} (r_{vi} - \mu_i)(r_{vj} - \mu_j)}{\sqrt{\sum_{v \in U_i \cap U_j} (r_{vi} - \mu_i)^2} \sqrt{\sum_{v \in U_i \cap U_j} (r_{vj} - \mu_j)^2}} \tag{1.23}$$

Es clara la interpretación de estas similitudes, si pensamos a los items representados como un vector $\vec{R}_{-,i} \in \mathbb{R}^n$ que es la columna i-ésima de la matriz R donde los datos faltantes los tratamos como 0 sólo a fines de interpretación geométrica y de notación, entonces por ejemplo la similitud coseno coincide con el coseno del ángulo θ_{ij} comprendido entre los vectores que representan ambos items, podemos entonces reescribirla como:

$$Cos(i,j) = \cos(\theta_{ij}) = \frac{\vec{R}_{-,i} \cdot \vec{R}_{-,j}}{\|\vec{R}_{-,i}\| \|\vec{R}_{-,ij}\|} \tag{1.24}$$

Con $\|.\| = \|.\|_2$

Cálculo de predicción

Finalmente resta calcular ahora la predicción del rating $\hat{r}_{uj}$ sobre el ítem j, para eso recordemos que $N_j(u)$ es el conjunto de k items más parecidos a j que han sido evaluados por u. Entonces la predicción P(u,j) que consideraremos será una suma pesada:

$$P(u,j) = \hat{r}_{uj} = \frac{\sum_{i \in N_j(u)} \text{Sim(i,j)} \cdot r_{ui}}{\sum_{i \in N_j(u)} |\text{Sim(i,j)}|} \tag{1.25}$$

La motivación detrás de esta suma es pesar sobre los propios ratings que ya proporcionó el usuario para los items similares, ya que dichos ratings son predictores más confiables.

En el cuadro 1 podemos ver el pseudocódigo en forma general de este algoritmo basado en items.

Algoritmo 1 k-NN

Entrada: E = conjunto de entradas de entrenamiento, k, u usuario activo, j ítem activo

Salida: Predicción $\hat{r}_{uj}$

para todo $i \in I$ **hacer**

$\mu_i \leftarrow \frac{1}{|U_j|} \sum_{v \in U_j} r_{vj}$

Calcular Sim(i,j)

fin para

Calcular $N_j(u)$

$\hat{r}_{uj} \leftarrow P(u, j)$

Variantes

Existen muchas variantes de este tipo de métodos, en general, las variantes más clásicas incluyen modificaciones en la forma del cálculo de la similitud, la forma del elegir el vecindario o como calcular la predicción [4]. En el cálculo de la predicción es usual la utilización de pesos vinculados a la frecuencia relativa de los items. Dos variantes usuales en el cálculo de la predicción del rating son centrar en la media de los ratings y también el de realizar una normalización de los ratings(Z-score) [43].

Ventajas y desventajas del método

Caben destacar algunos puntos que tienen a favor los métodos basados en vecindarios:

- Simplicidad: La implementación de estos métodos es relativamente sencilla y funciona de manera muy intuitiva, sin tener la necesidad de ajustar una gran cantidad de parámetros más que el tamaño de los vecindarios.
- Explicabilidad: El modelo en el que nos basamos es sencillo de interpretar, pero también de justificar. Por ejemplo, al usuario se le puede explicar que la recomendación de un ítem proviene de haber visto tales otros items.
- Eficiencia y estabilidad: Los métodos basados en vecindarios son algorítmicamente poco costosos y rápidos de ejecutar. No solo eso sino que también son estables al agregar nuevos usuarios y nuevos items [30].

Sin embargo, por otro lado, el alcance de esta técnica es limitado, ya que para poder calcular la similitud entre dos items(usuarios), dependemos de que estos hayan sido evaluados por varios usuarios al mismo tiempo, mientas que sabemos que dos items pueden ser similares aunque esto no haya sucedido. Además, dado que solo se pueden recomendar items clasificados por vecinos, la cobertura de tales métodos

también puede ser limitada, sobretodo sesgada a los items más vistos ya que tendrán muchas más valoraciones que los items menos populares.

Además, hay que sumar a esto la sensibilidad a la gran cantidad de datos faltantes que en esta técnica nos son tenidos en cuenta, esto representa la mayor dificultad para los sistemas de recomendación en general. En este escenario es más probable que dos usuarios(items) no tengan items evaluados en común, con lo cual nunca los podremos comparar, y así encontraremos pocos usuarios(items) parecidos.

1.4.2. Slope One

El esquema llamado *Slope One* desarrollado en [8] consiste en una técnica basada en un modelo de regresión. Supongamos que dados dos usuarios u y v con sus respectivos ratings dados como vectores filas de la matriz de ratings R, llamémoslos R_u y R_v respectivamente, donde cada elemento de cada uno de esos vectores es un rating de un ítem $i = 1, 2, \cdots, m$. La motivación de esta técnica es modelar la variación("diferencial") de la popularidad entre los items para los usuarios y luego utilizar esto como un predictor lineal para los ratings de un usuario. Veamos un ejemplo para entender la idea,

Ejemplo 1.4. Supongamos que tenemos tres items y dos usuarios A y B con los siguientes datos:

	ítem 1	ítem 2	ítem 3
Usuario A	3	5	?
Usuario B	4	3	3

Queremos predecir en este caso el rating del usuario A sobre el ítem 3, entonces lo que propone el método es mirar las diferencias de ratings para el otro usuario que conocemos B para modelar la variación de dichos items dentro de un mismo usuario. Las diferencias son:

$$\begin{aligned} Dif(B_3, B_1) &= 3 - 4 = -1 \\ Dif(B_3, B_2) &= 3 - 3 = 0 \end{aligned}$$

Luego propondremos para predecir el rating de A usar sus ratings sobre los otros items, modificados por esta diferencial de popularidad y promediaremos sobre todos los items que evaluó A, resultando:

$$P(A_3) = \frac{(A_1 + Dif(B_3, B_1)) + (A_2 + Dif(B_3, B_2))}{2} = \frac{(3 + (-1)) + (5 + 0)}{2} = 3.5$$

□

Formalmente, queremos hallar un predictor lineal de la forma $f(x) = x + b$ para predecir los ratings de v a partir de los ratings de u, de tal forma que minimice $\sum_i (r_{ui} - r_{vi})^2$. Derivamos e igualamos a cero y obtenemos $b = \sum_i \frac{r_{ui} - r_{vi}}{m}$ donde m es la cantidad de items.

Esto motiva la siguiente definición, consideremos U_{ij} el conjunto de los usuarios que calificaron ambos items, sea el desvío promedio del ítem i con respecto al ítem j definido como:

$$\text{dev}_{ji} = \frac{1}{|U_{ij}|} \sum_{u \in U_{ij}} r_{uj} - r_{ui} \tag{1.26}$$

Entonces sea un ítem i evaluado por u, resulta que $\text{dev}_{ji} + r_{ui}$ es un predictor para r_{uj}, por lo tanto es razonable definir al predictor de r_{uj} como:

$$\hat{r}_{uj} = \frac{1}{|R_j^u|} \sum_{i \in R_j^u} \text{dev}_{ji} + r_{ui} \tag{1.27}$$

Donde $R_j^u = \{i \in I_u | i \neq j, U_{ij} \neq \emptyset\}$ es el conjunto de items relacionados con el ítem j, es decir los items que fueron evaluados al menos una vez por un usuario que también evaluó a j. Finalmente queremos aproximar $\mu_u = \frac{1}{|I_u|} \sum_{i \in I_u} r_{ui} \approx \frac{1}{|R_j^u|} \sum_{i \in R_j^u} r_{ui}$, para esto necesitamos suponer que hay una cierta densidad en las evaluaciones de los datos, esto es que casi cualquier par de items (i,j) fue evaluada al menos una vez. Si pedimos esto entonces $R_j^u = I_u$ si $j \notin I_u$ y $R_j^u = I_u - \{j\}$ si $j \in I_u$ y el predictor slope one $\hat{r}_{uj}^{S1}$, despejando r_{ui}, queda definido como:

$$\hat{r}_{uj}^{S1} = \mu_u + \frac{1}{|R_j^u|} \sum_{i \in R_j^u} \text{dev}_{ji} \tag{1.28}$$

1.4.3. Factorización de matrices

Los modelos de factores latentes como la factorización de matrices es un acercamiento totalmente diferente a los descritos previamente. Estas técnicas son consideradas parte del estado del arte en lo que a filtros colaborativos basados en modelos respecta [43] y tratan de capturar características(factores) latentes que expliquen las valoraciones de los items. La intuición para la aplicación de factorización de matrices es que las porciones de las filas y columnas de la matriz de ratings R están altamente correlacionadas y como resultado, los datos tiene redundancias incorporadas, entonces la matriz de datos resultante a menudo se puede aproximar bastante bien por una matriz de bajo rango[2].

La descomposición en valores singulares (SVD) ha demostrado ser un técnica útil en identificar factores latentes en semántica en el procesamiento natural del lenguaje [20]. Sin embargo, la principal dificultad en su aplicación en los sistemas de recomendación es la alta proporción de datos faltantes de la matriz R de ratings

[2]Para profundizar sobre la intuición detrás de la factorización de matrices se puede ver en Aggarwal et. al. [43].

de usuarios-items para la cual no está definida la descomposición SVD tradicional. Frente a esta dificultad las primeras técnicas completaban la matriz para conseguir una matriz más densa de ratings. El problema con este acercamiento es la complejidad algorítmica ya que incrementar la cantidad de información de la matriz lleva a un aumento muy significativo de los tiempos de ejecución del sistema de recomendación. Para evitar este inconveniente, en los trabajos más recientes se realiza el modelado solo con los ratings observados S, no sólo mejorando los tiempos algorítmicos sino que también sorteando el problema de los datos faltantes. A continuación describiremos una de las técnicas más populares, la descomposición SVD regularizada.

SVD

Sea $R \in \mathbb{R}^{n \times m}$ la matriz de ratings usuarios-items, sabemos que podemos factorizar la matriz R de forma aproximada como[3]:

$$R \approx \mathbf{P}\mathbf{Q}^T \tag{1.29}$$

Donde $\mathbf{P} \in \mathbb{R}^{n \times s}$ y $\mathbf{Q} \in \mathbb{R}^{m \times s}$, y s es un parámetro que representa la cantidad de factores latentes. La idea de trasfondo del método es que la i-ésima fila de $\mathbf{P}$, que llamaremos el vector de factores del usuario, contiene las k entradas correspondientes a la afinidad del usuario i hacia los s factores latentes que intuitivamente representan algún concepto de los items. Las filas de $\mathbf{Q}$ son las representaciones a su vez de los s conceptos de cada ítem. En este sentido s es la dimensión del espacio de factores que codifican a las relaciones entre usuarios e items.

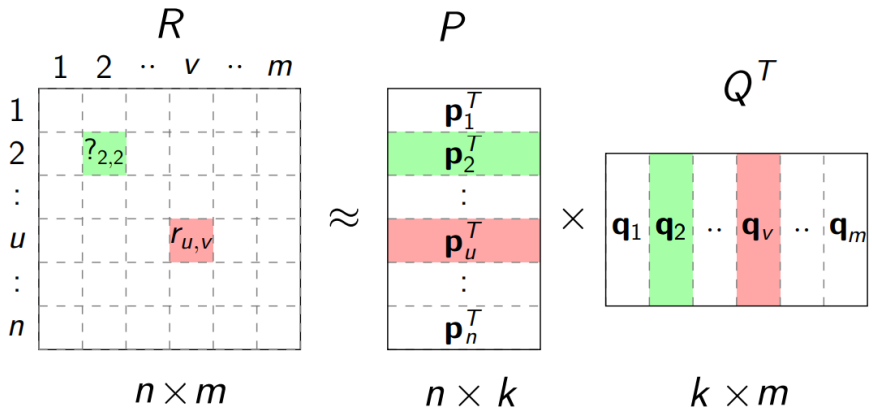


Figura 1.3: Representación de un modelo de factores latentes en general.

Entonces si R fuera una matriz densa se podría proponer el siguiente problema

[3]El término SVD no es estrictamente correcto sino que se trata de una representación de bajo rango de la matriz R [33]. Sin embargo, en el área la utilización de este término para este tipo de factorización es ampliamente aceptada.

de optimización irrestricto:

$$\min_{\text{P, Q}} \quad \frac{1}{2}\left\|R - \mathbf{P}\mathbf{Q}^T\right\|^2 \tag{1.30}$$

Donde $\|.\|$ es la norma de Frobenius. Si llamamos $\mathbf{p}_u \in \mathbb{R}^s$ a las filas de $\mathbf{P}$ y $\mathbf{q}_j \in \mathbb{R}^s$ a las filas de $\mathbf{Q}$, podemos escribir el problema de manera equivalente como:

$$\min_{P_u, Q_j} \quad \frac{1}{2}\sum_{(u,j)}(r_{uj} - p_u \cdot q_j^T)^2 = \frac{1}{2}\sum_{(u,j)}(r_{uj} - \sum_{i=1}^{s} p_{ui} \cdot q_{ji})^2 \tag{1.31}$$

Planteado de esta forma, se pueden calcular P y Q mediante algún algoritmo de optimización irrestricto como el descenso por gradiente. Entonces la predicción del rating $\hat{r}_{uj}$ será:

$$\hat{r}_{uj} = \sum_{i=1}^{s} p_{ui} \cdot q_{ji} \tag{1.32}$$

Este primer acercamiento supone, como primer dificultad, que se conocen todas las entradas de la matriz R, y aunque completemos la matriz de alguna forma, el otro problema que enfrentamos es la poca cantidad de datos, que puede llevar a que el modelo sobreajuste a los datos.

El método que consideraremos en esta tesis es el descrito en la sección 5.3.1 de [31] y mejorado en [12]. En este modelo, la función a minimizar, tendrá en cuenta sólo las entradas conocidas de la matriz separadas para entrenamiento, a diferencia de lo considerado en (1.31) donde la suma se realiza sobre todas las entradas de la matriz.

El modelo siempre trata de capturar de alguna manera las interacciones entre los usuarios y los items, pero suele haber efectos intrínsecos a cada usuario y a cada ítem que modifican su puntuación, un sesgo particular a cada uno de ellos. Por ejemplo podría ser que un usuario tenga tendencias a evaluar siempre altos los items más que otros usuarios, o tal vez que un ítem por su popularidad tenga mayor puntaje. Por esta razón es la que se introduce una predicción de base incluyendo sesgos para los usuarios $\{b_u : u \in U\}$ y para los items $\{b_j : j \in I\}$ y el promedio μ sobre todos los ratings observados.

Finalmente, se introduce un término de regularización $\lambda > 0$, para evitar el sobreajuste de los ratings. Entonces, el cálculo de los parámetros, se resuelve como un problema de minimización que tiene como función de pérdida:

$$\min_{b,p,q} \quad \sum_{(u,j)\in E}(r_{uj} - \mu - b_u - b_j - \mathbf{p}_u\mathbf{q}_j^T)^2 + \lambda(b_u^2 + b_j^2 + \|\mathbf{p}_u\|^2 + \|\mathbf{q}_j\|^2) \tag{1.33}$$

La optimización de la función de pérdida (1.33) se realiza típicamente por medio del método de descenso por gradiente estocástico (*stochastic gradient descent* SGD). Para esto necesitaremos elegir una tasa de aprendizaje $\alpha > 0$, es decir el tamaño del paso del SGD, y alguna condición de convergencia bajo la cual terminar el algoritmo. Esta condición, usualmente, depende de la cantidad de iteraciones o

depende de la norma del error. En el recuadro Algoritmo 2 encontraremos un posible pseudocódigo del algoritmo SVD con SGD.

Algoritmo 2 SVD + SGD

Entrada: Matriz de ratings R, learning rate α, λ
Salida: $\mathbf{b}, \mathbf{P}, \mathbf{Q}$

```
Iniciar aleatoriamente b, P, Q
E = {(u, j) : entradas observadas para entrenamiento}
mientras No se cumpla la condición de convergencia hacer
  Reordenar aleatoriamente E en Ẽ
  para todo (u, j) ∈ Ẽ hacer
    e_uj = r_uj − r̂_uj = r_uj − μ − b_u − b_j − p_u^T q_j
    b_u ← b_u + α(e_uj − λ · b_u)
    b_j ← b_j + α(e_uj − λ · b_j)
    p_u ← p_u + α(e_uj · q_j − λ · p_u)
    q_j ← q_j + α(e_uj · p_u − λ · q_j)
  fin para
  Calcular condición de convergencia
fin mientras
```

La constante λ, que controla el nivel de la regularización y la tasa de aprendizaje, generalmente se determinan por validación cruzada, utilizando procedimientos como una búsqueda de grilla o una búsqueda aleatoria. Por ejemplo, los valores estipulados en [31] basados en la competencia de Netflix son $\lambda = 0.02$ y $\alpha = 0.005$.

Variantes

Una modificación sencilla para este algoritmo usualmente utilizada, es diferenciar el tamaño de los pasos que realiza el gradiente estocástico, introduciendo una tasa de aprendizaje independiente para cada variable así como un parámetro de regularización para cada variable. Introducir esta variación produce mejores resultados para la predicción [13], al coste de un mayor tiempo de cómputo.

Se han aplicado numerosas mejoras a este algoritmo [12], pero entre ellas la variante más destacable, mostrando un rendimiento general superior a SVD, es el modelo desarrollado por Koren et. al. [14] conocido como SVD++. Dicho sistema realiza una normalización sobre los ratings, agrega algunas variables adicionales al modelo y luego utiliza un modelo basado en vecinos para mejorar el rendimiento del sistema. También se lo suele enmarcar dentro de una clase de modelos conocidos como *factorización no negativa de matrices* [43].

Ventajas y desventajas

La principal ventaja que poseen los modelos de factores latentes es que tienen un rendimiento muy superior a los modelos clásicos basados en items o en usuarios

[43]. No solo eso, sino que tiene una excelente escalabilidad a todo tipo de matrices y de problemas.

El principal problema que aqueja a los modelos de factores latentes, es la escasa interpretabilidad de los resultados. Al calcular el vector de factores de un usuario u, buscábamos medir de alguna forma la afinidad del usuario por las características del ítem. Pero esto no siempre se podrá interpretar como una característica explícita en todos los casos.

Una de las formas de mejorar la interpretabilidad del modelo es lo que se conoce como factorización no negativa de matrices(MNF) [43]. Pero estos modelos sufren de varias dificultades, entre las cuales está no poder aplicar el modelo para cualquier tipo de matriz.

Capítulo 2

Máquinas de factorización

A continuación presentaremos el modelo de máquinas de factorización, un predictor general introducido por S. Rendle en [21] y veremos como aplicarlo en el contexto de sistemas de recomendación. Veremos la motivación y un ejemplo de como aplicarlo. Luego lo veremos como se compara con otros métodos, contaremos como se implementa y cuál es su complejidad algorítmica.

2.1. Máquinas de Factorización

Como mencionamos antes, el problema del filtrado colaborativo puede ser visto como una generalización del problema de regresión o de clasificación, entonces tendría sentido aplicar técnicas relacionados a estas problemáticas. Una de las técnicas de clasificación más populares en el área del aprendizaje automático son las *máquinas de vectores de soporte* o SVM (Support Vector Machines) que poseen grandes ventajas como un buen funcionamiento en escenarios de datos ralos. Sin embargo, cuando se utilizan SVM con núcleos no lineales, en general no logran estimar los parámetros del modelo de manera precisa.

Inspirados en estos métodos, es como se definen las *máquinas de factorización* o *Factorization Machines* (FM), las cuales son predictores generales inspirados en la factorización de matrices y en las SVM pero que a diferencia de estos últimos, pueden estimar de manera más confiable los parámetros de un modelo cuando los datos son extremadamente ralos [21].

En las maquinas de factorización lo que trataremos de hacer de manera análoga a lo hecho en SVD, es capturar las interacciones entre las variables y para esto introduciremos una forma diferente de representar los datos en una matriz obtenida a partir de la matriz de ratings R, pensando a los ratings como una función $y(x)$ de los n usuarios y m items:

$$y(x) : \mathcal{B}^n \times \mathcal{B}^m \longrightarrow \mathcal{R}$$

Donde $\mathcal{B} = \{0, 1\}$. Para construir ahora el vector de variables x consideremos nuevamente a S el conjunto de los ratings r_{uj} observados, y llamemos $q = |S|$ la cantidad de entradas especificadas en R, entonces podemos indexar a los ratings

$S = \{r^{(1)}, r^{(2)}, \cdots, r^{(q)}\}$. Ahora consideremos el i-ésimo rating con $i = 1, \ldots, q$ correspondiente al rating r_{uj}, construiremos el vector $x^{(i)} \in \mathcal{B}^{n+m}$ de variables asociado a ese rating de la siguiente manera: en las primeras n posiciones del vector indicaremos el usuario que dio el rating con un 1 y el resto de las entradas serán 0, y las siguientes m indicarán el ítem evaluado de la misma forma, quedando entonces el vector de variables de la siguiente forma:

$$r_{uj}^{(i)} = x^{(i)} = (\underbrace{0, \cdots, 0, \overbrace{1}^{\text{u-ésimo usuario}}, 0, \cdots,}_{\text{Usuarios}} \underbrace{0, \cdots, 0, \overbrace{1}^{\text{j-ésimo ítem}}, 0, \cdots, 0}_{\text{items}}) \tag{2.1}$$

Luego la matriz del modelo tendrá tantas filas como observaciones tenga la matriz R. Los ratings y, pensados como función de x, los guardaremos en un vector $y \in \mathbb{R}^q$ donde la i-ésima posición es el i-ésimo rating, es decir $y_i = r_{uj} = y(x^{(i)})$. Una gran ventaja que representa modelar los datos de esta manera es conseguir deshacernos del problema de valores faltantes de la matriz de ratings, así, construimos una nueva matriz rala del modelo X con todas sus entradas especificadas.

Ejemplo 2.1. Consideremos un sistema de recomendación con 5 usuarios y 5 items y T ratings conocidos, la matriz de datos tendría la siguiente forma:

	Usuarios					items					Rating
x	u_1	u_2	u_3	u_4	u_5	i_1	i_2	i_3	i_4	i_5	r
$x^{(1)}$	1	0	0	0	0	0	0	1	0	0	3
$x^{(2)}$	1	0	0	0	0	0	0	0	1	0	5
$x^{(3)}$	0	1	0	0	0	0	1	0	0	0	1
$x^{(4)}$	0	0	1	0	0	0	0	0	0	1	2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$x^{(T)}$	0	0	0	0	1	0	0	0	1	0	4

Donde la observación $x^{(1)}$ representa que el usuario u_1 califico al ítem i_3 con tres puntos y así sucesivamente. □

Entonces, estructurando los datos de esta forma, podemos ver al problema de filtrado colaborativo como un problema de regresión en el cual queremos estimar los ratings a partir del conjunto de datos originales S, reescritos como $S = \{(x^{(1)}, y_1), (x^{(2)}, y_2), \cdots, (x^{(q)}, y_q)\}$ donde los usuarios e items representan las variables regresoras, y lo que queremos predecir es la función de rating y. En este contexto definimos las máquinas de factorización:

Definición 2.1. Dado $x \in \mathbb{R}^p$, (donde $p = n + m$ en nuestro ejemplo) el vector de variables, definimos el modelo de una *máquina de factorización* de grado $d = 2$ como:

$$y(x) = w_0 + \sum_{i=1}^{p} w_i x_i + \sum_{i=1}^{p} \sum_{j=i+1}^{p} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j \tag{2.2}$$

Donde $w_0 \in \mathbb{R}$, $\mathbf{w} \in \mathbb{R}^p$ y $\mathbf{v}_i$ es la i-ésima fila de $\mathbf{V} \in \mathbb{R}^{p \times s}$, con s un parámetro que define la dimensión de la factorización, la cantidad de factores latentes de la misma forma que vimos en SVD. De esta forma, una máquina de factorización de grado 2 trata de capturar todas las interacciones de a pares entre las variables. En este modelo, los hiperparámetros a estimar representan:

i w_0 es el sesgo global

ii w_i representa el peso de la i-ésima variable.

iii $w_{ij} = \langle \mathbf{v}_i, \mathbf{v}_j \rangle$ modela la interacción entre la i-ésima variable y la j-ésima.

El modelo planteado de esta manera, coincide con el modelo de SVD planteado en (1.33) al evaluar el predictor (2.2) en una observación x, como los únicos valores no nulos corresponden al usuario u y al ítem j, el modelo queda:

$$y(x) = w_0 + w_u + w_i + \langle \mathbf{v}_u, \mathbf{v}_j \rangle \tag{2.3}$$

Sin embargo, el principal interés en las máquinas de factorización viene dado por la posibilidad de incluir en el modelo otras variables que involucren más que solo usuarios e items, como veremos más adelante.

Generalización

La definición de una FM de grado 2 se puede generalizar a un grado $d \in \mathbb{N}$ en general como se puede ver en [21] y en [37]. La ventaja detrás de utilizar una FM de mayor grado es ver interacciones de mayor orden entre las variables del modelo, por ejemplo, una FM de grado 3, incluirá la interacción de tres variables a la vez $x_i x_j x_k$, lo cual puede mejorar la expresividad del modelo. Sin embargo el costo algorítmico a priori será mayor a medida que se aumente el grado de la FM.

2.2. Predicción sensible-de-contexto

El gran potencial de las máquinas de factorización se encuentra en la posibilidad de agregar información contextual (categórica o no) en el vector de variables x de una manera práctica dentro de la matriz del modelo. Este tipo de técnicas son conocidas como *sistemas sensibles al contexto* [43] [27]. En general las variables de contexto que se suelen utilizar, involucran tanto información del usuario, del ítem, temporal o del resto del los otros ratings que se conocen del usuario. Por ejemplo si se trata de un sistema de recomendación de inmuebles podríamos saber que cantidad de habitaciones tienen los mismos o la superficie de la propiedad, en el caso de películas podríamos saber a que género pertenece la misma, el director, los actores, etc. En cuanto a la información que se suele utilizar respecto a los usuarios están la edad, el sexo, educación, entre otros.

Formalmente, supongamos que tenemos d conjuntos de variables categóricas (las variables también pueden ser continuas) $C_1, \ldots, C_d$, cada uno de ellos con $c_1, \ldots, c_d$ categorías, de tal forma que la función de ratings y a estimar es:

$$y : \mathcal{B}^n \times \mathcal{B}^m \times C_1 \times \ldots \times C_d \to \mathcal{R} \tag{2.4}$$

Donde $x \in \mathbb{R}^{n+m+c_1+\ldots+c_d}$, por ejemplo si sólo tuviéramos una variable categórica C con |C| la cantidad de categorías, el vector resultante sería:

$$x^{(i)} = (\underbrace{0, \ldots, 0, 1, 0, \ldots, 0}_{n \text{ (Usuarios)}}, \underbrace{0, \ldots, 0, 1, 0, \ldots, 0}_{m \text{ (items)}}, \underbrace{0, \ldots, 0, 1, 0, \ldots, 0}_{|\mathrm{C}|}) \tag{2.5}$$

Esto forma de escribir los datos se conoce como *one-hot encoding*. Al incluir información contextual, podremos capturar interacciones no solo entre items y usuarios, sino también las interacciones entre los atributos de los items y de los usuarios, y toda otra información contextual que incluyamos.

Ejemplo 2.2. Supongamos nuevamente que tenemos un sistema de recomendación de películas[1] donde tenemos el sistema almacena el rating que le otorga un usuario $u \in U$ a una película $i \in I$ en un determinado momento $t \in \mathbb{R}$. Supongamos que tenemos el siguiente conjunto de usuarios y películas:

$$\begin{aligned} U = &\ \{\text{Alicia(A)},\ \text{Bruno(B)},\ \text{Cecilia(C)}, \ldots\} \\ I = &\ \{\text{Titanic(TI)},\ \text{Nothing Hill(NH)},\ \text{Star Wars(SW)},\ \text{Star Trek(ST)}, \cdots\} \end{aligned}$$

Y supongamos que los ratings observados con año y mes son:

$$\begin{aligned} S = \{&(A, TI, 2010-1, 5), (A, NH, 2010-2, 3), (A, SW, 2010-4, 1), \\ &(B, SW, 2009-5, 4), (B, ST, 2009-8, 5), \\ &(C, TI, 2009-9, 1), (C, SW, 2009-12, 5)\} \end{aligned}$$

Entonces la matriz de los datos quedaría como figura en el recuadro 2.1. En este caso consideramos como variables adicionales no solo el tiempo en el cual se evaluó la película, sino que también tomamos en cuenta el orden y el resto de películas evaluadas. En el caso del tiempo lo hemos codificado en meses desde Enero del 2009, aunque hay muchas otras formas de codificarlo. Por ejemplo miremos la segunda fila de las variables $x^{(2)}$, Alicia califico a Nothing Hill con 3 puntos 14 meses después de la fecha de referencia, es decir, en Febrero del 2010, habiendo solo evaluado hasta ese momento Titanic. Como solo poseemos tres observaciones para Alicia, esta observación representa el 33 % del total, entonces en amarillo representamos las tres películas que ha calificado (Titanic, Nothing Hill y Star Wars) y redondeamos dicho valor a 0.3 para cada una de la observaciones. □

[1] Este ejemplo fue extraído de Rendle [21]

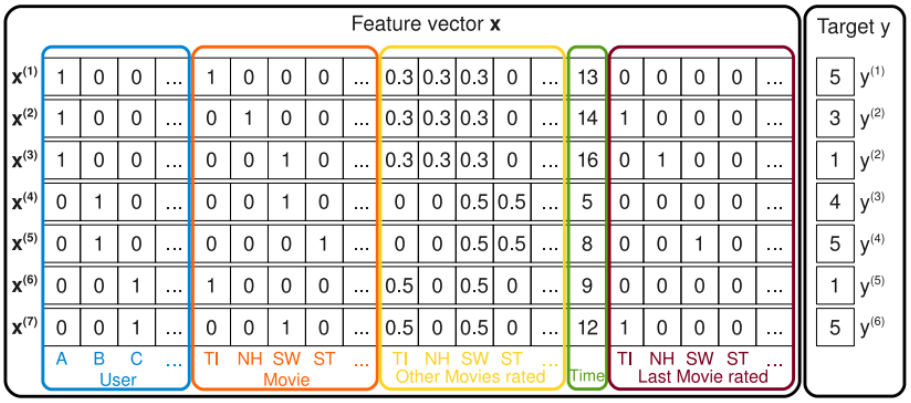


Figura 2.1: Así es como quedaría la matriz del ejemplo con la estructura de datos descrita. En azul vemos la primeras n columnas que corresponden a los usuarios, en naranja las m columnas que corresponden a las películas, en amarillo nuevamente m columnas que representan todos los items evaluados por el usuario de dicha observación. Por ultimo en verde la codificación temporal de la observación y el ultimo grupo, en marrón, representa la anterior película vista antes de la película activa correspondiente a dicho vector. Imagen extraída de [21].

2.2.1. Relación con regresión polinomial

A primera vista el modelo propuesto de las máquinas de factorización de orden tiene una ecuación similar a un modelo de regresión polinomial de segundo grado (o una SVM con núcleo de orden 2):

$$y^{(RP)}(x) = w_0 + \sum_{i=1}^{p} w_i x_i + \sum_{i=1}^{p} \sum_{j=1}^{p} w_{i,j} x_i x_j \tag{2.6}$$

Donde los parámetros del modelo a estimar son:

$$w_0 \in \mathbb{R}, \quad \mathbf{w} \in \mathbb{R}^p, \quad \mathbf{W} \in \mathbb{R}^{p \times p}$$

Si comparamos este modelo con el modelo que describimos en la ecuación (2.2), la diferencia que surge entre ellos es la forma de parametrizar las interacciones $w_{i,j}$. Para las regresión cuadrática estas interacciones son a priori independientes, mientas que en las FM asumimos que estas interacciones se pueden factorizar de la forma $\langle \mathbf{v}_i, \mathbf{v}_j \rangle$, ya que supusimos que la matriz de datos es de bajo rango($w_{i,j} \approx \langle \mathbf{v}_i, \mathbf{v}_j \rangle$). Por ejemplo, para un modelo de regresión clásico no existe una relación a priori entre dos interacciones $w_{i,j}$ y $w_{i,l}$, mientras que en las FM, al poder factorizar ambas como $\langle \mathbf{v}_i, \mathbf{v}_j \rangle$ y $\langle \mathbf{v}_i, \mathbf{v}_l \rangle$ respectivamente, existirá una dependencia que estará dada por $\mathbf{v}_i$.

Como resultado de esta suposición, las FM tiene como ventaja tener que estimar menos parámetros, en la regresión polinomial estaremos estimando $\mathbf{W} \in \mathbb{R}^{p \times p}$ por cual el orden de parámetros es $O(p^2)$. Mientras que las FM tendrán que estimar $O(ps)$ parámetros, que son muchos menos ya que s es la cantidad de factores latentes que estamos suponiendo que tiene $\mathbf{W}$.

2.3. Cálculo del modelo

De forma similar a lo que hemos planteado para SVD, para calcular el modelo queremos considerar una función de pérdida a minimizar. En este caso consideraremos la función de pérdida cuadrática:

$$l(y, \hat{y}) = \|y - \hat{y}\|_2 \tag{2.7}$$

Donde y es el rating e $\hat{y}$ es nuestro predictor de orden dos. Introduciremos nuevamente un término de regularización dado por $\lambda > 0$ para evitar que nuestro modelo sobreajuste a los datos, resultando de esta manera la función de perdida a optimizar:

$$\sum_{i=1}^{q} l(y_i, \hat{y}(x_i)) + \frac{\lambda}{2}(\|w\|^2 + \|\mathbf{W}\|^2) \tag{2.8}$$

Para calcular los parámetros del modelo, podemos utilizar el descenso estocástico por gradiente o algún otro algoritmo de minimización. Para esto necesitamos calcular el gradiente de una FM de grado 2:

$$\frac{\partial}{\partial\theta}\hat{y}(x) = \begin{cases} 1, & \text{si } \theta = w_0 \\ x_i, & \text{si } \theta = w_i \\ x_i(\sum_{j=1}^{p} v_{jk}x_j) - v_{ik}x_i^2, & \text{si } \theta = v_{ik} \end{cases} \tag{2.9}$$

Entonces si llamamos $e(x) = y(x) - \hat{y}(x)$ los pasos de actualización del SGD estarán dados por:

$$\theta \leftarrow \theta(1 - \alpha\lambda) + \alpha \cdot e(x) \cdot \frac{\partial}{\partial\theta}\hat{y}(x) \tag{2.10}$$

Donde $\alpha > 0$ es la tasa de aprendizaje, que determina la longitud de paso del algoritmo.

La complejidad del modelo se encuentra en $O(sp^2)$, donde p representa la cantidad de variables y s es la dimensión de la factorización. Sin embargo veamos que puede computarse en orden lineal respecto a s y a p debido a la simetría del término cuadrático, es decir en $O(sp)$, para eso primero hagamos la siguiente observación:

Lema 2.2. *El cálculo de la ecuación* (2.2) *se puede realizar en O(sp).*

Demostración. Consideremos la siguiente igualdad:

$$\sum_{i=1}^{p}\sum_{j=1}^{p}\langle \mathbf{v}_i, \mathbf{v}_j\rangle x_i x_j = \sum_{i=1}^{p}\sum_{j=i+1}^{p}\langle \mathbf{v}_i, \mathbf{v}_j\rangle x_i x_j + \sum_{i=1}^{p}\sum_{j=1}^{i}\langle \mathbf{v}_i, \mathbf{v}_j\rangle x_i x_j \tag{2.11}$$

Si desarmamos el segundo término obtenemos que:

$$\sum_{i=1}^{p}\sum_{j=1}^{i}\langle \mathbf{v}_i, \mathbf{v}_j\rangle x_i x_j = \sum_{i=1}^{p}\sum_{j=1}^{i-1}\langle \mathbf{v}_i, \mathbf{v}_j\rangle x_i x_j + \sum_{i=1}^{p}\langle \mathbf{v}_i, \mathbf{v}_i\rangle x_i x_i \tag{2.12}$$

Podemos hacer un cambio de indices en el primer término de la sumatoria $j = i - 1 \Rightarrow i = j + 1$, y nos queda que:

$$\sum_{i=1}^{p}\sum_{j=1}^{i-1}\langle \mathbf{v}_i, \mathbf{v}_j\rangle x_i x_j = \sum_{j=1}^{p}\sum_{i=j+1}^{p}\langle \mathbf{v}_j, \mathbf{v}_i\rangle x_j x_i = \sum_{i=1}^{p}\sum_{j=i+1}^{p}\langle \mathbf{v}_i, \mathbf{v}_j\rangle x_i x_j \tag{2.13}$$

Reemplazando (2.12) y (2.13) en (2.11) y despejando obtenemos:

$$\sum_{i=1}^{p}\sum_{j=i+1}^{p}\langle \mathbf{v}_i, \mathbf{v}_j\rangle x_i x_j = \frac{1}{2}\sum_{i=1}^{p}\sum_{j=1}^{p}\langle \mathbf{v}_i, \mathbf{v}_j\rangle x_i x_j - \frac{1}{2}\sum_{i=1}^{p}\langle \mathbf{v}_i, \mathbf{v}_i\rangle x_i x_i \tag{2.14}$$

$$= \frac{1}{2}\Big(\sum_{i=1}^{p}\sum_{j=1}^{p}\sum_{l=1}^{s} v_{il}v_{jl}x_i x_j - \sum_{i=1}^{p}\sum_{l=1}^{s} v_{il}^2 x_i^2\Big) \tag{2.15}$$

$$= \frac{1}{2}\sum_{l=1}^{s}\Big(\Big(\sum_{i=1}^{p} v_{il}x_i\Big)\Big(\sum_{j=1}^{p} v_{jl}x_j\Big) - \sum_{i=1}^{p} v_{il}^2 x_i^2\Big) \tag{2.16}$$

$$= \frac{1}{2}\sum_{l=1}^{s}\Big(\Big(\sum_{i=1}^{p} v_{il}x_i\Big)^2 - \sum_{i=1}^{p} v_{il}^2 x_i^2\Big) \tag{2.17}$$

El término final tiene complejidad del orden de $O(ps)$. ■

Entonces, gracias al Lema 2.2 si $x \in \mathbb{R}^p$ podemos reescribir a $\hat{y}$ como:

$$\hat{y}(x) = w_0 + \sum_{i=1}^{p} w_i x_i + \frac{1}{2}\Big[\sum_{l=1}^{s}\Big(\Big(\sum_{i=1}^{p} v_{il}x_i\Big)^2 - \sum_{i=1}^{p} v_{il}^2 x_i^2\Big)\Big] \tag{2.18}$$

Si para cada $l = 1, \ldots, s$ calculamos previamente en cada paso el valor de $\sum_{i=1}^{p} v_{il}x_i$, podemos calcular el valor del gradiente en tiempo constante. Entonces para cada iteración que haga SGD sobre el conjunto de entrenamiento E, la complejidad será de $O(ps)$.

Capítulo 3

El problema del arranque en frío

En este capítulo introduciremos el problema de arranque en frío en sus dos versiones, contaremos la forma más clásica de mitigarlo a través de entrevistas y discutiremos las ventajas y desventajas de estas técnicas. Luego propondremos un modelo basado en máquinas de factorización, utilizando información contextual del usuario para poder mejorar las predicciones en este escenario.

3.1. Estableciendo el problema

Cuando un usuario (o un ítem) ingresa por primera vez al sistema, el mismo no posee ninguna información referente a sus intereses. Debido a esto, un sistema basado en filtros colaborativos, no puede presentar items al usuario que sean de su preferencia. Esta, es una de las mayores dificultades en los sistemas de recomendación y es lo que se conoce como el problema del arranque en frío o *cold-start*[1].

Existen dos formas clásicas del problema de arranque en frío:

Usuario Nuevo Un nuevo usuario ingresa al sistema por primera vez, para el cual no tenemos ningún historial de evaluaciones previas sobre los items del sistema.

ítem Nuevo Un nuevo ítem es agregado al sistema, por ejemplo en un sistema de recomendación de libros podría tratarse de un nuevo lanzamiento o en el caso de películas, una película estreno. Al igual que en el caso del usuario, al ser nuevo el ítem ningún usuario lo ha evaluado, por lo tanto no poseemos ratings previos en el historial del sistema que nos permita recomendarlo.

Bajo cualquiera de estos escenarios, el filtrado colaborativo puro, es decir sin información contextual, en general, no puede conseguir resultados robustos sobre la interacción entre los usuarios y los items, ya que no hay información disponible sobre la preferencia del usuario que sirva de base para las recomendaciones. En esta tesis nos centraremos en el problema de arranque en frío para un nuevo usuario, que de ahora en adelante llamaremos el problema del nuevo usuario por simplificación.

[1] También se suele utilizar el término de Cold-Start para los casos de items (Cold ítem) o usuarios (Cold User) para los cuales se conocen muy pocos ratings.

Existen múltiples acercamientos al problema, por ejemplo las técnicas basadas en contenido puede ayudar a cerrar la brecha de usuarios(o items) existentes a nuevos usuarios(o items), al inferir similitudes entre ellos a partir de características de los usuarios(items) como puede ser información contextual o información geográfica [6] [40]. Por lo tanto, pueden hacer recomendaciones para usuarios nuevos que tienen características similares a las de otros usuarios para los cuales sí conocemos su comportamiento. También es usual encontrar sistemas que utilizan técnicas basadas en conocimiento, es decir que recurren a la interacción con el usuario, pidiendo que evalúe un cierto conjunto de items como parte del registro en el mismo sistema, por ejemplo un sistema que utiliza esta modalidad es el de Netflix o el de MovieLens[2] [5].

3.2. Modelado del usuario vía entrevista

Uno de los métodos más populares para el tratar el problema del nuevo usuario es el de utilizar una entrevista con el usuario antes de que comience efectivamente a utilizar el sistema. En esta entrevista, le iremos presentando items para que el usuario evalúe y así poder ir recolectando información acerca de sus preferencias que podremos utilizar en nuestro modelo. La principal razón de la popularidad de esta metodología se debe a que no es computacionalmente costosa, y a la vez podremos utilizar las verdaderas preferencias del usuario, para poder predecir su comportamiento en general. Sin embargo, este escenario trae consigo dos principales cuestiones que son críticas, primero, cómo elegir los items para presentar al usuario, y segundo, qué cantidad de items le requeriremos al usuario que evalue antes de terminar la entrevista.

Para responder estas cuestiones, tenemos que tener en cuenta dos aspectos. Para empezar, los items que vayamos a presentar tienen que tener un cierto valor 'informativo' en el siguiente sentido: si presentamos un ítem que no posee valoraciones negativas del todo o viceversa, esto es, un ítem sobre el que haya un consenso, preguntarle al usuario acerca de dicho ítem puede carecer de importancia. Por ejemplo, en un sistema de recomendación de películas, preguntar por un 'clásico' como Volver al Futuro(1985)[3] puede no ser tan informativo como preguntar por otra película que tenga más variabilidad en sus ratings.

Por otro lado, hay que también considerar la dificultad del proceso de entrevista, ya que si es muy costoso para el usuario, léase muchos items para evaluar, el usuario no utilizará el sistema o abandonará la entrevista en el transcurso de la misma [5]. Los motivos por los cuales una entrevista puede prolongarse demasiado, es que los items que le son presentados al usuario para evaluar no fueron vistos por el mismo. Esto motiva a tener en cuenta la popularidad de los items a presentar, bajo la premisa de que mientras más popular sea un ítem, más probables es que nuestro nuevo usuario la haya visto. Entonces, ya no sólo es importante el rendimiento en la

[2]https://movielens.org/

[3]Según Google, a más del 95 % de los usuarios de Google les gustó Volver al Futuro

predicción que pueda hacer el sistema de los gustos de un usuario sino que también tendremos que tener presente el esfuerzo del usuario.

3.2.1. Selección no personalizada

La selección de items no personalizada, simplemente establece que para cada usuario le mostraremos los mismos items sin utilizar ninguna otra información personal.

Popularidad

Utilizando la información de la cantidad de valoraciones que tiene cada ítem en el sistema, clasificaremos a los items en forma descendente desde el más visto hasta el menos visto. Una vez hecho esto, iremos presentando los items en orden de popularidad hasta cumplir con la cuota de items a evaluar.

Observación: Es claro, que esta estrategia tiene como principal motivador lograr que el usuario consiga calificar rápidamente la cantidad preestablecida de items que pediremos evaluar. Sin embargo, a priori, podría sufrir del problema mencionado anteriormente de que los items presentados no sean informativos. Por otro lado, es posible que al preguntar por los items más populares, estemos agregando un gran sesgo. Esto es decir, los items que son más populares, son los más fáciles de recomendar ya que poseen una gran cantidad de evaluaciones. Mientras que los items que tienen pocas evaluaciones sufren exactamente lo contrario, ya que una mala evaluación tendrá mayor peso para un ítem con pocas evaluaciones que para uno popular [5].

Entropía

La entropía es una medida utilizada para medir el contenido de información teórico que posee un ítem [2] [15]. La entropía de un ítem i la definimos de la siguiente forma:

$$Ent(i) = \sum_{r \in \mathcal{R}} P(r_i = r) \cdot \log(P(r_i = r)) \tag{3.1}$$

Donde $P(r_i = r)$ es la proporción de valoraciones r que tuvo el ítem i sobre todas las valoraciones que recibió. Mientras más grande sea la entropía de un ítem, mayor será el contenido potencial de información [5].

Luego de calcular esto para cada ítem, ordenamos en forma descendente los items para presentárselos al usuario por el valor de su entropía, ya que a mayor entropía, mayor es el supuesto contenido teórico de dicho ítem.

Observación: El acercamiento por entropía supone dos dificultades que no son claras de manejar. La primera es que hacer con los valores faltantes de evaluaciones del ítem que sabemos que se deben a múltiples factores. Segundo, ordenar por entropía supone que el ítem es más informativo, sin embargo, puede suceder que un ítem con dos o tres ratings diferentes puede tener mayor entropía que uno que

posea muchos más ratings pero con menor dispersión. De esta manera, utilizar la entropía para medir la información no siempre va a coincidir con una mayor utilidad al sistema.

Estrategia Balanceada

Como vimos anteriormente, la entropía pura puede terminar preguntando por items que muy poca gente ha visto por lo cual es difícil que el usuario nuevo pueda evaluarlo, y la popularidad puede no ser informativa. Es decir que estas dos características no parecen ser dependientes una de la otra [5]. Esto nos motiva a considerar la siguiente medida propuesta en Rashid et al [5] para ordenar los items a presentar:

$$LogPopEn(i) = \log(Pop(i)) \cdot Ent(i) \tag{3.2}$$

Siendo $Pop(i)$ la popularidad del ítem i medida por cantidad de usuarios que evaluaron dicho ítem. Una vez calculada esta medida, se ordena de manera descendente los items según este valor obtenido y se los presentaremos al usuario.

Observación: Como dijimos anteriormente, existe un intercambio y posible independencia entre la entropía y la popularidad, lo cual esta estrategia balanceada apunta a capturar. Podríamos considerar también no utilizar el logaritmo en la popularidad pero en general la popularidad es varias magnitudes mayor a la entropía, por lo cual dominaría completamente nuestra medida sobre los items.

Aleatoria

En la selección aleatoria, presentaremos los items con probabilidad uniforme sobre todos los items que posee el sistema. Esta estrategia la utilizaremos para poder comparar el rendimiento de los otros métodos.

3.2.2. Selección personalizada

En la selección personalizada, el objetivo es que cada valoración que el usuario aporte sobre un ítem sea utilizada para seleccionar el/los próximos items a presentarle para evaluar.

Selección ítem-ítem

En esta estrategia [5], presentaremos items con alguna de las estrategias antes enunciadas hasta que el usuario evalúe un ítem que conoce. Una vez evaluado ese ítem, calculamos su similitud con otros items y comenzamos a presentarle items que se encuentran en la vecindad de los que ya ha evaluado. A medida que el usuario logra evaluar más items, vamos actualizando la lista, sin repetir los items que ya hemos mostrado.

Observación: Esta técnica apunta nuevamente a encontrar de manera rápida items que el usuario pueda evaluar, actualizando los items a mostrar gracias a los ratings que el usuario ha provisto. Sin embargo, existe una posible desventaja, al

mostrar al usuario, items similares a los que ya ha evaluado, puede suceder que el sistema muestre items demasiado correlacionados entre sí. Por ejemplo, en un sistema de recomendación de películas, si el usuario ya ha evaluado a *Star Wars*-Episodio IV(1977), ya sea positivamente o no, es altamente probable que *Star Wars*-Episodio V(1980) u otra película de la saga, pertenezca a la vecindad de la misma y que el usuario tenga la misma preferencia por dicha película. De esta manera, colectaríamos demasiadas películas que le gusten o demasiadas películas que no le gusten al usuario.

3.3. Alternativas

El proceso de entrevistas, posee dos grandes desventajas:

- **Re-entrenamiento**: Para poder realizar recomendaciones a los usuarios, utilizamos los resultados de las entrevistas para re-entrenar el modelo y poder realizar predicciones con el nuevo modelo. Realizar esto no resulta escalable para grandes sistemas de recomendación
- **Dificultad para el Usuario**: Aunque el objetivo de las diferentes formas de seleccionar los items es también el de minimizar el esfuerzo que le requiere al usuario poder evaluar items, puede que esto no sea algo que el usuario este dispuesto a realizar, lo cual podría hacer que desista de utilizar el sistema.

Bajo estas dificultades existen múltiples enfoques que utilizan variables de contexto relacionadas al usuario y a los items para poder realizar recomendación sin utilizar las entrevistas [40]. De esta forma evitamos ambos problemas pero a costo de perder precisión en las predicciones.

Propuesta

En este contexto, propondremos una forma diferente de modelar un usuario, inspirados en la técnica de vecinos más cercanos y aprovechando las facilidades y versatilidad que otorgan las máquinas de factorización para involucrar este tipo de información. Para esto lo que haremos será construir un vector x de variables, utilizando su información contextual y sus vecinos en un sentido similar al de kNN.

Formalmente, supongamos que u es un usuario nuevo que ingresa a un sistema ya entrenado con una máquina de factorización que utiliza información contextual del usuario. En este caso, nuestro modelo asumirá que la función de ratings depende del ítem, usuario y otras d variables de contexto como planteamos en la ecuación (2.4) $y(x) : \mathbb{R}^p \rightarrow \mathcal{R}$, donde $p = n + m + d$. Supondremos que esta información relacionada a las características del usuario(edad, sexo, profesión, dirección, ingresos, etc.) la podemos cuantificar de alguna forma en un vector $f(u) \in \mathbb{R}^d$, donde d es la cantidad de características a las que tenemos acceso ya preprocesado. Por ejemplo, si poseemos la profesión podemos asignarles etiquetas numéricas, o si se trata de la dirección podemos codificarlo con etiquetas de provincias, o regiones.

Si quisiéramos ahora calcular la predicción del rating para ese usuario sobre un ítem j, podemos utilizar nuestro modelo ya entrenado $\hat{y}$, donde el vector $x^{(u,j)} \in \mathbb{R}^p$ es:

$$x^{(u,j)} = (\underbrace{0,\ldots,0}_{\text{Usuario}}, \underbrace{0,\ldots,0,1,0,\ldots,0}_{\text{j-ésimo ítem}}, \underbrace{f_1,\ldots,f_d}_{\text{f(u)}}) \tag{3.3}$$

Las primeras n posiciones, tendría solo ceros ya que el usuario no se encuentra en el sistema, las siguientes m posiciones codifican el ítem y las ultimas d posiciones caracterizan las variables del usuario. Las únicas interacciones que tendría $\hat{y}(x^{(u,j)})$ serían con el mismo ítem y los usuarios de iguales características, por lo que podríamos estar perdiendo mucha expresividad del modelo, que quedaría:

$$\hat{y}(x^{(u,j)}) = w_0 + w_j + \sum_{t=1}^{d} w_t f_t + \sum_{t=1}^{d}\sum_{t'>t}^{d} \langle \mathbf{v}_t, \mathbf{v}_{t'} \rangle f_t f_{t'} \tag{3.4}$$

Donde $w_0 \in \mathbb{R}, \quad \mathbf{w} \in \mathbb{R}^p, \quad \mathbf{V} \in \mathbb{R}^{p\times s}$, con s cantidad de factores considerados por nuestro modelo. Notar el pequeño abuso de notación que hicimos en $\sum_{t=1}^{d} w_t f_t$ donde w_t representan a las coordenadas de w que corresponden a las f_t, es decir que en realidad w_t es un w_{j_t} con $j_t = p - t + 1, \ldots, p$. Realizamos el mismo abuso en los $\mathbf{V}_t$.

Lo que propondremos entonces, es construir un vector x para representar al usuario nuevo, utilizando los usuarios que más se parezcan a el, con respecto a sus variables demográficas. De esta forma estamos recurriendo a la misma lógica que describimos con kNN, esto es, asumir que usuarios con características personales similares, tendrán similares preferencias sobre los items.

Para esto, consideremos $N_c(u)$ el conjunto de los c usuarios más cercanos (según alguna métrica a definir) al usuario u dentro del conjunto del los usuarios que están en el sistema. Entonces pensaremos al usuario u, el vector con el cual lo representaremos, como una combinación de sus vecinos en $N_c(u)$. La primer forma que consideraremos es modelar con pesos uniformes a los c usuarios más similares, entonces el vector de representación para un ítem j en las primeras n coordenadas (las correspondientes a los usuarios) estarán dadas por:

$$\bar{x}_v^{(u,j)} = \begin{cases} 1/c & \text{si } v \in N_c(u) \\ 0 & \text{caso contrario} \end{cases} \tag{3.5}$$

De esta forma el vector $x^(u, j)$ que utilizaremos será:

$$x^{(u,j)} = (\underbrace{\bar{x}^{(u,j)}}_{\text{Usuario}}, \underbrace{0,\ldots,0,1,0,\ldots,0}_{\text{j-ésimo ítem}}, \underbrace{f_1,\ldots,f_d}_{\text{f(u)}}) \tag{3.6}$$

Luego la predicción para este usuario quedará de la siguiente forma:

$$\hat{y}(x^{(u,j)}) = w_0 + w_j + \frac{1}{c}\sum_{v\in N_c(u)} w_v + \sum_{t=1}^{d} w_t f_t + \sum_{t=1}^{d}\sum_{t'>t}^{d}\langle \mathbf{v}_t, \mathbf{v}_{t'}\rangle f_t f_{t'} + \frac{1}{c^2}\sum_{v\in N_c(u)}\sum_{v'\in N_c(u):v'>v}\langle \mathbf{v}_v, \mathbf{v}_{v'}\rangle \tag{3.7}$$

Si comparamos ambas predicciones, (3.4) y (3.7) vemos que aparecen ahora dos términos extras que involucran la relación con los vecinos del usuario u pesadas sobre la cantidad de vecinos que hemos considerado. Esta aproximación, al involucrar más interacciones entre las variables, tiene la posibilidad de mejorar la expresividad del modelo en este contexto de arranque en frío.

Una segunda propuesta que también consideramos, de manera exactamente análoga a lo recién descrito, es cambiar la forma en la que pesamos los usuarios vecinos del usuario activo. En vez de pesarlos de manera idéntica, lo que haremos será considerar los inversos de las distancias más algún parámetro de contracción $\beta \geq 1$, de esta forma el vector de variables quedaría codificado para los usuarios como:

$$\bar{x}_v^{(u,j)} = \begin{cases} \frac{1}{d(u,v)+\beta} & \text{si } v \in N_c(u) \\ 0 & \text{caso contrario} \end{cases} \tag{3.8}$$

Al hacer esto lo que logramos, es darle mayor importancia al usuario más similar a u y quitarles peso a los usuarios que no sean tan parecidos. En este caso la predicción quedaría muy similar a (3.7) pero en vez de tener los pesos de $\frac{1}{c}$ fuera de las sumatorias, aparecerían los inversos de las distancias dentro de las sumas.

El término de contracción que utilizaremos será $\beta = \sqrt{c}$, el cual tiene varios objetivos como el de evitar dividir por cero. Pero también nos sirve en el caso de que por ejemplo el usuario tenga muchos usuarios que se encuentran a distancia 0 ya que en ese caso los términos que involucran a los usuarios quedarían sumados entre sí, sin realizar un promedio, lo cual nos llevaría a predicciones erróneas.

La distancia que consideraremos entre u y v será la norma L_1 ($\|.\|_1$) de la diferencia entre los vectores de variables de usuarios $f(u)$:

$$d(u,v) = \|f(u) - f(v)\|_1 = \sum_{j=1}^{d} |f(u)_j - f(v)_j| \tag{3.9}$$

También podemos considerar utilizar la norma L_2, lo cual dependerá del problema que se quiere resolver. Si las variables de contexto son categóricas, utilizaremos la norma L_1.

Capítulo 4

Resultados experimentales

En este capítulo expondremos los procedimientos llevados a cabo para evaluar el rendimiento en el problema de predicción, ranking y arranque en frío de las máquinas de factorización de orden 2. Luego debatiremos los resultados obtenidos y los compararemos con las técnicas clásicas descritas en el capítulo uno: kNN, Slope One y SVD.

4.1. Datos

Para los experimentos utilizaremos el conjunto de datos de *MovieLens-100k* [38] de ratings de un sistema de recomendación de películas implementado por el grupo de investigación *GroupLens* de la Universidad de Minnesota. Este conjunto de datos contiene 100000 ratings del 1 al 5 que dieron 943 usuarios sobre 1642 películas, en el cual cada usuario evaluó al menos 20 películas. Además también nos proporciona datos básicos sobre los usuarios: edad, sexo, profesión(un total de 21 profesiones) y un código postal de EEUU. Sobre los items el conjunto de datos cuenta con información sobre el género(19 géneros posibles), fecha de estreno y fecha de lanzamiento de la versión compacta.

Preprocesamiento para FM

Los datos de contexto que utilizamos para las máquinas de factorización tuvieron que ser procesados para tener la estructura que necesitamos. Las variables del usuario que consideramos fueron:

- **Edad** La edad la categorizamos por rangos $[a_i, a_{i+1})$ con $i = 1, \ldots, 9$ y con eso construimos un vector $edad(u) \in \mathbb{R}^9$ de tal forma que $edad(u)_i = 1$ si y solo si $edad(u)_i \in [a_i, a_{i+1})$. En este caso los intervalos para las edades fueron predefinidos. En el gráfico 4.2 podemos observar la distribución de las edades.

- **Sexo** El sexo lo categorizamos como dos variables binarias masculino y femenino.

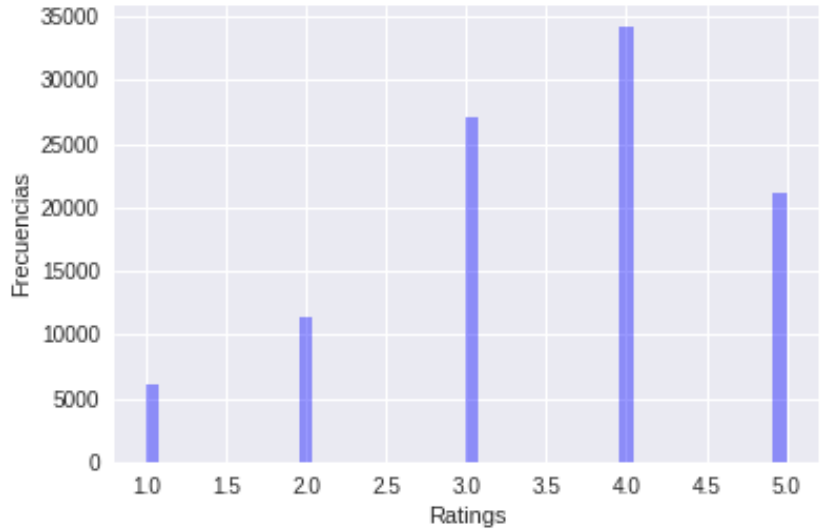


Figura 4.1: Distribución de ratings de MovieLens.

- **Profesión** La profesión la codificamos en un vector binario de longitud 21 (que es la cantidad de profesiones que admite el sistema), donde la i-ésima posición es 1 solo si corresponde a la profesión del usuario. De lo contrario esa posición es 0.

En el caso de los items, la única variable que utilizamos fue el género de la película, que lo codificamos de las misma manera que hicimos con las profesiones, pero esta vez, el vector tendrá longitud 19. Notaremos $f(u)$ al vector que representa todas la variables del usuario y $f(i)$ a las del ítem, en nuestro caso solo los géneros.

4.2. Implementación

Para realizar las simulaciones y experimentos con todas les técnicas descritas en el capitulo uno, utilizamos el paquete para sistemas de recomendación Surprise [41] implementado sobre la librería SciPy [1] de Python. Además también utilizamos la librería Scikit-learn [23], una popular librería de aprendizaje automático implementada en Python y como herramienta de visualización utilizamos Seaborn.

Para las máquinas de factorización existen varias implementaciones en Python, por ejemplo FastFM [36] y LightFM [32] son dos de ellas, también contamos con la implementación LibFM en C++ [27] del propio Rendle. Todas ellas implementan varias funciones de pérdida además de pérdida cuadrática considerada y varios optimizadores además de SGD. Elegimos utilizar una implementación de propósito académico llamada TFFM [39], basada en TensorFlow [35]. Dentro de esta implementación, utilizamos como optimizador el algoritmo basado en gradiente de primer orden *Adam* [29].

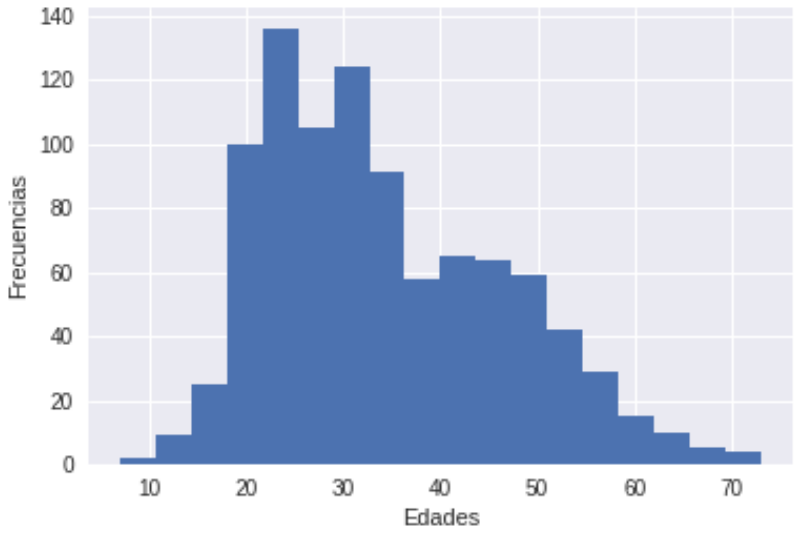


Figura 4.2: Distribución de las edades

El código fuente de las simulaciones descritas en esta tesis pueden encontrarse en el repositorio de *Github* [42].

4.3. Simulación 1 - Predicción y Recomendación

El objetivo de este primer experimento es ver como se comportan las máquinas de factorización frente a las técnicas clásicas descritas en el capítulo uno, en los problemas clásicos de predicción y recomendación.

Para las máquinas de factorización consideramos tres conjuntos diferentes de variables que nos determinarán tres máquinas diferentes de factorización. Estos conjuntos fueron:

- **FM-Us**: Las variables utilizadas fueron (Usuario(u), Pelicula(i), f(u))
- **FM-It**: Las variables utilizadas fueron (Usuario(u), Pelicula(i), f(i))
- **FM-All**: Las variables utilizadas fueron (Usuarios(u), Peliculas(i), f(u), f(i))

Procedimiento

Para evaluar y poder comprar los rendimientos de los algoritmos utilizamos un procedimiento de validación cruzada repetida. Como los modelos cuentan con varios hiperparámetros que queremos calcular, realizamos una búsqueda aleatoria de los mismos dentro de esta validación siguiendo la metodología sugerida por Cawley et. al. [19] y por Bergstra et. al. [25] la cual describiremos a continuación.

Primero extraeremos un conjunto de datos de test T al azar, los cuales no utilizaremos para entrenar el algoritmo, en nuestro caso, esos datos corresponden un 20 % del total de los datos S. El resto de los datos E los utilizaremos para entrenamiento, los dividiremos en 5 conjuntos disjuntos (*5-fold cross validation*) con las cuales nos generamos 5 particiones de conjunto de test y entrenamiento como se ejemplifica en la figura 4.3, las cuales usaremos para buscar la mejor combinación de hiperparámetros. Para esto repetiremos la siguiente estrategia una determinada cantidad de veces (nosotros utilizamos 20 repeticiones):

- Elegimos un conjunto de parámetros θ de forma aleatoria de una distribución predefinida.
- Realizamos la validación cruzada de 5 iteraciones con esos parámetros con los datos de entrenamiento E.
- Reportamos el promedio de los errores en cada uno de las particiones (*folds*), en este caso elegiremos el RMSE.

Una vez realizado este procedimiento, elegimos el conjunto de parámetros θ que obtuvo el mejor error y reentrenamos nuestro modelo pero con todos los datos en E y reportamos el error sobre el conjunto de test. Este procedimiento lo repetimos unas 100 veces, eligiendo al azar siempre el conjunto T, y reportaremos el promedio y el desvío obtenido como medida del rendimiento del modelo que estemos evaluando.

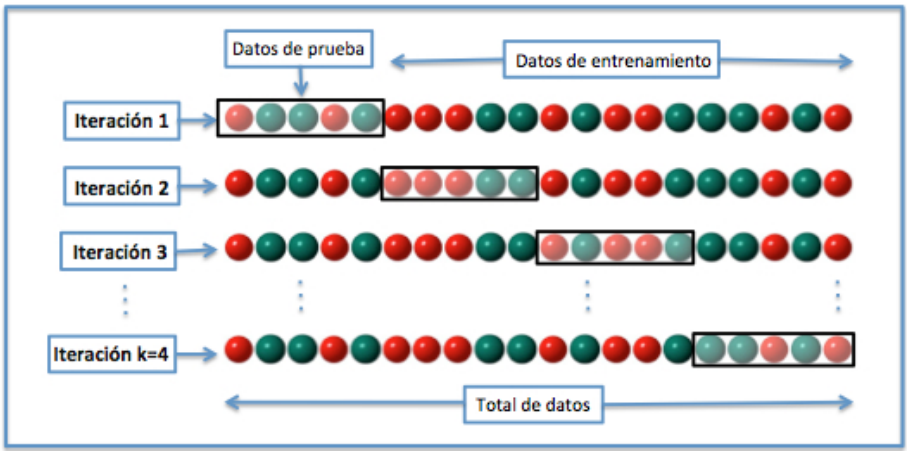


Figura 4.3: Validación cruzada de 4 iteraciones. Imagen extraída de Wikipedia.

Cabe destacar que la elección de las distribuciones de los hiperparámetros a determinar θ, dependerán de cada algoritmo, en la implementación se pueden verificar las elecciones realizadas. Lo único que consideraremos fijo serán las *épocas* que utilizará el algoritmo de minimización utilizado, las cuales representan la cantidad de iteraciones que realizaremos en los respectivos SGD, utilizándolo como criterio de parada para el algoritmo y también como herramienta para disminuir el sesgo del mismo [19].

Resultados - Predicción

En el recuadro 4.1 reportamos los resultados obtenidos con los diferentes modelos propuestos, tanto para el RMSE como para el MAE, junto al desvío de los errores obtenidos en las repeticiones.

Algoritmo	RMSE	MAE
kNN	0.9941 ± 0.0049	0.7853 ± 0.0058
kNN con Z-Score	0.9273 ± 0.0076	**0.7244** ± 0.0042
Slope One	0.9446 ± 0.0045	0.7424 ± 0.0039
SVD	0.9254 ± 0.0084	**0.7279** ± 0.0057
FM-us	0.8877 ± 0.0176	0.7393 ± 0.0081
FM-it	0.8775 ± 0.0166	0.7329 ± 0.0054
FM-all	**0.8684** ± 0.0098	0.7325 ± 0.0082

Cuadro 4.1: RMSE y MAE para las diferentes técnicas.

En los resultados se puede observar como los modelos de máquinas de factorización tienen un rendimiento superior a las otras técnicas según el RMSE. Sin embargo, al observar el valor del error absoluto, los resultados son más parejos ya que tanto kNN con el Z-Score como SVD obtienen resultados similares e incluso mejores.

Por otro lado, se observa que los rendimientos entre los diferentes modelos de FM, no son significativamente diferentes en este experimento en comparación a la diferencia con el resto de las técnicas, siendo que el modelo que utiliza todos las variables **FM-all**, el que obtiene el mejor rendimiento de los tres modelos.

A partir de estos resultados es que nos gustaría también analizar como se comportan las máquinas de factorización en el caso de contar con menos datos, y para eso repetimos el proceso anterior con la salvedad de que no utilizaremos una búsqueda de parámetros sino que utilizamos los obtenidos previamente y entrenando los algoritmos en un conjunto de datos menor al original.

En la figura 4.4 podemos observar los resultados obtenidos sobre el conjunto de test, donde utilizamos diferentes fracciones de los datos de entrenamiento E para cada algoritmo. Estos resultados muestran que aunque el rendimiento es superior para lo visto anteriormente, cuando reducimos la cantidad de datos de forma aleatoria, a las máquinas de factorización les resulta más costoso obtener mejores resultados que los otros dos modelos con los cuales los comparamos, siendo que recién al utilizar el 70 % de los datos, el rendimiento de las mismas supera los otros modelos.

Resultados - Recomendación

Para medir la recomendación, utilizaremos MAP@k descrita en (1.9) y también NDCG@k descrita en (1.17) para diferentes valores de $k = 5, 10, 20, 30, 50$. En este experimento, solo consideraremos **FM-all**. Los resultados obtenidos para MAP se pueden apreciar en el gráfico 4.5 mientras que los obtenidos para NDCG pueden observarse en el gráfico 4.6.

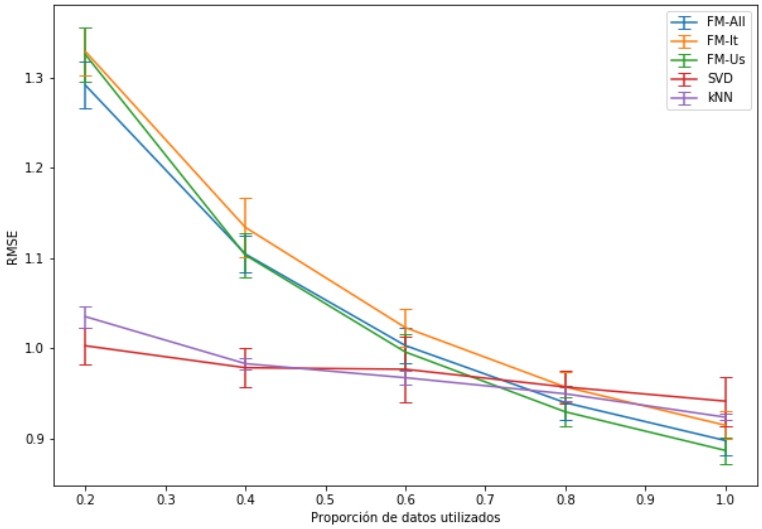


Figura 4.4: Rendimiento de los modelos según el RMSE con distintos cantidad de datos de entrenamiento.

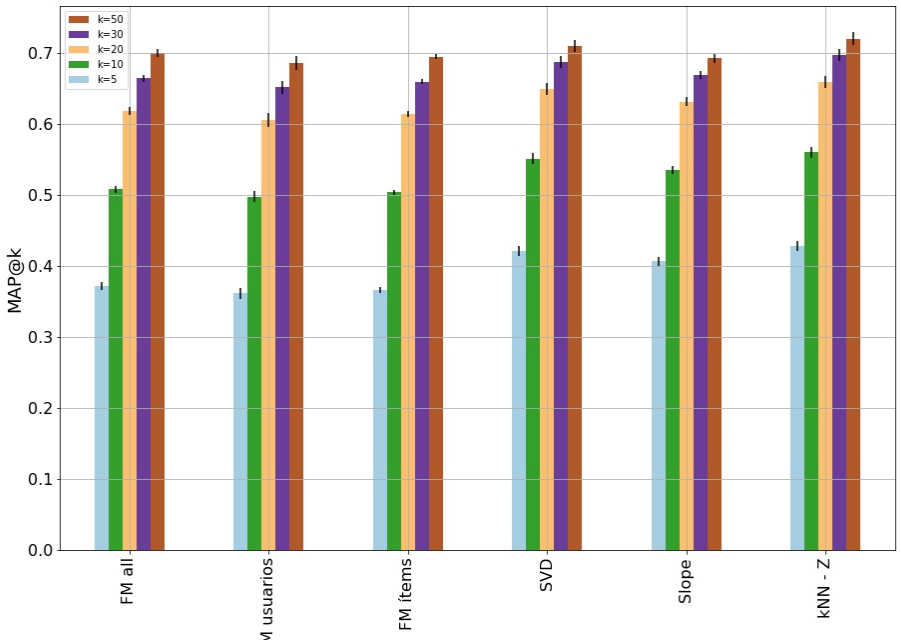


Figura 4.5: Errores según MAP@k en el Top-k.

Según la métrica MAP, las máquinas de factorización no pudieron rankear los items tan bien como otros modelos, por ejemplo SVD. Mientras que para NDCG, los resultados obtenidos no varían mucho, siendo de nuevo inferior el rendimiento de

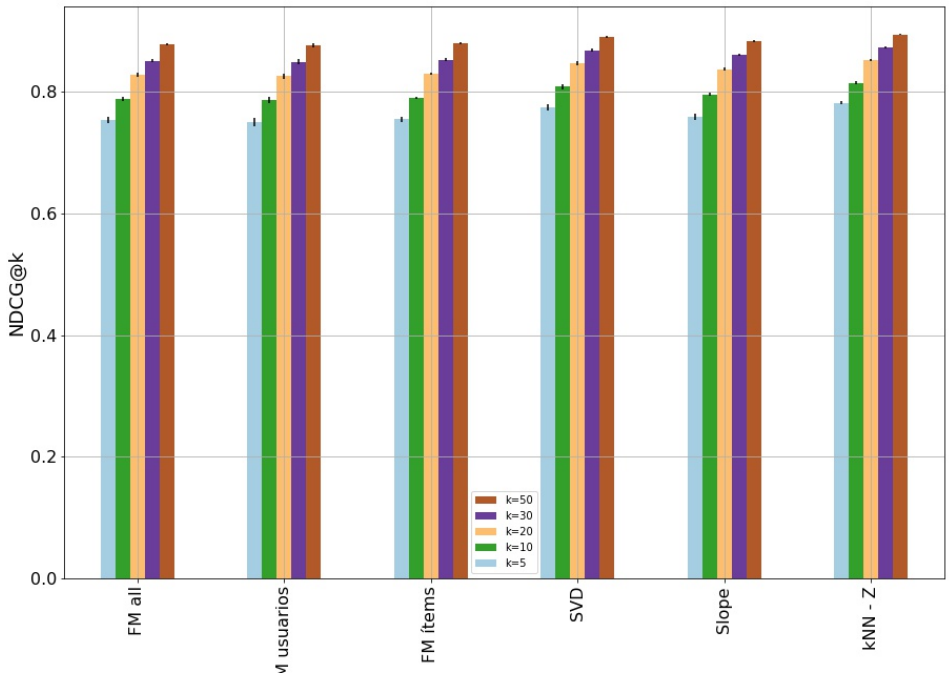


Figura 4.6: Errores según NDCG@k en el Top-k

las máquinas de factorización frente a modelos más sencillos como kNN con Z-Score.

4.4. Simulación 2 - Entrevistas

El objetivo de esta simulación es analizar el rendimiento de las máquinas de factorización en el problema de arranque en frío, utilizando una estrategia de entrevistas para modelar el perfil del usuario como lo visto en el capítulo tres. De esta forma, podremos analizar si el modelo de **FM-all** logra mejorar la expresividad del sistema frente a SVD, el modelo de mejor rendimiento de los comparados, en un escenario de arranque en frío. Para esto, simularemos el funcionamiento de un sistema de recomendación como MovieLens, el cual le solicita a un usuario nuevo al ingresar por primera vez al sistema, que evalúe una determinada cantidad de items. En esta simulación elegimos emular el comportamiento de SVD con una máquina de factorización como fue descrito en (2.3) para obtener resultados más confiables.

Procedimiento

Para este escenario, dividimos el conjunto de datos S entre conjunto de datos de entrenamiento E y de test T de forma disjunta en los usuarios, llamaremos U_E y U_T a los usuarios que están en el conjunto de entrenamiento y test respectivamente. Esto es para poder imitar el proceso del ingreso de un nuevo usuario en el sistema,

por lo que $U_E \cap U_T = \emptyset$. Para esto seleccionamos de manera aleatoria a los usuarios para agregar a T, quitándolos de S junto a todos sus ratings, manteniendo una proporción de entre el 20 % y el 25 % de datos en T. Las observaciones restantes contenidas en S, constituirán el conjunto de entrenamiento E.

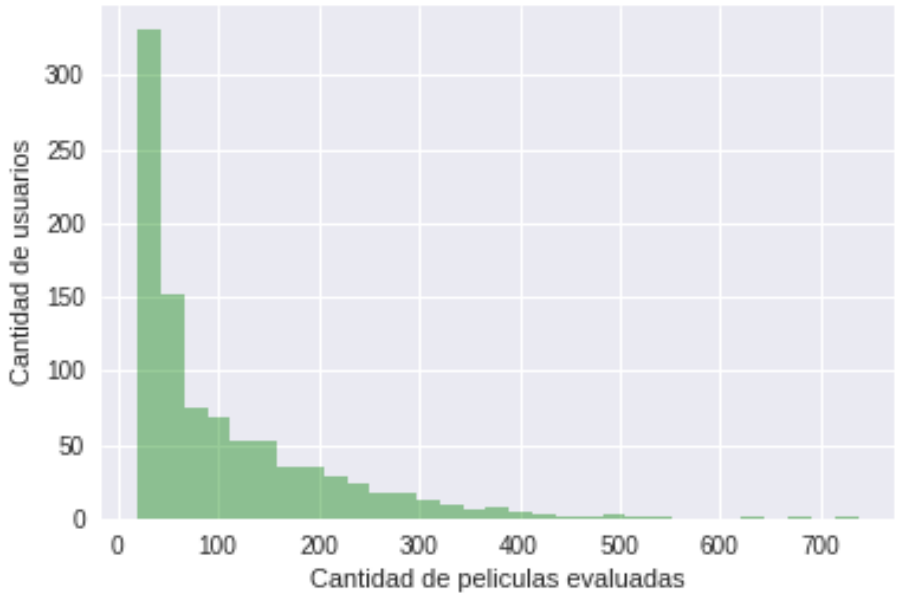


Figura 4.7: Cantidad de películas que evaluaron los usuarios en el conjunto de datos

La selección de usuarios para el arranque en frío, estuvo sujeta a que los usuarios hubiesen evaluado al menos 100 películas y menos de 200, siguiendo un procedimiento similar al realizado en Rashid et. al. [15]. Esto lo realizamos para asegurarnos de poder solicitar a los nuevos usuarios que evalúen una cierta cantidad de items, ya que de lo contrario podría haber usuarios en U_E que tienen menos observaciones que la cantidad de películas que le pedimos evaluar, emulando lo que haría un sistema de recomendación. El corte superior de 200 fue elegido para evitar perder usuarios en el conjunto de datos que sean muy significativos, ya que es poca la cantidad de usuarios que evaluaron muchas películas como se puede observar en 4.7. En esta simulación, de un total de 943 usuarios, 788 usuarios pertenecen a U_E y 155 usuarios pertenecen a U_T.

Para presentar las películas para que el nuevo usuario evalúe, elegimos la estrategia balanceada definida en 3.2.1, que mostró un rendimiento superior a las mencionadas en la precisión de la predicción [5]. Esta elección se debe a que el objetivo de esta simulación no es la de comparar las estrategias de selección de items, sino evaluar los rendimientos en las predicciones. Le presentamos películas hasta que logre evaluar l de ellas, donde $l = 10, 15, 20, 25$. Una vez que el usuario nuevo evaluó la cantidad de películas solicitadas, llamamos $\tilde{E}$ al nuevo conjunto de entrenamiento para ese usuario y $\tilde{T}$ al conjunto de test sin esos ratings.

Finalmente compararemos los errores entre **FM-all** y SVD, utilizando los mismos parámetros que en la primer simulación, pero para diferentes cantidades de factores $s = 15, 25, 50, 100$ con el objetivo de analizar su impacto en el problema. Para calcular el error, utilizamos el RMSE y MAE, pero en vez de hacerlo sobre todo el conjunto de testeo T, lo hicimos sobre cada usuario en el conjunto U_T y luego promediamos sobre los usuarios. Esto tiene sentido ya que queremos ver cómo es el rendimiento individual del sistema para el escenario de un nuevo usuario.

El procedimiento realizado, se encuentra detallado en el algoritmo 3.

Algoritmo 3 Procedimiento de entrevista

para todo Usuario $u \in U_T$ **hacer**
  mientras u no haya evaluado l películas **hacer**
    Presentar la próxima película i
    si u pudo evaluar i **entonces**
      Agregar al usuario u junto a i y el rating provisto al conjunto E
      Remover la observación de T
    fin si
  fin mientras
  Entrenar modelo **M** con $\tilde{E}$
  Calcular el error de predecir el resto de las películas de u en $\tilde{T}$
fin para
Promediar los errores obtenidos para cada usuario en U_T

Resultados

En el cuadro 4.2 se encuentran los errores obtenidos al calcular las predicciones directamente con el conjunto T, lo cual claramente tiene un mal rendimiento como era de esperar ya que no tiene información de los usuarios nuevos. Mientras que el modelo de **FM-all**, logra resultados que son más razonables al utilizar la información contextual de los items y de los usuarios. Además, quisimos ver si el aumento de la cantidad de factores s mejora la expresividad de los modelos. En estos casos se puede observar que no hay una mejora en el error al aumentar los factores.

	Errores	$s = 15$	$s = 25$	$s = 50$	$s = 100$
SVD	RMSE	12.935 ± 0.112	13.047 ± 0.116	13.139 ± 0.113	13.193 ± 0.112
	MAE	3.424 ± 0.0157	3.439 ± 0.0163	3.452 ± 0.0159	3.460 ± 0.0155
FM	RMSE	1.8505 ± 0.042	1.8985 ± 0.036	1.9474 ± 0.036	1.9758 ± 0.060
	MAE	1.1184 ± 0.013	1.1340 ± 0.011	1.1507 ± 0.011	1.1601 ± 0.019

Cuadro 4.2: Errores según la cantidad de factores para SVD y **FM-all** sin encuestas

En las figuras 4.8, 4.9, 4.10 y 4.11, graficamos los errores (RMSE y MAE) en función de la cantidad de películas solicitadas l, para cada elección de factores.

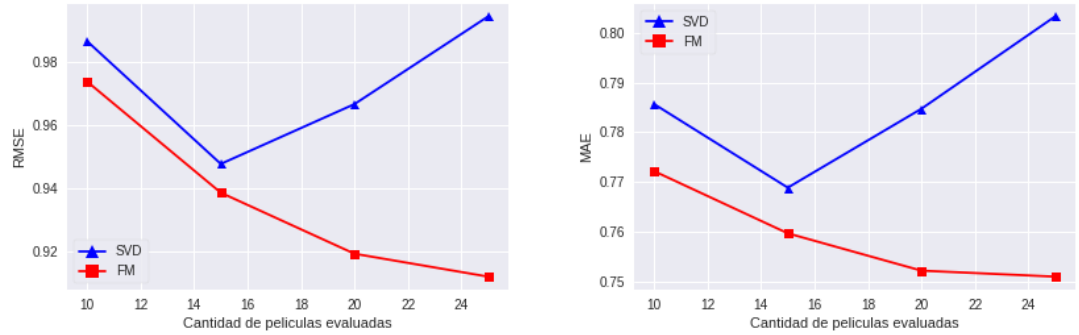


Figura 4.8: Errores con $s = 15$

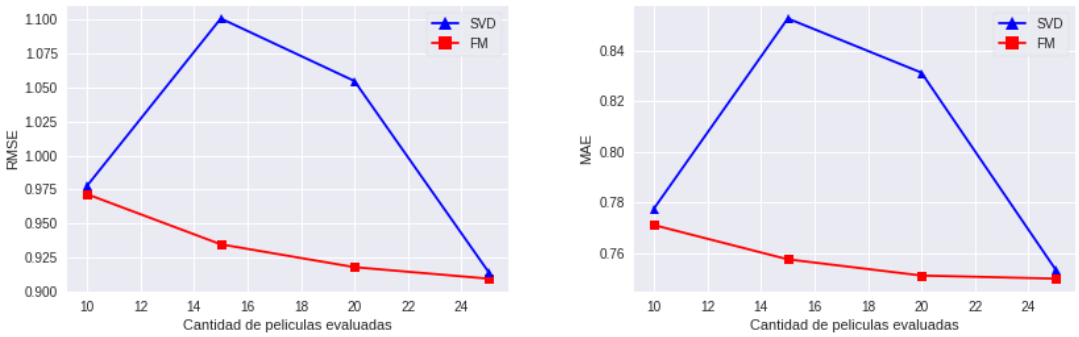


Figura 4.9: Errores con $s = 25$

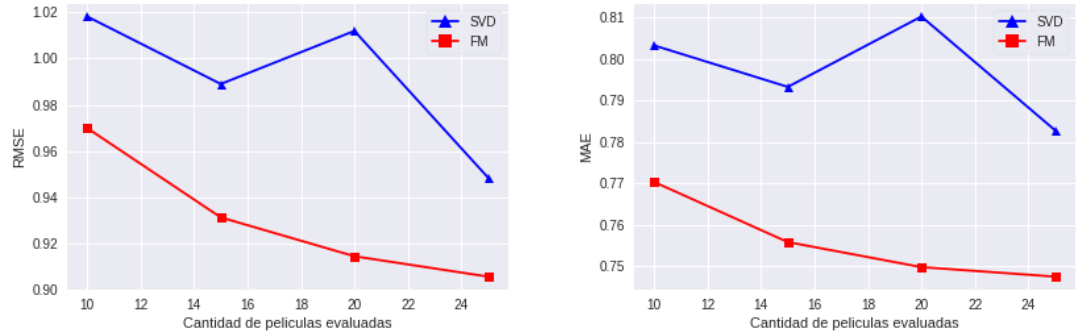


Figura 4.10: Errores con $s = 50$

En todas ellas, el modelo de **FM-all** tiene un rendimiento superior que el modelo SVD en ambas métricas. En los casos donde consideramos pocos factores, es donde más se destacan esas diferencias de rendimiento a favor de **FM-all**. En el caso de 25 factores, SVD logra emparejar su rendimiento a FM al pedirle al usuario que evalúe 10 películas, pero luego el error empeora al solicitar más ratings, volviendo a rendir de igual manera que FM al evaluar 25 películas. Este comportamiento con baja cantidad de factores, evidencia la dificultad que tiene el modelo clásico de SVD

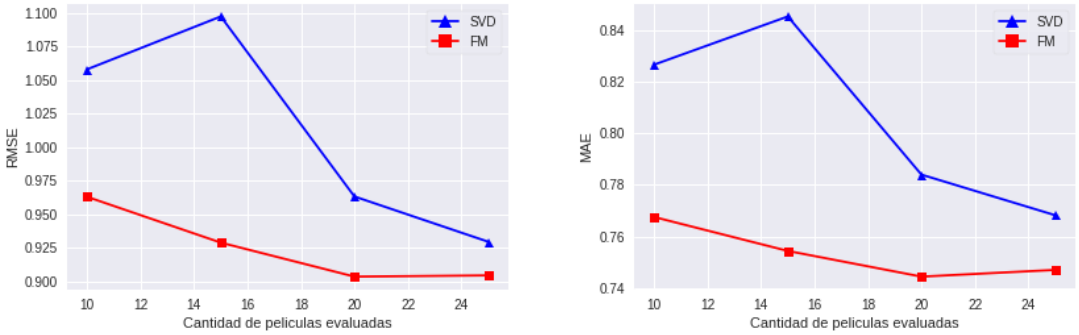


Figura 4.11: Errores con $s = 100$

para predecir ratings en un escenario de arranque en frío.

En los casos donde consideramos 50 y 100 factores, el modelo de FM tiene un comportamiento superior a SVD con pocos ratings provistos por los usuarios(10 y 15), mostrando como el modelo propuesto tiene una expresividad superior a SVD, ya que con poca información provista por los usuarios, consigue mejores predicciones en promedio. Una vez que los usuarios proveen más ratings, SVD empieza a mejorar su rendimiento pero aún por debajo del de FM.

En la figura 4.12 podemos ver una comparación de los diferentes modelos de FM considerados según la cantidad de factores. En todos ellos observamos cómo mejora el rendimiento a medida que los usuarios proveen más ratings. El aumento en la cantidad de factores para la FM, se vio traducido en una mejora de los errores, pero cuando la cantidad de ratings provistos fueron pocos (10 y 15), esa mejora no es tan significativa como lo es cuando aumenta la cantidad de películas evaluadas l.

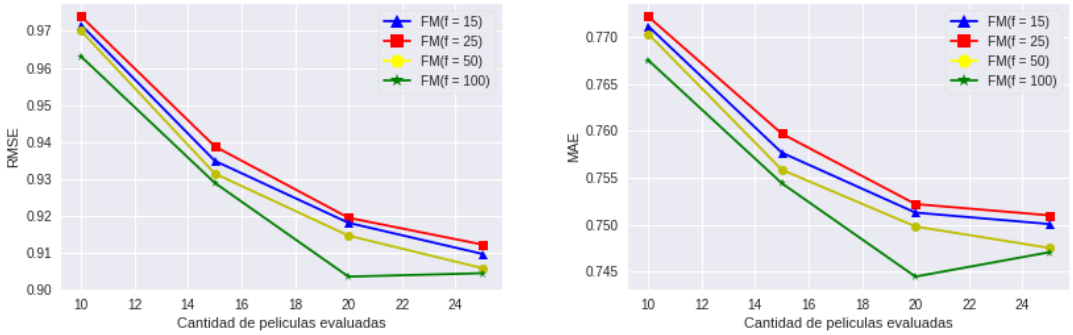
Estos resultados evidencian no sólo cómo las FM pudieron obtener buenos resultados sin la necesidad de muchos ratings, sino también que con pocos factores los resultados son los suficientemente mejores en comparación con el otro modelo considerado, lo cual nos vuelve a hablar de la expresividad de estas técnicas. Esto es una gran ventaja, ya que el entrenamiento de una FM con mayor cantidad de factores es más costoso computacionalmente pues la complejidad depende de los mismos.

4.5. Simulación 3 - Arranque en Frío

El objetivo de esta simulación es el de analizar el rendimiento de las propuestas realizadas basadas en los vecinos más cercanos para el caso del arranque en frío para un nuevo usuario en la sección 3.3, sin recurrir al proceso de encuestas.

Procedimiento

Para esta simulación, realizamos el mismo procedimiento que el utilizado en la simulación de entrevistas, sin la restricción de seleccionar los usuarios para el



(a) Errores RMSE (b) Errores MAE

Figura 4.12: Comparación del rendimiento de los modelos de FM con las diferentes cantidad de factores según la cantidad de entrevistas realizadas.

conjunto de entrenamiento E según la cantidad de ratings. En esta simulación, de un total de 943 usuarios, 786 usuarios pertenecen a U_E y 157 usuarios pertenecen a U_T.

Una vez construidos estos conjuntos, entrenamos nuestra FM y calculamos el error en T de la misma manera que lo hicimos en la simulación dos, pero esta vez no necesitamos re-entrenar el modelo, dado que no modificamos nuestros conjuntos, utilizaremos el modelo ya entrenado para predecir. Luego calculamos el promedio de esos errores sobre los usuarios. A esta forma de predecir los ratings, utilizando el vector de variables como fue descrito en (3.3), lo llamamos **M1**.

En este caso, la FM que consideramos es **FM-all**, que utiliza todas las variables contextuales. La única salvedad que realizamos fue la de no categorizar la edad, quedando entonces las variables:

$$x \in \mathcal{B}^{786} \times \mathcal{B}^{1642} \times \mathbb{R}^{25} \times \mathbb{R}^{19}$$

Para cada usuario u, calculamos $N_c(u)$ a los c vecinos más cercanos de u. Llamamos **M-uniforme** a la predicción obtenida al modelar al usuario como (3.5), pesando a los vecinos modelo de forma uniforme. **M-pesado** a la predicción obtenida al modelar el usuario como (3.8), donde pesamos a los vecinos por los inversos de las distancias al usuario activo.

Resultados

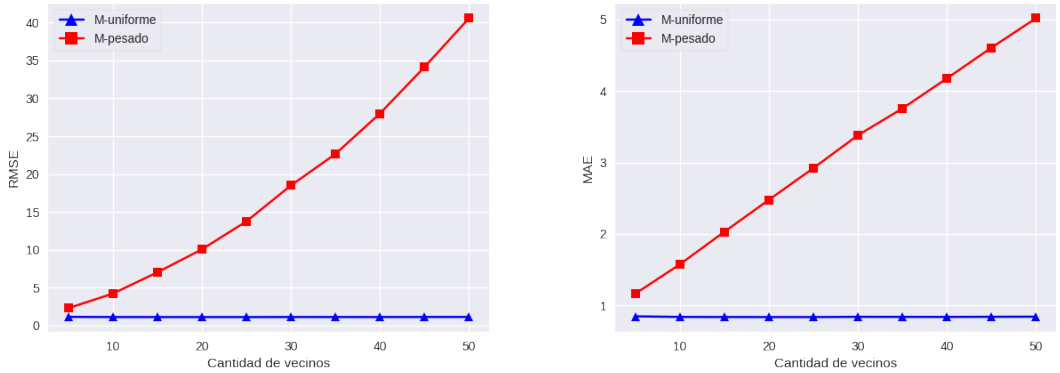
Los resultados obtenidos con la predicción **M1** fueron:

$$\text{RMSE} = 1.864, \quad \text{MAE} = 1.125$$

Estos valores los utilizamos para comparar el rendimiento de nuestros modelos de vecinos **M-uniforme** y **M-pesado**.

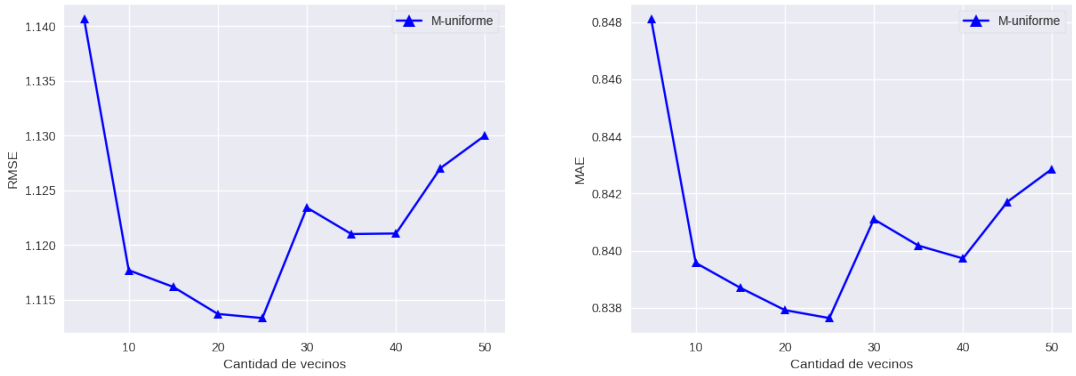
A continuación, en la figura 4.13 se observan los gráficos de los resultados obtenidos para ambos modelos según la cantidad de vecinos c utilizados y en la

figura 4.14 vemos un detalle de los errores en el caso de la predicción según **M-uniforme**.



(a) Errores RMSE (b) Errores MAE

Figura 4.13: Errores de **M-uniforme** y **M-pesado**



(a) Error RMSE (b) Error MAE

Figura 4.14: Detalle de los errores de **M-uniforme**

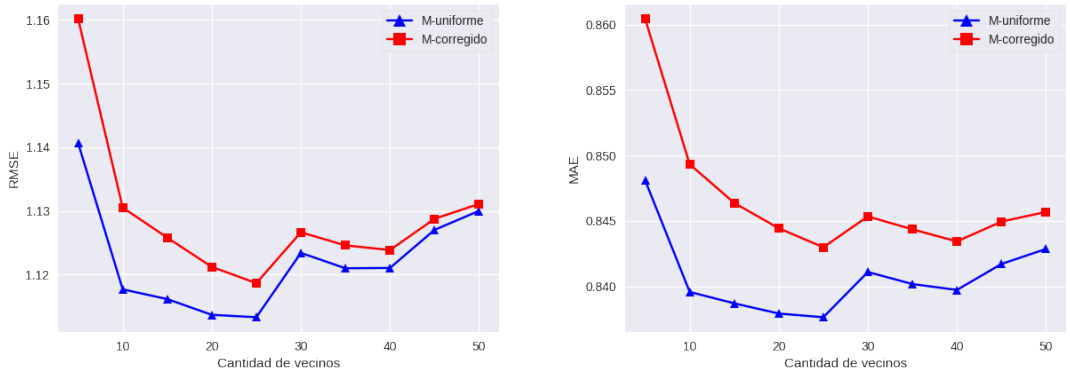
Los primeros resultados obtenidos muestran que la predicción del modelo **M-pesado**, obtuvo errores extremadamente altos en ambas métricas y una tendencia clara de aumentar a la hora de considerar más vecinos, lo cual al principio parecería ir en contra de la intuición. Este modelo con estos resultados, claramente no nos es útil. No obstante, al observar la forma de modelar al usuario nuevo y estudiar las vecindades de los usuarios que obtuvieron errores de gran magnitud, se pueden determinar algunas causas probables. Al observar en detalle los usuarios que obtuvieron errores de gran magnitud, pudimos ver que estos usuarios, poseían en su mayoría vecinos que se encontraban a distancia cero o a lo sumo uno. Esto provoca que a la hora de realizar la predicción, los términos de $\hat{y}$ que involucran a los factores de los usuarios, queden multiplicados por $\frac{1}{\sqrt{c}}$ en vez de $\frac{1}{c}$. Entonces en estos casos,

al multiplicar por $(\sqrt{c})^{-1}$, el modelo no calcula un promedio de los parámetros $\mathbf{w}$ y $\mathbf{V}$ correspondientes, dando lugar a la aparición de valores extremos en los errores.

Para evitar este problema, proponemos elegir parámetro de contracción al mismo c. De esta forma la predicción sobre un usuario u, es una combinación entre las dos ideas originales. Si u es el nuevo usuario, lo modelaremos:

$$x_v^{(u,j)} = \begin{cases} \frac{1}{d(u,v)+c} & \text{si } v \in N_c(u) \\ 0 & \text{caso contrario} \end{cases} \tag{4.1}$$

Llamaremos a este modelo **M-corregido**.



(a) RMSE (b) MAE

Figura 4.15: Errores de **M-uniforme** y **M-corregido**

En las figuras 4.15a y 4.15b observamos las comparativas entre **M-uniforme** y **M-corregido**, según la cantidad de usuarios considerados como vecindario. En ambos casos se ve que los errores tienen un comportamiento similar. Aumentar la cantidad de vecinos mejora al principio los errores, pero no siempre, ya que luego de considerad 25 vecinos, los errores de ambos empeoran.

Sin embargo, a pesar de que la corrección realizada mejoró significativamente el resultado el rendimiento de **M-corregido**, el rendimiento de **M-uniforme** fue superior. Por lo tanto, esto nos permite suponer que la inclusión de las distancias, a la hora de modelar por utilizando los usuarios vecinos, no ayuda al rendimiento del sistema sobre nuevos usuarios tanto como los usuarios que fueron considerados en sí.

Modelo	RMSE	MAE
M1	1.864398 ± 0.0754	1.125662 ± 0.0476
M-uniforme(k=25)	1.113308 ± 0.0352	0.837641 ± 0.0254
M-corregido(k=25)	1.118658 ± 0.0387	0.842989 ± 0.0219

Cuadro 4.3: Resultados obtenidos con los modelos propuestos para la mejor cantidad de vecinos.

Ambos métodos obtienen su mejor error cuando se consideraron entre 20 y 30 vecinos. Los resultados obtenidos, representan una mejora significativa a tratar de predecir los ratings de usuarios nuevos sin realizar ningún tipo de ajuste por la información de contexto, como figura en el cuadro 4.2.

4.6. Conclusiones

Considerando los resultados obtenidos, queda claro que las máquinas de factorización poseen un gran potencial en lo que respecta a los sistemas de recomendación, sobretodo en el problema de predicción de ratings, en el cual obtuvo resultados mucho mejores según el error cuadrático medio. Por otro lado, los errores medios (MAE) obtenidos no fueron mejores en comparación con los modelos más clásicos. Sin embargo, esto es explicable por la función de pérdida que consideramos para optimizar con SGD. Al haber utilizado el error cuadrático como función de pérdida, esos resultados no siempre serán equiparables ya que ponderan cosas diferentes [10] [28].

En el caso de recomendación, no obtuvimos tan buenos resultados como para el problema de predicción, ya que a penas se equipara en rendimiento a Slope One. Esto es también explicado por la función de pérdida utilizada, ya que en naturaleza, aunque ambos problemas están vinculados, no son del todo equivalentes y se pueden abordar otras técnicas para este objetivo, que incluyen considerar funciones de pérdida específicas para el problema de ranking. Aún así, los resultados obtenidos son comparables con los de las técnicas clásicas, pero además las máquinas de factorización tienen ventajas de diferente índole frente a modelos como kNN como por ejemplo que encuentran representaciones vectoriales para los usuarios y los items que pueden ser utilizadas luego en otros modelos.

Los resultados obtenidos en las simulaciones 2 y 3, muestran que las máquinas de factorización realizan predicciones confiables bajo datos ralos en el escenario específico de arranque en frío. Vimos también que incluso con pocos factores podemos obtener buenos resultados frente a técnicas como SVD.

Por otro lado, nuestra propuesta parece ayudar a mitigar el problema de arranque en frío para un nuevo usuario, reduciendo el error de predicción de ratings al utilizar un modelado del usuario basado en sus características poblacionales con un bajo costo computacional en comparación al modelado por entrevistas. Este escenario, se vería probablemente beneficiado de una ingeniería de variables, ya que al involucrar mayor cantidad de variables y de mejor calidad podremos mejorar la expresividad del modelo.

En resumen, las máquinas de factorización lograron obtener resultados superiores en el escenario de predicción de rating, mientras que en el escenario de ranking obtuvieron resultados a la par de las técnicas más clásicas. Mientras que en el escenario de arranque en frío, los resultados fueron muy superiores a los obtenidos por las otras técnicas, logrando mejoras significativas con nuestras propuestas e incluso en el escenario de encuestas.

4.7. Trabajo Futuro

Queda como trabajo a futuro, a la luz de los resultados, la implementación de otras funciones de pérdida objetivo que involucren el ranking de los items. Dos de ellas, frecuentemente utilizadas, son la BPR descrita en Rendle et. al. [18] o WARP, descrita en Weston et. al. [24]. Estas técnicas abordan el problema de ranking directamente sin hacer un uso directo de los ratings desde un punto de vista más cercano a un problema de clasificación.

Además, en el escenario de arranque en frío, restaría realizar una comparación entre las técnicas más modernas que no utilizan entrevistas y nuestro modelo propuesto utilizando las máquinas de factorización. Se pueden ver algunas alternativas descritas en Son [40].

Bibliografía

[1] Eric Jones, Travis Oliphant, and Pearu Peterson. {SciPy}: open source scientific tools for {Python}. `https://www.scipy.org/`(v. 1.15.0).

[2] Paul B Kantor and Jung Jin Lee. The maximum entropy principle in information retrieval. In *Proceedings of the 9th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 269–274. ACM, 1986.

[3] Jonathan L Herlocker, Joseph A Konstan, Al Borchers, and John Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 230–237. ACM, 1999.

[4] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.

[5] Al Mamunur Rashid, Istvan Albert, Dan Cosley, Shyong K Lam, Sean M McNee, Joseph A Konstan, and John Riedl. Getting to know you: learning new user preferences in recommender systems. In *Proceedings of the 7th international conference on Intelligent user interfaces*, pages 127–134. ACM, 2002.

[6] Andrew I Schein, Alexandrin Popescul, Lyle H Ungar, and David M Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 253–260. ACM, 2002.

[7] Jonathan L Herlocker, Joseph A Konstan, Loren G Terveen, and John T Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.

[8] Daniel Lemire and Anna Maclachlan. Slope one predictors for online rating-based collaborative filtering. *Proceedings of the 2005 SIAM International Conference on Data Mining*, pages 471–475, 2005.

[9] Slobodan Vucetic and Zoran Obradovic. Collaborative filtering using a regression-based approach. *Knowledge and Information Systems*, 7(1):1–22, 2005.

[10] Cort J Willmott and Kenji Matsuura. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate research*, 30(1):79–82, 2005.

[11] James Bennett, Stan Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, NY, USA, 2007.

[12] Arkadiusz Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD cup and workshop*, volume 2007, pages 5–8, 2007.

[13] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Major components of the gravity recommendation system. *ACM SIGKDD Explorations Newsletter*, 9(2):80–83, 2007.

[14] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434. ACM, 2008.

[15] Al Mamunur Rashid, George Karypis, and John Riedl. Learning preferences of new users in recommender systems: an information theoretic approach. *Acm Sigkdd Explorations Newsletter*, 10(2):90–100, 2008.

[16] Markus Weimer, Alexandros Karatzoglou, Quoc V Le, and Alex J Smola. Cofi rank-maximum margin matrix factorization for collaborative ranking. In *Advances in neural information processing systems*, pages 1593–1600, 2008.

[17] Ron Kohavi, Roger Longbotham, Dan Sommerfield, and Randal M Henne. Controlled experiments on the web: survey and practical guide. *Data mining and knowledge discovery*, 18(1):140–181, 2009.

[18] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 452–461. AUAI Press, 2009.

[19] Gavin C Cawley and Nicola LC Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, 11(Jul):2079–2107, 2010.

[20] Yehuda Koren. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 4(1):1, 2010.

[21] Steffen Rendle. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 995–1000. IEEE, 2010.

[22] Fidel Cacheda, Víctor Carneiro, Diego Fernández, and Vreixo Formoso. Comparison of collaborative filtering algorithms: Limitations of current techniques and proposals for scalable, high-performance recommender systems. *ACM Transactions on the Web (TWEB)*, 5(1):2, 2011.

[23] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011. `https://scikit-learn.org/`.

[24] Jason Weston, Samy Bengio, and Nicolas Usunier. Wsabie: Scaling up to large vocabulary image annotation. In *IJCAI*, volume 11, pages 2764–2770, 2011.

[25] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[26] Phil Chen and Dan Posch. Collaborative filtering on very sparse graphs a recommendation system for yelp.com, 2012.

[27] Steffen Rendle. Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(3):57, 2012.

[28] Tianfeng Chai and Roland R Draxler. Root mean square error (rmse) or mean absolute error (mae)?–arguments against avoiding rmse in the literature. *Geoscientific model development*, 7(3):1247–1250, 2014.

[29] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[30] Christian Desrosiers and George Karypis. A comprehensive survey of neighborhood-based recommendation methods. In *Recommender systems handbook*, pages 107–144. Springer, 2015.

[31] Yehuda Koren and Robert Bell. Advances in collaborative filtering. In *Recommender systems handbook*, pages 145–186. Springer, 2015.

[32] Maciej Kula. Metadata embeddings for user and item cold-start recommendations. In Toine Bogers and Marijn Koolen, editors, *Proceedings of the 2nd Workshop on New Trends on Content-Based Recommender Systems co-located with 9th ACM Conference on Recommender Systems (RecSys 2015), Vienna, Austria, September 16-20, 2015.*, volume 1448 of *CEUR Workshop Proceedings*, pages 14–21. CEUR-WS.org, 2015.

[33] Yuan Lu and Jie Yang. Notes on low-rank matrix factorization. *arXiv preprint arXiv:1507.00333*, 2015.

[34] Guy Shani and Asela Gunawardana. Recommender systems handbook. chapter Evaluating recommendation systems, pages 257–297. Springer, 2015.

[35] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016. `https://www.tensorflow.org/?hl=es`.

[36] Immanuel Bayer. fastfm: A library for factorization machines. *Journal of Machine Learning Research*, 17(184):1–5, 2016.

[37] Mathieu Blondel, Akinori Fujino, Naonori Ueda, and Masakazu Ishihata. Higher-order factorization machines. In *Advances in Neural Information Processing Systems*, pages 3351–3359, 2016.

[38] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(4):19, 2016.

[39] Alexander Novikov Mikhail Trofimov. tffm: Tensorflow implementation of an arbitrary order factorization machine. https://github.com/geffy/tffm, 2016.

[40] Le Hoang Son. Dealing with the new user cold-start problem in recommender systems a comparative review. 2016.

[41] Nicolas Hug. Surprise, a python library for recommender systems. http://surpriselib.com, 2017.

[42] Gastón Bujía. Factorization machines. `https://github.com/gastonbujia/tesis-maquinas-factorizacion`, 2018.

[43] Charu Aggarwal. *Recommender Systems, the textbook*. Springer, 2016.