
**Universidad de San Andrés**

**Departamento de Economía**

**Licenciatura en Economía**

# *La oferta en el siglo XXI: una revisión de literatura sobre sistemas de recomendación semisupervisados*

**Autora: Natalia Fonzo**

**Legajo: 27078**

**Mentora de Tesis: Marcela Svarc**

**Buenos Aires, 13 de julio de 2020**

# Índice

# 1 Introducción

En una era en que la información se ha transformado en materia prima para muchas de las compañías más importantes del mundo, los algoritmos de recomendación cobran un papel central cuando de explotar datos se trata. Los recomendadores no sólo han sido ampliamente estudiados por la academia desde comienzos de los años noventa, la industria también demostró gran interés en el tema. Hoy, son el corazón de casi todos los servicios provistos en línea: redes sociales, *e-commerce*, noticias, música y entretenimiento son algunos ejemplos.

Se trata de una herramienta de *machine learning* cuyo propósito es aprender la preferencia de los usuarios y predecir -recomendarles- una selección sólo de ítems relevantes o de su interés, como pueden ser canciones, videos, productos a la venta o posteos.

Este trabajo tiene por objetivo exponer el estado del arte de la literatura sobre sistemas de recomendación *semisupervisados*, aquellos que combinan aprendizaje supervisado y no supervisado. El primero se nutre de una estructura de datos etiquetados o *labeled*, en la que para cada usuario conocemos su valoración de los ítems. El segundo, de datos no etiquetados o *unlabeled*, es decir, que contamos con cierta información acerca de los usuarios de nuestra base, pero no con la variable que queremos eventualmente predecir, también llamada *output* -en este caso, sus preferencias sobre los ítems. El aprendizaje semisupervisado potencia, entonces, el desempeño de los recomendadores al permitirles aprovechar todos los datos disponibles, con y sin etiqueta. Esta forma de abordar el problema de escasez de información que los algoritmos de recomendación típicamente enfrentan comenzó a desarrollarse, a nuestro mejor saber, en el 2014 [1]. Ponemos el foco en una literatura muy joven.

La implementación de recomendadores, principalmente en la industria del *e-commerce* y del entretenimiento *online*, ha permitido una hiperpersonalización sin precedentes de la oferta disponible. En 1998, en una entrevista [2] que brindó al Washington Post, Jeff Bezos, CEO y fundador de Amazon, afirmó "si tenemos 4,5 millones de clientes, no deberíamos tener una tienda; deberíamos tener 4,5 millones de tiendas". La eficiencia de ofrecer un determinado ítem no a todo el que ingresa a un comercio, sino al cliente que va específicamente a buscarlo habilitó el desarrollo de mercados de nicho sub-explotados anteriormente, en tanto disminuyó los costos de transacción asociados a la venta de esta clase de productos [3]. De hecho, uno de los objetivos iniciales de Amazon a fines de los noventa era "mejorar la vida de las personas al ayudarlas a encontrar cosas que no hallarían de otro modo" [2].

Este fenómeno ha hecho posible que las firmas produzcan y vendan de manera rentable una gama de productos mucho más amplia que nunca. Como los costos de operar en la "cola larga" del mercado[1] han caído, productos que antes eran difíciles de comercializar hoy conquistan fuerza económica: agregados, representan un volumen considerable frente a los de producción masiva y el principio de Pareto[2] pierde cierta relevancia [3]. Este nuevo modelo de negocio ha catapultado a las empresas que lo adoptaron. De hecho, las más exitosas de Internet capitalizan en este sector. Según Reed Hastings, CEO de Netflix, sólo el 30% de sus películas rentadas son últimos lanzamientos; el resto, películas en catálogo [3, p. 109]. Para el 2008, casi el 40% de

[1] "Cola larga" es el término que usa Chris Anderson [3] para referirse a aquellos mercados de nicho que por sí solos no alcanzan gran volumen de ventas y que, por tener altos costos de transacción asociados, su comercialización típicamente deja un bajo margen de ganancia. Un ejemplo podría ser el mercado de palos de golf: los costos de tener un local de venta al público de palos de golf pocas veces pueden ser afrontados por las compras de clientes interesados en el golf que de casualidad pasen por enfrente. Este es el caso para la mayoría de los bienes en el esquema tradicional *offline*: son producidos en pequeña escala y comercializados con bajos márgenes de ganancia, y solo unos pocos bienes de producción masiva dominan el mercado.

[2] El Principio de Pareto establece que, para ciertos fenómenos, el 80% de los resultados se debe al 20% del esfuerzo. En nuestro caso, el 80% de los ingresos provienen del 20% de los productos [3, p. 130-131].

las ventas de libros de Amazon eran lecturas de nicho y el excedente del consumidor se quintuplicaba para ese entonces desde el 2000, consecuencia de este cambio en la forma de ofertar: herramientas de búsqueda, *reviews*, calificación pública de los productos y motores de recomendación [4], [5], [6].

Nos proponemos, entonces, entender el estado del arte de una de las más exitosas y difundidas aplicaciones de *machine learning* en los negocios, un método que no sólo ha revolucionado, sino que empuja hoy industrias millonarias. Así, la pregunta que guíe este trabajo puede formularse como: ¿cuáles son, cómo funcionan y qué otras aplicaciones tienen los más nuevos algoritmos empleados por los grandes unicornios tecnológicos?

En la próxima sección introducimos los principales conceptos de aprendizaje semisupervisado. También, presentamos las distintas clases existentes de sistemas de recomendación y algunas herramientas de las que disponen para fortalecerse. En la Sección 3, una revisión de literatura: tres aplicaciones de recomendadores semisupervisados. Por último, ponemos a prueba un recomendador con un caso práctico.



## 2 Machine learning: algunos conceptos

La gran cantidad datos digitales recolectados dio lugar a la necesidad de métodos automatizados para su análisis. El propósito del aprendizaje automático o *machine learning* es, entonces, aprender del comportamiento de los datos para predecir nuevos resultados o bien detectar patrones ocultos [7].

## 2.1 Aprendizaje semisupervisado

Se utiliza lo que hemos definido como aprendizaje no supervisado cuando el objetivo es descubrir patrones en una muestra aleatoria $(\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n)$ donde $\boldsymbol{x}_i = (x_{1i}, x_{2i}, \dots, x_{pi})$ con $i \in \{1,2, \dots, n\}$, esto es, una matriz con $n$ observaciones de $p$ variables. Entre otros ejemplos, se encuentran el *clustering* -para definir grupos de observaciones de la muestra, de manera que observaciones similares pertenezcan al mismo grupo y observaciones diferentes, a distintos-, y el análisis de componentes principales, también conocido como PCA por sus siglas en inglés -para conocer qué combinación lineal de las $p$ variables concentra en mayor medida información sobre los datos.

Ahora bien, se emplea el aprendizaje supervisado cuando no sólo contamos con la información de la muestra $(\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n)$, sino que además conocemos sus etiquetas, esto es, la variable de respuesta o el *output* que tuvo cada observación de dicha muestra de entrenamiento. Lo que ahora se busca es predecir las etiquetas de observaciones de una nueva muestra, para la cual no contamos con esta información. En este caso, usualmente se requiere aprender un mapeo $f$ de $(\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n)$ a $\boldsymbol{Y}$, con $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \dots, \boldsymbol{y}_n)$ el vector de etiquetas conocidas para cada observación $i \in \{1, 2, \dots, n\}$. Así, para una nueva observación $\boldsymbol{x}_j = (x_{1j}, x_{2j}, \dots, x_{pj})$, se predecirá su etiqueta como $\hat{\boldsymbol{y}}_j = \hat{f}(\boldsymbol{x}_j)$. Regresiones y clasificadores son los principales métodos de esta clase. Árboles de decisión, redes neuronales y recomendadores también pueden ser entrenados de forma supervisada.

Mientras los datos etiquetados son difíciles y costosos de conseguir, los no etiquetados pueden ser relativamente fáciles de recolectar. El entrenamiento semisupervisado [8], [9] ataca este

problema alimentándose de ambos en la medida en que estén disponibles, para construir mejores y más precisos modelos. En este caso, los datos en $(\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n)$ pueden dividirse en dos partes. Por un lado, tenemos las observaciones etiquetadas $X_L \coloneqq (\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_L)$, donde $\boldsymbol{x}_l = (x_{1l}, x_{2l}, \dots, x_{pl})$ con $l \in \{1,2, \dots, L\}$ es la $l$-ésima observación de las $p$ variables, y para las que conocemos sus respectivas etiquetas $\boldsymbol{Y_L} \coloneqq (\boldsymbol{y}_1, \boldsymbol{y}_2, \dots, \boldsymbol{y}_L)$. Por otro lado, las restantes $n - L$ observaciones $X_U \coloneqq (\boldsymbol{x}_{L+1}, \boldsymbol{x}_{L+2}, \dots, \boldsymbol{x}_n)$, donde $\boldsymbol{x}_u = (x_{1u}, x_{2u}, \dots, x_{pu})$ con $u \in \{L+1, \dots, n\}$, para las que desconocemos el *output*.

*Co-training*, *self-training* y los *graphed-based* son algunos métodos de aprendizaje semisupervisado que los recomendadores toman para sí.

### *2.1.1 Self-training*

Un modelo ajustado de manera semisupervisada con *self-training* [8] primero aprende de una pequeña cantidad de datos etiquetados. Luego, es utilizado para etiquetar los datos sin etiqueta. Un subconjunto $X_{U'} \subset X_U$, en particular compuesto por aquellas observaciones etiquetadas con mayor confianza por el modelo, y sus etiquetas predichas correspondientes $\hat{Y}_{U'}$ serán considerados en la siguiente instancia como información etiquetada para reentrenar el algoritmo, y así repetir sucesivamente el procedimiento. Notar que de esta forma el modelo utilizará sus propias predicciones para reajustar sus parámetros y así mejorar la predicción en la próxima iteración.

### *2.1.2 Co-training*

De manera similar, el procedimiento de *co-training* [8] emplea dos o más modelos diferentes, también llamados *co-trainers*, que una vez ajustados inicialmente con datos etiquetados

utilizarán sus mejores predicciones sobre los datos sin etiquetas como información para reentrenarse. Por ejemplo, si la mayoría de los *co-trainers* coincide en un determinado *output* estimado $\hat{y}_u$ para cierta observación $\boldsymbol{x}_u$ originalmente *unlabeled*, entonces el par $(\boldsymbol{x}_u, \hat{\boldsymbol{y}}_u)$ se incorpora a $X_L$ en la próxima iteración. Luego de las repeticiones necesarias, las predicciones de todos modelos conjuntamente ajustados coincidirán para mayoría de los datos. La predicción final sobre nuevas observaciones será un ensamble de las hechas por cada *co-trainer*.

Lo interesante de este método es que al permitirle a algoritmos especializados en distintos aspectos de los datos enseñarse mutuamente, el modelo ensamblado terminará superando en *performance* a cualquier otro que de manera independiente también emplee la totalidad de la información disponible.

### *2.1.3 Métodos gráficos*

Los métodos gráficos [8, p. 18], [9, p. 191] consisten en estructurar los datos en forma de grafo. Se emplean en aprendizaje semisupervisado pues permiten incorporar al modelo información acerca de la *relación* que pueda haber entre ítems, como pueden ser libros, más allá de sus etiquetas conocidas, como ha de ser si cierto usuario los ha comprado o no o qué calificación les ha dado.

En nodos, se representan las observaciones *labeled* y/o *unlabeled* de nuestra base de datos. Estarán unidos por las aristas, que llamaremos *edges*, y que representan la similitud entre los nodos que conectan. Los *edges* pueden, además, contener ponderadores que contemplen cuán similares son las dos observaciones en cuestión. Un grafo puede definirse, entonces, como $\mathcal{G} = \{\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{E}}\}$, donde $\boldsymbol{\mathcal{V}}$ es el conjunto de nodos y $\boldsymbol{\mathcal{E}}$, el de aristas. Adicionalmente, definimos el conjunto de ponderadores $\boldsymbol{W} = \{w_\varepsilon | \varepsilon \in \boldsymbol{\mathcal{E}}\}$. Puede visualizarse como muestra la *Figura 1*.

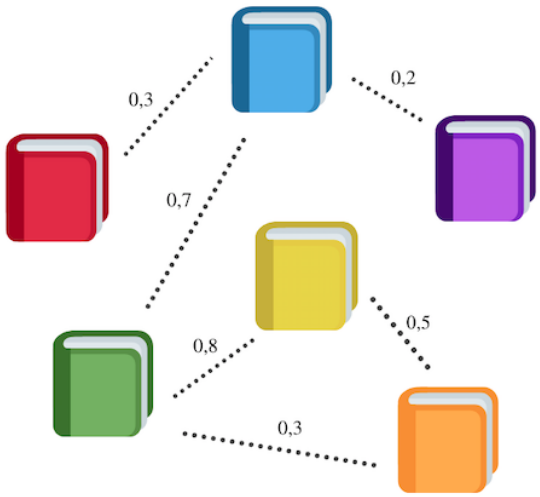


***Figura 1:*** *Ejemplo de un grafo para un set de libros disponibles en stock.*

Típicamente, el objetivo será estimar una función $f$ que permita etiquetar las observaciones *unlabeled* de $\boldsymbol{\mathcal{V}}$ respetando la similitud entre ítems y de manera que las predicciones para los nodos *labeled* coincidan lo más posible con las etiquetas reales. En el apartado 3.2 revisaremos un *paper* que incorpora de manera central esta estructura en su recomendador semisupervisado.

Por su parte, el *paper* del apartado 3.3, utiliza grafo de manera más auxiliar: con ellos busca aprender de manera no supervisada la relación entre usuarios, por un lado, y entre ítems, por otro, para que una forma simplificada de estos grafos -*graph embeddings*- sean consumidos en última instancia por su recomendador. *Graph Laplacian regularization* es el método usado para traducir nodos, aristas y sus pesos a un espacio vectorial de menor dimensión procurando conservar al máximo la estructura del grafo y la información en ella contenida.

## 2.2 Otras herramientas al alcance: redes neuronales

Redes neuronales es la herramienta que Yang, Carl et al. [10] utilizan, en el trabajo que presentaremos en la sección 3.3, para conciliar la perspectiva semisupervisada con los

mecanismos de recomendación. Una red neuronal es una regresión o clasificación hecha en dos o más etapas [11, p.302], típicamente representada con estructura de red como vemos en la *Figura 2*.

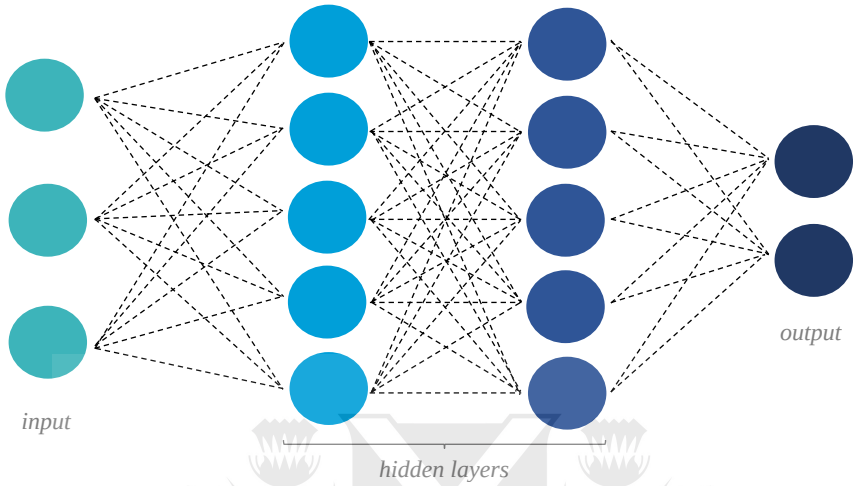


***Figura 2:*** *Representación de una red neuronal con dos hidden layers.*

En esencia, se trata de extraer combinaciones lineales de los *inputs* y repetir este proceso tantas veces como sea necesario para finalmente modelar el target a predecir como una función no lineal de la última capa de combinaciones. Las capas intermedias de este proceso se llaman *hidden layers*: la red neuronal de la *Figura 2* tiene dos.

## 2.3 Sistemas de recomendación

Los algoritmos de recomendación se proponen aprender los patrones subyacentes en los datos para poder predecir una selección de ítems que sean relevantes o de interés para los usuarios.

*Content-based* y *collaborative filtering* son los dos principales enfoques existentes [12]. Como se ilustra en la *Figura 3*, el supuesto detrás del primero es que, si a un usuario le gustó cierto ítem en el pasado, le gustarán ítems similares a éste en el futuro.

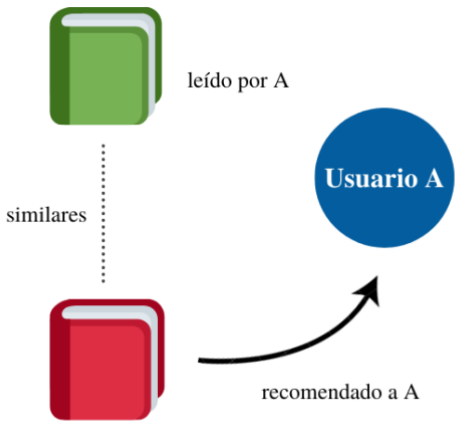


***Figura 3:*** *Recomendadores del tipo content-based*

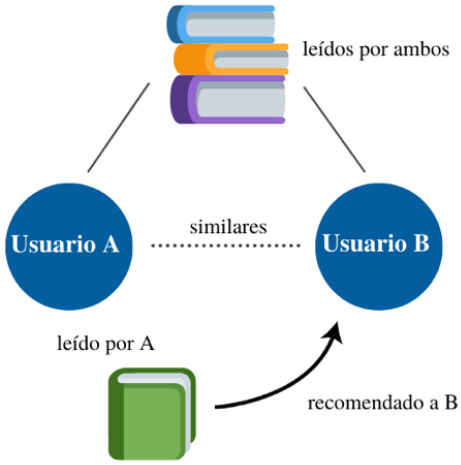


***Figura 4:*** *Recomendadores del tipo collaborative filtering*

Para hacer una recomendación, estos algoritmos necesitan entonces definir su métrica de similitud [13]. Consideremos la matriz

$$\boldsymbol{A} = [s_{11}\ s_{12}\ \dots\ s_{1F};\ s_{21}\ s_{22}\ \dots\ s_{2F};\ \dots\ \dots\ \dots\ \dots;\ s_{M1}\ s_{M2}\ \dots\ s_{MF}\ ], \qquad (1)$$

donde cada fila representa un ítem disponible y cada columna, una característica posible. Digamos que se trata de $M$ aplicaciones publicadas en el App Store y de $F$ características: cada una de las categorías -juegos, salud, viajes, utilidades, etc.- y cada una de las empresas desarrolladoras corresponde a una columna, de manera tal que $\boldsymbol{s}_{mf} = \{0,1\}$ con $m \in \{1, 2, \dots, M\}$ y $f \in \{1, 2, \dots, F\}$. Consideremos, además, un vector $\boldsymbol{u}_i = (u_{i1}, u_{i2}, \dots, u_{iF})$ con $i \in \{1, 2, \dots, N\}$, que representa las preferencias del usuario $i$ y que está constituido por unos y ceros según si éste descargó en el pasado aplicaciones con alguna de las $F$ características. Ahora bien, una métrica de similitud entre el comportamiento pasado del usuario $i$ y la aplicación $s_m$ podría ser, por ejemplo, el producto escalar $\langle \boldsymbol{s}_m, \boldsymbol{u}_i \rangle = \sum_{f=1}^{F} s_{mf}\, u_{if}$. Notar que una característica que aparece tanto en el vector de la aplicación, como en el del usuario, contribuye en 1 a la

sumatoria. De esta manera, cuanto más parecido sea el historial de descargas del usuario con las características de una aplicación, más alto será el producto escalar.

Una desventaja [14] de los recomendadores *content-based* es que sólo pueden hacer recomendaciones basadas en los intereses existentes de un usuario: aunque excelentes a la hora de captar intereses muy específicos, son limitados a la hora de expandir los gustos de un usuario.

Una recomendación más fortuita es la que hacen los recomendadores del tipo *collaborative filtering*. Como se ilustra en la *Figura 4*, el supuesto detrás es que si dos usuarios tuvieron gustos similares en el pasado también los tendrán en el futuro. A grandes rasgos, el filtrado de ítems relevantes del extenso conjunto de alternativas se hace colaborativamente entre las preferencias de los consumidores, de manera tal que estos algoritmos incorporan en su aprendizaje la similitud que hay entre usuarios e ítems simultáneamente.

Las preferencias pueden ser representadas como una matriz $N \times M$, donde $N$ es la cantidad de usuarios y $M$, la cantidad de ítems. Si bien las entradas de esta matriz pueden ser numéricas, como en el caso de los *ratings* de Netflix, en general se trata de una variable binaria: "clickeado", visto, comprado, "likeado". Más aún, la mayoría de las entradas, hemos visto, suelen estar vacías. El objetivo de los recomendadores es, de alguna forma u otra, completar esta matriz de *feedback*. El método empleado para este propósito por los recomendadores del tipo *collaborative filtering* se llama *matrix factorization*. Este consiste en traducir una matriz incompleta de *feedbacks* usuario-ítem en una matriz numérica completa que respete razonablemente las preferencias reveladas por los usuarios en la primera. El ejemplo a continuación, extraído de [15], ilustra cómo.

Como se muestra en la *Figura 5*, dada una matriz de *feedback* binario $\boldsymbol{H} \in \mathbb{R}^{N \times M}$, que indica si $N$ personas vieron o no ciertas $M$ películas, el objetivo del modelo será aprender una matriz de *emebeddings*[3] de usuarios, $\boldsymbol{H}_u \in \mathbb{R}^{N \times d}$, y otra para películas, $\boldsymbol{H}_p \in \mathbb{R}^{M \times d}$, de manera tal que $\boldsymbol{F} = \boldsymbol{H}_u \boldsymbol{H}_p^T$ sea una buena aproximación de $\boldsymbol{H}$. Notar que la fila $i$ de $\boldsymbol{H}_u$ es el *embedding* correspondiente al usuario $i$ y que la fila $j$ de $\boldsymbol{H}_p$ es el *embedding* de la película $j$. Notar, también, que el elemento en la posición $(i, j)$ de la matriz **F** corresponde a $\langle \boldsymbol{H}_{ui}, \boldsymbol{H}_{pj}^T \rangle$, esto es, el producto escalar entre la fila $i$ de $\boldsymbol{H}_u$ y la columna j de $\boldsymbol{H}_p^T$.

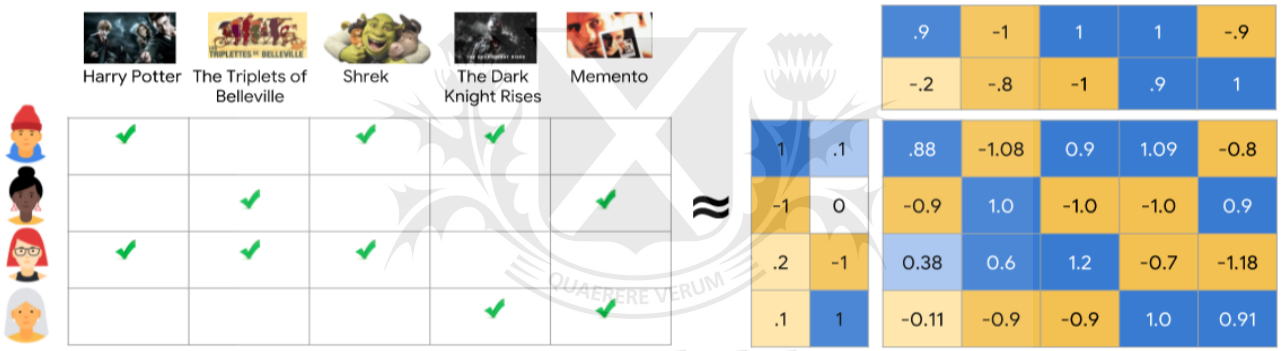


***Figura 5:*** *Matriz de feedback* $\boldsymbol{H}$ *y su aproximación, el producto matricial entre las dos matrices de embeddigns* $\boldsymbol{H}_u$ y $\boldsymbol{H}_p^T$. *Fuente: [15]*

Entre otras, una forma de obtener las matrices de *embeddings* es escogiendo aquellas que minimizan la suma de los errores cuadrados de todas las posiciones conocidas u observadas de $\boldsymbol{H}$ respecto de $\boldsymbol{F}$, esto es:

$$\min_{\boldsymbol{H}_u \in \mathbb{R}^{N \times d}, \boldsymbol{H}_p \in \mathbb{R}^{M \times d}} \sum_{(i,j) \in obs} (\boldsymbol{H}_{ij} - \langle \boldsymbol{H}_{ui}, \boldsymbol{H}_{pj}^T \rangle)^2 \qquad (2)$$

[3] Técnica por la cual información de vectores de un espacio de gran dimensión se vuelca sobre un espacio de dimensión reducida [16].

# 3 Recomendadores semisupervisados: panorama actual

La falta de información, el cambio en las preferencias de los usuarios, las predicciones *cold-start* -esto es, de usuarios o ítems nuevos en la base- o el sobreajuste de los datos que puede estancar al modelo en una predicción óptima localmente, son sólo algunos ejemplos de los desafíos que los recomendadores de cualquier tipo frecuentemente enfrentan. Como hemos mencionado anteriormente, el enfoque semisupervisado ayuda a atacar el primero. A continuación, una revisión literaria de tres publicaciones con aplicaciones interesantes de recomendadores semisupervisados.

## 3.1 Recomendando productos: aprendizaje *stream-based*

*Stream-based semi-supervised learning for recommender systems*

En este *paper* [17], los autores proponen una estructura de aprendizaje semisupervisado para recomendadores usuario-ítem *stream-based* -esto es, que incorporan nueva información sobre la marcha, como puede ser un nuevo producto, usuario o *feedback*. Su objetivo es paliar el problema de escasez de datos etiquetados típico de estos métodos, a la vez que les permiten una ágil adaptabilidad a nuevos *inputs* o cambios de preferencia. Más aún, la capacidad del modelo de introducir constantemente nueva información le permitirá potenciar los beneficios del aprendizaje semisupervisado [17, p. 772]: no solamente aprenderá de las observaciones sin etiqueta, sino que también podrá aprender de las etiquetas que a estas les vaya asignando.

Para ello, los autores desarrollaron una extensión del algoritmo de *matrix factorization* BRISMF [18], que llaman extBRISMF. Lo usarán para etiquetar la matriz de preferencias de dimensión *creciente* -pues nuevos usuarios e ítems aparecen en el flujo de información. Como

mencionamos, lo entrenarán de manera semisupervisada, en particular emplean *co-training* y *self-training*.

### *3.1.1 Estructura de aprendizaje semisupervisado*

Los datos consisten en una matriz de *ratings* para cada combinación usuario-ítem. Para entrenar el recomendador, será necesario dividirla en cuatro partes como se ilustra en la *Figura 6.* En primer lugar, el algoritmo operará en modo *batch* y aprenderá con datos etiquetados e información estática qué productos le suele gustar a cada usuario en esta porción de la información (*batch traning set*). Una vez listo el entrenamiento inicial, se testea, con datos desconocidos para el modelo, el aprendizaje realizado y se reajusta su capacidad predictiva a partir de los errores que cometa al intentar predecir los *ratings* de combinaciones ítem-usuario que nunca ha usado (*batch testing set*). En esta última instancia, el algoritmo cambia a modo *stream*, es decir, trabajará con información en flujo: predecirá los *ratings* de cada ítem-usuario uno por uno e intentará mejorar en la próxima predicción con cada error cometido. Por último, el modelo, que estará ya operando en modo *stream*, comienza el aprendizaje semisupervisado (*stream training set*). Veremos más adelante cómo funciona esta etapa. Una porción de datos es reservada para evaluar la *performance* y testear la significatividad final (*test stream set*).

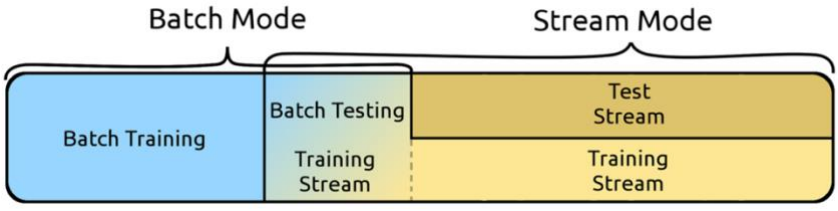


***Figura 6:*** *Divisón del dataset en modo batch y stream, ambos con set de entrenamiento y testeo. Fuente: [17]*

### *3.1.2 Enfoque co-training*

De acuerdo con este enfoque, se corren paralelamente múltiples recomendadores *stream-based*, que se especializarán en distintos aspectos de los datos y pueden enseñarse entre sí. Los datos del *bach tranining set* se dividen en $N$ subconjuntos -no necesariamente disjuntos-, como se muestra en la *Figura 7*. Cada *co-trainer* de $\boldsymbol{C} = \{Co - Tr_1, Co - Tr_2, \dots, Co - Tr_N\}$ será inicialmente entrenado de manera autónoma con un subconjunto de los datos y reajustado en la etapa de *batch test* como fue explicado anteriormente.

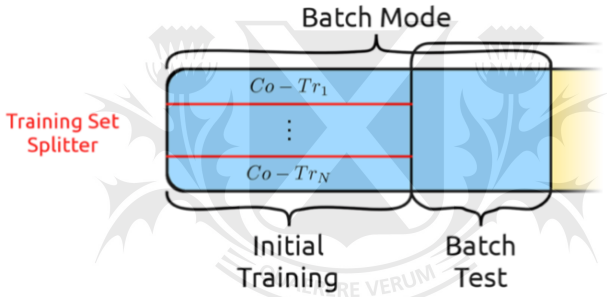


***Figura 7:*** *Divisón del batch trainign set entre N porciones para el entrenamiento de los N co-trainers. Fuente: [17]*



Una vez en el *stream training set*, un flujo de *ratings* se irá procesando uno a uno. El aprendizaje semisupervisado consistirá en ir intercalando observaciones *labeled* y *unlabeled*, como se ilustra respectivamente con las secciones amarillas y verdes de la *Figura 8*.

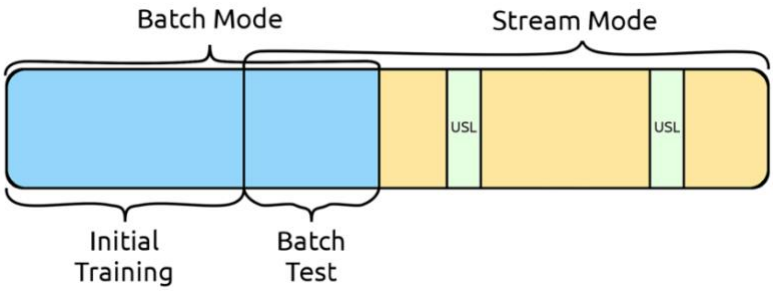


***Figura 8:*** *Composición del stream trainign set por datos etiquetados (amarillos) y datos no etiquetados (verdes). Fuente: [17]*

Como se ve en la *Figura 9*, los $N$ *co-trainers* irán avanzando por el eje $t$. Cada uno hará su predicción del primer *rating* del flujo, $\hat{\boldsymbol{r}}_{1Co-Tr_k}$, y luego recibiriá el verdadero valor de $\boldsymbol{r}_1$, que utilizará para actualizarse. El procedimiento coninuará de manera incremental: para cada *rating* $\boldsymbol{r}_t \in (\boldsymbol{r_1}, \ldots, \boldsymbol{r}_m)$ se obtendrán las $N$ predicciones $Co - Tr_k(\boldsymbol{r}_t) = \hat{\boldsymbol{r}}_{t\,Co-Tr_k}$ y el verdadero valor de $\boldsymbol{r}_t$ se les proveerá a los *co-trainers* para ayudarlos a mejorar en el próxima vez.

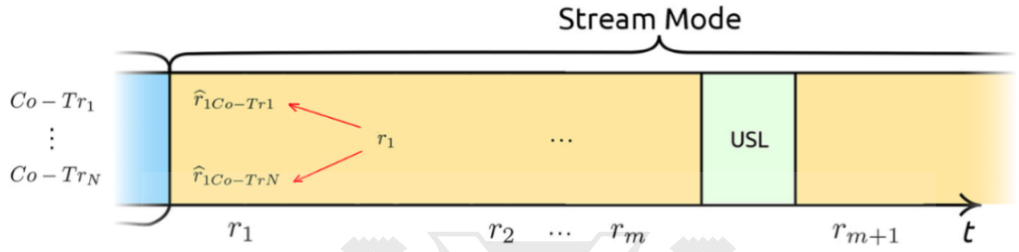


***Figura 9:*** *Entrenamiento supervisado durante el stream traning set.*
*Fuente: Adaptado de [17]*

Luego de $m$ observaciones, datos *unlabeled* aparecerán en el flujo. Dada la abundancia relativa de datos sin etiqueta, y en la medida en que no es posible procesarlos en su totalidad, se empleará un selector de observaciones *unlabeled*. Este podría, por ejemplo, decidir usar algunos pares usuario-ítem aleatorios, o bien quedarse con aquellas observaciones para las cuales el desacuerdo entre *co-trainers* es mayor, de manera que el aprendizaje en esta instancia sea más rico. Una vez elegidos los $z$ datos sin etiqueta $\boldsymbol{r}_1^u, \boldsymbol{r}_2^u, \ldots, \boldsymbol{r}_z^u$, los $N$ *co-trainers* predecirán el *rating* de la primera observación sin etiqueta, $\hat{\boldsymbol{r}}_{1Co-Tr_k}^u$, como se ve en la *Figura 10*.

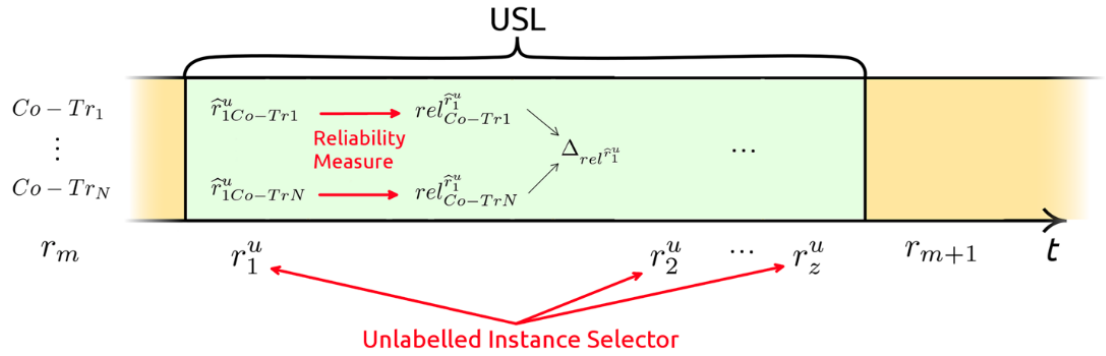


***Figura 10:*** *Entrenamiento no supervisado durante el stream traning set.*
*Fuente: [17]*

Esta vez, en lugar de utilizar el verdadero valor -que desconocemos- para intentar mejorar en la siguiente observación, los recomendadores se entrenarán entre sí: los peores aprenderán de las etiquetas de los mejores. Para ello, es necesaria una herramienta que nos permita evaluar la confianza que tenemos en cada una de las $N$ predicciones, definida como una función que mapea a cada *co-trainer* y su predicción entre el 0 y el 1, representando 1 la total confianza y 0, lo contrario. Esta función podría, por ejemplo, otorgar mayor confiabilidad a aquellos modelos que etiqueten parecido a la mayoría de los datos y menor a aquellos con una predicción muy distitna a la promedio, o podría beneficiar a los modelos menos sensibles a nueva información, o bien asignar aleatoriamente una medida de confianza a cada *co-triner* y su predicción.

Una vez calculadas las $N$ medidas de confianza $rel(\hat{\boldsymbol{r}}^{u}_{1\ Co-Tr_1}), rel(\hat{\boldsymbol{r}}^{u}_{1\ Co-Tr_2}), \dots, rel(\hat{\boldsymbol{r}}^{u}_{1\ Co-Tr_N})$, se computan todas las diferencias ente ellas:

$$\Delta rel(\hat{\boldsymbol{r}}^{u}_{1}, a, b) = \left| rel(\hat{\boldsymbol{r}}^{u}_{1\ Co-Tr_a}) - rel(\hat{\boldsymbol{r}}^{u}_{1\ Co-Tr_b}) \right|, \forall\ a, b \in \{1, \dots, N\}\ con\ a \neq b \qquad (3)$$

Tomando las $q$ mayores diferencias obtenidas en (3), estaremos eligiendo pares de *co-trainers* donde uno es muy confiable y el otro, poco confiable: el segundo tomará la etiqueta del primero como la verdadera, para evaluar su error y mejorar su predicción en el próximo *rating*. Este proceso se itera para todas las observaciones sin etiqueta de esta sección. Nuevas etapas de aprendizaje supervisado y no supervisado vendrán a continuación.

Una vez atravesadas todas las etapas de aprendizaje de la *Figura 8*, los *co-trainers* aún así pueden diferir en sus predicciones, por lo tanto, es necesario definir una función de agregación: al ponerse en práctica, ¿cuál será el *rating* definitivo estimado para un par usuario-ítem? Se tratará de un promedio ponderado de las $N$ etiquetas, donde el peso para cada modelo podría

ser, por ejemplo, la raíz de su error cuadrático medio, alguna función de confianza como la usada previamente, entre otros.

### *3.1.3 Enfoque self-training*

Un entrenamiento similar podría hacerse con el enfoque de *self-training*. Este es un caso particular del anterior. La diferencia es que, ahora, no necesitamos segmentar el *batch training set*: la totalidad será utilizada por el modelo. Además, la diferencia entre las medidas de confianza no podrá ser calculada, pues para cada *rating* contamos con una única predicción. Las etiquetas con mayor confianza serán tomadas como las verdaderas por el algoritmo y aprenderá de ellas en el próximo paso. Asimismo, no será necesario usar una función que agregue predicciones.

## 3.2 Etiquetando posteos: recomendaciones *cold-start*

*Semi-supervised Tag Recommendation - Using untagged resources to mitigate cold-start problems*

Los recomendadores de etiquetas, mejor conocidos como *Tag Recommender Systems*, sirven para simplificarle al usuario el proceso de etiquetado de posteos. Redes sociales como Facebook, Twitter, Instagram y YouTube, entre otras aplicaciones web con etiquetado social[4] disponible los utilizan. Las sugerencias *cold-start*, esto es, cuando se recomiendan *tags* para nuevos usuarios -aquellos que nunca han posteado con *tags* antes- o para nuevos contenidos -aquellos que nunca fueron etiquetados por usuarios del sistema-, es uno de los mayores desafíos

[4] El etiquetado social [19] es el sistema que le permite a los usuarios de aplicaciones web subir contenido -como pueden ser artículos, fotos o videos- junto con una lista de palabras clave libremente elegidas que sirven para clasificarlo. Tiene como objetivo ordenar el contenido del sitio y ayudar a sus usuarios a encontrar los recursos que les interesan.

de estos algoritmos, especialmente cuando la mayoría de los usuarios nunca etiquetan sus posteos y cuando la mayoría de los posteos nunca han sido etiquetados.

Los autores de este *paper* [20] proponen un recomendador semisupervisado basado en un grafo que busca mitigar aquel problema. Antes de su desarrollo, este problema típicamente se sorteaba recomendando los *tags* más populares o basadando las etiquetas en el contenido a publicar. Sin embargo, estas soluciones no son personalizadas y, más aún, la segunda no es genérica en tanto requiere un algoritmo distinto para cada tipo de contenido. Este trabajo aborda el problema de manera personalizada e independiente del contenido.

El sistema que los autores porponen no solamente podrá explotar la información estructural detrás de los datos *unlabeled* -en este caso, posteos sin *tags*-, sino que esto además le permitirá predecir etiquetas para los posteos hechos por nuevos usuarios o con nuevos recursos. El objetivo será convertir estos posteos en observaciones *labeled* que puedan utilizarse luego como información para sugerir *tags* precisas cuando, en una nueva ocasión, estos nuevos usuarios vayan a hacer una publicación o dichos nuevos recursos vayan a ser publicados. Notar que llamamos observación "*labeled*" o "etiquetada" a los posteos que tienen "*tags*" o "etiquetas" asociadas.

### *3.2.1 Datos: tags, nuevos usuarios y posteos*

Los datos disponibles para este modelo son un conjunto de usuarios $\boldsymbol{U}$, un conjunto de recursos $\boldsymbol{R}$, un conjunto de *tags* $\boldsymbol{T}$, y el conjunto de las relaciones ternarias que entre ellos haya habido $\boldsymbol{Y} \subseteq \boldsymbol{U} \times \boldsymbol{R} \times \boldsymbol{T}$. Sea $\boldsymbol{X}$ el set de todos los posteos, i.e., todos los pares de usuarios y recursos: $\boldsymbol{X} = \{\boldsymbol{x} = (u,r) |\, u \in \boldsymbol{U}, r \in \boldsymbol{R}\}$. Puede entonces definirse $\boldsymbol{X}_t \coloneqq \{(u,r) \in \boldsymbol{U} \times \boldsymbol{R} \mid \exists t \in \boldsymbol{T}: (u,r,t) \in \boldsymbol{Y}\}$ como el conjunto de todos los posteos publicados con etiquetas y $\boldsymbol{X}_s = \{(u,r) \in \boldsymbol{U} \times \boldsymbol{R} \mid \nexists t \in$

$\boldsymbol{T}$: $(u, r, t) \in \boldsymbol{Y}\}$ como el conjunto de todos los posteos sin etiquetas, de manera tal que $\boldsymbol{X}_t \uplus \boldsymbol{X}_s = \boldsymbol{X}$. Para el entrenamiento, el algoritmo tendrá acceso a todos los posteos con etiqueta y sus verdaderos *tags*, $\boldsymbol{T}|\boldsymbol{X}_t$. También aprenderá de los datos en $\boldsymbol{X}_s$.

El trabajo se refiere a un nuevo usuario $u$ como aquel que nunca ha posteado con etiquetas en el sistema, formalemente: $\boldsymbol{X}_t \cap (\{u\} \times \boldsymbol{R}) = \emptyset$. De igual manera, definen a los nuevos recursos $r$ como aquellos nunca antes etiquetados por ningún usuario del sistema: $\boldsymbol{X}_t \cap (\boldsymbol{U} \times \{r\}) = \emptyset$. Notar que, por definición, todos los posteos de nuevos usuarios o con nuevos recursos estarán en $\boldsymbol{X}_s$. Entonces, dado un posteo $\boldsymbol{x}_s \in \boldsymbol{X}_s$ el objetivo del modelo será predecir un conjunto de etiquetas $\hat{T}(\boldsymbol{x}_s) \subset \boldsymbol{T}$, incluso para los de nuevos usuarios o con nuevos recursos.

### *3.2.2 Métodos gráficos semisupervisados*

Este recomendador operará con datos estructurados en forma de grafo. Como se explica en la sección 2.1.3, esto le posibilita consumir, además de las etiquetas conocidas, la estructura de los datos *unlabeled* como información en sí mima. Los datos serán representados como un grafo $\boldsymbol{G} \coloneqq (\boldsymbol{X}, \mathcal{E})$, cuyos *edges* contienen ponderadores $w$: $\boldsymbol{X} \times \boldsymbol{X} \rightarrow \mathbb{R}$, de acuerdo a cuán similares sean dos posteos relacionados. Se asume que dos posteos están relacionados si comparten el usuario o el recurso. Así se definen los conjuntos $\boldsymbol{\mathcal{R}}_u = \{(\boldsymbol{x}, \tilde{\boldsymbol{x}}) \in \boldsymbol{X} \times \boldsymbol{X} \mid usuario(\boldsymbol{x}) = usuario(\tilde{\boldsymbol{x}})\}$ y $\boldsymbol{\mathcal{R}}_r = \{(\boldsymbol{x}, \tilde{\boldsymbol{x}}) \in \boldsymbol{X} \times \boldsymbol{X} \mid recurso(\boldsymbol{x}) = recurso(\tilde{\boldsymbol{x}})\}$, de manera que cada *edge* del grafo corresponde a una relación en $\boldsymbol{\mathcal{R}} = \boldsymbol{\mathcal{R}}_r \cup \boldsymbol{\mathcal{R}}_u$.

Como hemos adelantado, la tarea del sistema será ir avanzando por el grafo para lograr etiquetar todos los posteos de $\boldsymbol{X}_s$, comenzando por la información disponible en $\boldsymbol{X}_t$. En una primera aproximación, los autores proponen etiquetar una observación $\boldsymbol{x}_s \in \boldsymbol{X}_s$ con los *k tags* con mayor probabilidad de elegirse para ese posteo, utilizando el algoritmo *Weigthed Average (WA)* para

estimarla. Así, la probabilidad de que una etiqueta $t$ sea elegida para un posteo $\boldsymbol{x}_s$, es decir, de que $t \in T(\boldsymbol{x}_s)$, se computa como la suma de todos los ponderadores de los posteos vecinos de $\boldsymbol{x}_s$ que comparten ese *tag* $t$, normalizada por la suma de los ponderadores de todos los vecinos de $\boldsymbol{x}_s$:

$$P(t|x_s) = \frac{\sum_{x' \in V_{x_s} |\ t \in T(x')} w(x_s, x')}{\sum_{x' \in V_{x_s}} w(x_s, x')} \tag{4}$$

donde $\boldsymbol{V}_{x_s} \coloneqq \{\boldsymbol{x}' \in \boldsymbol{X} \mid (\boldsymbol{x}_s, \boldsymbol{x}') \in \boldsymbol{\mathcal{R}}\}$ es el conjunto de posteos vecinos de $\boldsymbol{x}_s$.

Ahora bien, ¿qué ocurre con este algoritmo si $\boldsymbol{x}_s$ tiene posteos *unlabeled* en sus vecinos? La única manera de incorporar esta información es transformar *WA* en un algoritmo iterativo, que los autores llamarán *WAOneShot*. Así, en la primera iteración, se etiquetan los posteos $\boldsymbol{x}_s$ utilizando sólo sus vecinos etiquetados; en la segunda iteración, las observaciones de $\boldsymbol{X}_s$ que continúen *unlabeled* se etiquetarán utilizando la información de sus vecinos etiquetados en la iteración previa. Esto se repite hasta que todo el conjunto $\boldsymbol{X}_s$ tenga etiquetas asignadas. El nombre de este algoritmo se debe a que los posteos nunca son re-etiquetados, una vez clasificados no vuelven a evaluarse sus etiquetas.

Notar que como *WAOneShot* utiliza (4), este algoritmo considera los *tags* de manera determinística: sin importar que en la primera iteración un conjunto de *tags* hayan sido asignados a la observación $\tilde{\boldsymbol{x}}_s$ con distintas probabilidades de ocurrencia, si en la siguiente iteración $\tilde{\boldsymbol{x}}_s$ es vecino directo de una observación todavía *unlabeled* $\tilde{\tilde{\boldsymbol{x}}}_s$, todos los *tags* $t \in \hat{T}(\tilde{x}_s)$ pesarán lo mismo a la hora de ser evaluados como potenciales etiquetas de $\tilde{\tilde{\boldsymbol{x}}}_s$. Para soslayar esta limitación, (4) puede extenderse a (5), y se llama *Probabilistic Weighted Average (PWA)*:

$$P(t|x_s) = \frac{\sum_{x' \in V_{x_s}} w(x_s, x') P(t|x')}{\sum_{x' \in V_{x_s}} w(x_s, x')} \quad (5)$$

De combinar la ecuación (5) con la implementación iterativa de *WAOneShot* resulta *PWAOneShot*, un algoritmo iterativo probabilístico y semisupervisado, que no sólo hace uso de las observaciones sin etiqueta, sino también de la incertidumbre asociada a sus propias predicciones para dichas observaciones, con el fin de etiquetar todos los posteos en $\boldsymbol{X}_s$. Notar que para *PWAOneShot* y *PWA*, $P(t|\boldsymbol{x}_t) = 1 \; \forall \boldsymbol{x}_t \in \boldsymbol{X}_t$.

Notar, además, que ninguno de los algoritmos hasta ahora explicados re-clasifican las etiquetas que asignan, incluso si en iteraciones posteriores se agregan *tags* a los posteos vecinos de algún posteo ya etiquetado. Por eso, los autores desarrollan *PWA**, una adaptación de *PWAOneShot* que, en cada iteración, reevalúa la probabilidad que cada posteo ya etiquetado $\ddot{\boldsymbol{x}}_s$ tiene de que cada etiqueta $t \in \boldsymbol{T}$ pertenezca a $T(\ddot{\boldsymbol{x}}_s)$ y, de ser necesario, remplaza los *tags* asignados $\hat{T}(\ddot{\boldsymbol{x}}_s)$. El algoritmo frena cuando converge, es decir, cuando la diferencia entre la probabilidad de la clasificación de la iteración $j$ y la de la iteración $j + 1$ es menor que un $\epsilon$ arbitrariamente chico; formalmente:

$$\sqrt{\frac{1}{|X_s| \cdot |T|} \sum_{x_s \in X_s} \sum_{t \in T} (P(t\,|x_s)^{i+1} - P(t|x_s)^i)^2} \; < \epsilon \quad (6)$$

### *3.2.3 Recomendaciones cold-start*

El sistema de recomendación semisupervisado que los autores proponen, *PWA**, es especialmente útil para abordar el problema de las recomendaciones *cold-start*, en tanto tiene

muchas fuentes de información para llegar a etiquetar estos posteos: datos *labeled*, datos *unlabeled*, provenientes de $\boldsymbol{\mathcal{R}}_u$ o incluso de $\boldsymbol{\mathcal{R}}_r$.

En la *Figura 11*, se ilustra este problema: los nodos grises representan posteos de $\boldsymbol{X}_s$ y los nodos blancos de $\boldsymbol{X}_t$. Notar que $u_8$ es un nuevo usuario y $r_6$ un nuevo recurso. Para recomendar *tags*, por ejemplo, para el posteo $(u_8, r_7)$, *PWA** utilizará información de $(u_1, r_7)$ y $(u_3, r_7)$ en $\mathcal{R}_{r_7}$, de sus propios posteos sin etiquetas $(u_8, r_6)$ y $(u_8, r_2)$ en $\mathcal{R}_{u_8}$ y también de $(u_4, r_2)$ en la medida en que los *tags* de este posteo sirvan para re-etiquetar a $(u_8, r_2)$. El algoritmo sirve incluso para etiquetar $(u_8, r_6)$, que si bien inicialmente está relacionado con posteos únicamente *unlabeled*, estos tienen relación con al menos un nodo *labeled* y *PWA** tiene la habilidad de propagar esta información por el grafo.

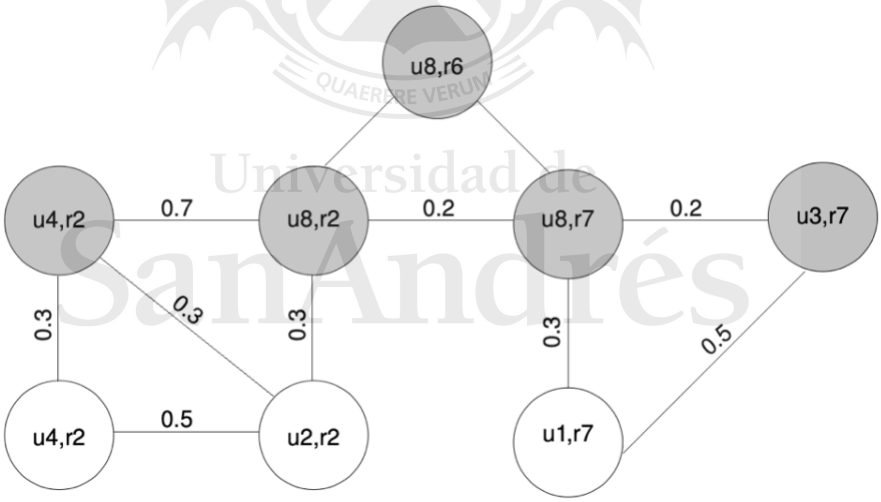


***Figura 11:*** *Grafo del nuevo usuario $u_8$ y del nuevo recurso $r_6$.*
*Fuente: [20]*

En la medida en que el grafo se compone de dos tipos de relaciones, cuando una no existe, hay un respaldo en la otra. El único escenario que *PWA** no soporta es aquel en que un nuevo usuario solamente ha posteado nuevos recursos.

## 3.3 Recomendando puntos de interés: un enfoque neuronal

*Bridging collaborative filtering and semi-supervised learning: a neural approach for POI recommendation*

La promimencia de aplicaciones que conocen la locación de sus usuarios, como Google Maps Uber o Yelp, y la recopilación de los historiales de *check-in*, esto es, si los usuarios visitaron o no cierto lugar, propulsó el surgimiento de recomendadores de puntos de interés o POI, por sus siglas en inglés. Estos algoritmos sugieren ubicaciones puntuales en un mapa a usuarios que pueden encontrarlas útiles o interesantes. Si bien tienen grandes beneficios para usuarios y empresas, son desafiantes cuando de construirlos y entrenarlos se trata: la escasez de información y el contexto de recomendación son los dos principales retos [10].

La escasez de información que sufren los recomendadores de puntos de interés es mucho mayor que la de otros sistemas de recomendación, pues la densidad de los datos de *check-in* es hasta diez veces menor que la de datos como los de Netflix[5]. Más aún, en lugar de contar con *feedback* explícito de *ratings* por parte de los usuarios, sólo *feedback* implícito y binario, obtenido del historial de *check-in* de los usuarios, está disponible en este caso. Son estas condiciones las que, según los autores, llevan directamente a la ineficiencia del enfoque tradicional de *matrix factorization* para este problema.

Además, el contexto del usuario en el que la recomendación tendrá lugar es una fuente de información rica y relevante para la recomendación en sí misma. La preferencia de los usuarios sobre cierto punto de interés está determinada por su movilidad y la distancia al mismo, por sus

[5] En octubre del 2006, Netflix publicó un *data set* con cien millones de *ratings* de películas generados anónimamente por sus usuarios y desafió a las comunidades de *machine learning*, ciencia de datos y ciencias de la computación del mundo a desarrollar un sistema de recomendación que superara la precisión del vigente en ese entonces a cambio de un millón de dólares. Este desafío es conocido como The NetflixPrize [21].

lazos sociales e incluso cambia a lo largo del día. La complejidad y variedad de contextos en los que estas recomendaciones ocurren dio lugar al surgimiento de múltiples recomendadores específicos para cada uno. Sin embargo, estos modelos ad-hoc son inestables a través de diferentes datos y carecen de un marco general que pueda fácilmente tener en cuenta varios contextos y beneficiarse automáticamente de los más importantes para producir recomendaciones personalizadas, satisfactorias y estables. Esto es exactamente lo que Yang, Carl et al. [10] proponen en su publicación.

### *3.3.1 Enfoque semisupervisado*

El aprendizaje semisupervisado resulta una forma conveniente de mitigar ambos problemas: aprender de forma no supervisada los contextos de recomendación permite incorporar al modelo más información que sólo la del historial de *check-in* de los usuarios.

En la recomendación de puntos de interés, el historial de *check-in* opera como información *labeled*: sabemos que un usuario efectivamente estuvo en un lugar y eso revela interés de su parte hacia ese POI. Información *unlabeled* sobre el contexto puede aprenderse de forma no supervisda y usarse para enriquecer la recomendación: la información social y geográfica que se extrae de la relación que hay, por un lado, entre usuarios y, por otro, entre POIs puede naturalmente representarse en dos grafos. Otra clase de contextos, como información temporal o categórica podría eventualmente incorporarse también.

El aporte principal de los autores es, entonces, una estructura de redes neuronales profundas llamada PACE (*Preference And Conetxt Embedding*), que integra con éxito en un mismo modelo los métodos de *matrix factorization* y de *graph Laplacian regularization*. Se trata de un marco que permite de forma generalizada realizar recomendaciones del tipo *Collaborative*

*Filtering* a la vez que respeta la información de varios contextos contenida en *graph-embeddings*.

### *3.3.2 Arquitectura neuronal*

Tal como anticipamos, el algoritmo propuesto se nutrirá de una matriz de *check-in* de dimensión $N \times M$, donde cada fila representa los POIs visitados por un usuario $u_i$ con $i \in N$ y cada columna, los visitantes de un POI $p_j$ con $j \in M$. El objetivo ulterior del modelo será aprender una matriz de preferencias de dimensión $N \times M$ compuesta por ceros y unos, que en esta ocasión indican si un usuario $u_i$ tiene interés en el POI $p_j$ o no. Para incorporar la información del contexto, será necesario que a la vez el modelo aprenda los grafos: por un lado, los ponderadores del grafo que vincula a todos los usuarios entre sí y, por otro, los del que vincula a los todos los POIs entre sí. Esto es lo que llamamos contexto social y georgráfico: qué tan parecidos son entre sí dos usuarios o POIs dados.

Como en otros modelos donde se incorpora tanto información etiquetada como no etiquetada, el algoritmo buscará minimizar una función de pérdida compuesta por la pérdida supervisada y la no supervisada. En general, estas funciones pueden escribirse como

$$\sum_{i \in \mathcal{L}} l(y_i, \hat{f}(\boldsymbol{x}_i)) + \lambda \sum_{i,j} a_{ij} \left\| \hat{f}(\boldsymbol{x}_i) - \hat{f}(\boldsymbol{x}_j) \right\|^2 \tag{7}$$

donde $\mathcal{L}$ es el conjunto de observaciones etiquetadas, $a_{ij}$ expresa la similitud entre los vectores $\boldsymbol{x}_i$ y $\boldsymbol{x}_j$, $\hat{f}(\cdot)$ es la función aprendida para la predicción y $l(\cdot)$ una función de pérdida cualquiera. Notar que el primer término representa cuán distinta es la predicción respecto de los verdaderos valores para las observaciones etiquetadas, esto es la pérdida supervisada; y el segundo, cuán disímiles son las predicciones para dos observaciones distintas ponderadas por su grado de similitud, es decir, la pérdida no supervisada.

En este caso, la función de pérdida tomará la forma

$$\boldsymbol{\mathcal{J}} = \mathcal{J}_P + \lambda_1 \mathcal{J}_{C_u} + \lambda_2 \mathcal{J}_{C_p} , \tag{8}$$

donde $\mathcal{J}_P$ es la pérdida sobre los datos etiquetados o la matriz de *check-in*, $\mathcal{J}_{C_u}$ y $\mathcal{J}_{C_p}$ son las pérdidas sobre el contexto de usuarios y de POIs respectivamente -análogas al segundo término de (7)- y $\lambda_1$ y $\lambda_2$, ponderadores del *trade-off* entre los tres objetivos. Veremos que la estructura neuronal de PACE, en esencia, busca minimizar los tres componentes de $\boldsymbol{\mathcal{J}}$ a la vez: el fin será aprender conjuntamente una función de predicción del interés $\hat{h}(\cdot)$ y los coeficientes de similitud de dos grafos de manera tal que $\boldsymbol{\mathcal{J}}$ sea mínimo, esto es, que la matriz de preferencias resultante señale que los usuarios que visitaron un cierto lugar tienen interés por éste, que usuarios similares reciban recomendaciones similares y que POIs similares sean recomendados a usuarios similares. Este aprendizaje conjunto se estructura como ilustra la *Figura 12*.

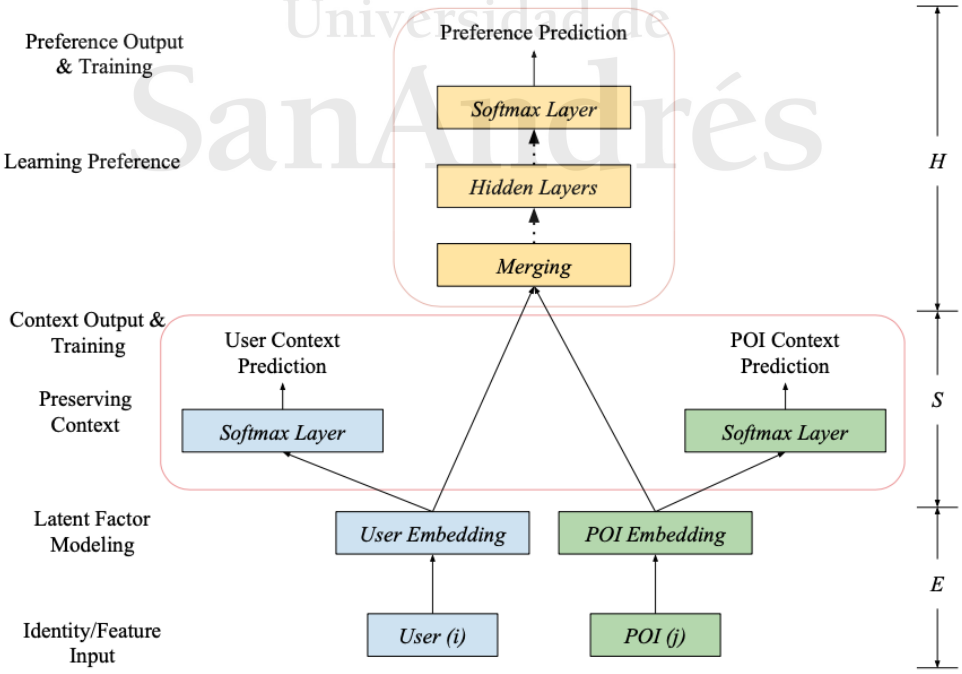


***Figura 12:*** *Arquitectura neuronal de PACE.*
*Fuente: [10]*

En una primera instancia $E$, el modelo aprende, a partir de la matriz inicial de *check-in*, dos matrices de *embeddings*, $\boldsymbol{E}_u \in \mathbb{R}^{N \times K_u}$ y $\boldsymbol{E}_p \in \mathbb{R}^{M \times K_p}$, a fin de reducir -horizontalmente- la dimensión de los vectores de usuarios de $M$ a $K_u$ y -verticalmente- la de los vectores de puntos de interés de $N$ a $K_p$. A continuación, $\boldsymbol{E}_u$ y $\boldsymbol{E}_p$ nutrirán, tal como se ilustra en la *Figura 12*, a tres redes neuronales que trabajarán simultáneamente en reducir la función de pérdida $\boldsymbol{\mathcal{J}}$. Notar que la instancia $\mathcal{H}$ se alimenta de una fusión de ambas matrices.

Cada red neuronal de la instancia $\mathcal{S}$ se ocupará de un grafo y buscará encontrar los ponderadores que minimizan $\boldsymbol{\mathcal{J}}$ a la vez que la red reuronal de la instancia $\mathcal{H}$ determina $\hat{h}(\cdot)$. Notar que dependiendo de cuál sea el contexto social y geográfico predicho, la función $\hat{h}(\cdot)$ tendrá unos parámetros u otros. Esto quiere decir que los esfuerzos de la instancia $\mathcal{S}$ por ajustar los contextos están contenidos en los esfuerzos de $\mathcal{H}$ por determinar $\hat{h}(\cdot)$. Es de esta forma que la información contextual aprendida no supervisadamente por los dos grafos enriquece a los datos crudos de la matriz de *check-in* y mejora la predicción finalmente.

En particular, la forma de $\hat{h}(\cdot)$ resultará del anidamiento de las transformaciones que la red neuronal en $\mathcal{H}$ vaya ejecutando sobre cada vector de la matriz que la alimenta. Finalmente, la matriz de preferencias se formará de los $N \times M$ resultados de $\hat{y}_{ij} = \hat{h}(\boldsymbol{E}_{ui}, \boldsymbol{E}_{pj})$, donde $\boldsymbol{E}_{ui}$ es la versión reducida del vector de *check-ins* de del usuario $u_i$ y $\boldsymbol{E}_{pj}$, la del punto de interés $p_j$.

## 4 Implementación en Python

En esta sección explicamos cómo puede implementarse un sistema de recomendación en Python. Un algoritmo semisupervisado escapa al alcance de este trabajo. Presentamos, entonces, un recomendador del tipo *Collaborative Filtering* desde el enfoque supervisado.

Para la construcción de este recomendador (ver Apéndice), se utilizó el *data set* [6] de un millón de observaciones de MovieLens [22]. Contiene un listado de películas, con su título y sus géneros, y una matriz de *ratings*: 6040 usuarios han votado 3706 películas con un puntaje entre 1 y 5. Los valores en 0 indican ausencia de *rating*.

A partir de la matriz de *ratings* incompleta $R$, el objetivo es predecir los *ratings* para todas las combinaciones de usuarios e ítems. Y, una vez hecho esto, construir una función que, dado un usuario, selecciona las películas que éste aún no ha visto -asumiendo que se trata de aquellas que no ha votado-, las ordena según el *rating* que se predijo que el usuario les hubiera asignado y devuelve las primeras diez.

Para completar el primer paso, se utilizó un procedimiento como el de *matrix factorization* explicado en la Sección 2.3. La función empleada (*Singular Value Decompostion*) realiza una descomposición de la matriz de *ratings* $R$ en dos matrices unitarias $U$ y $V$ y una matriz diagonal tal que $R \cong U\Sigma V^T$. La matriz $U$ captura cuánto le gustan a cada usuario las distintas características de las películas. Por su parte, la matriz $V$ explica cuán relevantes son esas características para cada película. La cantidad de características que se utilizaron para la predicción de los *ratings* es un parámetro a seleccionar a la hora de correr esta función.

Obtenidas las matrices parciales, se procedió a hacer la multiplicación matricial que recupera la matriz completa de *ratings* predichos, $\hat{R} = U\Sigma V^T$, y a construir la función de recomendación.

[6] Disponible en https://grouplens.org/datasets/movielens/

# 5 Conclusión

Los algoritmos de recomendación no sólo han dado lugar al surgimiento de unicornios tecnológicos como Amazon o Netflix: el perfeccionamiento de su desempeño ha terminado por revolucionar el comportamiento de la oferta en ciertos mercados. La industria y la academia han estudiado de cerca este problema desde los noventa. Pero en estos últimos años, la incorporación del enfoque semisupervisado ha cambiado por completo las reglas del juego.

Con la intención de tener un mejor entendimiento de estas herramientas, hemos recorrido algunas publicaciones que proponen sistemas de recomendación semisupervisados para distintos propósitos: *e-commerce*, *social tagging*, puntos de interés. Además, mostramos cómo puede implementarse fácilmente un recomendador supervisado.

Se podría extender el alcance de trabajo estudiando el impacto económico del fenómeno. También, podría hacerse una revisión literaria desde un enfoque computacional o indagar en aplicaciones fuera de los negocios.

## Apéndice

A continuación, el código en lenguaje Python del recomendador[7] implementado.

```
import pandas as pd
import numpy as np

# Cargamos los datos: una lista de las películas y la matriz de ratings

ratings_list = [i.strip().split("::") for i in open('ratings.dat', 'r').readlines()]
movies_list = [i.strip().split("::") for i in open('movies.dat', 'r', encoding='UTF-
8', errors='ignore').readlines()]

ratings_df = pd.DataFrame(ratings_list, columns = ['UserID', 'MovieID', 'Rating',
'Timestamp'], dtype = int)
movies_df = pd.DataFrame(movies_list, columns = ['MovieID', 'Title', 'Genres'])

# Conviertimos UserID y MovieID de string a numeric en ambos dataframes, pues más
adelante lo vamos a necesitar para poder ejecutar un .merge() de los dos datasets

movies_df['MovieID'] = pd.to_numeric(movies_df['MovieID'])
ratings_df['MovieID'] = pd.to_numeric(ratings_df['MovieID'])
ratings_df['UserID'] = pd.to_numeric(ratings_df['UserID'])

# Convertimos la matriz de ratings en una que tenga los usuarios en filas y las
películas en columnas

R_df = ratings_df.pivot(index = 'UserID', columns = 'MovieID', values =
'Rating').fillna(0)

# Para poder utilizar esta matriz en el recomendador necesitamos 1. convertir el data
frame en un numpy array y 2. normalizar por la media de cada usuario

R = R_df.to_numpy()
R = R.astype(np.float)
user_ratings_mean = np.mean(R, axis = 1)
R_demeaned = R - user_ratings_mean.reshape(-1, 1)

# A continuación, descomponemos la matriz de ratings

from scipy.sparse.linalg import svds
U, sigma, Vt = svds(R_demeaned, k = 50)
sigma = np.diag(sigma)

# Usaremos su producto matricial para predecir la matriz de ratings completa con la
que, finalmente, haremos las recomendaciones

all_user_predicted_ratings = np.dot(np.dot(U,sigma),Vt) +
user_ratings_mean.reshape(-1,1)
preds_df = pd.DataFrame(all_user_predicted_ratings, columns = R_df.columns)
```

[7] Disponible en https://github.com/nataliafonzo/LaOfertaEnElSigloXXI/blob/master/Matrix%20factorization.ipynb

```
# Creamos la función que, consumiendo la matriz de ratings predichos, recomienda a
un usuario las películas que más podrían gustarle y aún no vio

def recommend_movies(predictions_df, userID, movies_df, original_ratings_df,
num_recommendations=5):

   # Obtenemos los ratings predichos para el usuario en cuestión y ordenamos sus
   ratings de mayor a menor
   user_row_number = userID - 1 #UserID comienza en 1 en lugar de 0
   sorted_user_predictions =
   predictions_df.iloc[user_row_number].sort_values(ascending=False)
   sorted_user_predictions = pd.DataFrame(sorted_user_predictions).reset_index()
   sorted_user_predictions['MovieID'] =
   pd.to_numeric(sorted_user_predictions['MovieID'])

   # De la información original de este usuario, obtenemos cuántas y cuáles
   películas ya vio
   user_data = original_ratings_df[original_ratings_df['UserID'] == userID]
   user_full = user_data.merge(movies_df, how = 'left', left_on = 'MovieID',
   right_on = 'MovieID').sort_values(['Rating'], ascending=False)

   # Recomendamos las primeras que no vio con mayor rating predicho
   recommendations =
   (movies_df[~movies_df['MovieID'].isin(user_full['MovieID'])].merge(sorted_user_pr
   edictions, how = 'left',left_on = 'MovieID',right_on = 'MovieID').rename(columns
   = {user_row_number: 'Predictions'}).sort_values('Predictions', ascending =
   False).iloc[:num_recommendations, :-1])

   return user_full, recommendations

#Corremos la función de recomendación para algún UserID

already_rated, predictions = recommend_movies(preds_df, 2, movies_df, ratings_df, 10)
already_rated.head()
predictions.head(10)
```

# Referencias


[1] M. Zhang, J. Tang, X. Zhang y X. Xue, "Addressing cold start in recommender systems: A semi-supervised co-training algorithm", en *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, 2014, pp. 73-82.

[2] L. Walker, (1998, Nov. 8). *Amazon Gets Personal With E-Commerce* [Online]. Disponible: https://www.washingtonpost.com/wp-srv/washtech/daily/nov98/amazon110898.htm

[3] C. Anderson. (Oct 2008). *The Long Tail: Why the future of the business is selling less of more* [Online]. Disponible: http://dl.motamem.org/long_tail_chris_anderson_motamem_org.pdf

[4] E. Brynjolfsson, Y. J. Hu y M. D Smith, "From Niches to Riches: The Anatomy of the Long Tail", *Sloan Management Review*, vol. 47, no. 4, pp. 67-71, Jul 2006.

[5] E. Brynjolfsson, Y. J. Hu y M. D Smith, "A longer Tail?: Estimating the Shape of Amazon's Sales Distribution Curve in 2008", en *Workshop of Information Systems and Economics*, 2009, pp. 1-6.

[6] H. Z Tang y T. Bresnahan. (May 2014) "The Collaborative Filtering Effect of Netflix Ratings for Indie Films versus Blockbusters and Heavy Users versus Casual Users". Disponible: https://economics.stanford.edu/sites/g/files/sbiybj9386/f/publications/henrytanghonorsthesismay2014.pdf

[7] K. P. Murphy. *Machine Learning: a probabilistic perspective*. Cambridge: MIT Press, 2012.

[8] Z. Xiaojin, "Semi-supervised Learning Literature Survey", 2005.

[9] O. Chapelle, B. Schölkopf y A. Zien, *Semi-Supervised Learning*. Cambridge: MIT Press, 2006.

[10] C. Yang, L. Bai, C. Zhang, Q. Yuan y J. Han, "Bridging collaborative filtering and semi-supervised learning: a neural approach for poi recommendation", en *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1245-1254.

[11] J. Friedman, T. Hastie y R. Tibshirani, *Elements of Statistical Learning*. New York: Springer series in statistics, 2001.

[12] S. K. Gorakala y M Usuelli, *Building a recommendation system with R*. Birmingham: Packt Publishing Ltd, 2015.

[13] Google. (2020). *Content-based Filtering* [Online] Disponible: https://developers.google.com/machine-learning/recommendation/content-based/basics

[14] Google. (2018). *Content-based Filtering Advantages and Disadvantages* [Online] Disponible: https://developers.google.com/machine-learning/recommendation/content-based/summary

[15] Google. (2020). *Matrix Factorization* [Online] Disponible: https://developers.google.com/machine-learning/recommendation/collaborative/matrix

[16] Google. (2020). *Embeddings* [Online] Disponible: https://developers.google.com/machine-learning/crash-course/embeddings/video-lecture?hl=en

[17] P. Matuszyk y M. Spiliopoulou, "Stream-based semi-supervised learning for recommender systems", *Machine Learning*, vol. 106, no. 6, p. 771-798, Feb 2017.

[18] G. Takács, I. Pilászy, B. Németh, y D. Tikk, "Scalable collaborative filtering approaches for large recommender systems", *The Journal of Machine Learning Research*, vol. 10, pp. 623-656, Mar 2009.

[19] L. B. Marinho et al., *Recommender systems for social tagging systems*. New York: Springer, 2012.

[20] C. Preisach, L. B. Marinho y L. Schmidt-Thieme, "Semi-supervised tag recommendation-using untagged resources to mitigate cold-start problems", en *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2010. pp. 348-357.

[21] Netflix. *Netflix Prize* [Online]. Diponible: https://www.netflixprize.com/

[22] F. Maxwell Harper y J. A. Konstan, "The MovieLens Datasets: History and Context", *ACM Transactions on Interactive Intelligent Systems (TiiS)*, vol. 4, no. 4, Dic 2015.