Universidad ORT Uruguay

Facultad de Ingeniería

Investigación e implementación de sistemas de recomendación para e – commerce de ropa

Entregado como requisito para la obtención del título de Maestría en Big Data

Alan Rytt - 166.675

Gonzalo Rodríguez - 238.334

Federico Young - 249.892

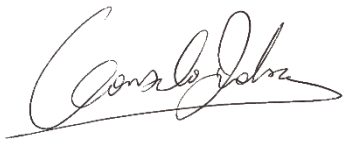
Tutores: Alexis Quintana y Fernando López

2021

Declaración de autoría

Nosotros, Alan Rytt, Gonzalo Rodríguez y Federico Young, declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos [nombre de la actividad curricular que origina la obra];
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;
- En la obra, hemos acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.



Alan Rytt	Gonzalo Rodríguez	Federico Young
24/09/2021	24/09/2021	24/09/2021

Agradecimientos

Queremos agradecer a nuestras familias que siempre han sido un apoyo incondicional a lo largo de este proceso.

También agradecer a nuestros tutores, Alexis Quintana y Fernando López, por habernos asesorado y guiado en el trascurso de proyecto. Mención especial también para “La Quia” quien confió en nosotros y permitió que utilicemos sus datos como parte de la experimentación en los motores de recomendación.

En el caso de la empresa “Rent the Runway” agradecer por disponibilizar sus datos a la comunidad, que nos permitieron realizar diferentes pruebas.

Por último, agradecer a la universidad ORT y a todos aquellos que colaboraron brindando información útil para el desarrollo de esta tesis.

Abstract

Los sistemas de recomendación en el comercio electrónico han tomado un papel preponderante en los últimos años como consecuencia del crecimiento en el volumen de datos y el avance de la tecnología de *Big Data*, a tal punto que hoy en día representa uno de los principales canales de ventas.

Los mismos tienen como objetivo mejorar la experiencia de los usuarios al brindar un contenido personalizado en función de los intereses y análisis de patrones de comportamiento, permitiendo mediante ello un aumento de la tasa de conversión. Los avances en inteligencia artificial y *machine learning* han permitido la adaptabilidad de estos sistemas a cualquier ecosistema electrónico.

El proyecto se centra en los *e-commerce* del rubro vestimenta, para obtener un estado del arte desde varios enfoques : filtrado colaborativo, filtrado basado en contenido y sistemas híbridos. Como objetivo se plantea obtener la cantidad de recomendaciones más relevantes y personalizadas de prendas por usuario.

Dentro de este marco, se realiza la experimentación de los siguientes algoritmos: *Market-basket*, Contenido basado en NLP (*natural language processing*), Contenido basado en atributos, KNN (*k-nearest neighbors*), matriz de similitud, basados en factorización matricial como ALS (A*lternative Least Square*) y SVD (*Singular Value Descomposition*), y diferentes variantes de sistemas híbridos.

Para medir el rendimiento se utilizó principalmente F1 (debido a que considera conjuntamente *Precision y Recall*) y MAR@K (*Mean average Recall at k*). Esta última es una métrica novedosa porque contempla el orden de relevancia de los productos recomendados según la preferencia predicha para el usuario. El mejor resultado fue obtenido con el modelo "*Switching hybrid*" el cual mejoro un 15% sobre el *baseline* planteado.

En líneas generales, los algoritmos basados en contenido mostraron una mejor performance que los colaborativos. Algunas de las posibles causas son: el problema de *cold start* de ítems, dado que se presentan varios productos con pocas transacciones, y la ventaja de poseer descripciones de mucho valor a través de las reseñas. Adicionalmente, se logró una leve mejora a los algoritmos de contenido con la implementación de los motores híbridos.

Palabras Clave

Sistema de recomendación, filtrado colaborativo, *content base*, híbrido, *cold start*, matriz de similaridad.

Indice

Índice de tablas

Índice de ilustraciones

1. Contexto General

En la actualidad, el volumen de compras por internet y la generación de datos desde los *e-commerce* están creciendo de forma exponencial, esto se debe a los beneficios que brindan al usuario y los avances en la confianza de las compras web.

La inteligencia artificial en el comercio electrónico es parte de cualquier estrategia para mejorar la experiencia por parte de los usuarios. Atrae y fideliza a los consumidores con un mejor entendimiento de sus comportamientos, para generar recomendaciones de los productos más relevantes y atractivos para el usuario.

Deesta forma surgen los motores de recomendación, como enlace entre la tecnología, los *e-commerce* y los usuarios. Ayudan a personalizar la oferta, mediante algoritmos basados en tiempo real, a partir de conductas anteriores, transacciones, datos de producto, entre otros. Evaluando el grado de interés de determinado usuario con los productos y sus similares.

Estos sistemas no solo se encargan de generar una excelente experiencia de compra para los usuarios, sino que ayudan a las empresas a potenciar sus ventas e invertir los recursos en estrategia de forma más efectiva.

Un dato que apoya lo anteriormente comentado es que según Invesp [1], una consultora especializada en analítica de *e-commerce,* "*la recomendación de productos personalizados en solo el 7% de los visitantes genera el 26% de las ganancias de la compañía*".

La personalización de este proceso es una parte fundamental en las estrategias de *e-commerce*. De acuerdo con Invesp [1] "*Amazon, uno de los mayores representantes de estos sistemas, genera cerca del 35% de sus ingresos con su motor de recomendación*".

"Se podría decir que, a mejores recomendaciones, carritos más llenos", indica Javier Pérez, experto en Big Data en Smartup [2].

Por lo anteriormente expresado, el presente estudio profundiza en esa problemática, así como en proponer una solución posible.

2. Visión emprendedora

Es importante destacar que, debido a la ambición y dedicación por parte del equipo del proyecto, se apunta a crear un emprendimiento donde se refleje lo aprendido, lo experimentado y las ganas de innovar en el mundo de los motores de recomendación.

Por tal motivo, la propuesta se proyecta a posicionarse como un socio clave en la recomendación inteligente y personalizada de productos basados en algoritmos de *machine learning.*

Se busca brindar un nuevo canal de ventas, específico y personalizado para las tiendas de ropa, donde se ofrezcan productos seleccionados mediante algoritmos de inteligencia artificial (IA), los cuales tengan la capacidad de adaptarse a las características del negocio.

Cabe destacar que como emprendedores, nos hemos presentado en paralelo a esta tesis a dos instancias para la obtención de fondos de validación al Carolan Fund a través del CIE, en donde como resultado se obtuvo un perfeccionamiento de la propuesta de valor.

3. Objetivos Específicos

- Realizar el relevamiento de la situación actual de los diferentes sistemas inteligentes de recomendación.

- Buscar un *dataset* con ejemplos reales de interacción entre los usuarios y las plataformas.

- Realizar la ingeniería de datos, a los efectos de ordenar, limpiar y seleccionar aquellos atributos que se entiendan sean los más apropiados para optimizar el rendimiento del motor de recomendación seleccionado.

- Implementar un motor de recomendación en el rubro de los *e-commerce* de ropa.

- Seleccionar un método de evaluación para validar la performance del motor de recomendación.

- Diseñar un marco arquitectónico de referencia para la implementación de un motor de recomendación.

- Investigar los problemas típicos a los que se enfrentan los motores de recomendación.

4. Marco teórico

4.1. Breve descripción de los distintos motores de recomendación

Los modelos básicos de sistemas de recomendación trabajan con dos tipos de datos: 1) Interacciones usuario-ítem, como *ratings* o patrones de comportamiento de los usuarios, 2) información de atributos de los usuarios y los ítems, como caracterización de los productos o perfiles de usuarios. Los primeros, son llamados métodos de filtrado colaborativo (o *collaborative filtering*), mientras que los segundos, son denominados métodos basados en contenido (o *content based*).

- ***Collaborative filtering -based*** , o CF de aquí en adelante

Las técnicas de filtrado colaborativo se basan en la premisa de que usuarios que evaluaron de la misma forma a un producto, vayan a estar de acuerdo sobre la evaluación de un nuevo producto en el futuro. El foco de los motores de CF son las relaciones entre usuarios e ítems a partir de datos históricos, construyendo matrices de similitud usuario-*items* con los valores de *ratings* o evaluaciones que dichos usuarios realizaron sobre los ítems.

El principal desafío que afrontan estos modelos es que la matriz sobre la cual trabajan suele ser dispersa, ya que es probable que solo una porción de los usuarios calificó a cada uno de los ítems, por lo que la idea se centra en imputar estos *ratings* que no fueron explícitos. Por ejemplo, dos usuarios (A y B) que tienen gustos parecidos (bajo la premisa de que *ratings* similares sobre los mismos ítems implica gustos similares). Además, solo el usuario A evalúo otro ítem, es probable que el usuario B evalúe al ítem con la misma calificación.

Según el modo en cómo son inferidos estos *ratings*, hay dos tipos de CF:

- *User-based*: Los *ratings* provistos por un set de usuarios similares a un usuario objetivo A son utilizados para hacer recomendaciones para A. Entonces, la idea principal es determinar usuarios similares a A e inferir los *ratings* que no fueron calificados por el usuario computando promedios ponderados de este set de usuarios similares.

- *Item-based*: Con el objetivo de hacer predicciones para un ítem B por el usuario A, el primer paso es determinar un set de ítems S que son similares al target ítem B. Los *ratings* por A sobre el set de ítems S son utilizados para predecir si a el usuario A le gustará el ítem B.

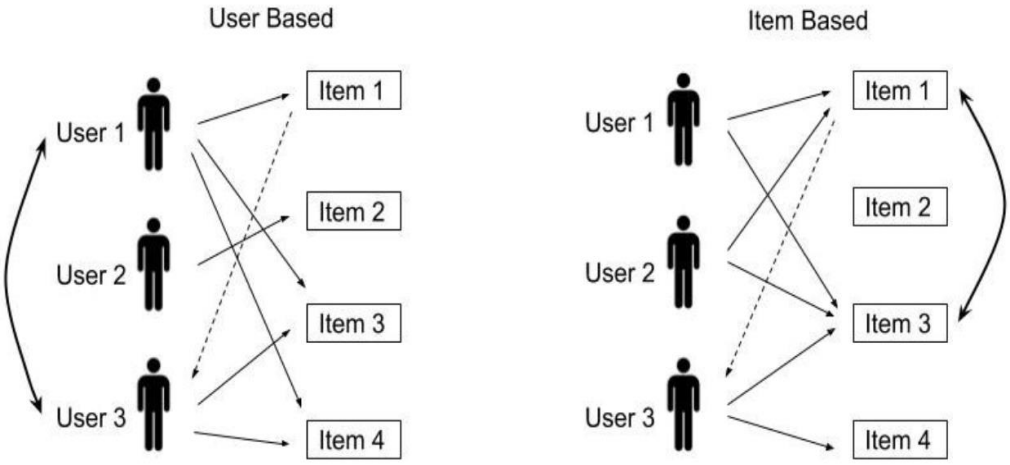


Ilustración 1 - Similaridades basadas en usuarios vs basadas en ítems [3]

- ***Content-based,*** o CB de aquí en adelante

En los modelos de recomendación basados en contenido, se utilizan las características descriptivas de los ítems para hacer las recomendaciones. El término "content" se refiere a estas descripciones. Los mecanismos para extraer dichos atributos pueden ser desde las características explícitas de los ítems (color, material, moda, etc) hasta a través de algoritmos de NLP identificando palabras claves en descripciones de los ítems o en *reviews* de los usuarios sobre los ítems.

La idea detrás de estos motores de recomendación es buscar similaridad entre ítems a través de dichos atributos. Si a un usuario A le gusto un ítem B, se buscarán los ítems similares a B en función de sus atributos y se realizará la recomendación con dichos ítems similares.

Una de las principales ventajas que tiene respecto a los colaborativos es cuando no se tiene suficiente información histórica de los ítems y sus respectivas calificaciones. Estos motores

funcionan bien para hacer recomendaciones de ítems nuevos, dado que no precisan datos históricos de los ítems, simplemente con los atributos que lo definen se buscan similitudes.

En lo que refiere a las desventajas, se puede destacar que la recomendación puede caer en un círculo cerrado de ítems con mismas características, y reducir la variedad de ítems recomendados. Si un ítem con un particular set de atributos nunca fue visto, es probable que el mencionado ítem nunca entre en una recomendación.

Adicionalmente se destaca el hecho de que, funciona bien para ítems nuevos, pero no así para usuarios nuevos.

- **Híbridos**

Los métodos híbridos de recomendación combinan algunos de los enfoques anteriores para cubrir las limitaciones individuales de los modelos. Más adelante se detallan en la sección 'Modelos híbridos de recomendación' las diferentes arquitecturas.

- **Otros modelos de recomendación (se brinda una breve descipcion ya que están fuera del alcance del marco de esta Tesis)**

    - ***Utility-based*:** Busca predecir la utilidad para los usuarios de diferentes ítems. Esto se realiza teniendo en cuenta las necesidades y limitaciones de cada usuario. Luego de calculado el score de utilidad, recomienda el de mayor score.

    - ***Demographic*:** Este tipo de motores de recomendación se basan en datos demográficos de los usuarios para recomendar.

    - ***Knowledge-based***: Este tipo de motor de recomendación se basa en formular las necesidades y preferencias del usuario para luego identificar si un elemento coincide o no con los criterios específicos de los usuarios.

4.2.*Rating* explícito vs *Rating* implícito

Algunos tipos de motores de recomendación, como es el caso de los CF, requieren de la utilización de *ratings* para identificar las preferencias de un usuario por los ítems.

Este rating puede ser capturado de dos formas:

- *Rating* explícito: Este ocurre cuando se le indica al usuario que explícitamente indique el *rating* de un ítem. Por ejemplo, cuando luego de una compra se pide que se indique qué tan satisfecho se esta con el ítem.

  Este tipo de *rating* tiene los siguientes problemas asociados:

  - No son fáciles de conseguir, ya que la gran mayoría de usuarios no califica los ítems.
  - Los usuarios suelen agregar *ratings* cuando los ítems no cumplen las expectativas, lo que produce sesgos no deseados en la información.
  - Son una expresión puntual de agrado o no agrado de un ítem. Por ejemplo, puede pasar que un usuario agregue un *rating* alto a un ítem , pero si a la semana le deja de funcionar el resto de los usuarios van a seguir pensando que la compra fue exitosa.
  - En [4] se han demostrado inconsistencias en los *ratings* de los usuarios cuando se los consulta para evaluar un mismo ítem en diferentes momentos.

- *Rating* implícito: Ocurre cuando no se solicita que el usuario indique el *rating* para un ítem, sino que se deduce de su comportamiento con el ítem. Por ejemplo, cuando se clickea en añadir un producto al carrito. El usuario demuestra tener un interés más alto de ese ítem en relación con el resto de los visualizados solamente.

  Este tipo de *rating* tiene las siguientes desventajas:

  - No sabe diferenciar de qué usuario son las preferencias. Si un usuario hoy compra algo para él y otro día para otra persona, las recomendaciones se harán en función de ambas preferencias sin discriminar quién es el usuario final.

La selección del criterio del *rating* es algo que no es trivial y que se debe evaluar en cada caso.

4.3.Matrices de similaridad

Un elemento muy utilizado en los motores de recomendación son las matrices de similaridad, debido a que resumen el grado de relación entre dos entidades, midiendo la semejanza o desigualdad entre los atributos comparados.

Se basa en vectorizar toda la información para cada entidad. En el caso práctico, se vectorizan todos los *ratings* asignados por cada usuario a todas las prendas de ropa alquiladas. Luego, con esta información se calcula la distancia para determinar la similitud entre vectores de usuarios. En base a los *ratings* de usuarios similares se construye el *rating* predicho. Este proceso también puede ser realizado con la información de ítems, siendo en este caso *item-based*.

Lo interesante de este método es que puede aplicarse tanto como algoritmo CB como CF. La diferencia radica en la información comparada en los vectores.

En cuanto al cálculo de similaridad, este es un valor que indica que tan relacionados están dos componentes. Se suelen utilizar medidas de distancia para representarla, por ejemplo: la correlación de *Pearson*, similitud del coseno, la distancia *Manhattan*, la distancia euclideana, etc. A continuación, se mencionan cada una de ellas:

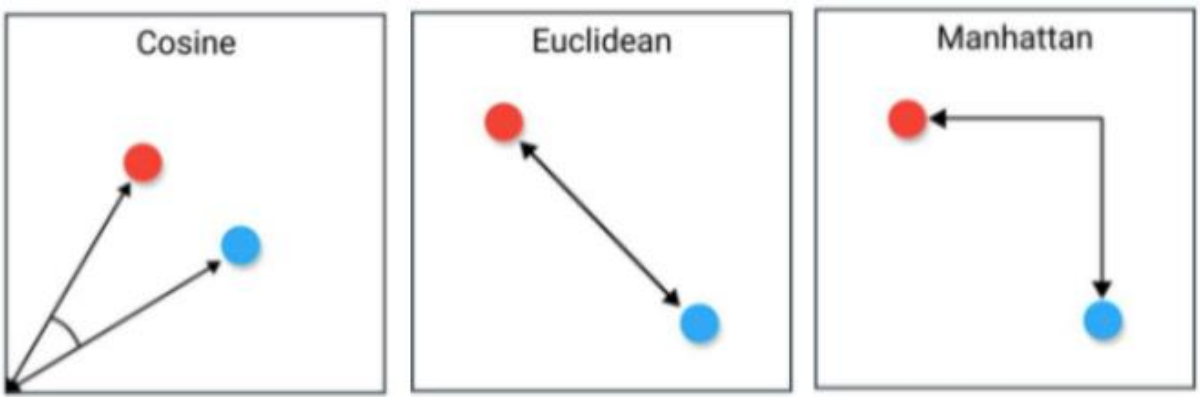


Ilustración 2 - Representaciones en dos dimensiones de las medidas Coseno, Euclidiana y Manhattan [5]

Similitud del coseno

Esta medida busca hallar el coseno del ángulo entre dos vectores mediante la siguiente fórmula:

$$cosine(x, y) = \frac{x.y^T}{||x||.||y||}$$

Este valor oscila entre 0 y 1. Siendo lo más cercano a 1 los vectores más parecidos por su contenido.

Una de las desventajas de este tipo de medida es que el coseno no toma en cuenta la diferencia en la escala de *rating* entre diferentes usuarios.

La similitud del coseno se suele utilizar cuando se presentan datos de alta dimensión y la magnitud de los vectores no es importante, y también en ejercicios de procesamiento de lenguaje natural, para ver la similaridad entre dos o N palabras.

Correlación de *Pearson*

Calcula una correlación donde su resultado se encuentra entre -1 y 1. Cuanto más cerca de 1 mayor correlación positiva se tiene entre los vectores, mientras que más cerca de -1, mayor correlación negativa se tienen entre los vectores. Por otro lado, 0 nos indica que no hay correlación entre los vectores.

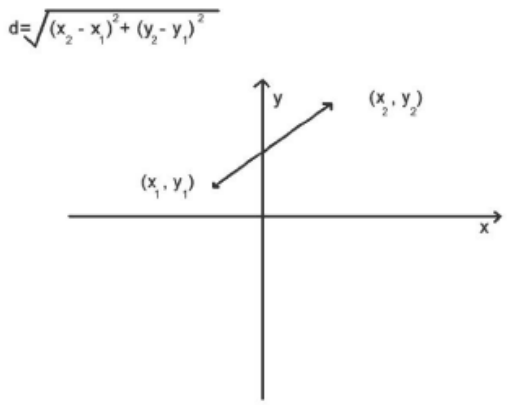
La correlación de Pearson tiene la siguiente fórmula matemática:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Distancia euclidiana

Esta distancia se calcula entre dos puntos de un plano de N dimensiones.

Por ejemplo, en un plano de dos dimensiones se vería de la siguiente forma:



Euclidean distance

La distancia euclidiana puede tomar valores entre 0 e infinito. Mientras más chica esta distancia más se parecen los vectores analizados.

Sus aplicaciones suelen ser en situaciones donde se tienen datos de baja dimensión y es importante medir la magnitud de los vectores. En estas presentan muy buena *performance* al utilizarse en algoritmos de *clusters* (*k-means,* por ejemplo).

Distancia de *Manhattan*

La distancia de *Manhattan* entre dos puntos, específicamente en dos dimensiones, es la suma de las diferencias absolutas de sus coordenadas cartesianas. Es preferida sobre la euclidiana cuando se trata de datos de alta dimensión:

$$d_1(p, q) = \|p_1 - q_1\| + \|p_2 - q_2\|$$

Con N dimensiones:

$$d_1(p, q) = \|p - q\| = \sum_{i=1}^{n} \|p_i - q_i\|$$

Uno de los problemas que tiene esta medida es que tiene mucho énfasis en la magnitud de la diferencia entre los vectores y no es muy precisa para diferenciar entre similaridades o disimilaridades.

La gran desventaja que poseen estas medidas es que deben ser re-calculada cada vez que se obtienen más datos.

Otra desventaja es que se debe tener información previa de los ítems, por lo que sufre del problema de entrada en frío (este tema se abordará en la sección *Cold Start*).

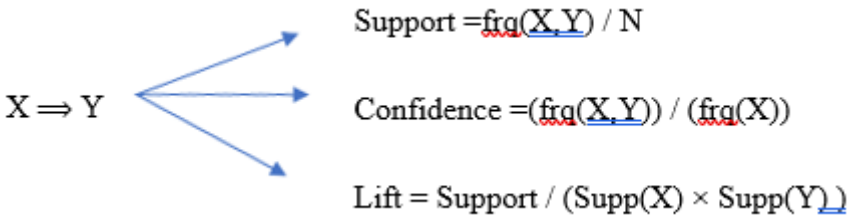
4.4. Algoritmos de recomendación

4.4.1. Market-Basket

El algoritmo *Market-Basket* se basa en buscar asociaciones entre ítems. Para construir el algoritmo se precisan un conjunto de transacciones, y cada transacción debe estar formada por un conjunto de productos que se hayan comprado juntos (*ítem set*). De esta forma se pueden construir reglas de asociación entre productos que vinculen la compra de uno o más de un producto, con la compra de otro u otros productos:

$$X: \{\text{Item A, Item B}\} \Rightarrow Y: \{\text{Item C, Item D}\}$$

Para medir la fuerza/importancia de cada regla de asociaciones el algoritmo está regido por tres variables fundamentales:



- *Support*: Mide el porcentaje de transacciones que contienen todos los ítems de un itemset. Cuanto más grande sea el support, más grande es la frecuencia en que se compran esos productos. Se buscan reglas de asociación con un alto support dado que podrán ser aplicadas a un gran número de transacciones.

- *Confidence*: La probabilidad de que una transacción que contiene los ítems en X también contenga los ítems en Y. A mayor confidence, mayor probabilidad que los ítems en Y sean comprados, por ende, mayor fuerza en la regla de asociación.

- *Lift*: La probabilidad de que todos los ítems se compran juntos (*Support*), sobre el producto de probabilidad de las veces que cada ítem aparece (independientemente de si hay asociación entre ellos o no). En otras palabras, mide la fuerza de la regla de asociación entre los ítems de X y los de Y. A mayor valor, mayor es la fuerza de asociación.

Para realizar el análisis con *Market-Basket* se decidió utilizar el algoritmo llamado 'Apriori', que trabaja en dos grandes pasos:

1) Busca itemsets que ocurren frecuentemente dentro de todo el conjunto de transacciones con un support mayor a un umbral predefinido.

2) Calcula el *confidence* de todas las posibles reglas de asociación dados por los itemset obtenidos en el paso 1, y se queda solo con aquellas que tienen un valor mayor a un umbral predefinido.

4.4.2. *Singular Value Descomposition* (SVD)

Es una técnica de factorización de matrices (reducción de dimensionalidad) muy popular en el ámbito de la ciencia de datos, particularmente en la aplicación de sistemas de recomendación. Disminuye la dimensión de la matriz de utilidad al extraer sus factores latentes con el objetivo de facilitar una representación clara y manejable de las relaciones entre usuarios e ítem.

En resumen, descompone una matriz A en tres matrices U, S, V de la siguiente manera:

$$SVD(A)_{nxm} = U_{nxn} \times S_{nxm} \times V^{t}_{mxm}$$

Matriz U

Esta matriz se compone en cada fila por los auto vectores calculados. Estos auto vectores se obtienen de una matriz cuadrada (nxn) resultante de multiplicar nuestra matriz inicial $A_{(nxm)}$ x $A^T_{(mxn)}$. Tendrá la siguiente forma:

$$U = \begin{bmatrix} \vec{v_1} \\ \vec{v_2} \\ \vdots \\ \vec{v_n} \end{bmatrix} = \begin{bmatrix} v_{1,1} & \cdots\cdots & v_{1,n} \\ v_{2,1} & \cdots\cdots & v_{2,n} \\ \vdots & \ddots & \vdots \\ v_{n,1} & \cdots\cdots & v_{n,n} \end{bmatrix}$$

Matriz V

Esta matriz se obtiene de la misma manera que obtenemos la matriz U, solo que con la diferencia de que obtenemos los auto vectores de una matriz cuadrada (mxm) resultante de multiplicar nuestra matriz inicial $A^T_{(mxn)}$ x $A_{(nxm)}$.

Matriz S

La matriz S es una matriz diagonal en donde se guardan los valores singulares, que se calculan realizando la raíz cuadrada de los valores calculados para U y V. Estos valores singulares serán puestos de manera decreciente y sus valores siempre serán mayores a cero. Tiene la siguiente forma:

$$S = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & \lambda_n \end{bmatrix}, \text{ donde } \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$$

Es importante destacar uno de los teoremas que aplica SVD y que la hace especialmente interesante no solo para los sistemas de recomendación sino también para muchas otras áreas.

Este teorema explica que reduciendo el número de valores singulares de la matriz S a los primeros k valores, se obtiene una aproximación de la matriz original A. Por lo que se permite reconstruir aproximadamente la matriz A a partir de las versiones reducidas de las matrices S, U, V cometiendo un cierto error.

Cuando se habla de sistemas de recomendación en base a usuarios se puede deducir a simple vista un problema de escalabilidad. Esta propiedad de los SVD permite superar esta barrera, lo que lo hace muy interesante su aplicación. Adicionalmente permite darnos rapidez en los cálculos matriciales al tomar matrices más chicas.

El teorema es el de *Eckart - Young* y permite asegurarnos una aproximación a la matriz original A se puede lograr poniendo 0 a los valores singulares más pequeños de la matriz S. Agregando los 0 podemos reducir el tamaño de la matriz U y V. En otras palabras, tomamos los primeros k- factores y al resto le asignamos 0.

En función de las matrices U, S y V se buscará hacer predicciones sobre ítems que el usuario aún no ha valorado.

Propuesta de mejora de Simon Funk

Es importante destacar que a priori SVD puede lograr un resultado muy bueno cuando los datos no son escasos, pero en situaciones reales se tienen matrices dispersas.

Es por esto que Simon Funk presenta una versión para solucionar este tipo de problemas de matrices dispersas, conocido como SVD Funk.

Esta variante, busca una manera de calcular los factores latentes utilizando solo los ratings conocidos. De esta manera las ecuaciones se reducen considerablemente.

La misma ignora la matriz S y actualiza los factores latentes sobre las matrices de usuarios e ítems mediante la siguiente función:

$$r_{u,i} = q_i^t \cdot p_u$$

Siendo:

- r = rating del usuario al ítem
- q^t = vector de factores latentes de un ítem.
- p = vector de factores latentes del usuario.

A continuación, se ejemplifica el resultado de aplicar SVD Funk. Esta representación no es matemática, sino que es un ejemplo para mostrar su funcionamiento.

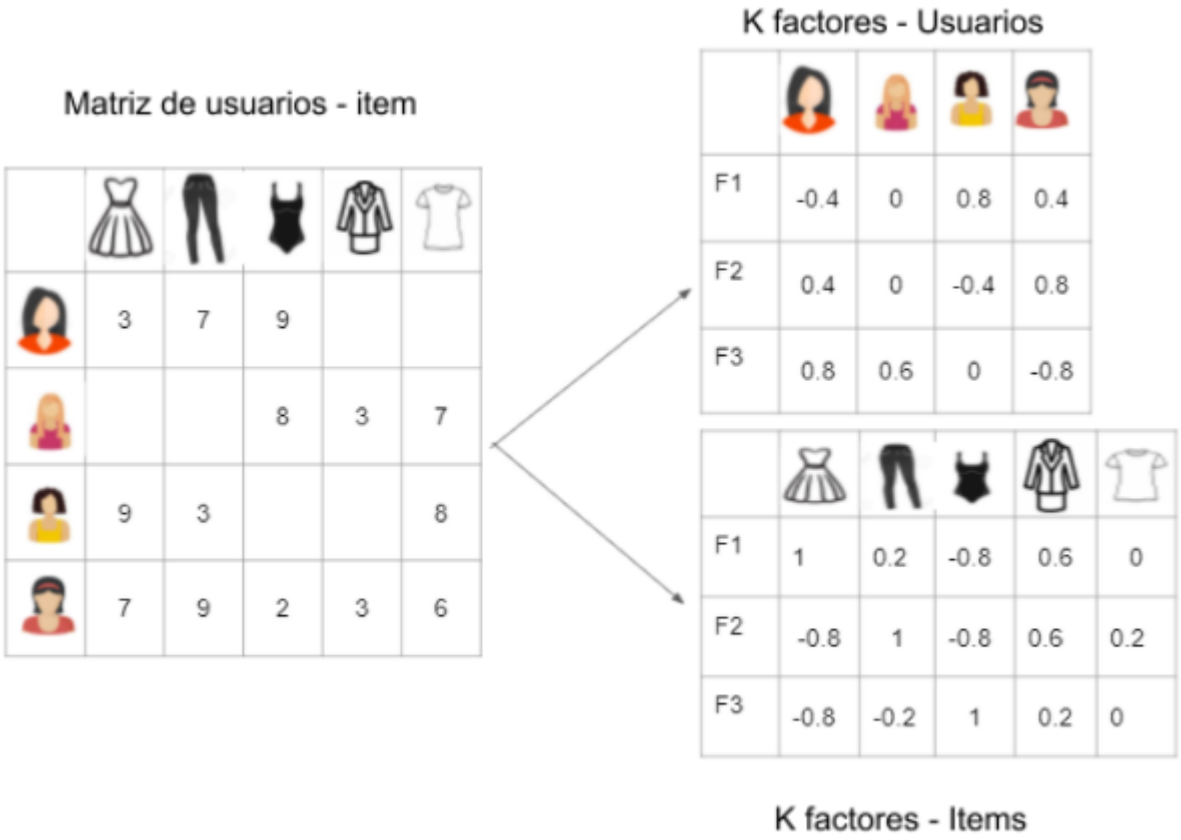


Ilustración 3- Ejemplo de *output* de matrices U y V desde una matriz A (fuente: elaboración propia)

Con el análisis se puede observar que tomar tres factores latentes para los ítems puede significar lo siguienta:

- Factor 1: Ítems asociados a bodas
- Factor 2: Ítems asociados trabajo
- Factor 3: Ítems asociados a playa

Desde la perspectiva del usuario, se pueden identificar los gustos de cada usuario dados por esta asociación. Por ejemplo, el usuario  tiene más preferencia por los ítems asociados a la playa mientras que el usuario  tiene más preferencia sobre la ropa de boda.

Una vez ejemplificado, se pueden clasificar a los usuarios e ítems según su grado de pertenencia a un determinado factor Esta relación implícita es la que se logra recabar con los factores latentes.

Proceso de predicción

A continuación, se presenta un ejemplo que busca ejemplificar la predicción de ratings faltantes.

1. Partiendo de la matriz A, *item*-usuario, se construyen dos matrices: la matriz U de usuario por factores latentes llenos de valores aleatorios, y la V^T de factores latentes por ítems con valores aleatorios.

Matriz A

	Item 1	Item 2	Item 3	Item 4
User 1	NaN	NaN	9	1
User 2	3	NaN	7	NaN
User 3	5	NaN	NaN	10
User 4	NaN	2	NaN	NaN

Tabla 1 Matriz A Usuarios x Ítems [6]

Matriz U

	Factor 1	Factor 2	Factor 3
User 1	0.8	1.2	-0.2
User 2	2	1.8	0.4
User 3	0.8	3	0.1
User 4	1	0.8	2.4

Tabla 2 Matriz U Usuarios x Factores [6]

Matriz V^T

	Item 1	Item 2	Item 3	Item 4
Factor 1	-1.8	1	-0.2	1
Factor 2	0.5	1.2	0.1	5
Factor 3	1.4	4	0.14	2

Tabla 3 Matriz V^T Factores x Ítems [6]

2. Si se busca en la matriz A, observando al primer usuario: el primer valor que tiene es 9, es el primer "valor real". Luego vamos a la matriz U y tomamos el valor de todos los factores latentes de ese usuario, posteriormente vamos a la matriz V^T y hacemos lo mismo para el ítem de la calificación real de usuario 1.

   Vector de la matriz U para el usuario 1: [0.8, 1.2, -0,2]

   Vector de la matriz V^T para el ítem 3: [-0.2, 0.1, 0.14]

3. Se realiza el producto de los vectores elemento a elemento y se obtiene la predicción.

   Valor real= 9

Valor predicho (resulta del producto de vectores) = -0.07

Error= (9+0.07)^2= 82.26

4. Luego se utiliza el descenso del gradiente para minimizar el error con la siguiente fórmula:

$$U(i) + \alpha\, \mathbf{2}(\boldsymbol{predicho} + \boldsymbol{real})\boldsymbol{x}\, \boldsymbol{V}(\boldsymbol{i})$$

	Factor 1	Factor 2	Factor 3
From U	0.8	1.2	-0.2
From V	-0.2	0.1	0.14

Tabla 4 Vectores de factores latentes para U y V [6]

Actualizamos 0.8 con α =0.1

$$\mathbf{0}.8 + 0.1x\ 2(0.07 + 9)\ x(-0.2) = 0.44$$

5. Se actualizan todos los valores:

	Factor 1	Factor 2	Factor 3
From U	0,44	1.38	0,053
From V	0,6	2.60	0.24

Tabla 5 Actualización de valores luego de aplicar la fórmula de error [6]

6. Se actualizan los valores de la matriz U y de la matriz V^T y se termina la primera iteración. Luego se realiza con todos los valores de las matrices y finalmente con la fórmula de la descomposición de la matriz A, llegamos a predecir los valores faltantes.

$$A \approx U V^T$$

Por último y no menos importante es que se debe tener en cuenta que SVD como cualquier otro algoritmo que recibe entrenamiento puede sufrir de *overfitting*. Por este motivo y al tratarse de un problema de regresión puede utilizarse regularizadores como Ridge y Lasso.

SVD presenta las siguientes desventajas:

- No maneja correctamente los productos que ingresan al inventario. Por ejemplo, para un ítem nuevo va a tener un puntaje de cero por lo que la predicción será un valor muy cercano a cero y claramente este no es el resultado esperado.
- Cuando un usuario nuevo aparece en nuestra base de datos se debe correr nuevamente el algoritmo de SVD para todos los datos lo que lo hace muy costoso.

4.4.3. Alternating Least Square (ALS)

ALS es un algoritmo basado en la ***factorización matricial***, que busca separar una matriz en un producto de ellas para obtener una aproximación de la matriz original. En cuanto al CF, se factoriza la matriz de interacción ***usario_id - item_id*** en el producto de dos matrices de menor dimensión:

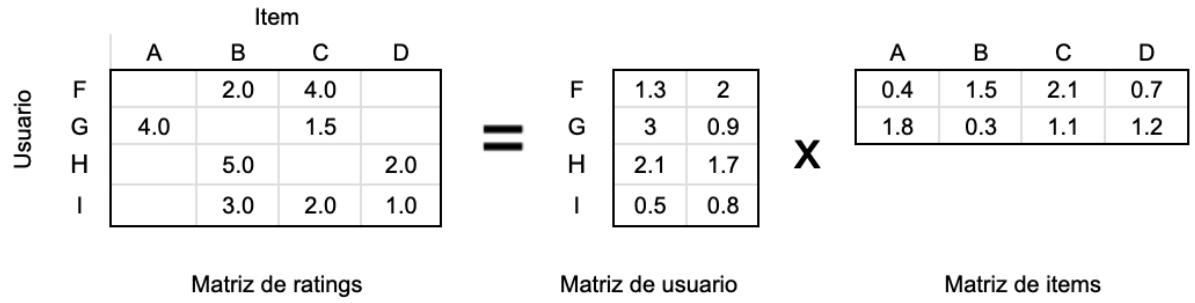


Ilustración 4 - Ejemplo de operaciones matriciales con ALS (fuente: elaboración propia)

El enfoque general es iterativo, donde en cada iteración una de las matrices de factores se mantiene constante, mientras que la otra se resuelve utilizando mínimos cuadrados.

La creación de vectores de factores latentes sucede de la siguiente forma: se fija un número k (por ejemplo, k =8), y explicamos cada usuario con un vector k dimensional Hu, y cada artículo i con un vector k dimensional Wi. Por ende, el *rating* del usuario u para el artículo i se calcula de la siguiente manera:

$$\tilde{r}_{ui} = \sum_{f=0}^{nfactors} H_{u,f} W_{f,i}$$

Los factores latentes son las características de los ítems y usuarios en un espacio de menor dimensionalidad, y la idea detrás de la factorización de matrices es usar los factores latentes para representar las preferencias de los usuarios en este espacio reducido.

Este tipo de factorización tiene grandes beneficios a la hora de hacer predicciones, dado que los ítems menos populares pueden llegar a tener representaciones tan significativas como los más populares y los modelos también aprenden a factorizar la matriz en representación de ítems y usuarios.

Las principales características de ALS son que:

- Es sencillo, se implementa en Apache spark ML para solucionar problemas de escalabilidad en algoritmos de filtrado colaborativo a gran escala y ayuda a la escasez de datos y matrices dispersas.

- Agrega término de regularización L2 (Ridge) a la función objetivo para evitar *overfitting*. La cantidad de factores latentes determina la cantidad de información abstracta que queremos mantener en el espacio de menor dimensión. Una factorización de matrices con un solo factor latente es un recomendador popular (recomienda los artículos con más interacciones sin ninguna personalización). Aumentar el número de factores latentes aumenta el grado de personalización con la que se realiza la recomendación, con el riesgo que con demasiados factores se llega a zona de *overfitting* o sobreajuste. Una estrategia para evitarlo es aplicar regularización (L2 en este caso).

  Este método de regularización agrega un coeficiente multiplicado por la norma de la magnitud como penalización a la función de error (diferencia entre *rating* real y *rating* predicho), por lo que penaliza los valores grandes. Esto es una diferencia en cuanto a SVD que utiliza la regularización L1.

$$\arg\min_{H,W} \|R - \tilde{R}\|_{\mathrm{F}} + \alpha\|H\| + \beta\|W\|$$

  Donde H y W son las matrices de usuarios e ítems reducidas, respectivamente, y 'α' y 'β' son los coeficientes de los factores latentes.

- Una de las principales diferencias con SVD es que el proceso de entrenamiento consiste en minimizar dos funciones de pérdida alternativamente. Primero fija la matriz de elementos y ejecuta el descenso de gradiente con la matriz de usuario, y luego viceversa.

- Presenta sus propios parámetros: maxIter (número máximo de iteraciones a ejecutar), rank (número de factores latentes en el modelo), regParam (parámetro de regularización).

Con respecto a las limitaciones de ALS, se puede destacar que es un algoritmo que requiere un tiempo de ejecución más elevado que el promedio de modelos, suele consumir más CPU en general, dado que sus operaciones con matrices son más complejas.

4.4.4. KNN - método basado en memoria

KNN es un método de *clustering*. El *clustering* o segmentación, se utiliza para encontrar grupos de usuarios que son similares. El mayor propósito de la clusterización se puede traducir en una optimización porque se quiere encontrar grupos de usuarios para reducir el número de veces que se calcula la similitud usuario-usuario.

Conceptualmente se visualiza de la siguiente forma: si el usuario X llega a su sitio, potencialmente tendría que iterar sobre todos los usuarios, pero al dividir a los usuarios en *clusters*, puedes buscar de qué clúster forma parte el usuario y calcular las similitudes entre esos usuarios como parte de un grupo más pequeño.

Un breve esquema de cómo es el *pipeline* del CF basado en vecinos cercanos del usuario:

1. De una matriz de *ratings* de usuarios e ítems se calcula la similaridad entre un usuario activo y el resto de los usuarios.
2. Se ordena a los usuarios por la similaridad respecto al usuario activo.
3. En base a la cantidad de vecinos cercanos establecida se seleccionan los vecinos cercanos al usuario activo.
4. Con los vecinos cercanos se calcula el *rating*.

Para el cálculo de similitud, se podrán seguir dos caminos para seleccionar los k más cercanos según los intereses a estudio:

- KNN basado en usuarios, donde se seleccionan los usuarios más similares al usuario objetivo.

- KNN basado en ítems, donde se seleccionan los ítems más similares al ítem objetivo.

Las predicciones se harán con los vecinos más similares recopilados del conjunto de datos de entrenamiento, las formas más comunes de calcular esto es: regresión y clasificación (explicados en la siguiente imagen). En cuanto a la clasificación, predice una valoración de 5 al ver qué valoraciones se dan más, mientras que la regresión predice 3,8 al tomar el promedio de las valoraciones basadas en los elementos.

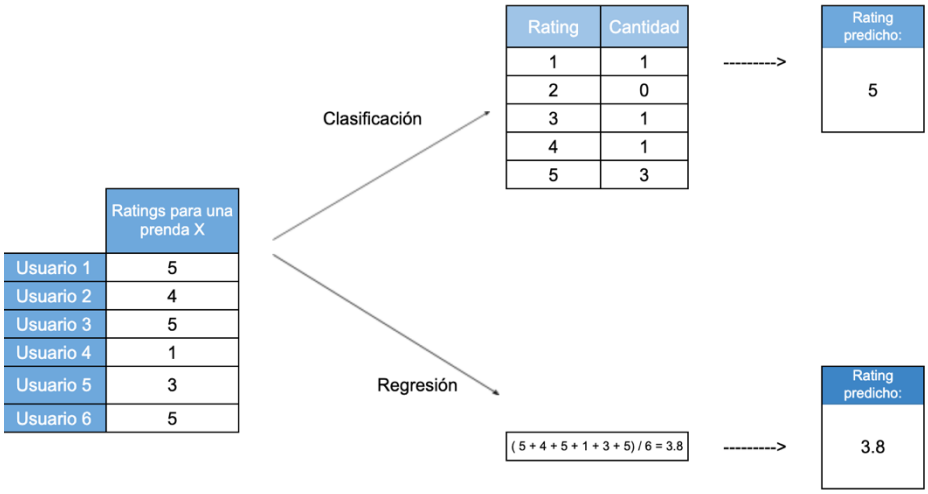


Ilustración 5 - Ejemplo de predicción de *rating* utilizando KNN para regresión y clasificación (fuente: elaboración propia)

4.4.5. Sistemas Híbridos de Recomendación

Todos los motores de recomendación vistos anteriormente sufren de alguna desventaja puntual. Por ejemplo, los CF tienen dificultades para el tratamiento del *cold-start* dado que no tienen datos históricos, y por otro lado, los CB pueden entrar en una burbuja cerrada de recomendación en función de los intereses del mismo usuario y no expandir la recomendación a otros ítems no vistos.

Los sistemas híbridos de recomendación abordan estos problemas y conservan las ventajas de cada modelo. Estos motores combinan varios modelos simples de recomendación para brindar las predicciones. No existe un camino estandarizado; algunos híbridos predicen usando algoritmos CB y CF de forma independiente, otros introducen motores CB dentro de CF y viceversa.

Estos sistemas híbridos de recomendación se clasifican en los siguientes tres grandes tipos:

Monolíticos

Son motores que contienen diferentes partes de cada tipo de recomendación, por ejemplo, matriz de similitud de un primer recomendador CB para detectar similitud entre ítems, y luego predicción de *rating* para el usuario de esos ítems seleccionados mediante un CF.

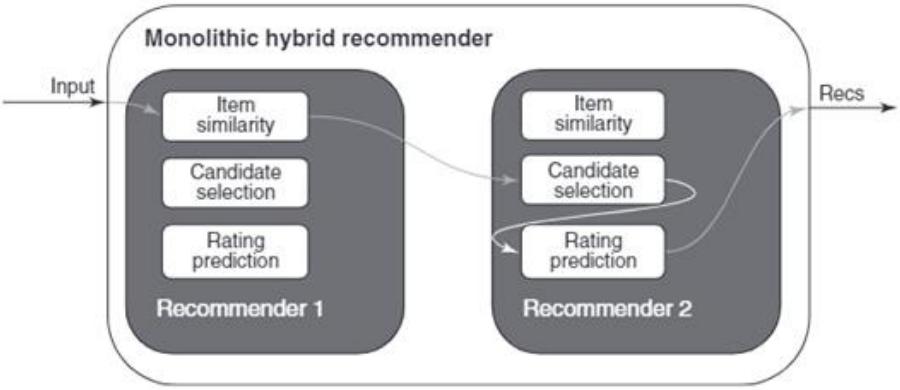


Ilustración 6 - Sistemas híbridos monolíticos [7]

Mixed

Un sistema *mixed-hybrid* retorna las recomendaciones dadas por varios algoritmos, una por una. Trata cada motor de recomendación como "caja negra" y brinda todas las recomendaciones individuales.

Es muy usado cuando se precisa un orden jerárquico de recomendaciones. Se le puede dar un score a cada recomendación y ordenarlas por dicho valor, para poder compararlos y brindar la mejor recomendación.

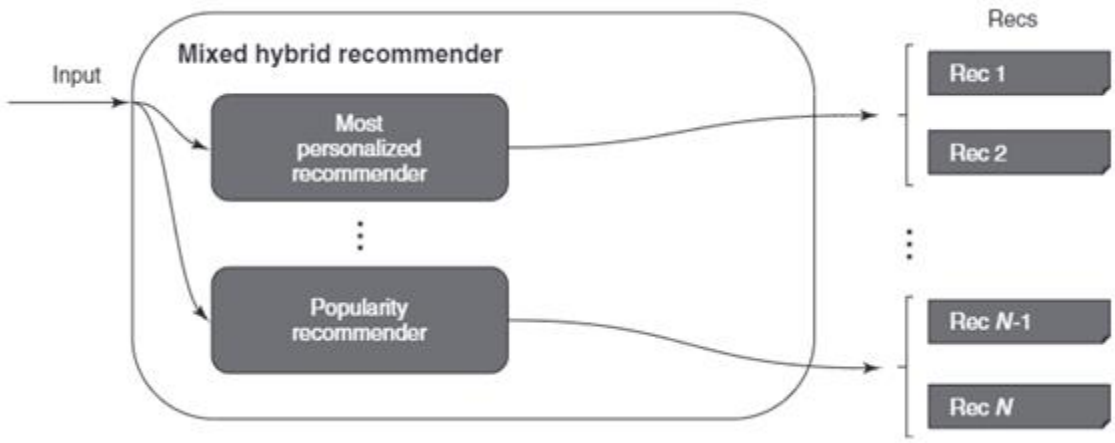


Ilustración 7 - Sistemas híbridos mixtos [7]

Ensemble

Se combinan predicciones de múltiples motores de recomendación antes de hacer la recomendación final.

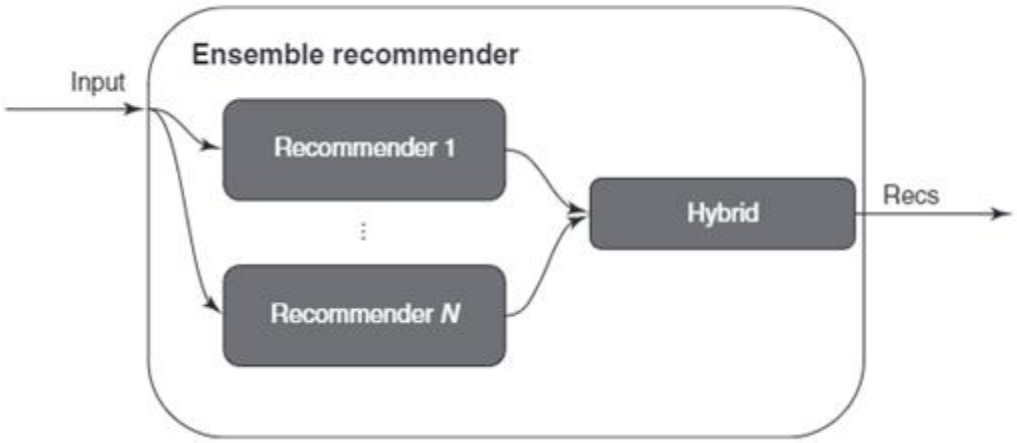


Ilustración 8 - Sistemas híbridos de ensamble [7]

Hay varias formas de combinar estos resultados provenientes de los diferentes sistemas de recomendación:

a) *Switched* ensemble recommender: Usa la mejor recomendación en función de una determinada condición. Por ejemplo, si existen dos mejores motores de recomendación para diferentes países, se usará una u otra recomendación dependiendo del país en el que esté. Esto se puede aplicar a tiempos del día, o en un diario, la sección de noticias nacionales se puede llenar con las últimas noticias, y la sección de noticias culturales con recomendaciones CB para libros específicos. Un caso bien práctico de uso en estos motores es para atacar los problemas de cold-start de los CF: Si el usuario calificó menos de 20 ítems se utiliza un algoritmo determinado, y si califico más de 20, se usa un CF.

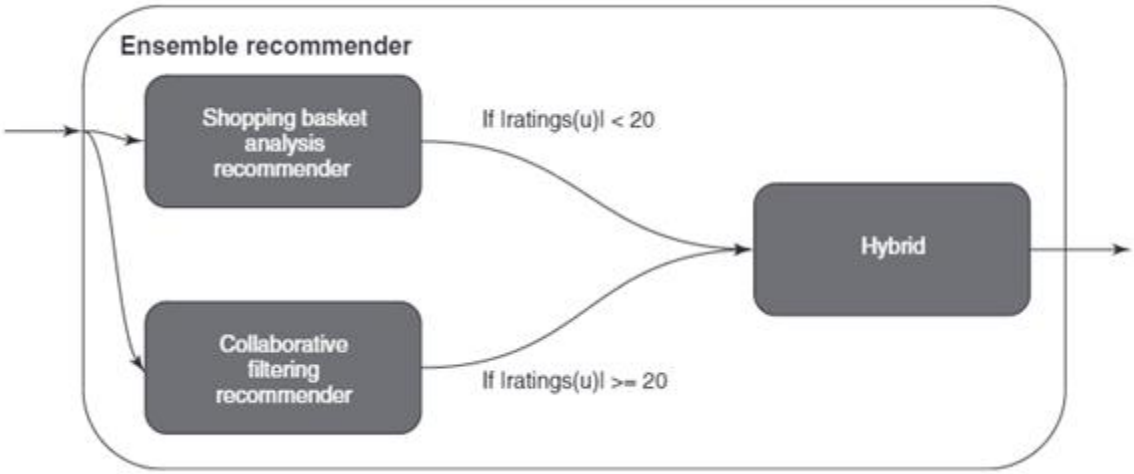


Ilustración 9 - Ejemplo de un sistema híbrido de ensamble con switch [7]

b) *Weighted* ensemble *recommender*: Se pueden combinar dos modelos de recomendación y obtener una final asignándole un peso relativo a cada una (no tienen por qué ser 50/50). Por ejemplo, un algoritmo CB busca los ítems más relacionados, pero sin importar la calidad de éstos. Por otro lado, un CF prioriza las recomendaciones en función de ítems de buena calidad. Se pueden entrenar ambos algoritmos y preguntarles a ambos para producir los mejores candidatos.

$$\hat{r} = 0.6 \times \hat{r}_{collaborative} + 0.4 \times \hat{r}_{content}$$

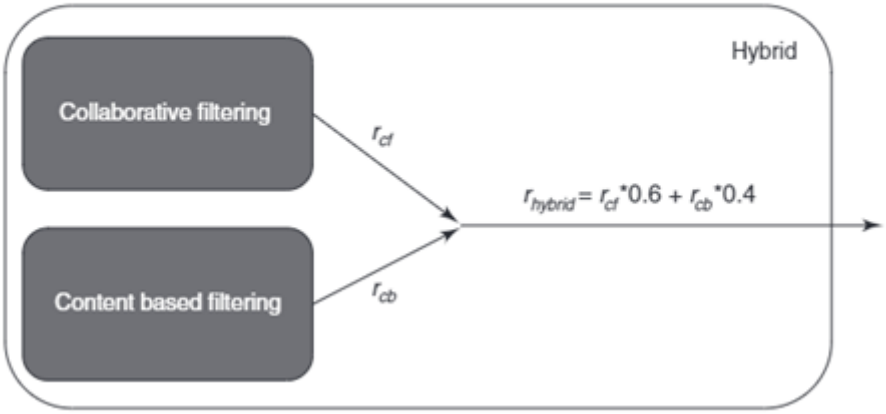


Ilustración 10 - Ejemplo de un sistema híbrido de ensamble con pesos [7]

c) *Linear regression*: Se crea una función que minimiza el error entre el *output* dado por la combinación de ambos motores de recomendación y el valor real.

$$RSS = \sum_{all\ ratings\ in\ training\ data} (f(u,i) - r)^2$$

$$f(u,i) = w_1 \times RecSys1(u,i) + w_2 \times RecSys2(u,i)$$

Hay varios métodos estadísticos para detectar cuales son los valores óptimos de los pesos. Incluso, éstos pueden variar en función de los ítems y de los usuarios.

4.5. Cold start

Como bien mencionamos anteriormente los sistemas de recomendación presentan muy buenos resultados al acortar la lista de ítems siempre y cuando se tenga información respecto a preferencias de los usuarios o información de los ítems. Por lo tanto, uno de los mayores

desafíos que tiene este tipo de sistemas es cuando no tiene suficiente información previa sobre los productos o sobre los usuarios. A esta situación se le denomina entrada en frio (*cold start*).

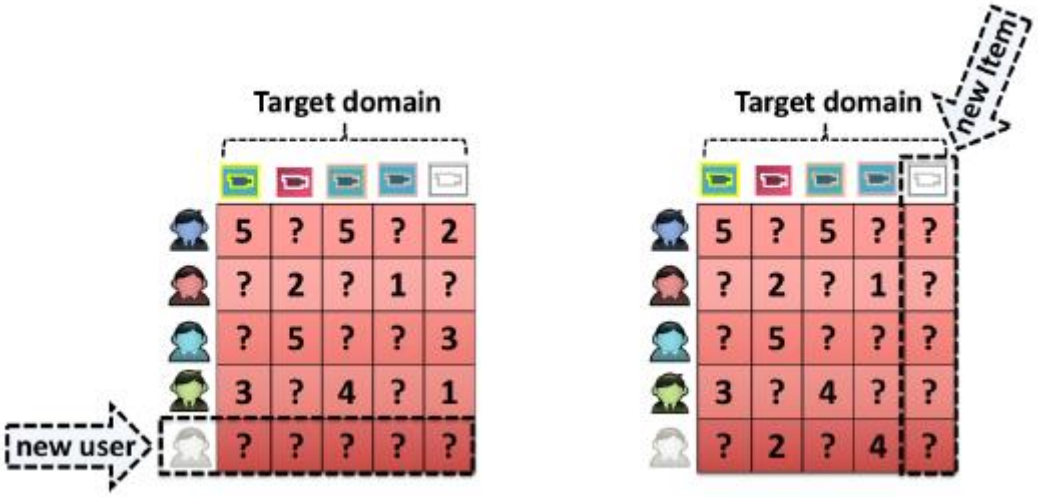


Ilustración 11 - Problemas de cold start [8]

Otro problema asociado a *cold start* es la dispersión de los datos, ya que en muchos casos los ítems poseen muy pocos *ratings* y la performance de los algoritmos puede verse seriamente afectada. Un ejemplo podría ser: El número de *rating* conocidos son mucho menores que los desconocidos. Intentar predecir estos últimos es complejo.

A su vez, los motores de recomendación en general sufren de problemas de escalabilidad de los mismos, ya que requieren de relaciones entre múltiples ítems y muchos usuarios. Lo que hace que los cálculos de estos algoritmos sean un verdadero desafío debido al crecimiento exponencial que pueden tener ambos.

En la bibliografía se plantean diferentes técnicas para solucionarlo. En este documento se busca hacer un acercamiento a las principales:

4.5.1. Active learning

Este es un proceso por el cual se le solicita a un nuevo usuario que pondere una selección de ítems.

Active learning no solo soluciona el *cold start*, sino que tiene dos principales ventajas: La primera es que iterativamente consigue más información sobre los ítems y la segunda radica

en que este método analiza la data disponible y decide cual data necesita ser recolectada. Permite de esta manera quitarles ruido a los datos y mejorar la calidad de los datos de entrada.

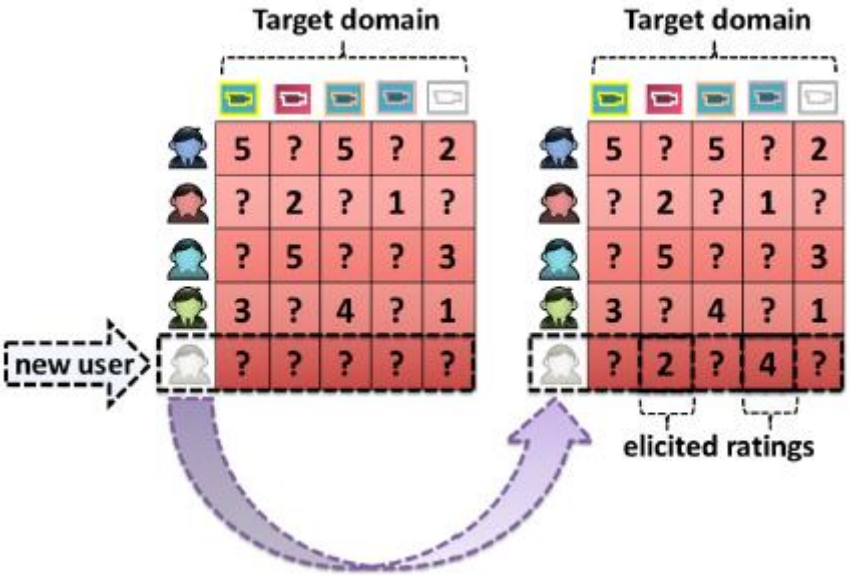


Ilustración 12 - Ejemplo de active learning sobre un usuario nuevo [8]

En la ilustración 14, se visualiza una solicitud de *rating* sobre una selección de ítems. Esto permite obtener un mínimo de data que describe las preferencias del usuario.

Si bien hablamos de que la solución se basa en que un usuario pondere una selección de ítems, la siguiente pregunta sería válida. ¿Cómo seleccionar los ítems a ponderar?

Existen numerosas técnicas que permiten obtener esta selección de ítems. A continuación, daremos una pequeña introducción de algunas de ellas:

- **Reducción de incertidumbre:** Se seleccionan los ítems con los *ratings* más diversos ya que el motor de recomendación tiene menos certeza sobre ellos. Es muy fácil recomendar ítems con *rating* alto y muy fácil excluir ítems con *rating* bajo. Por lo que en ítems en donde los *ratings* son muy diversos la obtención de más datos puede ser muy informativa y puede disminuir la incertidumbre del sistema al calcular predicciones.

- **Reducción del error**: Esta técnica se basa en conseguir *ratings* de ítems que ayuden directamente a reducir el error de predicción. Esto puede ocurrir porque ítems con diversos *ratings* pueden estar poco correlacionados con los *ratings* de otros ítems. A diferencia de la estrategia anterior, ésta puede ignorar los ítems con *ratings* diversos y enfocarse más en *ratings* que puede mejorar la precisión de la predicción.

- **Adaptación al usuario:** Esta estrategia trata de personalizar el proceso en función de las características particulares de los usuarios, donde según el usuario se le solicita diferentes ítems para ponderar. Siguiendo el principio básico de que distintos tipos de usuarios tienen diferentes preferencias, recolectar los *ratings* de sus preferencias mejora la calidad y cantidad de los datos.

- **Probabilidad de adquisición:** Esta estrategia busca maximizar la probabilidad de que un usuario pondere un *rating*. Indicando que pondere aquellos ítems que le puedan agradar más al usuario. Por ejemplo: Si a un usuario se le pido que pondere un tipo de vestido que nunca usaría, puede aumentarse el ruido de los datos. Por lo que resulta crucial para esta estrategia tomar en cuenta la probabilidad de que el usuario esté familiarizado con este tipo de prendas.

- **Árboles de decisión:** Esta estrategia utiliza algoritmos de árboles de decisión para identificar qué ítems seleccionar. Cada nodo contiene un ítem a ser propuesto a un nuevo usuario. Por lo que cada nodo representa un grupo de usuarios afines que valoró de manera similar ese ítem. El nodo separa a los usuarios en tres grupos: Quienes dieron un alto *rating* al ítem, bajo *rating* al ítem, y los que no le dieron *rating*. Con los nodos construye un árbol de decisión basado en la reducción del error de predicción. Una vez construido el árbol, el usuario se pasea por el mismo en función de las respuestas que vaya dando.

- **Utilizando modelos predictivos:** Se basa en la predicción de ratings, donde los ítems con mayor score predicho se le otorgan al usuario para que los pondere realmente. La ventaja de esta técnica es que se seleccionan ítems probables para el usuario.

- **Híbrido:** Esta estrategia busca combinar algunas de las estrategias anteriores. Permite enfocarse en mejorar ambas métricas a la vez.

4.5.2. Recomendadores semánticos

Una de las soluciones tradicionales al *cold start* están basadas en la utilización de algoritmos CB. Estos algoritmos buscan construir las preferencias de los usuarios con los atributos semánticos del contenido de los ítems.

Este mecanismo es muy utilizado para resolver el problema de nuevos ítems. Ya que interpreta como similares aquellos ítems con similares contenidos (atributos semánticos), que hayan sido recientemente agregados o que ya tengan meses en el stock.

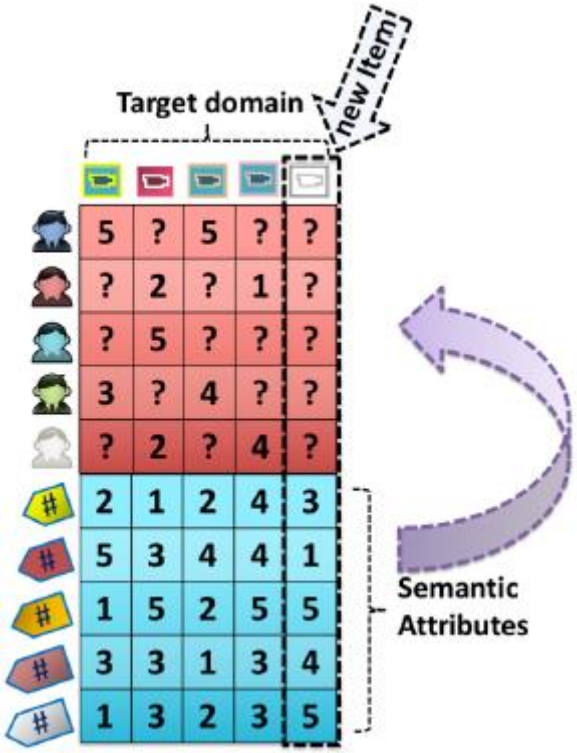


Ilustración 13 - Ejemplo de recomendadores semánticos para solucionar cold start de ítems [8]

Los atributos semánticos varían según el caso, pueden ser desde la categoría del ítem a la descripción del ítem.

4.5.3. Recomendaciones basadas en atributos visuales

Otra posible solución al cold start para nuevos ítems es el enriquecimiento de los perfiles de los ítems con data adicional.

Un ejemplo del mismo es la utilización de redes neuronales para identificar atributos visuales de los ítems y así agregar más datos a los ítems.

En el siguiente ejemplo se muestra como a diversos ítems multimedia se les agrega nuevos atributos visuales para solucionar el problema del cold start.

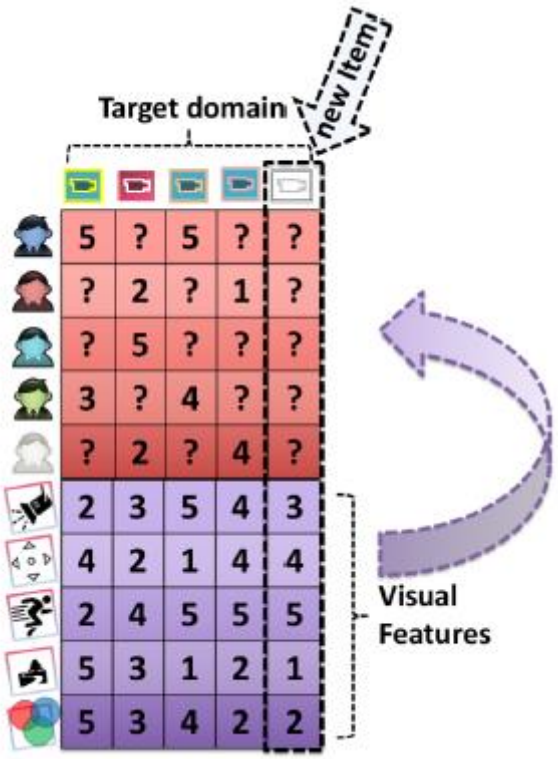


Ilustración 14 - Ejemplo de recomendadores basados en atributos visuales para solucionar cold start de ítems [8]

En nuestro caso, estos atributos visuales pueden ser variaciones de colores, tipo de prenda, atributos propios del estilo de prenda, etc.

Una de las limitaciones a este mecanismo es que asume que existen atributos semánticos y que los visuales son un agregado.

4.5.4. Recomendadores basados en personalidad de los usuarios

Una solución natural a este problema es utilizar información adicional del usuario para brindar un perfil inicial. Si bien existen muchas estrategias para hacer esto, la más común es identificar la personalidad del usuario.

Para recolectarla existe mucha literatura respecto a modelos psicológicos que buscan comprender la personalidad de una persona. Uno de los modelos más conocidos es el modelo de cinco factores. Este modelo describe a la persona en cinco grandes factores: Apertura, Conciencia, Extraversión, Agradabilidad y Neuroticismo (en inglés las siglas son OCEAN).

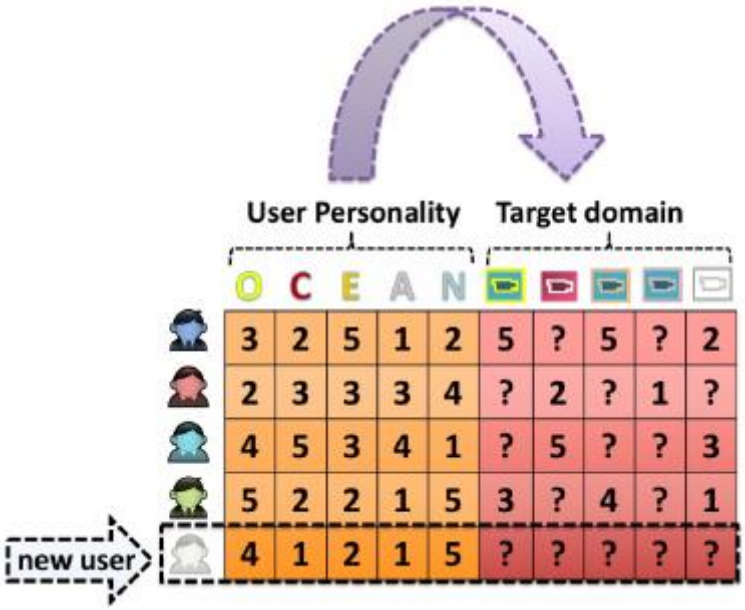


Ilustración 15 - Ejemplo de recomendadores basados en personalidad de usuarios para solucionar cold start de usuarios [8]

Usuarios con las mismas personalidades tienden a compartir preferencias.

Una de las limitantes a esta solución es que requieren de cuestionarios de personalidad que son muy demandantes para el usuario. Por esto es que en la actualidad existe mucha investigación con *machine learning* en este aspecto para obtener indicadores de personalidad a través de redes sociales de los usuarios.

4.5.5. Recomendaciones basadas en cruzamiento de dominios

En algunos casos la compañía posee datos de sus usuarios referentes a múltiples dominios o puede conseguirlos mediante acuerdos. Este cruzamiento puede otorgar recomendaciones mucho más precisas.

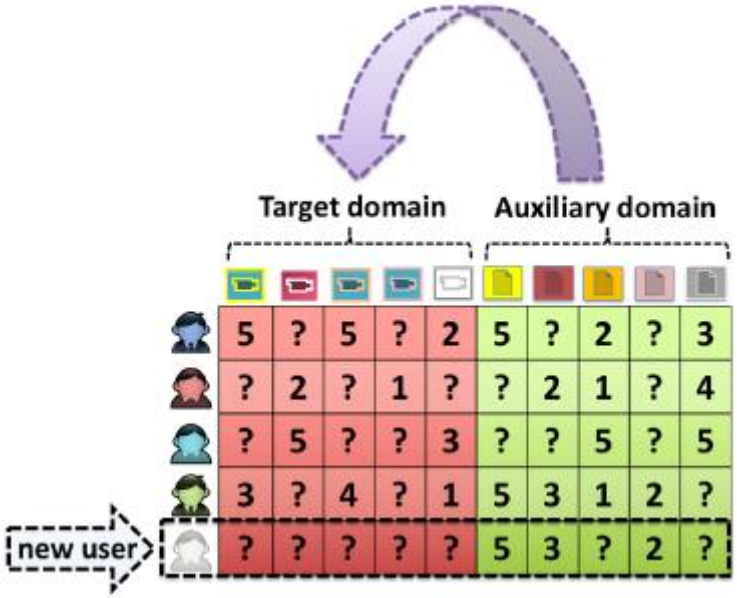


Ilustración 16 - Ejemplo de recomendadores basados en cruzamiento de dominios para solucionar *cold start de* usuarios [8]

4.5.6. Resumen *cold start*

A continuación, se resumen las soluciones al problema de *cold start* comentadas anteriormente.

Es importante destacar que éstas son algunas de las soluciones más utilizadas para resolver el problema de *cold start* que sufren los motores de recomendación.

Solución	Soluciona *cold start* para nuevos usuarios	Soluciona *cold start* para nuevos ítems
Active learning	SI	SI
Atributos semánticos		SI

Atributos visuales		SI
Atributos de personalidad	SI	
Cruzamiento de dominios	SI	

Tabla 6 Resumen soluciones *cold start* [8]

5. Experimentación: La Quia

La Quia es una empresa uruguaya dedicada a la venta de indumentaria femenina con más de 20 años de trayectoria. Tiene una fuerte presencia en las redes sociales y su venta online se encontró exponenciada en el último tiempo debido a la pandemia del COVID.

Para la realización de este caso de uso se utilizó información de su tienda electrónica. La obtención de los datos fue mediante una conexión a la API de Google Analytics.

5.1. Motivación y objetivos

La principal motivación para experimentar con La Quia se basa en simular un caso de uso real donde se potencie al negocio desde la inteligencia artificial. Hubo varios desafíos en el camino, desde interactuar con el cliente para comentarles de qué trata nuestro proyecto, obtener los datos desde su página web y observar de la forma que estaban estructurados, hasta crear un sistema de recomendación.

El objetivo radica en estudiar y analizar los datos obtenidos de Google Analytics para crear un motor de recomendación, que basado en el historial de actividad de los usuarios, recomiende ítems que sean de su interés y ajustados a su perfil.

5.2. Descripción del *dataset*

Este *dataset* contiene las interacciones que realiza un usuario con los productos de la tienda.

Ejemplo de flujo de interacciones capturadas:

1) Ver el producto
2) Agregar al carrito
3) Ir al checkout

El periodo analizado es desde el 31/12/2020 al 31/05/2021.

5.2.1. Composición del *dataset*

- clientId: ID del cliente asignado por Google. Este ID es referente a la sesión del usuario y no al usuario en sí. Es decir, si este usuario cambia de dispositivo el ID es diferente.
- dateHourMinute: Es la concatenación de la fecha con la hora y el minuto cuando el usuario realizo el evento en cuestión con una prenda.
- productSku: SKU correspondiente al producto. Este es el ID del producto.
- productName: Nombre del producto.
- shoppingStage: Estados efectuados hasta el momento sobre el producto.

Representación del contenido de este *dataset*:

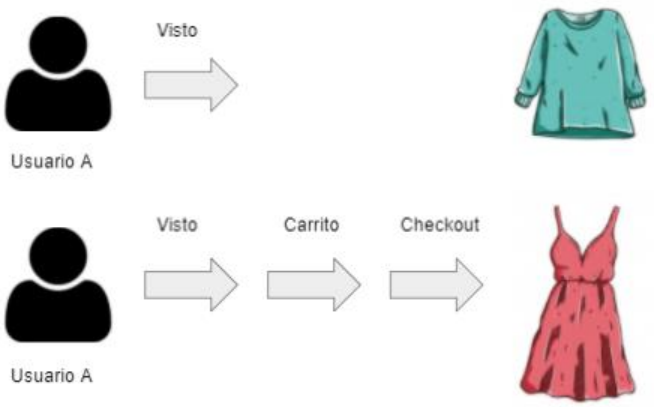


Ilustración 17 - Representación del estado del usuario en el *dataset* (fuente: elaboración propia)

En la ilustración 19 se visualiza los diferentes grados de preferencia del usuario A hacia los productos.

5.2.2. Supuestos tomados respecto a este *dataset*

Existen numerosos estados posibles cuando un usuario interactúa con un producto, los cuales fueron clasificados en cinco grandes grupos como se muestra a continuación:

Categoría	Visto	Ver detalles	Carrito	Checkout	Compra
ADD_TO_CART			X		
ALL_VISITS	X				
CHECKOUT_1				X	
CHECKOUT_1_ABANDONMENT				X	
CHECKOUT_3_WITHOUT_CHECKOUT_2				X	
CHECKOUT_ABANDONMENT				X	
NO_CART_ADDITION		X			
NO_PRODUCT_VIEW	X				
PRODUCT_VIEW	X				
TRANSACTION					X
TRANSACTION_WITHOUT_CHECKOUT					X

Tabla 7 Clasificación de estados (fuente: elaboración propia)

5.3. Transformaciones necesarias

Previamente a la implementación del algoritmo, se realizaron ciertas transformaciones a los datos y determinados análisis para un mejor entendimiento del set de datos.

5.3.1. *Rating* Implícito

Debido a que es de interés la aplicación de CF, es que debemos obtener un *rating* de preferencia del usuario hacia el ítem. Por lo que se creó a la variable "Ranking", en función de la variable categórica 'shoppingStage', con valores en una escala del 1 al 5 de forma manual, para formar el **rating *implícito***. Este ayuda a medir la valoración de los usuarios a los ítems, dado que 'shoppingStage' es el estado efectuado en un *pipeline* de compra. Suponiendo una relación positiva entre grado de avance en la serie de eventos realizados sobre una prenda e interés sobre la misma, cuanto más alto es la variable Ranking, significa que el usuario está en una etapa más próxima a comprar el producto y por ende, tiene más interés en el producto.

A continuación, se presenta la asignación manual de ranking en función de la categorización de shopping stage:

	shoppingStage	Ranking
0	NO_PRODUCT_VIEW	0
1	NO_SHOPPING_ACTIVITY	0
2	ALL_VISITS	1
3	PRODUCT_VIEW	1
4	NO_CART_ADDITION	2
5	ADD_TO_CART	3
6	ADD_TO_CART_WITH_VIEW	3
7	CART_ABANDONMENT	3
8	CHECKOUT	4
9	CHECKOUT_1	4
10	CHECKOUT_2	4
11	CHECKOUT_3	4
12	CHECKOUT_3_ABANDONMENT	4
13	CHECKOUT_ABANDONMENT	4
14	CHECKOUT_WITH_CART_ADDITION	4
15	CHECKOUT_1_ABANDONMENT	4
16	CHECKOUT_WITHOUT_CART_ADDITION	4
17	CHECKOUT_2_ABANDONMENT	4
18	CHECKOUT_3_WITHOUT_CHECKOUT_2	4
19	CHECKOUT_2_WITHOUT_CHECKOUT_1	4
20	TRANSACTION	5
21	TRANSACTION_WITHOUT_CHECKOUT	5

Tabla 8 Transformación de shopping stage en ranking numérico (fuente: elaboración propia)

5.3.2. NewSKU

Otra transformación necesaria fue crear una variable "newSKU", donde unificamos las prendas iguales que tienen diferente ID. Se encontró que en el *dataset* existían prendas donde al variar el color o el talle se cambiaba su SKU. Esto sucede ya que el SKU del producto es asignado aleatoriamente por el desarrollador y es independiente al SKU del producto en la tienda física.

Esta tarea permitió identificar dentro de los datos transacciones del mismo producto, pero insumió mucho más tiempo del proyectado, ya que requirió que unifiquemos en forma manual los distintos SKU en un único código (NweSKU).

5.4. Implementación y resultados

El primer algoritmo a implementar fue *market-basket*. Este algoritmo busca las asociaciones entre ítems en una misma compra.

Previo a realizar la implementación del algoritmo, decidimos quedarnos con los ítems con "Ranking" 4 y 5 para buscar asociaciones de "preferencia" entre los ítem y los usuario.

Los resultados que obtuvimos fueron los siguientes:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	l
10	(35743)	(37575)	0.034810	0.037975	0.028481	0.818182	21.545455	0.027159	5.291139	
11	(37575)	(35743)	0.037975	0.034810	0.028481	0.750000	21.545455	0.027159	3.860759	
8	(35744)	(35743)	0.028481	0.034810	0.012658	0.444444	12.767677	0.011667	1.737342	
9	(35743)	(35744)	0.034810	0.028481	0.012658	0.363636	12.767677	0.011667	1.526673	
12	(35744)	(37575)	0.028481	0.037975	0.012658	0.444444	11.703704	0.011577	1.731646	
13	(37575)	(35744)	0.037975	0.028481	0.012658	0.333333	11.703704	0.011577	1.457278	
2	(35993)	(35237)	0.123418	0.037975	0.012658	0.102564	2.700855	0.007971	1.071971	
3	(35237)	(35993)	0.037975	0.123418	0.012658	0.333333	2.700855	0.007971	1.314873	
0	(35993)	(34957)	0.123418	0.041139	0.012658	0.102564	2.493097	0.007581	1.068445	
6	(35993)	(35617)	0.123418	0.041139	0.012658	0.102564	2.493097	0.007581	1.068445	
1	(34957)	(35993)	0.041139	0.123418	0.012658	0.307692	2.493097	0.007581	1.266174	
7	(35617)	(35993)	0.041139	0.123418	0.012658	0.307692	2.493097	0.007581	1.266174	
4	(35993)	(35276)	0.123418	0.079114	0.018987	0.153846	1.944615	0.009223	1.088320	
5	(35276)	(35993)	0.079114	0.123418	0.018987	0.240000	1.944615	0.009223	1.153398	
14	(35993)	(35836)	0.123418	0.072785	0.012658	0.102564	1.409142	0.003675	1.033183	

Tabla 9 Resultados *Market-Basket* La Quía (Fuente: elaboración propia)

En la tabla anterior se ven las relaciones entre los productos con sus respectivas medidas de relación, donde a priori, los datos son bastante confusos dado que los resultados en “support”, “confidence”, y “lift” son más bajos de lo que esperábamos para la mayoría de las asociaciones.

Principalmente nos llamó la atención que los valores de “lift” eran altos para pocos ítems y, para los que son altos, el support es muy bajo. A continuación, observamos en un gráfico los lift de las parejas de ítems observados.

```
In [34]: #Histograma de los lift resultantes
lift_rules = rules.lift

plt.hist(lift_rules)
plt.style.use('ggplot')
plt.show()
```

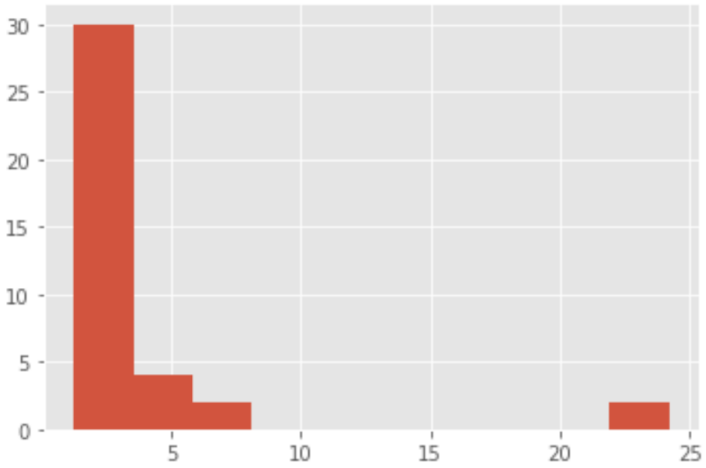


Ilustración 18 - Histograma de Lifts resultantes (fuente: elaboración propia)

A partir de este análisis, realizamos una investigación de por qué podía estar sucediendo esto. En este análisis, detectamos que Google Analytics tiene la limitante de que el usuario no se mantiene en el tiempo, sino que es un ID por usuario que se mantiene durante 7 días. Esto significa que, luego de pasados 7 días, el ID del usuario se reinicia y cambia de identificador, perdiendo la trazabilidad por usuario. Por este motivo, sumado a la baja cantidad de transacciones y la falta de metadata única para los SKU, es que decidimos cambiar el *dataset* de trabajo.

5.5. Resumen de obstáculos presentados y lecciones aprendidas

En esta sección se busca resumir los diversos obstáculos presentados y las lecciones aprendidas sobre este *dataset*.

- La asignación del score de *rating* implícito es algo que depende de la información obtenida de cada tienda y debe ser elegida cuidadosamente. Pudiendo ponderar distinto según los objetivos de cada empresa.

- Los productos deben tener una clave única o una tabla que permita vincular el mismo producto en sus diferentes talles y colores con un solo identificador.

- Los usuarios deben ser identificables a lo largo del tiempo. Debido a que la tienda no guarda registro del usuario, la variable usuario incluida en los datos es creada y mantenida por Google Analytics donde cada usuario se vuelve anónimo luego de 7 días desde su primera conexión a la sesión.

5.6.Recomendaciones a la tienda

Más allá de que no hemos podido continuar con este caso, creemos pertinente indicar todas aquellas recomendaciones que fuimos observando de los datos de la tienda.

- Pocos usuarios culminan las transacciones a través de las etapas de checkout de la página web.

  Las categorías del gráfico son las siguientes:

  - PRODUCT_VIEW: Productos visualizados.
  - ADD_TO_CART: Productos agregados al carrito productos.
  - CHECKOUT_1: Etapa de ingreso de datos de facturación.
  - CHECKOUT_1_ABANDONMENT: Sesiones que abandonan al momento de completar los datos de facturación.
  - CHECKOUT_2: Etapa de ingreso de datos de envío.
  - CHECKOUT_2_ABANDONMENT: Sesiones que abandonan al momento de completar los datos del envío.
  - CHECKOUT_3: Etapa de ingreso de datos para el pago.
  - CHECKOUT_3_ABANDONMENT: Sesiones que abandonan al momento de completar los datos del pago.

- TRANSACTION: Sesión con compra finalizada.
- TRANSACTION_WITHOUT_CHECKOUT: Sesión con compra finalizada directa (sin ingresar en los pasos de *checkout*). Esta situación ocurre cuando el usuario ya es cliente y tiene sus datos de *checkout* previamente guardados.

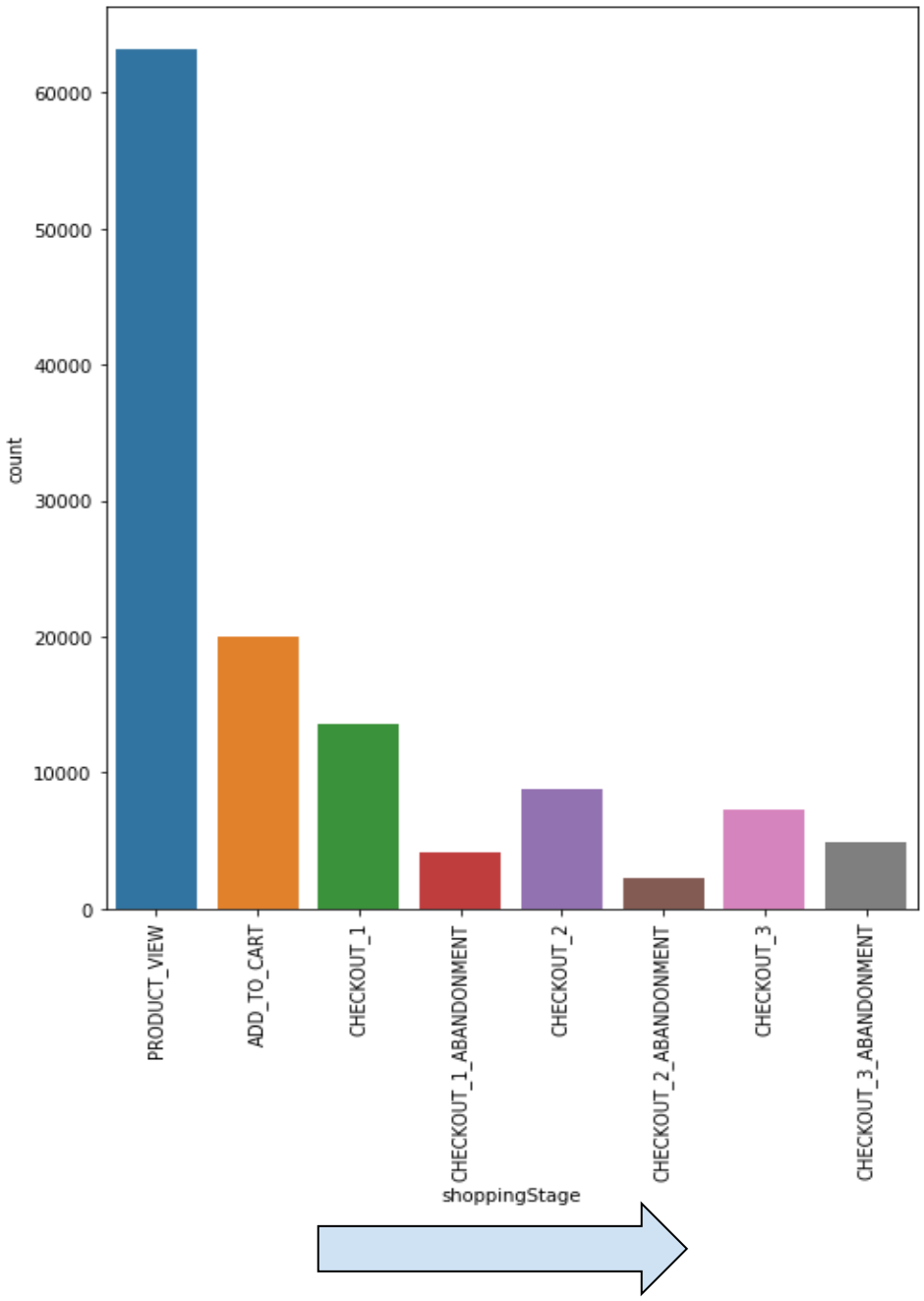


Ilustración 19 - Distribución de los estados en el *dataset* (fuente: elaboración propia)

El gráfico anterior representa el trayecto de las sesiones en la tienda de ropa. Se debe interpretar de la siguiente manera. Se visualiza que una gran masa de usuarios visualizo productos, y de éstos 20.000 agregaron productos a sus carritos. Luego 13.500 sesiones ingresaron a la etapa de facturación, de las cuales 4.000 abandonaron. A la siguiente etapa (CHECKOUT_2) ingresan 8.700 y abandonan en esta misma etapa 2.200. En la etapa de pago (CHECKOUT_3) ingresan 7.200 sesiones, de las que abandonan 5.000.

Si bien notamos una caída normal en cada uno de los *checkouts* 1 y 2, nuestra recomendación es que la empresa fortalezca la etapa del pago para evitar el abandono. Una buena estrategia es agregar descuentos por tiempo limitado en esta etapa de compra.

Otra estrategia sería implementar procedimientos que estimen el tiempo de envío. Este tipo de datos permite reducir el abandono de clientes.

Estas dos últimas estrategias se ven sustentadas en la segunda edición del monitor de *e-commerce* de UES [9] donde se afirma: "*Los envíos de la mercadería "garantizados" juegan un rol clave para aumentar el nivel de conversión. Las facilidades de envíos y de pago representan un 33% de los motivos por los cuales los uruguayos eligen el canal online. Por su parte las promociones y los descuentos en los precios constituyen un 44% de la intención de compra, siendo uno de los motivos principales para optar el canal de ventas online."*

Nota: Los números presentados en el grafico son aproximados ya que existen algunas categorías no representadas en el gráfico anterior por ser poco materiales. Ejemplo: CHECKOUT_2_WITHOUT_CHECKOUT_1 = 21 sesiones.

A su vez, encontramos que las compras realizadas por los usuarios con datos de *checkout* guardados superan el doble a las que se ingresan los datos del checkout manualmente. Cuando un usuario está culminando una compra tiene la opción de registrarse para que los datos del checkout se carguen automáticamente.

Por lo que recomendamos ofrecer descuentos a primeros clientes (aquellos que tienen que completar manualmente los datos del checkout), con tal de registrarse y acelerar compras futuras.

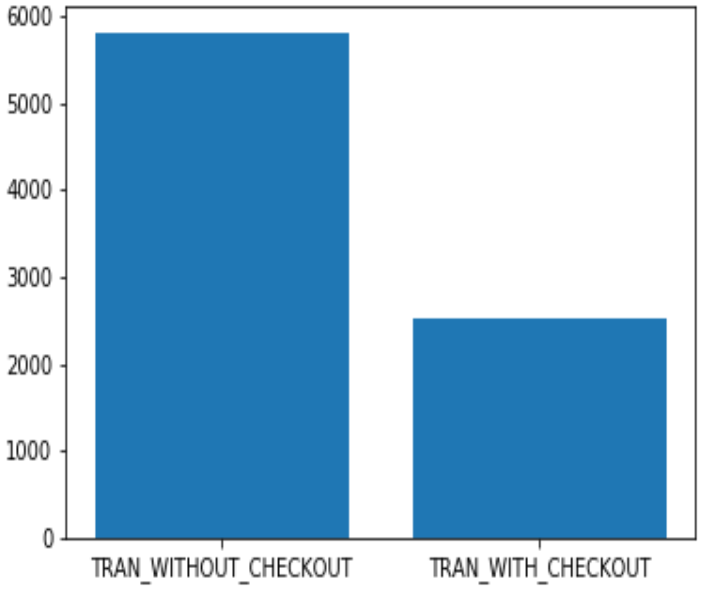


Ilustración 20 - Cantidad de transacciones con y sin etapa de checkout (fuente: elaboración propia)

- Mantener la trazabilidad del usuario.

Hoy en día la única trazabilidad del usuario que se tiene es aquella que mantiene Google Analytics. Por lo que más allá del conocimiento que tiene de sus clientes en la tienda física, no puede ofrecer ofertas especiales a sus clientes, ni adentrarse en alternativas de motores de recomendación como filtrado colaborativo.

Aunque la empresa no se mostró afín a agregar pasos en el proceso con tal de almacenar los usuarios, nos comentaron que existe la posibilidad de pagar una mensualidad a la API de registro de información de *checkout*, en donde a través de ella se puede obtener los datos de cada usuario que realiza una compra.

Con esta información se puede plantear la posibilidad de recorrer el camino inverso en Google Analytics y empezar a recabar información valiosa de los usuarios.

Esquemáticamente, esta API permitirá recorrer la información brindada por Google Analytics de la siguiente forma:

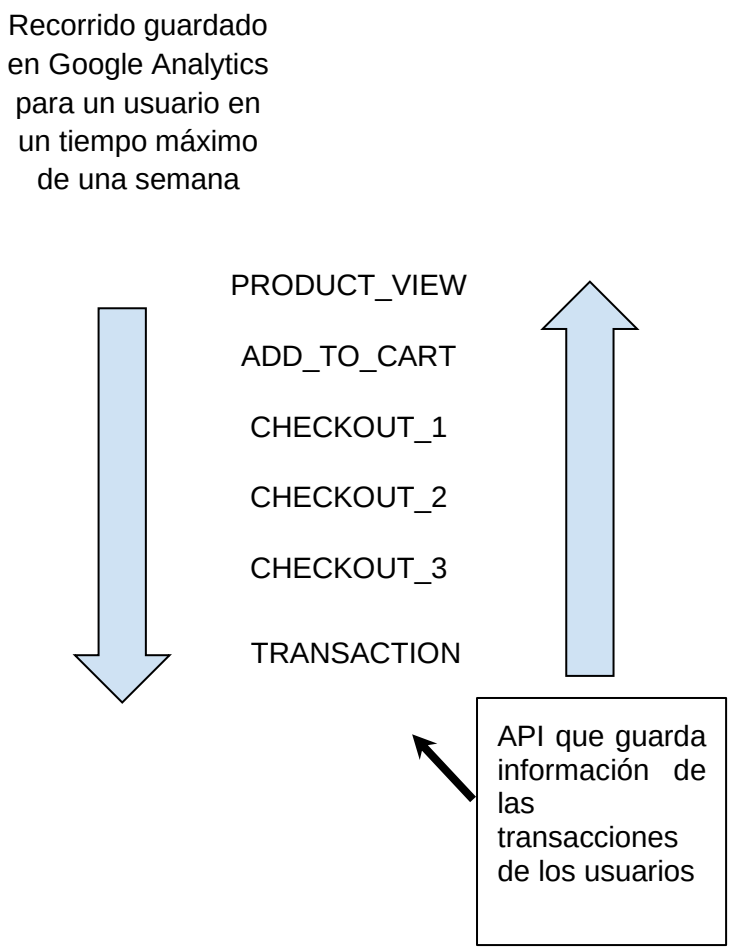


Ilustración 21 - Esquema de las APIs para recorrer los datos de las transacciones y almacenar información trazable de los usuarios (fuente: elaboración propia)

Esta solución tiene la ventaja de que es transparente en la obtención de datos del usuario, pero tiene la desventaja de que únicamente obtiene datos trazables cuando se efectúa una compra. Es decir, por cada compra tenemos la información de 7 días del usuario, pero si entra en otro momento el usuario queda sin identificar.

- Unificar los códigos SKU del ítem de la tienda.

Actualmente los ítems de la tienda mantienen SKU distintos, generados sin ningún criterio en común.

Ejemplo:

29963	CONJUNTO MONAMOUR VERDE
29947	CONJUNTO MONAMOUR BEIG

Tabla 10 Ejemplo de diferentes códigos SKU para un mismo producto (fuente: elaboración propia)

Esto hace que sea muy costoso el rastreo de un ítem y sus diferentes colores dentro de los datos extraídos de Google Analytics.

Una posibilidad sería mejorar el criterio de asignación de códigos SKU. A continuación, se presenta una propuesta:

CÓDIGO ÚNICO PRODUCTO - CÓDIGO COLOR - CÓDIGO TALLE

A la fecha, la empresa nos comentó que ya está en proceso de mejora junto con su programador.

- Utilización de *market-basket* como criterio para encontrar productos que se suelen vender en conjunto

Si bien en nuestro análisis se encontraron pocos ítems que se compran en conjunto bajo el criterio de *market-basket* (bajo support). Estas "parejas" de ítems fueron reportados a la tienda para que pueda hacer publicidad o promociones con el fin de aumentar sus ventas.

En caso de que la tienda quiera continuar con este criterio se debe solucionar el punto anterior de unificación de SKU, ya que requirió de un trabajo importante para identificar que ítems son los mismos en sus diferentes variaciones.

6. Experimentación: Rent the Runway

‘Rent the Runway’ es una compañía de servicio en línea que permite alquilar y comprar en forma online prendas y accesorios de grandes marcas de lujo (aproximadamente 123 marcas) en el rubro textil.

Su principal servicio es el alquiler de prendas por un periodo de 4 días aproximadamente, donde su valor puede variar entre un 10% a 15% del valor total de la pieza, también ofrecen un seguro de prenda a 5 dólares en caso de cualquier contratiempo. El giro de esta empresa nos favorece en el sentido de que al analizar el *dataset*, es posible contar con un histórico de prendas que tiene poca rotación de inventario. Traduciéndose en un posible motor de recomendación con mucha información de los usuarios.

Dados los problemas explicados anteriormente en referencia al uso de los datos de La Quía, decidimos continuar la implementación con el *dataset* ‘Rent the Runway’ obtenido a partir de Kaggle [10].

El cambio de la fuente de datos fue uno de los hitos más importantes de este proyecto, ya que pudimos detectar los problemas de poca variedad de productos y de no conservación de usuarios web a tiempo. A su vez, la búsqueda del nuevo *dataset* también implicó un desvío respecto al plan inicial trazado.

Si bien el *dataset* está enfocado en análisis de predicciones de talles (*fit-size recommendation*), contiene información descriptiva de los productos y calificaciones realizadas por los usuarios, que nos dan una base a partir de la cual poder evaluar diferentes alternativas de recomendación.

El *dataset* contiene la siguiente información:

- *item*_id: Identificador único del producto.
- user_id: Identificador único del usuario.
- weight: Peso del consumidor.
- height: Altura del consumidor.
- rented for: Propósito por el cual la ropa es rentada la prenda por el consumidor.

- body type: Tipo de cuerpo del consumidor.
- category: Categoría del producto.
- *rating*: Puntuación dada al producto por el cliente.
- review_text: Reseña escrita por el consumidor.
- review_summary: Resumen de la reseña escrita por el consumidor.
- size: El tamaño estándar del producto.
- age: Edad del consumidor.
- bust size: Medidas del busto del consumidor.
- *fit*: Tipo de *fit*. Las categorías son: *fit*, large y small.
- review_date: Fecha en la que la reseña fue escrita.

	fit	user_id	bust size	item_id	weight	rating	rented for	review_text	body type	review_summary	category	height	size	age	review_date
0	fit	420272	34d	2260466	137lbs	10.0	vacation	An adorable romper! Belt and zipper were a lit...	hourglass	So many compliments!	romper	5' 8"	14	28.0	April 20, 2016
1	fit	273551	34b	153475	132lbs	10.0	other	I rented this dress for a photo shoot. The the...	straight & narrow	I felt so glamourous!!!	gown	5' 6"	12	36.0	June 18, 2013
2	fit	360448	NaN	1063761	NaN	10.0	party	This hugged in all the right places! It was a ...	NaN	It was a great time to celebrate the (almost) ...	sheath	5' 4"	4	116.0	December 14, 2015
3	fit	909926	34c	126335	135lbs	8.0	formal affair	I rented this for my company's black tie award...	pear	Dress arrived on time and in perfect condition.	dress	5' 5"	8	34.0	February 12, 2014
4	fit	151944	34b	616682	145lbs	10.0	wedding	I have always been petite in my upper body and...	athletic	Was in love with this dress !!!	gown	5' 9"	12	27.0	September 26, 2016

Tabla 11 Primeros datos incluidos en el *dataset* de Rent the Runway (fuente: elaboración propia)

6.1.Exploración y Transformación de Datos

El *dataset* contiene 192.544 transacciones, 5.850 productos y 105.571 usuarios. Estos usuarios representan un subconjunto del total de usuarios posibles, correspondientes a aquellos que han

realizado todo el ciclo completo de tráfico incluido una instancia final de *review* y *rating* sobre su compra. En resumen, el análisis y recomendación se basará en aquellos usuarios que:

1) Ingresaron a la web
2) Interaccionan con el/los productos/s
3) Compraron/alquilaron al menos una prenda
4) Expusieron una *review* y un *rating* de su transacción

Ilustración 22 - Esquema de las diferentes interacciones de usuarios con la página web (fuente: elaboración propia)

El *dataset* presenta la siguiente distribución de cantidad de productos comprados y/o alquilados por los diferentes usuarios:

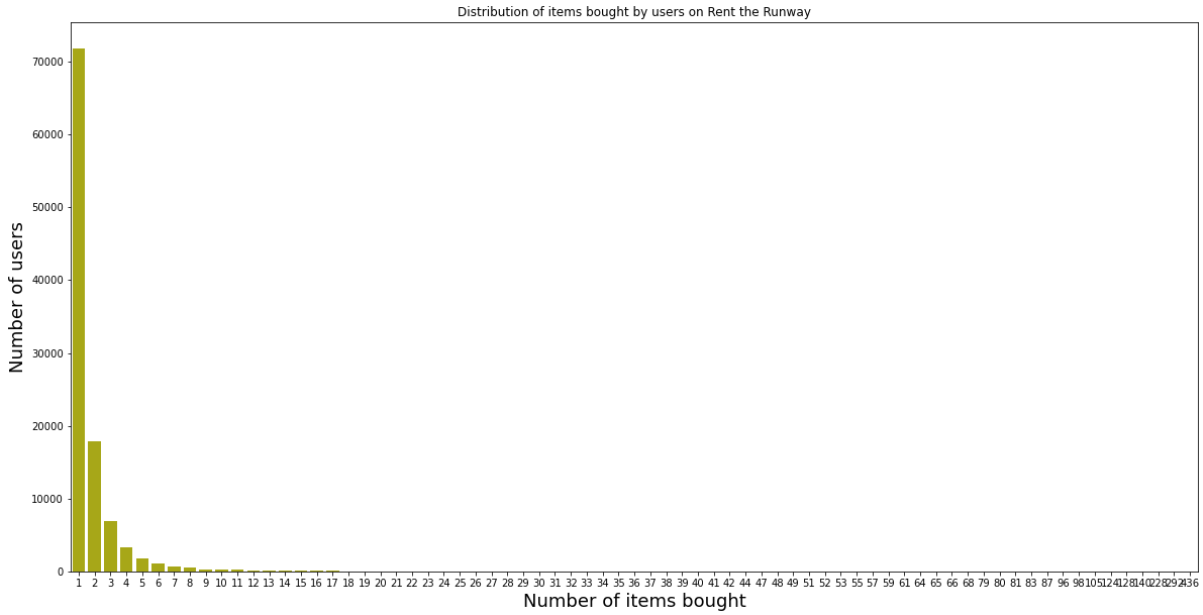


Ilustración 23 - Distribución de los ítems comprados por los usuarios (fuente: elaboración propia)

Aproximadamente el 66% de los usuarios compraron y/o alquilaron un solo un producto. Esto agrega cierta complejidad al momento de buscar asociaciones entre usuarios y se podría ver como un problema de *cold start* (Cold start) si se busca predecir ítems para estos.

6.1.1. Rented for

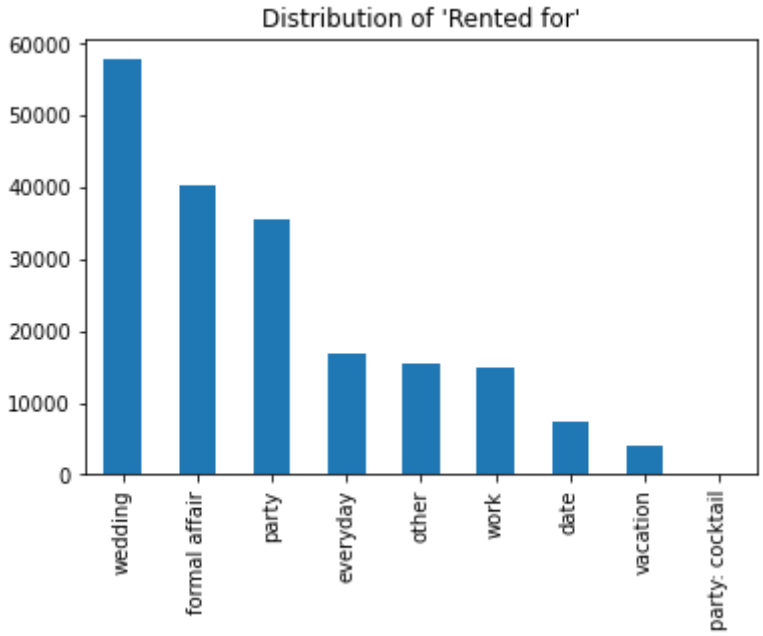


Ilustración 24 - Distribución de la categoría 'Rented for' (fuente: elaboración propia)

Existen 9 categorías en función del tipo de evento para el cual fue alquilada la prenda. Como consideración, a categoría ‘party: cocktail’ la asignaremos a la categoría ‘party. Se asume que debido al nombre se puede juntar con la categoría party y dado la poca cantidad de registros no impacta de forma significativa en los resultados.

6.1.2. Body type

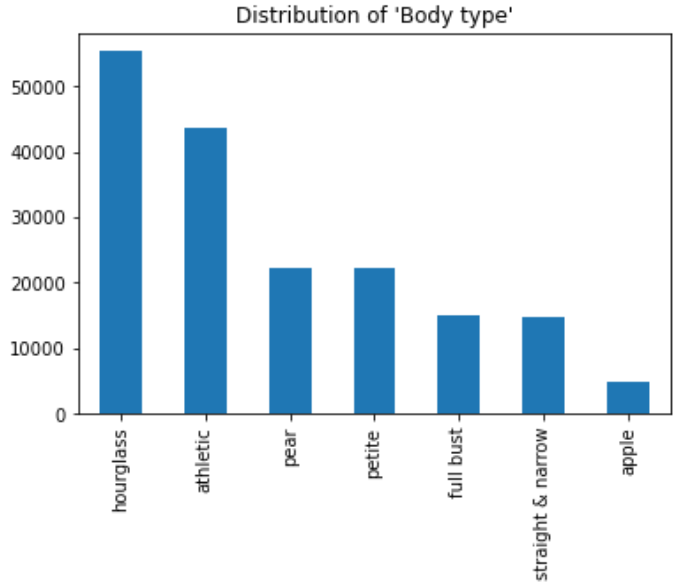


Ilustración 25 - Distribución de la categoría 'Body type' (fuente: elaboración propia)

El *dataset* contiene 7 categorías que se definen en función del tipo de cuerpo.

6.1.3. Category

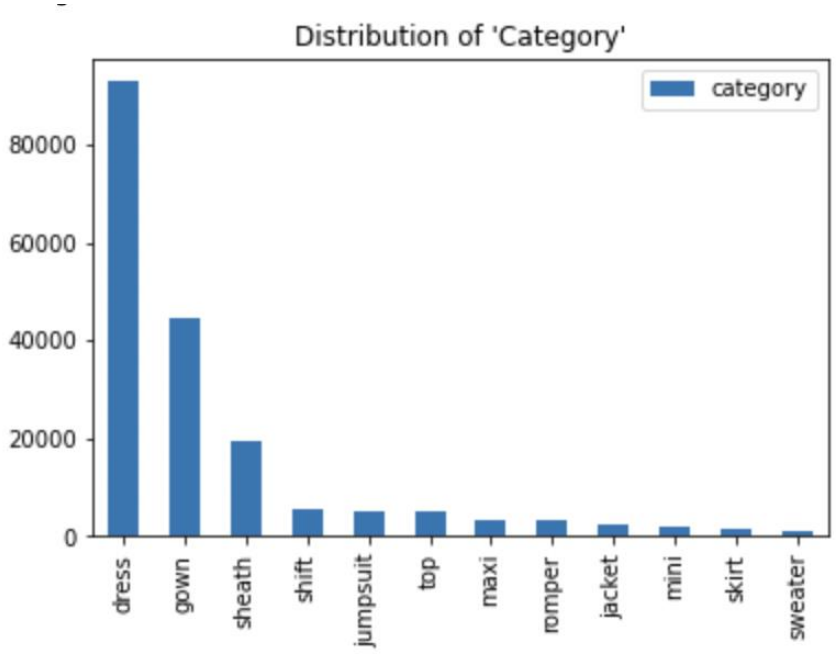


Ilustración 26 - Distribución de la variable 'Category' (fuente: elaboración propia)

En total hay 68 categorías, pero varias de ellas tienen pocas transacciones. Si se consideran las 56 categorías menos representativas, estas apenas llegan al 0.5% del total de las transacciones. En el gráfico se muestran aquellas categorías principales, y se puede apreciar 3 categorías dominantes: *'dress', 'gown*' y '*sheath'*, las cuales representan el 81% de las transacciones.

En un segundo enfoque, se trabaja con algoritmos de NLP (Procesamiento de Lenguaje Natural) sobre las *reviews* de los usuarios, extrayendo los *tokens* de cada *reviews* y asignándoles a la caracterización de los ítems (junto con la variable 'category').

6.1.4. Valores faltantes

El *dataset* contiene valores faltantes en diferentes columnas:

	missing_count	missing_percent
weight	29982	15.571506
bust size	18411	9.561970
body type	14637	7.601899
age	960	0.498587
height	677	0.351608
rating	82	0.042588
rented for	10	0.005194

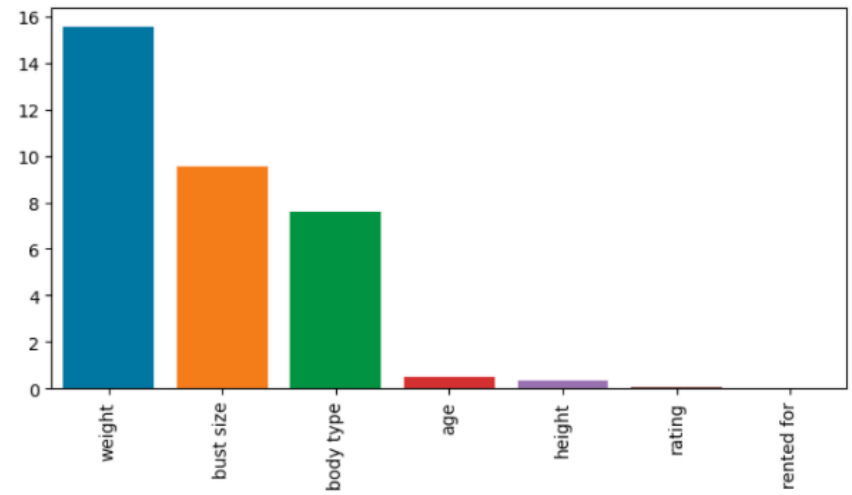


Ilustración 27 - Porcentaje de valores faltantes por columna (fuente: elaboración propia)

Debido a que las variables utilizadas en la primera implementación (‘*body type’*, *‘rating’* y *'rented for’*) tienen un porcentaje bajo de nulos, se optó por quitar estos registros del *dataset* ya que son poco representativos.

6.1.5. *Rating* explícito

En este caso se cuenta con un *rating* explícito en la variable ‘*rating*’ que hace referencia a la calificación que el usuario adjudica a la prenda subyacente. Este *rating* se efectúa luego de utilizado el servicio.

6.2. Implementación de algoritmos

6.2.1. Separación de datos en *train* y *test*

Previo a la implementación de algoritmos se dividieron los datos en: datos de entrenamiento (*train*) y datos para el testeo (*test*), de la siguiente forma:

1. Separamos los datos de aquellos usuarios que tienen una sola transacción. Este proceso se realizó debido a que, si bien es de interés para esta tesis la investigación de problemas de cold start, el objetivo de la implementación es no lidiar con ellos.

2. Del *dataset* resultante del punto anterior separamos un 30% para *test* y un 70% para *train* de manera aleatoria. Se utilizó el parámetro "*stratify*" que asegura que los usuarios de *test* tengan por lo menos un alquiler / compra en *train*. Dicho de otra manera, se busca no tener que predecir una recomendación para un usuario en *test* que no tenga alquiler / compras anteriores en los datos de *train*.

3. Se adicionó los usuarios con una sola transacción (separados en el punto 1) a los datos de *train*, con el fin de disponer de más datos para entrenar.

6.2.2. Métricas para evaluación de resultados

Es necesario definir, previo a comenzar con la implementación, como va a ser la evaluación de los distintos motores de recomendación.

Es importante destacar que se decidió quedarse con los *ratings* en *test*, mayores o iguales a 8, dado que, conceptualmente son los ítems que los usuarios califican como preferentes, y cuando realizamos el cálculo de los *TP (True Positives)* es fundamental contemplar ítems relevantes.

RMSE (*Root-mean-square-error*)

La raíz del error cuadrático medio es una medida de error muy útil y usada en problemas de regresión. Mide la desviación estándar de los residuales (errores en las predicciones).

Básicamente, compara el valor predicho para un conjunto de datos con el valor real, y calcula la diferencia para cada par (predicción – valor real). Se suman las diferencias, tantas como predicciones se hayan realizado, se computa el promedio y finalmente la raíz cuadrada.

Como se detalla más adelante, esta métrica la utilizamos para evaluar el rendimiento entre los diferentes algoritmos de filtrado colaborativo. En estos motores, para realizar la recomendación final se predice el *rating* y se compara contra el *rating* real que el usuario asigno al ítem correspondiente.

$$RMSE = \sqrt{\frac{\sum_{u,j}^{N} \hat{r_{u,j}} - r_{u,j}}{N}}$$

En nuestro caso, siendo $r_{u,j}$ los *ratings* por el usuario u al ítem j y N la cantidad total del set de datos a evaluar.

Precision

Esta métrica permite medir la calidad de la recomendación. Es la fracción de los ítems que son relevantes sobre todos los ítems recomendados.

Responde a la siguiente pregunta: ¿Cuántos elementos recomendados fueron realmente comprados por los usuarios?

La fórmula es la siguiente:

$$Precision = \frac{tp}{tp + fp}$$

TP = True Positive. Son los casos en donde las recomendaciones fueron efectivamente compradas por el usuario.

FP = False Positive. Representa el resto de los ítems recomendados que no fueron comprados por el usuario.

Ejemplo gráfico:

Ilustración 28 - Ejemplo de compras de un usuario sobre la lista de recomendaciones (fuente: elaboración propia)

En el ejemplo anterior los ítems comprados por el usuario son los marcados con un círculo y representan los TP. La cantidad de recomendaciones realizadas son el TP + FP.

Precision de ejemplo = 2 / 7

Recall

Es una medida de completitud, determina la fracción de ítems relevantes recomendados de la totalidad de ítems relevantes para el usuario.

Responde a la pregunta de: ¿Cuántos ítems comprados por los usuarios fueron recomendados?

La fórmula es la siguiente:

$$Recall = \frac{tp}{tp + fn}$$

TP = True Positive. Estos son los casos en donde las recomendaciones fueron efectivamente compradas por el usuario.

FN = False Negative. Representa aquellos ítems que fueron comprados por los usuarios, pero no fueron recomendados.

Ejemplo gráfico:

Ilustración 29 - Ejemplo de compras de un usuario por la lista de recomendaciones y por fuera de la misma (fuente: elaboración propia)

En el caso anterior podemos observar como un ítem comprado se encontraba en la recomendación al usuario (TP) y el otro ítem comprado no fue recomendado (FN).

Recall de ejemplo = 1 / 2

F1

Las métricas *Recall* y *Precision* son muy importantes, pero tienen el problema de que suelen maximizar hacia lados contrarios. Si queremos maximizar *Precision* es evidente que buscaremos sólo recomendar la cantidad de ítems en los que tengamos alto grado de seguridad de que el usuario quiere comprar, pero esto haría que el resto de los ítems comprados no sean recomendados bajando el score del *Recall*. Por otro lado, para maximizar el *Recall* se debe recomendar muchos ítems que pueden no ser comprados, pero esto haría bajar el score de *Precision*.

Dado que nos interesa tener una recomendación completa y exacta es que debemos tener ambas métricas en cuenta. La manera elegida para lograr este objetivo fue basarnos en la métrica F1.

La diferencia que tiene la métrica F1 del resto de las medidas F(n) es que *Precision* y *Recall* tienen el mismo peso, por lo que tienen la misma importancia en el resultado final de la métrica.

La fórmula es la siguiente:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

MAR@K (Mean Average *Recall* at K)

Esta métrica busca medir el *Recall* ponderando cada true positive (TP) en función del orden en que se muestra en la recomendación. Esta métrica es una buena opción en caso de empate entre modelos que obtuvieron el mismo F1 pero que internamente muestran los resultados en órdenes distintos según la preferencia de los usuarios.

Por ejemplo, si un ítem efectivamente comprado o alquilado por el usuario se encuentra en la posición 3 de la recomendación, el TP equivale a 1/3.

6.2.3. *Baseline* elegido (Popular)

Previo a comenzar, se definió un *baseline* que demuestre cómo es el estado de la organización respecto a su motor de recomendación actual. Lamentablemente no se cuenta con esta información por lo que se realizó una búsqueda alternativa para encontrarlo.

La búsqueda consistió en la realización de un filtro de popularidad, donde se muestre a todos los usuarios los N productos que contengan la suma de *ratings* más altos en los datos de entrenamiento.

Esquemáticamente, los pasos para su obtención fueron:

1. Se agruparon los ítems en *train* sumando sus *ratings*.
2. Se ordenaron descendentemente por la suma de *ratings*.
3. Se tomaron las primeras N recomendaciones.

Los resultados fueron los siguientes:

n_recomendaciones	pop_recall	pop_precision	pop_F1
5	0.03061	0.00800	0.012685
10	0.05119	0.00669	0.011833
20	0.08138	0.00532	0.009987
50	0.13561	0.00354	0.006900

Tabla 12 Resultados de *Precision*, *Recall* y F1 del baseline (fuente: elaboración propia)

Debido a sus buenos resultados en la predicción de ítems de usuarios en *test* es que se consideró que la empresa podría estar utilizando este método como motor de recomendación. La decisión fue tomar este método como baseline.

6.2.4. Algoritmos

6.2.4.1. Content

Esta clase de recomendaciones están basadas en información de los contenidos de los elementos. Por ejemplo, descripciones de los productos y atributos. En este *dataset*, como caracterización objetiva de los productos se tiene la variable 'category', que detalla la categoría de cada ítem. Para implementar las estrategias de filtrado basado en contenido se siguieron dos grandes enfoques:

1) Caracterización de los ítems en función de variables categóricas creadas por los usuarios ('rented for' y 'body type') más la variable 'category'.

2) Caracterización de los ítems en función de las *reviews* aplicando NLP. En este modelo se aplicaron dos enfoques, matriz de similitud *item-item* y KNN.

6.2.4.1.1. Basado en variables categóricas

Para implementar este algoritmo utilizamos variables categóricas para caracterizar cada uno de los ítems.

En este enfoque, las variables que utilizaremos para categorizar los ítems son: 'Rented for', 'Body type' y 'category'. Las primeras dos son atributos asignados por los usuarios, donde esto trae aparejado que un mismo ítem puede tener diferentes asignaciones en cada una de estas dos variables. Por otro lado, 'category' es un atributo que describe a cada producto, y no depende de categorizaciones de usuarios.

La estrategia utilizada es crear una nueva columna ('soup') que tenga en cuenta todas las categorizaciones que tiene cada *item*_id a lo largo del *dataset* de entrenamiento. Básicamente, cada valor sería un campo de texto con varias palabras, tantas como cada una de las veces que aparece categorizado dicho *item*_id en el *dataset* (simplemente para darle una ponderación a la cantidad de veces que un *item*_id figura caracterizado con tal descripción).

	item_id	soup
0	123373	formal affair gown petite formal affair gown ...
1	123793	wedding gown full bust wedding gown athletic ...
2	124204	party dress athletic other dress hourglass pa...
3	124553	wedding dress hourglass party dress straight ...
4	125424	formal affair dress hourglass wedding dress a...
...	...	...
5668	2963850	work skirt full bust other skirt athletic vac...
5669	2964470	work top hourglass everyday top pear everyday...
5670	2965009	vacation coat hourglass everyday coat hourglass
5671	2965924	date jacket petite work jacket hourglass othe...
5672	2966087	work pants athletic everyday pants pear party...

Tabla 13 Asignación de metadata (soup) para cada ítem id (fuente: elaboración propia)

La razón por la que realizamos dicha estrategia se basa en que los ítems quedan explicados por las categorías por las que más fueron utilizados.

Por ejemplo, si un vestido es utilizado por muchos usuarios para eventos del tipo casamiento, el mismo va a contener mayoritariamente la palabra "*wedding*" dentro de su sopa de palabras. Gracias a esto cuando se quiere recomendar un ítem asociado a casamiento la recomendación del content va a ser aquella con muchas veces la palabra "*wedding*".

A partir de esta "sopa" de palabras que categorizan a cada *item*_id, creamos una vectorización de este atributo y generamos la matriz de similaridad entre ítems.

Ejemplo de aplicación de content para este caso:

Usuario 377.141 compró el ítem 2334570:

	user_id	item_id	rating	review_summary
9900	377141	2334570	10.0	Super cute and comfy

Tabla 14 Datos históricos del user_id 377.141 (fuente: elaboración propia)

Este ítem en *train* tiene la siguiente categorización:

```
'party': 11,
'jumpsuit': 21,
'athletic': 7,
'work': 2,
'hourglass': 6,
'petite': 5,
'date': 3,
'other': 3,
'wedding': 2,
'pear': 2,
'straight': 1,
'&': 1,
'narrow': 1})
```

Ilustración 30 - Descripción del ítem 2334570 (fuente: elaboración propia)

Como se puede visualizar, este ítem mayoritariamente tiene la categoría *jumpsuit* y fue utilizado mayoritariamente para fiestas (*party*).

La recomendación basada en el ítem sería la siguiente:

```
content_recommender(2334570)

4022    2155094
4281    2291737
4490    2396750
5286    2816095
3891    2087286
5315    2829293
4383    2340996
4613    2463317
4334    2315001
4641    2476239
Name: item_id, dtype: int64
```

Ilustración 31 - Recomendación para ítem 2334570 (fuente: elaboración propia)

Donde el primer ítem que recomienda (2155094) esta categorizado de la siguiente forma en *train*:

```
'date': 10,
'jumpsuit': 87,
'athletic': 28,
'work': 1,
'straight': 12,
'&': 12,
'narrow': 12,
'everyday': 2,
'petite': 12,
'vacation': 4,
'party': 46,
'other': 15,
'hourglass': 23,
'wedding': 5,
'pear': 7,
'full': 3,
'bust': 3,
'formal': 4,
'affair': 4,
'apple': 2})
```


Ilustración 32 Descripción ítem 2155094 (fuente: elaboración propia)

Como podemos visualizar, el ítem recomendado tiene similares proporciones de recomendaciones, ya que es categorizado como jumpsuit y fue utilizado mayoritariamente para fiestas (party)

Este método obtuvo los siguientes resultados:

n_recomendaciones	content_soup_recall	content_soup_precision	content_soup_F1
5	0.01155	0.00302	0.004788
10	0.02010	0.00263	0.004651
20	0.03219	0.00210	0.003943
50	0.06248	0.00163	0.003177

Tabla 15 Resultados de *Precision*, *Recall* y F1 de content basado en variables categóricas (fuente: elaboración propia)

Esta experimentación se encuentra por debajo del resultado obtenido en el *baseline*, por lo que decidimos buscar otra información relevante con la cual recomendar.

6.2.4.1.2. Basado en *reviews*

Antes de comenzar con el desarrollo de este algoritmo, es necesario definir algunos conceptos de NLP que serán nombrados durante la explicación:

- Documento o bloque de texto: Es un objeto o una unidad.
    - Puede presentar una frase, o un conjunto de estas mismas. Ejemplo: Una *review* realizada por un usuario.
- Palabras: Son las variables del documento, con algunas propiedades:
    - Normalización: forma base + transformación
    - Posición en el documento
    - Frecuencia
- Corpus: Es un conjunto de documentos. Ejemplo: Todas las *reviews* del *dataset*
- Vocabulario: Un *set* finito de palabras (subset del vocabulario total del lenguaje) que usamos para el problema puntual

Matriz similitud *item-item*

Para la implementación de este algoritmo y vectorización del corpus se usó TF-IDF (*Term Frequency Inverse Document Frequency*). Es un método de vectorización que ayuda a detectar la importancia de una palabra en un documento. A diferencia de *Bag of words*, que simplemente vectoriza las palabras en función de la cantidad de ocurrencias, TF-IDF es una variante más compleja que refleja la importancia de una palabra en un documento perteneciente a un corpus.

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}}$$

Frecuencia de la palabra ***i*** en el documento ***j***

$$idf(w) = log(\frac{N}{df_t})$$

Frecuencia inversa de la palabra ***w*** en todo el corpus

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

tf_{ij} = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

Peso TF-IDF para la palabra ***w*** en el documento ***i j***

Ilustración 33 - Ilustración sobre las Fórmulas para el armado del TF – IDF [11]

Para su implementación se combinaron las variables 'category', 'review_summary' y 'review_text' en un solo campo, denominado '*item*_review_text'.

item_id	item_review_text
123373	gown simple elegant warned loved dress fit wel...
123793	gown glam miles fashion inspo holiday party os...
124204	dress loads compliments night long stood among...
124553	dress love one flattering great color get enou...
125424	dress dress amazing hourglass figures flatteri...

Tabla 16 Ejemplo de variable '*item*_review_text' para algunos ítems (fuente: elaboración propia)

Como primera etapa de preprocesamiento se quitaron los caracteres especiales incluidos en las *reviews*.

En una segunda etapa se quitaron las palabras más comunes del vocabulario, las que en definitiva no agregan valor a ser analizadas (denominadas, *stop words*). Debido a que el *dataset* contiene *reviews* en inglés se quitaron palabras como *the, is, at, which*, entre otras.

En una tercera etapa se probaron diferentes alternativas para obtener la raíz de las palabras, entre ellas lematización y *stemming* (métodos de normalización que buscan obtener el lema y la raíz de la palabra, respectivamente). Se obtuvieron mejores resultados solo con s*temming*, por lo que el preprocesamiento se realizó con esta última instancia.

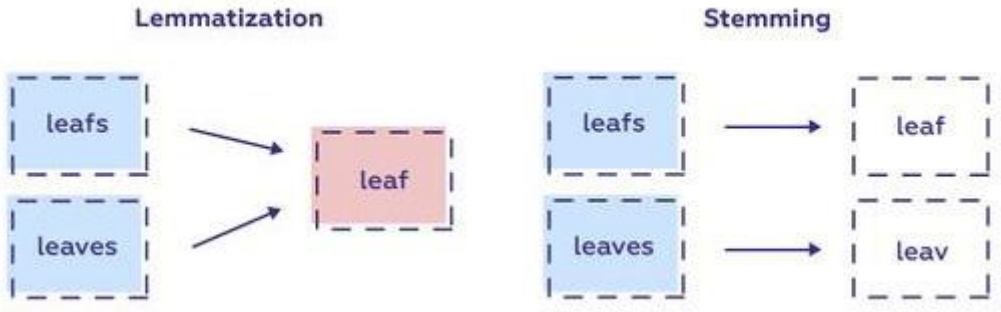


Ilustración 34 - Ejemplo de lematización y *steamming* [12]

Previo al entrenamiento se generó un vector para cada ítem donde para cada palabra resultante de los procesos anteriores se le calcula TF - IDF explicado al principio de esta sección. Uno de los parámetros más importantes a la hora de realizar TF- IDF es n-gram, el mismo indica que cantidad de palabras reales que son una palabra a analizar. Por ejemplo, si se indica la utilización de unigramos o bigramos, la frase: “Me encanta esta remera”, se separaría en las siguientes palabras: [me, encanta, esta, remera, me encanta, encanta esta, esta remera]. Sobre cada una de estas palabras se medirá la importancia para cada ítem (documento) en función del resto de los ítems (corpus). Para esta implementación el mejor resultado fue logrado utilizando unigramos y bigramos.

Con la matriz resultante del punto anterior se calculó la similitud de coseno y la distancia euclídea entre ítems.

Los resultados obtenidos para cada una de las distancias calculadas y las métricas mencionadas anteriormente son los siguientes:

Similitud del coseno:

	n_recomendaciones	cos_recall	cos_precision	cos_F1
0	5	0.03503	0.00916	0.014523
1	10	0.05506	0.00720	0.012735
2	20	0.08642	0.00565	0.010607
3	50	0.14258	0.00373	0.007270

Tabla 17 Resultados con Similitud del coseno (fuente: elaboración propia)

Distancia euclídea:

	n_recomendaciones	euc_recall	euc_precision	euc_F1
0	5	0.00010	0.00003	0.000046
1	10	0.00016	0.00002	0.000036
2	20	0.00048	0.00003	0.000056
3	50	0.00110	0.00003	0.000058

Tabla 18 Resultados con distancia euclídea (fuente: elaboración propia)

Se pudo observar con la similitud del coseno una mejora respecto al *baseline*.

KNN

En este algoritmo también se utilizó como elemento de caracterización para los ítems las columnas de *reviews* y 'category', tal como se explicó en el punto anterior.

El *pipeline* de este algoritmo es:

1) Vectorización TF-IDF de 'category', 'review_summary' y 'review_text'.
2) Conversión a columnas *dummy*. Matriz ítems vs palabras.
3) Búsqueda de relaciones entre ítems con KNN y n_neighbors=3. Para calcular estas relaciones se utilizó la distancia euclídea.

4) N recomendaciones.

Los resultados obtenidos para las diferentes N recomendaciones fueron los siguientes:

n_recomendaciones	euc_recall	euc_precision	euc_F1
5	0.02600	0.00680	0.010780
10	0.03974	0.00519	0.009181
20	0.06219	0.00406	0.007622
50	0.10432	0.00273	0.005321

Tabla 19 Resultados KNN (fuente: elaboración propia)

Si comparamos la métrica F1 con el desarrollo del punto anterior (matriz *item-item* con TF-IDF) obtuvimos peores resultados, por lo que al momento de implementar el híbrido avanzaremos con TF-IDF basado en *reviews*.

6.2.4.2. Collaborative

Para cada una de las implementaciones vamos a utilizar **RMSE** como comparación y evaluación de modelos.

6.2.4.2.1. Matriz de similitud

Para implementar y evaluar los algoritmos colaborativos, se deben calcular previamente las matrices que vinculan los usuarios con los ítems y sus respectivos *ratings*.

Como se mencionó en la sección correspondiente dentro del marco teórico (Matrices de similitudes), para el cálculo de la matriz de similitud existen diferentes alternativas para medir similitudes y semejanzas entre dos entidades (ejemplo: distancia euclídea, distancia del coseno, distancia de Pearson entre usuarios o entre ítems). Este tipo de modelos colaborativos son basados en memoria, ya que recomiendan basados únicamente en la similaridad.

En un principio, vamos a implementar un par de opciones para medir similitudes simplemente a modo comparativo, para luego sí generar la matriz con los diferentes métodos para el cálculo de la distancia y poder evaluar los respectivos rendimientos.

1er *Collaborative* (*mean rating*)

En este primer CF vamos a construir una matriz con los usuarios y los ítems, como filas y columnas respectivamente. De esta forma, tenemos los *ratings* dados por cada uno de los usuarios para todos los ítems valorados. En esta implementación, se completarán los missing values con los promedios de *ratings* sobre cada ítem, y en el caso de que algún ítem no tenga ninguna calificación, se le asignará el valor medio de *ratings*: 5.

Con estos datos, podemos calcular el *rating* estimado para cada ítem de *test*, y compararlo contra los *ratings* de los ítems en *test*. Con este método, computamos el RMSE y lo utilizamos como métrica para medir la performance entre los CF (ver RMSE entre collaboratives)

2do *Collaborative* (Matriz de similitud del coseno)

Como en el caso anterior, construimos la matriz con los usuarios y los ítems como las filas y las columnas, respectivamente.

En este caso además creamos la matriz de similitud entre usuarios usando la distancia del coseno. La similitud entre usuarios se construye a partir de los *ratings* dados por cada usuario. A partir de esta similitud entre usuarios, para cada *missing value* en un par *item-user*, vamos a la matriz de similitud del coseno, y ponderando el grado de similitud con los usuarios que le dieron un *rating* a ese ítem, calculamos el *rating* promedio y se lo computamos a dicho missing value.

Nuevamente, con estos datos, podemos calcular el RMSE entre el *rating* estimado y el *rating* predicho para cada ítem (ver RMSE entre collaboratives)

6.2.4.2.2. Implementación *Alternating Least Squares* (ALS)

Como comentamos anteriormente en el marco teórico, dada las limitaciones de infraestructura de ALS, decidimos realizar una implementación en la API de Apache Spark para python (PySpark) dado que ofrece una paralelización del modelo de ALS para el filtrado colaborativo en la predicción de *ratings*, y presenta un excelente rendimiento y escalabilidad.

A diferencia del modelo anterior que es basado en memoria. En este lo primero a realizar, es la creación del modelo, donde se le indican determinados parámetros para identificar a los *usuarios*, los ítems, los *ratings*, la estrategia de *coldstart* y otros. Luego se realiza la selección del modelo mediante un análisis de regresión con ajuste automático de hiper parámetros, con *param grid search.* Computando los posibles valores de los hiper parámetros del modelo y ejecutando las combinaciones de ellos de forma iterativa, para luego comparar y obtener el modelo óptimo. Los parámetros utilizados fueron:

- ***Rank***: Cantidad de factores latentes
- **regParam:** Regularización.

```
# Add hyperparameters and their respective values to param_grid
param_grid = ParamGridBuilder() \
            .addGrid(als.rank, [10, 50, 100, 180]) \
            .addGrid(als.regParam, [.01, .1, .4, .6]) \
            .build()
```

```
# Define evaluator as | and print length of evaluator
evaluator = RegressionEvaluator(
            metricName="rmse",
            labelCol="rating",
            predictionCol="prediction")
print ("Num models to be tested: ", len(param_grid))
```

```
Num models to be tested:  16
```

Ilustración 35 - Búsqueda de parámetros a través del método “Grid Search” (fuente: elaboración propia)

Esta comparativa se realiza mediante la evaluación del cálculo del error cuadrático medio (RMSE) ya que es la métrica comúnmente más utilizada como principal métrica en problemas de regresión, como se mencionó anteriormente.

Finalmente realizamos el *fit* del modelo, y computamos las predicciones con el mejor modelo llegando a un RMSE de 1.983, representando un error promedio al comparar los valores predichos del conjunto de entrenamiento con los valores reales.

6.2.4.2.3. Implementación *Singular Value Decomposition* (SVD)

Iniciamos definiendo un modelo de SVD, con los siguientes parámetros:

- **n_factors**= 400 (cantidad de factores latentes)
- **lr_all** = 0.01 (learning rate)
- **n_epochs** = 50 (número de veces que se ejecuta el algoritmo)

Continuamos realizando un split de *train* para realizar el *cross validation* y entrenamos el modelo.

Definimos una función denominada "***Score***", que se encarga de computar el cálculo de RMSE mediante la obtención de la predicción (y_pred) y el valor real (y_true). Para el cálculo de la predicción, creamos un zip con los pares de *item_id* y *user_id* en *test*, e inicializar una iteración para cada par donde calculamos su predicción y donde se va agregando a una lista el valor estimado por el algoritmo. Una de las desventajas de este modelo es que tenemos que computar la predicción del *rating* para todos los ítems por usuario.

Una vez llamada la función **RMSE**, obtenemos un valor de 1.445, representando el error promedio de la predicción del algoritmo.

6.2.4.2.4. RMSE entre *collaboratives*

El siguiente gráfico muestra el error cuadrático medio (RMSE) entre diferentes algoritmos collaborative, donde se observa un mejor rendimiento en la implementación de SVD.

Modelo	RMSE
1er collaborative	1.519
2do collaborative	1.662
SVD	1.445
ALS	1.984

Tabla 20 Comparación RMSE para los modelos de CF (fuente: elaboración propia)

El método con menor RMSE es SVD. Con este criterio, decidimos avanzar en el desarrollo del motor para realizar el cómputo de *Recall* y *Precision* a fines de tener una métrica comparativa estandarizada.

Definimos la función *recomendation_all_user*, cuyo *output* nos va a presentar una lista de listas, donde cada elemento de esa lista es el conjunto de las mejores N recomendaciones de ítems para cada usuario.

Dichas recomendaciones se calcularon varias veces con diferentes N (5, 10, 20, 50) para ver la variación de los resultados de *Precision*, *Recall* y F1:

n_recomendaciones	recall	precision	F1
5	0.00219	0.00057	0.000905
10	0.00455	0.00059	0.001045
20	0.00832	0.00054	0.001014
50	0.01748	0.00046	0.000896

Tabla 21 *Precision*, *Recall* y F1 para SVD (fuente: elaboración propia)

Adicionalmente se realiza la optimización de parámetros (*fine tuning*) del algoritmo SVD.

6.2.4.2.5. Optimización de parámetros - SVD

La optimización de parámetros se centró en cuatro parámetros clave:

- n_epochs = Determina la cantidad de épocas necesarias para que el modelo aprenda. La sobreutilización de este parámetro puede llevar al sobreajuste del modelo a los datos de *train*.
- lr_all = Determina la tasa de aprendizaje (*learning rate*) en cada iteración para todos los parámetros.
- reg_all = Determina el parámetro de regularización.
- n_factors = Número de factores utilizados en la factorización de matrices.

Debido a la cantidad de pruebas a realizar, se optó por no aplicar el método "*Grid Search*" el cual utiliza un enfoque de combinaciones exhaustivas, sino que se optó por utilizar "*RandomizedSearch*" que computa las métricas de evaluación para varias combinaciones de parámetros realizando *cross validation*. La combinación de parámetros se selecciona mediante un muestreo randómico del espacio de parámetros.

El resultado del RMSE en *test* fue el siguiente:

Modelo	RMSE
SVD	1.445
SVD optimizado	1.420

Tabla 22 SVD vs SVD Optimizado (fuente: elaboración propia)

La diferencia entre los parámetros utilizados son las siguientes:

Parametros	SVD	SVD optimizado
n_epochs	50	5
lr_all	0.01	0.0105
reg_all	0.02	0.138
n_factors	400	10

Tabla 23 Parámetros utilizados SVD vs SVD Optimizado (fuente: elaboración propia)

La conclusión más importante observada en esta búsqueda es que nuestro modelo sin optimizar estaba sobre ajustando. Lo pudimos observar por la gran diferencia en la cantidad de *epochs* como en la cantidad de factores utilizados. A su vez, se puede visualizar esta conclusión con el aumento del componente regularizador.

6.2.4.3. Híbrido

Luego de experimentar con algoritmos CB y CF decidimos aplicar un método híbrido con el objetivo de encontrar un modelo que supere los resultados de nuestros algoritmos por separado.

Modelo 1 - Híbrido Monolítico

Los sistemas de recomendación híbridos monolíticos son aquellos que tienen dos o más recomendadores que funcionan de manera lineal. Es decir, el *output* de un recomendador es el *input* del otro recomendador.

Por lo tanto, debido al buen resultado de *Recall* en el algoritmo CB la primera aproximación en este trabajo fue utilizar un algoritmo CB que separe una gran cantidad de ítems según uno de los ítems mejor puntuados por el usuario en *train* para que luego se le aplique a cada ítem una predicción de *rating* mediante el mejor algoritmo de CF (SVD). Finalmente, el híbrido separa las N recomendaciones con mayor rating.

El *pipeline* de este híbrido realiza los siguientes pasos:

1) Tomar un usuario y un ítem como *input.*
2) Usar el algoritmo de content para obtener los 50 ítems a recomendar.
3) Se predice los *ratings* que los usuarios hubiesen asignado a cada uno de estos ítems usando un algoritmo de collaborative.
4) Se retorna las N primeras recomendaciones con los valores de *ratings* predichos más altos.

El objetivo es obtener N recomendaciones (5, 10, 20 y 50), medir *Precision*, *Recall* y F1. Con este último, vamos a medir y comparar con el F1 del CB. Se recuerda que hasta el momento el algoritmo CB es el que obtuvo los mejores resultados

Los resultados de *Precision*, *Recall* y F1 se muestran en la siguiente tabla:

n_recomendaciones	hybrid_recall	hybrid_precision	hybrid_F1
5	0.01332	0.00348	0.005518
10	0.02877	0.00376	0.006651
20	0.05568	0.00364	0.006833
50	0.14258	0.00373	0.007270

Tabla 24 Híbrido monolítico: Resultados de *Precision*, *Recall* y F1 (fuente: elaboración propia)

En la siguiente tabla se resumen el resultado del F1 para el modelo híbrido vs content:

n_recomendaciones	content_F1	hybrid(Model 1)_F1
5	0.014523	0.005518
10	0.012735	0.006651
20	0.010607	0.006833
50	0.007270	0.007270

Tabla 25 Comparación F1 Híbrido monolítico vs Content (fuente: elaboracion propia)

Los resultados indican que la adición de un algoritmo colaborativo a la solución impacta negativamente en los resultados del modelo híbrido respecto al CB. Por otro lado, a diferencia del resto de los modelos donde el F1 baja a medida que se ofrecen más recomendaciones, en este caso aumenta hasta que la cantidad de ítems a recomendar es 50, donde la recomendación es la misma para ambos modelos, lo cual es de esperar ya que la diferencia es el orden de los *items* recomendados en la lista.

Modelo 2 – Híbrido Monolítico (N recomendaciones con CB y orden de preferencias con CF)

Los resultados de la experimentación anterior nos hicieron comprender que la cantidad de ítems resultantes del motor de recomendación basado en CB es la mejor solución hasta el momento y que no aporta valor que los N elementos sean seleccionados por el algoritmo CF.

Luego de visualizar los resultados de F1 cuando la cantidad de ítems seleccionados son los mismos, se continuó la experimentación con la implantación de un motor híbrido donde el primer motor de CB devuelva las N recomendaciones y CF únicamente ordene los ítems según el *rating* predicho.

El *pipeline* de este híbrido realiza los siguientes pasos:

1. Tomar un usuario y un ítem como *input.*
2. Usar el algoritmo CB para obtener los N ítems más parecidos.
3. Sobre estos ítems se predicen los *ratings* que los usuarios hubiesen asignado a cada uno de estos ítems usando un algoritmo CF.
4. Se retornan las N recomendaciones ordenadas por los valores de *ratings* más altos.

Debido a que los ítems recomendados son los mismos entre el híbrido y CB, es que las métricas *Recall*, *Precision* y F1 no son de utilidad para resolver qué modelo elegir. La solución es la utilización de la métrica MAR@K explicado en la sección "Métricas para evaluación de resultados" para evaluar el ordenamiento de los productos recomendados.

El resultado de aplicar esta métrica fue el siguiente:

n_recomendaciones	MAR@K_hybrid(Model 2)	MAR@K_content
5	0.017874	0.022960
10	0.019077	0.026181
20	0.019239	0.028760
50	0.014002	0.030893

Tabla 26 Comparación MAR@K Híbrido monolitico vs Content (fuente: elaboración propia)

Como podemos observar el ordenamiento propuesto por el motor de recomendación de CB continúa siendo la mejor opción hasta el momento.

Modelo 3 – *Switch* de Híbridos

Una de las conclusiones extraídas del punto anterior es que puede no haber información suficiente para la utilización de un motor de recomendación híbrido que aplique CF para todos los usuarios y por lo tanto aplicarlo estaría quitando valor a la recomendación.

Esto tiene sentido ya que se debe recordar que estamos frente a *ratings* explícitos que son de usuarios que compraron y que luego de comprar decidieron agregar una *review*. Pudiendo tener pocos *ratings* por usuario por ítem lo que hace difícil la aplicación de CF.

Debido a lo anteriormente comentado una buena solución sería aplicar switch de híbridos donde dado un switch el motor deja de aplicar CB (nuestra mejor solución hasta el momento) y empieza aplicar un motor híbrido. Sería razonable que para aquellos usuarios que tengan información suficiente este switch aporte valor y el motor híbrido tenga mayor MAR@K.

Para evaluar a partir de donde tenemos información suficiente de los usuarios se comenzó por separar *train* en *train* y *validation*. Posteriormente se midió el RMSE entre el *rating* predicho y el verdadero para los usuarios agrupados por cantidad de transacciones en *train*. Por ejemplo, si un usuario tiene 4 transacciones su RMSE por ítem en *validation* se suma al promedio del grupo de usuarios con 1 a 5 transacciones.

Los resultados de este análisis fueron los siguientes:

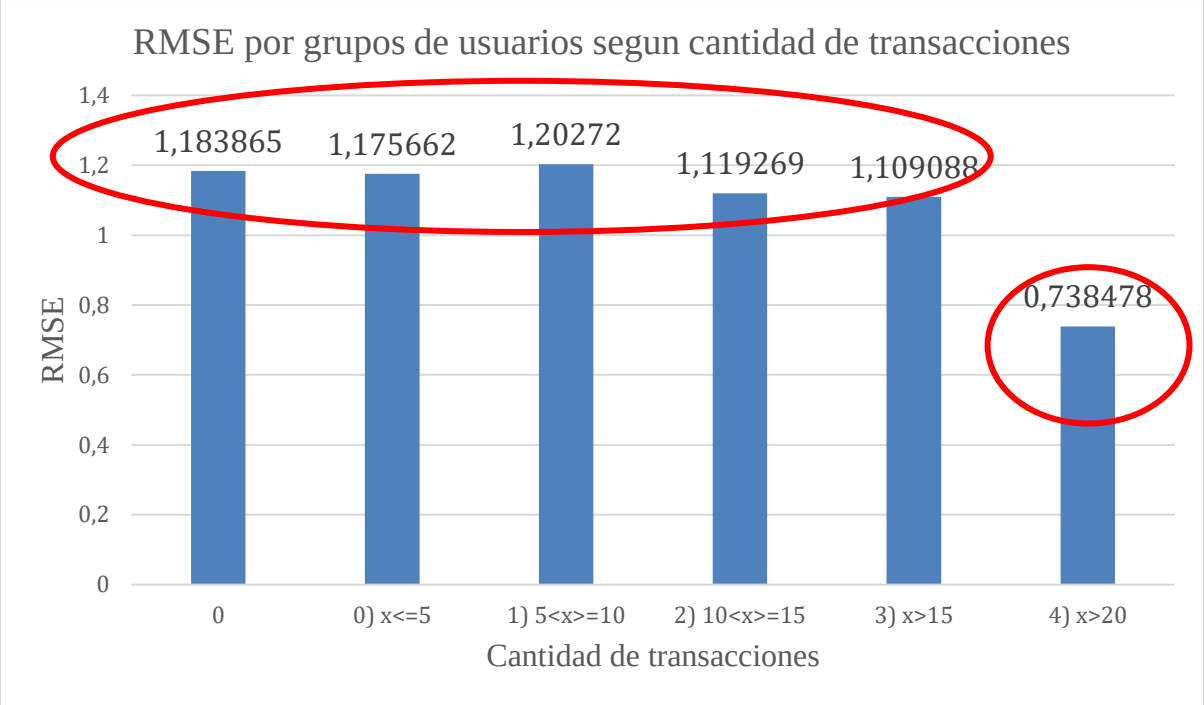


Ilustración 36 - RMSE por cantidad de transacciones (fuente: elaboración propia)

Los resultados demostraron una posible mejora en la utilización de un híbrido para los usuarios que tengan más de 20 transacciones en *train*. El problema que tiene esta conclusión es que solo aplicaría para el 0.55 % de los usuarios en *test*.

Para la prueba se adaptó el *pipeline* para agregar swtiching hybrid:

1. Tomar un usuario y un ítem como *input*
2. Usar el algoritmo CB para obtener los N ítems más parecidos
3. Se revisa si el usuario tiene más de 20 transacciones en *train*
    a. Si el usuario tiene más de 20 transacciones: se predice los *ratings* de cada uno de estos ítems usando un algoritmo CF y se ordena por los *ratings* predichos más altos.

b. Si el usuario no tiene más de 20 transacciones: el orden de las recomendaciones son las mismas que las ejecutadas por el algoritmo CB.

Los resultados de la métrica MAR@K son los siguientes:

n_recomendaciones	MAR@K_content	MAR@K_hybrid(Model 3)
5	0.022960	0.022962
10	0.026181	0.026183
20	0.028760	0.028762
50	0.030893	0.030894

Tabla 27 Comparación MAR@K Híbrido monolítico (Modelo 3) vs Content (fuente: elaboración propia)

Se puede visualizar una pequeña mejora en la métrica luego de aplicar híbrido a los usuarios con más de 20 transacciones. Lo que nos lleva a la conclusión de que dada la naturaleza del *dataset* se está sufriendo de poca información significativa para el armado de filtros colaborativos.

Aunque la mejora es poca decidimos quedarnos con este último modelo híbrido ya que propone un ordenamiento personalizado para usuarios que tengan información suficiente. A diferencia de La Quia donde el recambio de productos es muy frecuente debido a la estacionalidad, Rent the Runway mantiene el stock de ítems debido a que su principal modelo de negocio es alquiler de ropa, por este motivo es de esperar que a través del tiempo este modelo hibrido genere mejores resultados.

Modelo 4 – Ensambles de híbridos

Uno de los problemas detectados en los modelos anteriores es su linealidad, el algoritmo CF siempre se ve influenciado por la salida del algoritmo CB.

Por lo tanto, se realiza un sistema ortogonal donde se proyecta los scores normalizados de ambos modelos. Por un lado, el CB con *reviews* y el CF (SVD) para poder calcular la distancia euclídea, partiendo desde [0, 0] y así generar las predicciones. Esto permitirá tener en cuenta ambos algoritmos a la hora de la selección de la recomendación.

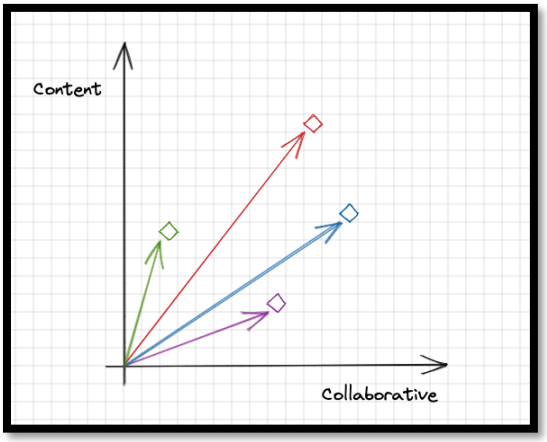


Ilustración 37 - Ejemplo de gráfico de ensamble de híbridos (fuente: elaboración propia)

Dicho ejercicio permite quitar el sesgo inicial que tiene el híbrido de utilizar CB en primera instancia, y luego CF para ordenar las recomendaciones. Finalmente se realiza la recomendación para 5 ítems (representan los de mayor distancia) y se obtienen los siguientes resultados. Destacamos que no se realizó para más recomendaciones, dado que con 5 recomendaciones no supera al Switch Hybrid:

Métrica	**Valor para 5 recomendaciones**
Precision	0.00135
Recall	0.00516
F1 Score	0.00214

Tabla 28 *Precision*, *Recall* y F1: Ensambles de híbridos (fuente: elaboracion propia)

Comparamos los resultados del MAR@K para 5 recomendaciones con el *Switching Hybrid*, el cual supera ampliamente:

Modelo	Cantidad de recomendaciones	MAR@K
Ensamble de híbridos (modelo 4)	5 recomendaciones	0.00277
Switching Hybrid (modelo 3)	5 recomendaciones	0.02296

Tabla 29 MAR@K Switching hybrid vs Ensamble de híbridos (fuente: elaboración propia)

Esta prueba final se plantea como un ejercicio exploratorio, sin embargo, creemos que con más información de los usuarios podría a llegar a preformar mejor.

6.2.4.4. Resultados Híbrido vs *Baseline*

En este apartado se muestran diversas métricas de evaluación aplicadas a motores de recomendación. Estas métricas pueden ser desde las tradicionales métricas de IA (*Precision*, *Recall* y F1) hasta métricas que miden las ventajas del algoritmo en relación con los objetivos empresariales o la experiencia del usuario. En ciertas situaciones estas últimas métricas pueden tener más importancia que el *trade off* entre *Precision* y *Recall*.

6.2.4.4.1. *Recall* vs *Precision* y F1

La definición de estas métricas y su importancia se realiza en el capítulo “Métricas para evaluación de resultados”.

Los resultados en comparación con el *baseline* son los siguientes:

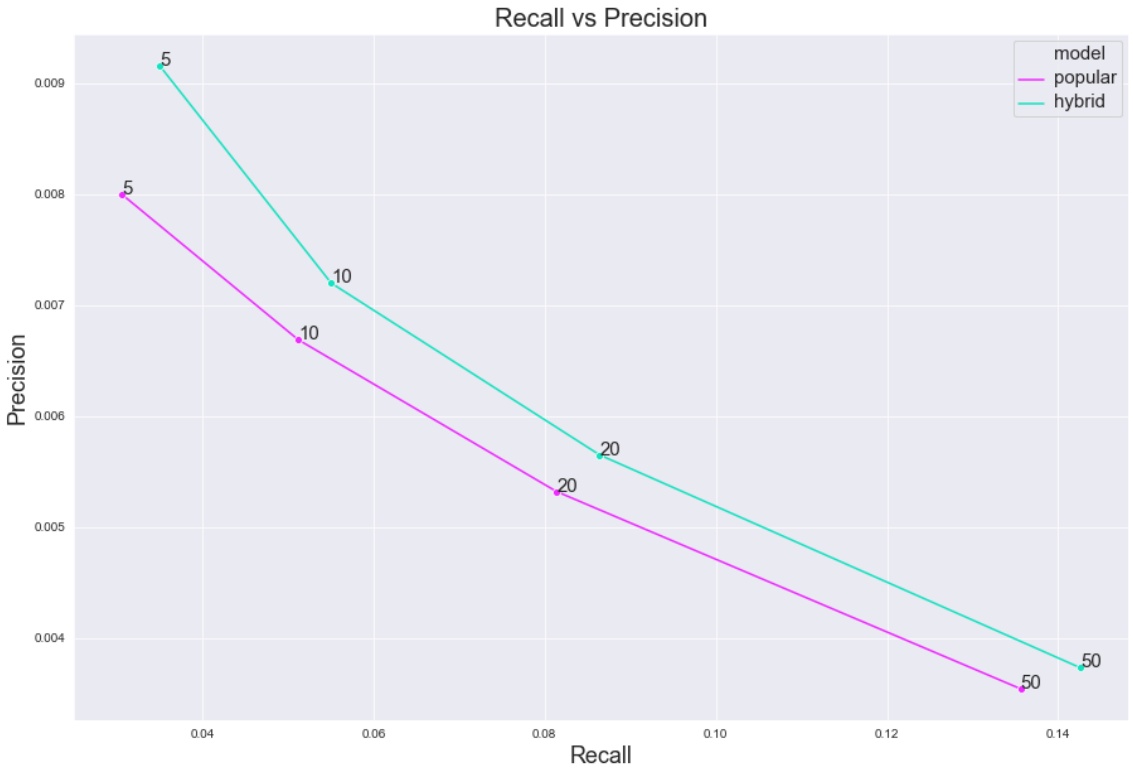


Ilustración 38 - *Recall* vs *Precision* del baseline vs híbrido (fuente: elaboración propia)

En la gráfica anterior se representan en cada punto la cantidad de recomendaciones por modelo evaluadas. Se puede observar como para todas las recomendaciones el modelo híbrido tiene mejores resultados que el baseline.

Como era de esperar *Recall* y *Precision* tienen comportamientos inversos al buscar su maximización. En las evaluaciones de motores de recomendación, algunos autores utilizan *Recall* como métrica, pero en nuestro caso decidimos tener en consideración *Precision* también debido a la necesidad de identificar la cantidad de recomendaciones que tenga mayores resultados y que no abrume al consumidor final. Debido a esto y lo explicado en su capítulo correspondiente es que a continuación se presenta la métrica F1.

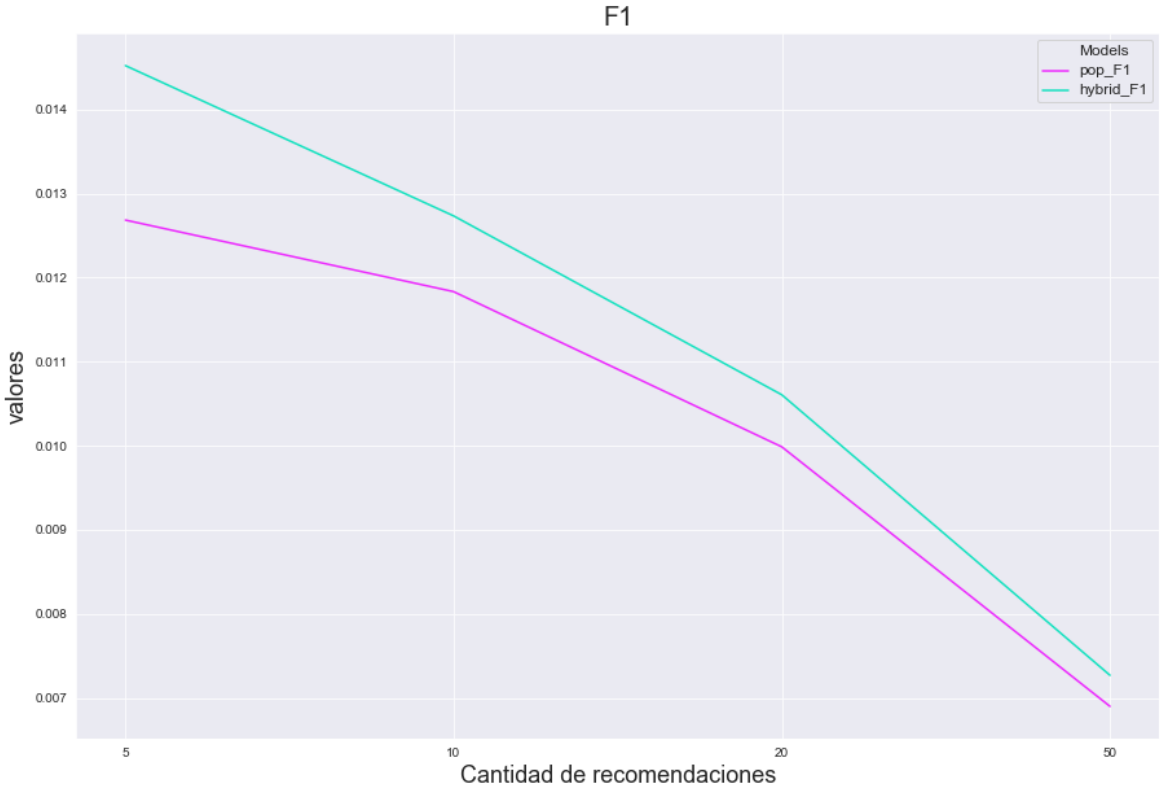


Ilustración 39 - F1 del baseline vs híbrido (fuente: elaboración propia)

n_recomendaciones	pop_F1	hybrid_F1	porcentaje_mejora
5	0.012685	0.014523	14.49
10	0.011833	0.012735	7.62
20	0.009987	0.010607	6.20
50	0.006900	0.007270	5.36

Tabla 30 Popular vs Híbrido (fuente: elaboración propia)

Como se muestra en los resultados anteriores para 5 recomendaciones la mejora es de aproximadamente un 15% respecto al baseline. Otra conclusión importante es que 5 recomendaciones se muestran como la cantidad óptima para recomendar a los consumidores finales.

En las siguientes subsecciones de este capítulo se mostrará como en otras métricas el recomendador propuesto supera ampliamente al *baseline* teniendo en cuenta objetivos de negocio y experiencia del usuario.

6.2.4.4.2. Prediction coverage

La finalidad es medir la cantidad de ítems distintos que se muestra en cada una de las predicciones. Esta métrica permite cuantificar la proporción de ítems que el sistema de recomendación utiliza para recomendar.

Esta métrica suele responder varias preguntas respecto al recomendador y al negocio:

¿Qué porcentaje de ítems aparecen en el top N de los usuarios? y en top N del motor de recomendación?

¿Cuántos artículos en nuestro catálogo hay que no se muestran a ningún usuario?

$$coverage = \frac{I}{N} \times 100\%$$

I = Cantidad de Ítems únicos que muestro a los usuarios de *test*.

N= Cantidad de ítems únicos en los datos de *train*.

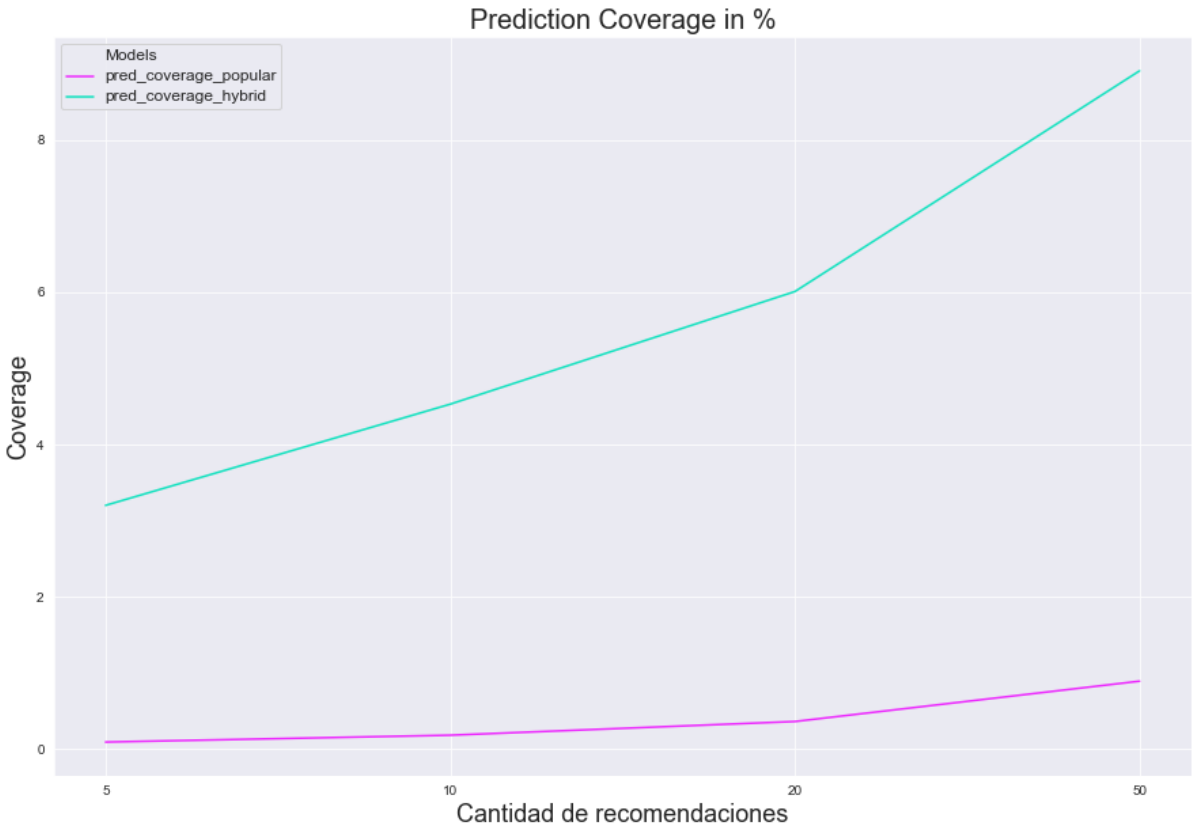


Ilustración 40 - *Coverage baseline* vs híbrido (fuente: elaboración propia)

n_recomendaciones	pred_coverage_popular	pred_coverage_hybrid	x_veces_mejora
5	0.09	3.20	35.56
10	0.18	4.53	25.17
20	0.36	6.01	16.69
50	0.89	8.91	10.01

Tabla 31 *Prediction coverage*: Híbrido vs Popular (fuente: elaboración propia)

Como se puede visualizar a medida que aumentamos la cantidad de recomendaciones el motor diseñado aumenta más que el *baseline*. En la tabla se puede ver como para 5 recomendaciones se muestra 35.56 veces más productos del catálogo que el *baseline*.

Esta métrica plantea un posible problema del motor de recomendación ya que se tiene 23.721 usuarios únicos a los que hacer 5 recomendaciones de productos, por lo que existen un total de 118.605 posibilidades de mostrar los 5.781 *items* de nuestro catálogo. En todas esas posibilidades vemos que para 5 recomendaciones mostramos un 3.2% del catálogo. Esto puede indicar diferentes acciones. Puede el motor de recomendación propuesto disponer de pocos datos sobre los cuales recomendar y por esto no utilizar la totalidad del catálogo disponible o puede indicarnos que las ventas de la empresa se basan en pocos productos, por lo que la necesidad de mantener stock sobre el resto de los ítems se vuelve innecesario y costoso. En nuestro caso optamos por considerar que puede tratarse de un problema relacionado con los datos dada la naturaleza del *dataset* de ser únicamente *ratings* explícitos, pero en una situación real podría ser necesarios cambios en la estrategia de la empresa.

Lo anterior puede plantear la necesidad de discutir con la empresa la preferencia de utilizar esta métrica como evaluador del recomendador. Debido al giro comercial de la empresa puede ser necesario mostrar más recomendaciones del catálogo ya que se trata de prendas rentadas y por ende únicas en su tipo. Por lo que esto podría dejar de lado la métrica propuesta para evaluación (F1) y adoptar esta métrica para selección de la solución y de la cantidad de recomendaciones. Adicionalmente cabe aclarar que estas cuestiones quedan abiertas ya que se tiene la desventaja de no disponer de pruebas A/B que nos permitan tomar esta decisión en función de más datos.

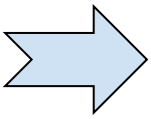
6.2.4.4.3. Personalization

Este indicador mide el grado de personalización en cada una de las recomendaciones a los usuarios.

Ejemplo de funcionamiento:

Usuario A: item X, item Y, item Z, item T

Usuario B: item X, item Y, item Z, item V



Indicador personalización: 25%

Por lo tanto, a mayor indicador las listas recomendadas a distintos usuarios son más personalizadas.

En el fondo el cálculo es estadístico y mira la superposición de listas de ítems recomendados a cada usuario, poniendo énfasis en cuán altamente correlacionadas están entre sí.

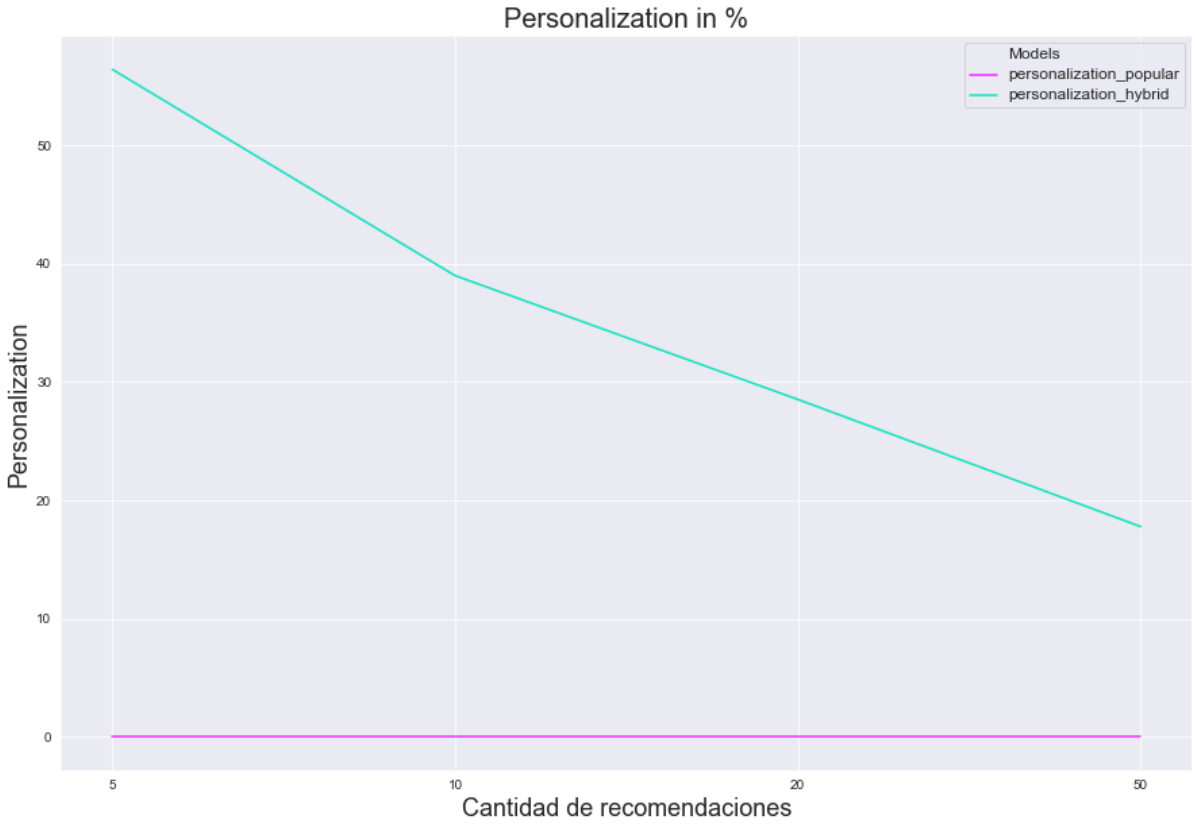


Ilustración 41 - *Personalization* del baseline vs híbrido (fuente: elaboración propia)

Debido a que nuestro *baseline* recomienda la misma lista para todos los usuarios el indicador se encuentra en cero, pero lo interesante es que para 5 recomendaciones nuestro modelo tiene un 56% de personalización. Este dato nos indica que más de la mitad de las recomendaciones son diferentes, lo que describe un excelente nivel de personalización de nuestro motor.

6.2.4.4.4. Novelty

Esta métrica mide la capacidad del sistema de recomendación para proponer elementos novedosos e inesperados que es poco probable que un usuario ya conozca. Para encontrarla mide la popularidad de los *items* recomendados, donde para definirla mide la cantidad de usuarios que consumieron ese ítem. Luego le aplica el inverso a la popularidad para encontrar la novedad.

Para la métrica utiliza la propia información del elemento recomendado y calcula la información media según la lista de N recomendaciones. Luego los promedia entre todos los usuarios.

Esta métrica busca resolver la siguiente pregunta: ¿Quiero recomendar productos populares o quiero recomendar artículos novedosos?

$$novelty = \frac{1}{|U|} \sum_{\forall u \in U} \sum_{\forall i \in topN} \frac{\log_2(\frac{count(i)}{|U|})}{|N|}$$

U = Cantidad de usuarios.

count(i) = Es la cantidad de usuarios que consumieron un ítem específico.

N = Cantidad de ítems recomendados.

Resultados de esta métrica:

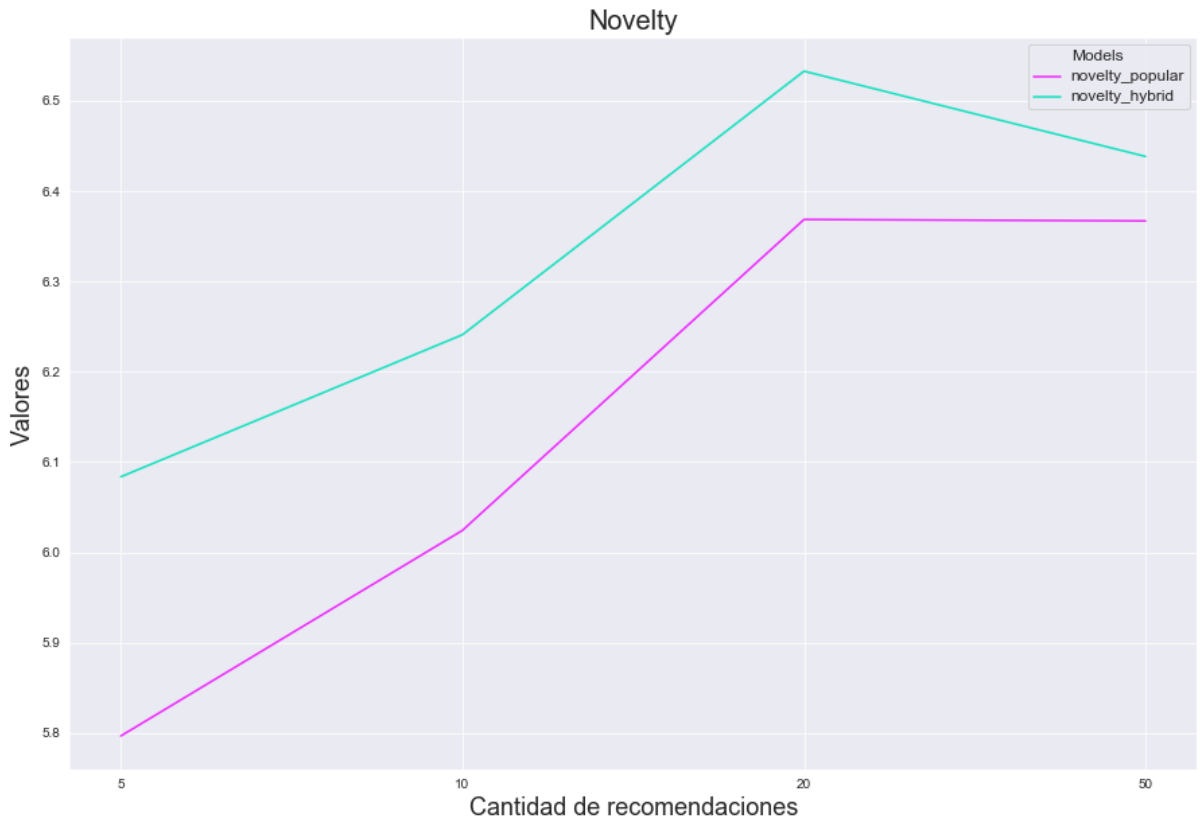


Ilustración 42 - *Novelty* del *baseline* vs híbrido (fuente: elaboración propia)

n_recomendaciones	novelty_popular	novelty_hybrid	porcentaje_mejora
5	5.796612	6.083591	4.95
10	6.024246	6.240895	3.60
20	6.368562	6.532790	2.58
50	6.366970	6.438332	1.12

Tabla 32 Novelty: Popular vs Híbrido (fuente: elaboración propia)

Se puede visualizar como nuestro recomendador supera al *baseline* siendo más novedoso sin importar cuántas recomendaciones se hagan. Podríamos intuir a la novedad como la inteligencia del recomendador en recomendar más variedad de catálogo que no es considerado como lo que se vende más. Por otro lado, se observa como nuestro recomendador híbrido comienza a bajar luego de las 20 recomendaciones, esto se puede deber a que comienza a recomendar ítems más populares debido a la falta de datos suficientes para un aprendizaje más amplio. Debido a que la cantidad de ítems recomendados según nuestro análisis es 5 y es donde se presenta mayor novedad, es que lo consideramos como un buen recomendador.

7. Arquitectura de la solución

Si bien el objetivo de esta tesis no es la realización de una solución sobre una arquitectura *Big Data*. A continuación, se desarrolla un marco arquitectónico de referencia sobre software libre para el soporte y utilización del motor de recomendación realizado.

Ilustración 43 - Arquitectura del motor de recomendación (fuente: elaboración propia)

La arquitectura presenta los siguientes componentes:

- Fuentes: En esta capa se representan las fuentes de datos. La fuente detectada es un archivo JSON con las *reviews* de compras de la tienda Rent the Runway.

- Capa de ingesta: Para esta capa se optó por Apache Kafka dado a que su utilización permite la ingesta en tiempo real, micro - *batch* y mega - *batch*. Apache Kafka es un sistema de almacenamiento publicador / subscriptor distribuido, particionado y replicado.

  Si bien para este proyecto la casuística plantea ingesta de datos *batch*, se debe tener una arquitectura preparada para soportar ingesta de datos en real time de los *e-commerce* asociados.

- Capa de almacenamiento:
    - Manejo del sistema: Se utilizará Apache Zookeeper para la administración de los estados y archivos de configuración del clúster.
    - Almacenamiento: La persistencia de la información es realizada con la utilización de HDFS de Apache Hadoop. El mismo es un sistema de archivos distribuidos.
    - Procesamiento: El procesamiento distribuido se realiza con Apache Spark. Su selección se debe a la rapidez con lo que se necesitan los resultados de las recomendaciones una vez ingresan las fuentes de datos.
      Para la aplicación de *machine learning* se utilizará la librería que Spark dispone para ello llamada MLlib.

      Un detalle a tener en cuenta para su utilización es que la solución está construida en python con la librería pandas. Pandas permite la manipulación y análisis de datos, pero el problema es que el mismo no corre de forma distribuida, por lo que para la utilización de Spark como procesamiento distribuido se debe realizar la correspondiente modificación al código.
    - Resultados en memoria: La tecnología utilizada es Redis. La misma es una base de datos No SQL clave - valor en memoria que es rápida y distribuida. Su uso permite mediante publicación y suscripción hacer disponible para cada usuario las recomendaciones a ser utilizadas.
- Capa de utilización:
    - API: Se prevé la utilización de un servicio API *rest* que permita consultar la base de datos en memoria y devolver en tiempo real las recomendaciones para cada usuario de la página web.
    - Consultas sobre los datos: Para realizar consultas sobre datos almacenados en HDFS se utilizará Apache Hive. El mismo permite hacer consultas con un lenguaje llamado *Hive Query Language* que es muy similar a SQL.

- *Reporting Batch*: Para el caso de *reporting batch* se optó por la utilización de Power BI por su facilidad para el armado de reportes dinámicos.

8. Conclusiones y lecciones aprendidas

Dentro de los dos principales grandes enfoques de motores de recomendación (CB y CF) se obtuvieron ampliamente mejores resultados con el enfoques de CB.

Por la naturaleza de los datos de Rent the Runway, existe poca cantidad de usuarios que tienen un número grande de transacciones históricas (solo el 0.55 % tienen más de 20 transacciones en el set de entrenamiento), era de esperar que los algoritmos basados en contenido tengan un mejor desempeño.

Debemos recordar que el set de datos utilizado contiene solo un conjunto muy específico de usuarios (aquellos que han comprado y han dejado una *review* y un *rating* explícito). Esto se traduce en una dificultad mayor para obtener relaciones de buena calidad entre ellos. Una opción de mejora puede ser incluir *raiting* implícito.

Al no tener algoritmos de CF con resultados relevantes, no fue sencilla la tarea de aplicar un modelo híbrido entre CB y CF que mejore los resultados originales obtenidos solo con CB. Dentro de los varios enfoques de híbrido utilizados, se logró mejorar apenas los resultados con el "Switch de hibridos", un modelo que aplica un enfoque híbrido o CB en función de la cantidad de transacciones históricas por usuario.

Como lecciones aprendidas, es importante contar con datos de ítems o productos únicos y debidamente caracterizados para la aplicación de los algoritmos de contenido y la búsqueda de relaciones entre ellos. Así mismo, la persistencia en los datos desde el punto de vista de la identificación de los usuarios, si no se tiene trazabilidad de usuarios, no se pueden buscar relaciones, y por ende, aplicar motores de recomendación CF. En nuestro caso, nos encontramos con estos problemas en el primer alcance de trabajo (con los datos de La Quia), lo que nos implicó realizar un cambio de rumbo durante el desarrollo de la tesis.

Como ultimó pero no menos importante, debemos destacar la utilización de la navaja de Ockham como elemento fundamental a la hora de elegir entre varios modelos. En la implementación de SVD quedo demostrado que luego de iterar en la búsqueda de parámetros el modelo más simple permite generalizar mejor.

9. Trabajo futuro

- Implementación de un motor de recomendación sobre una arquitectura *Big Data*.
- Profundización del método “redes neuronales” para motores de recomendación.
- Implementación de redes neuronales convolucionales para la extracción de características visuales de imágenes de prendas de ropa.
- Realizar *testing* A/B con una tienda de ropa para comprobar el funcionamiento del motor de recomendación.
- Investigación de grafos aplicados a motores de recomendación.

10. Repositorio de trabajo

En el siguiente enlace se deja acceso al repositorio en GitHub donde se encuentra el código de todas las implementaciones realizadas.

Link: GitHub Repositorio Sistemas de Recomendación

En [10] se encuentra el acceso a los datos de Rent the Runway.

11. Bibliografía

[1] - L. Ross, “The importance of cross selling and *E-commerce* Product Recommendations - Statistics and Trends [Infographic],” Invesp, 10-Jun-2019. [Online]. Available: https://www.invespcro.com/blog/*e-commerce*-product-recommendations/. [Accessed: 19-Sep-2021].

[2] - M. Blanco, “Amazon, los sistemas de recomendación y cómo disparar las ventas de tu *e-commerce*,” *Blog Smartup*, 22-Oct-2019. [Online]. Available: https://blog.smartup.es/sistemas-de-recomendacion-*e-commerce*/. [Accessed: 19-Sep-2021].

[3] - Ankita Prasad, “The Mathematics of Recommendation Systems” Medium, 10-Sep-2020. [Online]. Available: https://levelup.gitconnected.com/the-mathematics-of-recommendation-systems-e8922a50bdea

[4] - W. Hill, L. Stead, M. Rosenstein, and G. Furnas, “Recommending and evaluating choices in a virtual community of use,” Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '95, May 1995.

[5] - “9 MEDIDAS DE distancia En CIENCIA de datos,” ICHI.PRO, 02-Feb-2021. [Online]. Available: https://ichi.pro/es/9-medidas-de-distancia-en-ciencia-de-datos-159983401462266. [Accessed: 22-Sep-2021]

[6] - M. Vinceslas, “How funk singular value decomposition algorithm work in recommendation engines?,” Medium, 09-Jan-2019. [Online]. Available: https://medium.datadriveninvestor.com/how-funk-singular-value-decomposition-algorithm-work-in-recommendation-engines-36f2fbf62cac. [Accessed: 22-Sep-2021].

[7] - K. Falk, Practical recommender systems. Shelter Island, NY: Manning, 2019.

[8] - M. Farshad Bakhshandegan and E. Mehdi, “Cold start solutions for recommendation systems,” ResearchGate, May-2019. [Online]. Available: https://www.researchgate.net/publication/332511384_Cold_Start_Solutions_For_Recommendation_Systems. [Accessed: 19-Sep-2021]. Preprint.

[9] - “Monitor *e-commerce* Segunda Edición,” UES | Monitor *e-Commerce* Segunda Edición, 06-Jul-2019. [Online]. Available: https://www.laentregaperfecta.com.uy/post/monitor-*e-commerce*-segunda-edici%C3%B3n. [Accessed: 19-Sep-2021].

[10] - R. Misra, “Clothing fit dataset for size recommendation,” Kaggle, 21-Aug-2018. [Online]. Available: http://www.kaggle.com/rmisra/clothing-fit-dataset-for-size-recommendation. [Accessed: 19-Sep-2021].

[11] J. Olloniego and F. Mayr, “Clase 8 NLP,” en curso de Deep Learning, Oct-2021.

[12] - Sciforce, “Text preprocessing for NLP and *machine learning* tasks,” Medium, 05-May-2020. [Online]. Available: https://medium.com/sciforce/text-preprocessing-for-nlp-and-machine-learning-tasks-3e077aa4946e. [Accessed: 19-Sep-2021].

[13] - “TF - IDF for bigrams & Trigrams,” GeeksforGeeks, 27-Sep-2019. [Online]. Available: https://www.geeksforgeeks.org/tf-idf-for-bigrams-trigrams/. [Accessed: 19-Sep-2021].

[14] - R. Banik, Hands-on recommendation systems with python: Start building powerful and personalized, ... recommendation engines with python. Birmingham: PACKT Publishing Limited, 2018.

[15] - R. Moya García, “SVD aplicado a sistemas de recomendación basados en filtrado colaborativo,” tesis para Máster en Ciencias y Tecnologías de la Computación, Universidad Politécnica de Madrid, Madrid, España, 2013.

[16] - J. A. Konstan and M. D. Ekstrand, “Additional Item and List-Based Metrics,” in *Recommender Systems: Evaluation and Metrics*, [Online]. Available: https://es.coursera.org/lecture/recommender-metrics/additional-item-and-list-based-metrics-hgutJ. [Accessed: 19-Sep-2021].

[17] - P. Bonillo, “Propuesta de una arquitectura de gestión de grandes volúmenes de datos para la analítica en tiempo real bajo software libre,” LinkedIn, 17-Sep-2018. [Online]. Available: https://www.linkedin.com/pulse/propuesta-de-una-arquitectura-gesti%C3%B3n-grandes-datos-la-bonillo-ramos/?originalSubdomain=es. [Accessed: 19-Sep-2021].