1. **Sistemas de Recomendación**

Los sistemas de recomendación aplican técnicas estadísticas y de descubrimiento de conocimiento al problema de hacer recomendaciones de productos basadas en datos de uso previamente registrados. La creación de recomendaciones personalizadas generadas automáticamente para productos que incluyen libros, canciones, programas de televisión y películas mediante el filtrado colaborativo ha recorrido un largo camino desde que Information Lense, creo el primer sistema que utiliza el filtrado social ya hace más de 30 años (Malone, Grant, Turbak, Brobst y Cohen 1987). Hoy en día, los sistemas de recomendación son una tecnología exitosa utilizada por los líderes del mercado en varias industrias (por ejemplo, Amazon, Netflix y Pandora). En el comercio minorista, tales recomendaciones pueden mejorar las tasas de conversión al ayudar al cliente a encontrar los productos que desea comprar más rápido, promover la venta cruzada al sugerir productos adicionales y puede mejorar la lealtad del cliente mediante la creación de una relación de valor agregado (Schafer, Konstan y Riedl 2001).

La Tabla 1 proporciona enlaces a varias implementaciones populares de código abierto que proporcionan código que pueden usar los investigadores. El alcance de la funcionalidad (disponible actualmente), así como el uso objetivo de los paquetes de software disponibles, varían mucho y muchos proyectos se han abandonado a lo largo de los años.

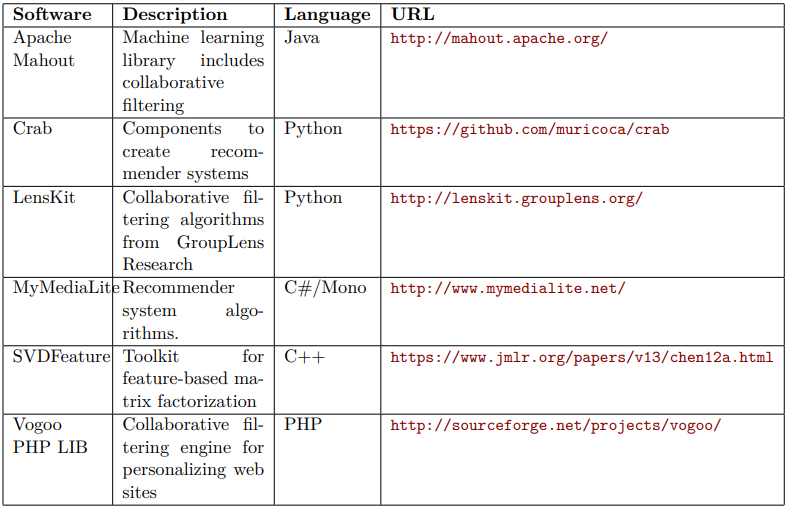


Tabla 1: Software del sistema de recomendación disponible gratuitos.

La mayoría del software disponible se centra en la creación de aplicaciones de recomendación para su implementación como un sistema de producción o implementan un método único como parte de un proyecto de investigación. El paquete de extensión de R **recommenderlab** utilizado para este proyecto fue diseñado para un propósito completamente diferente. Su objetivo es proporcionar una infraestructura de investigación integral para los sistemas de recomendación. La atención se centra en el manejo de datos consistente y eficiente, la fácil incorporación de algoritmos nuevos y existentes, la configuración de experimentos y la evaluación de los resultados. El lenguaje de programación de código abierto R, un entorno de software popular para la informática estadística y los científicos de datos (R Core Team 2018), se utiliza como plataforma, ya que permite al investigador implementar e integrar fácilmente algoritmos escritos en una amplia gama de lenguajes de programación. incluyendo R, Python, Java, C/C++ y ya proporciona todas las herramientas estadísticas necesarias, lo cual es importante para proporcionar un entorno de investigación útil.

Aunque se desarrolló para respaldar las propias necesidades de investigación y enseñanza del autor en 2010, se demostró que existía la necesidad de un software centrado en la investigación y el paquete **recommenderlab** resultó ser bastante popular. El paquete se utiliza en varios cursos universitarios para demostrar los conceptos básicos del desarrollo del sistema de recomendación.

Finalmente, varios investigadores emplearon el paquete para desarrollar y probar sus propios algoritmos (p. ej., Chen, Chao y Shah 2013; Buhl, Famulare, Glazier, Harris, McDowell, Waldrip, Barnes y Gerber 2016; Beel, Breitinger, Langer, Lommatzsch y Gipp 2016; Lombardi y Vernero 2017).

Los sistemas recomendadores se agrupan en 2 grandes grupos:

1. **Basados en Contenido**: toman en cuenta el contexto y características tanto de los usuarios como los ítems. Ej: modelos de clasificación para saber si un usuario elegirá o no una película en base a sus características, las características de la película y el comportamiento pasado del usuario.
2. **Filtros Colaborativos**: modelos basados exclusivamente en el comportamiento de los usuarios con respecto a los ítems. Ej: rating dado por parte de los usuarios a las películas. Se pueden utilizar los ratings para encontrar usuarios (o películas) similares para recomendar en base a la similitud.

El paquete **recommenderlab** se centra en el filtrado colaborativo que se basa en la idea de que, dados los datos de calificación de muchos usuarios para muchos elementos (por ejemplo, de 1 a 5 estrellas), uno puede predecir la calificación de un usuario para un elemento que no conoce (ver, por ejemplo, Goldberg, Nichols, Oki y Terry 1992) o crear para cada usuario las denominadas listas top-N de elementos recomendados (ver, por ejemplo, Sarwar, Karypis, Konstan y Riedl 2001; Deshpande y Karypis 2004). La premisa es que los usuarios que estuvieron de acuerdo con la calificación de algunos elementos generalmente también tienden a estar de acuerdo con la calificación de otros elementos. **Recommenderlab** proporciona implementaciones de muchos algoritmos populares, incluidos los siguientes.

* **El filtrado colaborativo basado en usuarios (UBCF)**: Encontrar usuarios similares y recomendar en base a los vecinos más cercanos.
* **El filtrado colaborativo basado en elementos** **(IBCF):** Encontrar similitud entre ítems y realizar recomendaciones en base a los items más similares a los preferidos por el usuario.
* **Singular Value Decomposition (SVD):** Los modelos de factores latentes usan la descomposición de valores singulares (SVD) para estimar las calificaciones faltantes usando métodos como SVD con imputación de media de columna, Funk SVD o mínimos cuadrados alternos.
* **El recomendador basado en reglas de asociación (AR):** usa reglas de asociación para encontrar elementos recomendados.
* **Aleternating last squares (ALS):** Factorización de matrices, utiliza el algoritmo alterating last squares.
* **Popular (POPULAR)**: Es un algoritmo no personalizado que recomienda a todos los usuarios los elementos más populares que aún no han calificado.
* **Aleatorio (RANDOM):** Crea recomendaciones aleatorias que se pueden utilizar como referencia para la evaluación del algoritmo de recomendación.
* **Re-recommended liked items (RERECOMMEND):** recomienda elementos que el usuario ha valorado altamente en el pasado. Estas recomendaciones pueden ser útiles para artículos que normalmente se consumen más de una vez (por ejemplo, escuchar canciones o comprar comestibles).
* **Recomendaciones híbridas (HybridRecommender):** agrupa las recomendaciones de varios algoritmos ponderándolas por un vector de pesos.

A continuación, se muestran a detalle algunos de estos algoritmos para el resto pueden encontrar información detallada en el libro de encuestas de Desrosiers y Karypis (2011).

1. **Filtros Colaborativos**

Para comprender el uso del software, son necesarias algunas definiciones formales. A menudo daremos ejemplos para un recomendador de películas, pero los ejemplos también se generalizan a otros tipos de artículos. Sea el conjunto de usuarios el conjunto de ítems. Las calificaciones se almacenan en una matriz usuarios-elementos, de tamaño , donde cada fila representa un usuario con y cada columna representa el item con . Usamos para denotar el vector de filas con las calificaciones del usuario . Las calificaciones usan una escala especifica.

Por ejemplo, Netflix usa de 1 a 5 estrellas. y las calificaciones estimadas pueden estar dentro de un intervalo de rango coincidente (p. ej., [1, 5]). Por lo general, solo se conoce una pequeña fracción de las calificaciones y para muchas celdas en , faltan los valores. Los valores faltantes representan películas que el usuario no calificó y que posiblemente tampoco haya visto todavía.

El filtrado colaborativo tiene como objetivo crear recomendaciones para un usuario llamado usuario activo . Definimos el conjunto de elementos desconocidos para el usuario como . Las dos tareas típicas son predecir las calificaciones de todos los elementos en o crear una lista que contenga los mejores N items recomendados de (es decir, una lista de recomendaciones de los N principales) para . Predecir todas las calificaciones que faltan significa completar la fila de la matriz de calificación donde los valores que faltan para los elementos en se reemplazan por calificaciones estimadas a partir de otros datos en . Desde este punto de vista, los sistemas de recomendación están relacionados con el problema de completar la matriz. La creación de una lista de los primeros N puede verse como un segundo paso después de predecir las calificaciones de todos los elementos desconocidos en y luego tomar los N elementos con las calificaciones predichas más altas. Algunos algoritmos omiten la predicción de calificaciones primero y pueden encontrar los N elementos principales directamente. Una lista de las mejores N-recomendaciones para un usuario es un conjunto parcialmente ordenado , donde y denota la cardinalidad del conjunto). Tenga en cuenta que puede haber casos donde la lista de los top-N principales contengan menos de N elementos. Esto puede pasar si o si el algoritmo CF no puede identificar N artículos para recomendar. La relación binaria es definida como si y solo si para todo . Además, se requiere que para garantizar que la lista N superior contenga solo los elementos con la calificación estimada más alta.

Por lo general, tratamos con una gran cantidad de elementos con calificaciones desconocidas, lo que hace que predecir primero los valores de calificación para todos ellos sea computacionalmente costoso. Algunos enfoques (p. ej., enfoques basados ​​en reglas) pueden predecir la lista de los primeros N directamente sin considerar primero todos los elementos desconocidos.

Los algoritmos de filtrado colaborativo generalmente se dividen en dos grupos, algoritmos CF basados ​​en memoria y CF basados ​​en modelos (Breese, Heckerman y Kadie 1998). Los CF basados ​​en memoria utilizan toda la base de datos de usuarios (o al menos una gran muestra de ella) para crear recomendaciones.

El algoritmo más destacado es el filtrado colaborativo basado en el usuario. La desventaja de este enfoque es la escalabilidad, ya que toda la base de datos de usuarios debe procesarse en línea para crear recomendaciones. Los algoritmos basados ​​en modelos usan la base de datos de usuarios para aprender un modelo más compacto (por ejemplo, grupos con usuarios de preferencias similares) que luego se usa para crear recomendaciones.

A continuación, presentaremos los conceptos básicos de la memoria conocida y los algoritmos de filtrado colaborativo basados ​​en modelos. Se puede encontrar más información sobre estos algoritmos en el reciente capítulo del libro de encuestas de Desrosiers y Karypis (2011).

**2.1 Usuarios-basado en Filtros Colaborativos**

La CF basada en el usuario (Goldberg et al. 1992; Resnick et al. 1994; Shardanand y Maes 1995) es un algoritmo basado en la memoria que intenta imitar el boca a boca analizando los datos de calificación de muchos individuos. La suposición es que los usuarios con preferencias similares calificarán los elementos de manera similar. Por lo tanto, las calificaciones que faltan para un usuario se pueden predecir encontrando primero un vecindario de usuarios similares y luego agregando las calificaciones de estos usuarios para formar una predicción.

La vecindad se define en términos de similitud entre usuarios, ya sea tomando un número dado de usuarios más similares (k vecinos más cercanos) o todos los usuarios dentro de un umbral de similitud dado. Las medidas de similitud populares para CF son el coeficiente de correlación de Pearson y la similitud del coseno. Estas medidas se definen entre dos vectores con valores x e y.

y

donde es el número de elementos en el vector de calificaciones y las calificaciones perdidas son saltadas para el cálculo. y son la desviación estándar y es la norma de un vector. Para calcular la medida entre dos usuarios, y , e representan los vectores de fila en con los vectores de perfil de los dos usuarios. Para calcular la similitud utilizando datos de calificación, solo se utilizan las dimensiones (elementos) que fueron calificados por ambos usuarios. Tenga en cuenta que el coseno y la correlación están en el rango [−1, 1], pero las medidas de similitud deben estar en el rango de [0, 1]. Nosotros, usamos la transformación

Usando la similitud, la vecindad para el usuario activo se puede seleccionar mediante un umbral en la similitud o tomando los k vecinos más cercanos. Una vez que se encuentran los usuarios en el vecindario, sus calificaciones se agregan para formar la calificación pronosticada para el usuario activo. La forma más fácil es simplemente promediar las calificaciones en el vecindario. Para el artículo esto es

En la Figura 1 se muestra un ejemplo del proceso de creación de recomendaciones por CF basado en el usuario.

A la izquierda se encuentra la matriz de calificación con 6 usuarios y 8 ítems y calificaciones en el rango de 1 a 5 (estrellas). Queremos crear recomendaciones para el usuario activo que se muestra en la parte inferior de la matriz. Para encontrar el k-vecindario (es decir, los k vecinos más cercanos), calculamos la similitud entre el usuario activo y todos los demás usuarios en función de sus calificaciones en la base de datos.

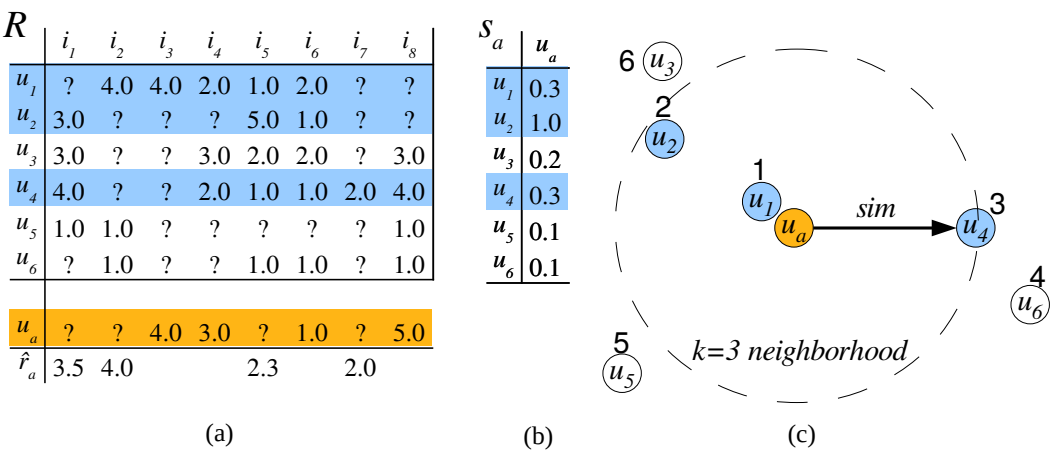


Figura 1: Ejemplo de filtrado colaborativo basado en usuarios con (a) matriz de calificación y calificaciones estimadas para el usuario activo, (b), similitudes entre el usuario activo y los otros usuarios (distancia euclidiana convertida en similitudes), y (b) la formación del vecindario de usuarios.

y luego seleccionar los usuarios con mayor similitud. A la derecha en la Figura 1, vemos una representación bidimensional de las similitudes (los usuarios con mayor similitud se muestran más cerca) con el usuario activo en el centro. Los vecinos más cercanos () se seleccionan y marcan en la base de datos a la izquierda. Para generar una calificación estimada agregada, calculamos las calificaciones promedio en el vecindario para cada elemento no calificado por el usuario activo. Para crear una lista de recomendaciones de los N principales, los elementos se ordenan por calificación prevista. En el pequeño ejemplo de la Figura 1, el orden en la lista de los primeros con es . Sin embargo, para una aplicación real, probablemente no recomendaríamos los elementos e debido a sus bajas calificaciones.

El hecho de que algunos usuarios en el vecindario sean más similares al usuario activo que otros (ver Figura 1 (b)) puede incorporarse como pesos en la Ecuación (3).

es la similaridad entre el usuario activo en el vecindario. Para los datos de calificación, el rendimiento del algoritmo de recomendación se puede mejorar eliminando el sesgo de calificación del usuario, donde algunos usuarios tienden a usar siempre calificaciones más altas, mientras que otros tienden a usar calificaciones más bajas. Esto se puede hacer normalizando los datos de calificación antes de aplicar el algoritmo de recomendación. Cualquier función de normalización puede ser usada para preprocesamiento. Dicha función debe ser reversible por para mapear la calificación predicha en la escala normalizada de regreso a la escala de calificación original. La normalización se usa para eliminar el sesgo de calificación individual por parte de los usuarios que constantemente siempre usan calificaciones más bajas o altas que otros usuarios. El método más popular es centrar las filas de la matriz de calificación de elementos de usuario por

donde es la media de todas las calificaciones disponibles en la fila del elemento-usuario en la matriz de calificaciones . Esto significa que las calificaciones ahora se miden para cada usuario por cuánto están por encima o por debajo de la calificación promedio del usuario. Lo contrario es simplemente,

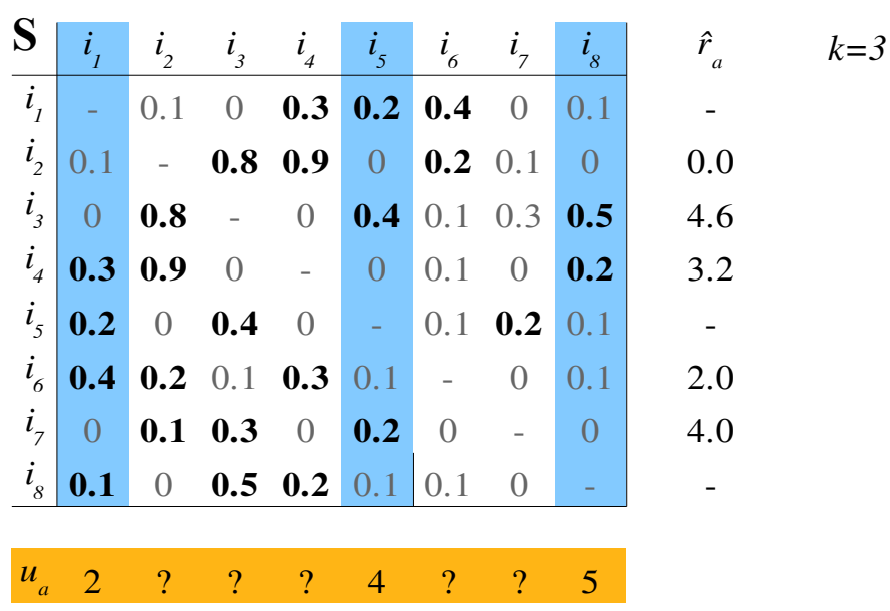


Figura 2: Filtrado colaborativo basado en elementos

En la literatura se pueden encontrar otros métodos, como la normalización del puntaje Z, que también tiene en cuenta la varianza de la calificación (ver, por ejemplo, Desrosiers y Karypis 2011). Los dos problemas principales de la FC basada en el usuario son que toda la base de datos del usuario debe mantenerse en la memoria y que debe realizarse un costoso cálculo de similitud entre el usuario activo y todos los demás usuarios en la base de datos.

**2.2 Elementos-basado en Filtros Colaborativos**

La FC basada en ítems (Kitts et al. 2000; Sarwar et al. 2001; Linden et al. 2003; Deshpande y Karypis 2004) es un enfoque basado en modelos que produce recomendaciones basadas en la relación entre los ítems inferidos de la matriz de calificación. La suposición detrás de este enfoque es que los usuarios preferirán artículos que sean similares a otros artículos que les gustan.

El paso de construcción del modelo consiste en calcular una matriz de similitud que contiene todas las similitudes entre elementos utilizando una medida de similitud determinada. De nuevo son populares la correlación de Pearson y la similitud del coseno y se utilizan las ecuaciones 1 y 2. Aquí, los vectores de calificación x e y son columnas de que representan las calificaciones de dos elementos.

Todas las similitudes por pares se almacenan en una matriz de similitud de tamaño . Para reducir el tamaño del modelo a con, para cada elemento solo se almacena una lista de los elementos más similares y sus valores de similitud. Los elementos k que son más similares al elemento il se denotan por el conjunto que puede verse como la vecindad del tamaño del elemento. Retener solo similitudes por elemento mejora significativamente la complejidad del espacio y el tiempo, pero potencialmente sacrifica algo de la calidad de la recomendación (Sarwar et al. 2001).

Para hacer una recomendación basada en el modelo, usamos las similitudes para calcular una suma ponderada de las calificaciones del usuario para elementos relacionados.

La Figura 2 muestra un ejemplo para n = 8 elementos con k = 3. Para la matriz de similitud , solo se almacenan las k = 3 entradas más grandes por fila (estas entradas están marcadas en negrita). Para el ejemplo, asumimos que tenemos calificaciones para el usuario activo para los elementos. Las filas correspondientes a estos elementos se resaltan en la matriz de similitud de elementos. Ahora podemos calcular la suma ponderada usando las similitudes (solo se usa la matriz reducida con las k = 3 calificaciones más altas) y las calificaciones de los usuarios. El resultado (debajo de la matriz) muestra que tiene la calificación estimada más alta para el usuario activo.

De manera similar a los algoritmos de recomendación basados en el usuario, el sesgo del usuario se puede reducir normalizando primero la matriz de calificación del elemento del usuario antes de calcular la matriz de similitud de elemento a elemento.

La FC basada en elementos es más eficiente que la FC basada en usuarios, ya que el modelo (matriz de similitud reducida) es relativamente pequeño () y se puede precalcular por completo. Se sabe que la FC basada en elementos solo produce resultados ligeramente inferiores en comparación con la FC basada en el usuario y son posibles modelos de orden superior que tienen en cuenta la distribución conjunta de conjuntos de elementos (Deshpande y Karypis 2004). Además, la FC basada en artículos se aplica con éxito en sistemas de recomendación a gran escala (por ejemplo, por Amazon.com).

**2.3 Usuarios y elementos-Basado en FC usando datos 0-1**

Hay menos investigación disponible para situaciones en las que no se dispone de una gran cantidad de datos de calificación detallados obtenidos directamente. Sin embargo, esta es una situación común y ocurre cuando los usuarios no quieren revelar directamente sus preferencias al calificar un elemento (por ejemplo, porque consume mucho tiempo). En este caso, las preferencias solo se pueden inferir analizando el comportamiento de uso. Por ejemplo, podemos registrar fácilmente en un supermercado qué artículos compra un cliente. Sin embargo, no sabemos por qué no se compraron otros productos. La razón puede ser una de las siguientes.

* El cliente no necesita el producto en este momento.
* El cliente no conoce el producto. Tal producto es un buen candidato por recomendación.
* Al cliente no le gusta el producto. Evidentemente, un producto de este tipo no debería ser recomendado.

Mild y Reutterer (2003) y Lee, Jun, Lee y Kim (2005) presentan y evalúan algoritmos de recomendación para esta configuración. El mismo razonamiento es válido para recomendar páginas de un sitio web dados los datos de flujo de clics. Aquí solo tenemos información sobre qué páginas se vieron, pero no por qué no se vieron algunas páginas. Esta situación conduce a datos binarios o más exactamente a datos 0-1 donde 1 significa que inferimos que el usuario tiene preferencia por un elemento y 0 significa que al usuario no le gusta el elemento o no lo conoce.

Pan, Zhou, Cao, Liu, Lukose, Scholz y Yang (2008) llaman a este tipo de datos en el contexto del filtrado colaborativo análogo a situaciones similares para clasificadores de datos de una clase, ya que solo la clase 1 es pura y contiene solo datos positivos. ejemplos La clase 0 es una mezcla de ejemplos positivos y negativos.

En el caso con definimos:

Dos estrategias para manejar los datos de una clase es asumir que todas las calificaciones que faltan (ceros) son ejemplos negativos o suponer que todas las calificaciones que faltan son desconocidas. Además, Pan et al. (2008) proponen estrategias que representan una compensación entre las dos estrategias extremas basadas en aproximaciones ponderadas de rango bajo de la matriz de calificación y en un muestreo de ejemplo negativo que podría mejorar los resultados en todos los algoritmos de recomendación.

Si asumimos que los usuarios suelen preferir solo una pequeña fracción de los elementos y, por lo tanto, la mayoría de los elementos sin calificación serán ejemplos negativos. entonces no tenemos valores perdidos y podemos usar los enfoques descritos anteriormente para datos de calificación de valor real. Sin embargo, si asumimos que todos los ceros son valores faltantes, entonces esto lleva al problema de que no podemos calcular las similitudes usando la correlación de Pearson o la similitud del coseno ya que las partes que no faltan de los vectores solo contienen unos. Una medida de similitud que solo se enfoca en hacer coincidir unos y, por lo tanto, evita el problema con los ceros es el *índice de Jaccard*:

donde e son los conjuntos de los ítems con 1 en los perfiles de usuario ua y ub, respectivamente.

El índice Jaccard se puede utilizar entre usuarios para el filtrado basado en usuarios y entre elementos para el filtrado basado en elementos como se describe anteriormente.

**2.4 Recomendaciones para Datos 0-1 Basado en Reglas de Asociación**

Los sistemas de recomendación que utilizan reglas de asociación producen recomendaciones basadas en un modelo de dependencia para elementos dados por un conjunto de reglas de asociación (Fu et al. 2000; Mobasher et al.2001; Geyer-Schulz et al. 2002; Lin et al. 2002; Demíriz 2004). La matriz de perfil binario se ve como una base de datos donde cada usuario es tratado como una transacción que contiene el subconjunto de elementos en I con una calificación de 1. Por lo tanto, la transacción se define como y toda la base de datos de transacciones es donde es el numero de usuarios. Para construir el modelo de dependencia, se extrae un conjunto de reglas de asociación sobre . Las reglas de asociación son de la forma donde y . Para el modelo solo usamos reglas de asociación con un solo elemento en el lado derecho de la regla. Para seleccionar un conjunto de reglas de asociación útiles, se utilizan umbrales sobre medidas de importancia e interés. Dos medidas ampliamente aplicadas son:

da el número de transacciones en la base de datos que contiene todos los elementos en . es el evento en el que el conjunto está contenido en una transacción.

Requerimos y y esto incluye las restricciones El conjunto de reglas que satisfacen estas restricciones forman el modelo de dependencia. Aunque encontrar todas las reglas de asociación dados los umbrales de apoyo y confianza es un problema difícil (el modelo crece exponencialmente en el peor de los casos con el número de elementos), están disponibles algoritmos que encuentran eficientemente todas las reglas en la mayoría de los casos (p. ej., Agrawal y Srikant 1994; Zaki 2000; Han, Pei, Yin y Mao 2004). También el tamaño del modelo se puede controlar mediante.

Para realizar una recomendación a un usuario activo dado el conjunto de elementos que le gustan al usuario y el conjunto de reglas de asociación (modelo de dependencia), son necesarios los siguientes pasos:

1. Encuentra todas las reglas coincidentes para las cuales en .
2. Recomendar N lados derechos únicos de las reglas de coincidencia con la confianza más alta (u otra medida de interés).

El modelo de dependencia es muy similar al FC basado en ítems con similitud basada en probabilidad condicional (Deshpande y Karypis 2004). Puede precalcularse completamente y reglas con más de un elemento en el lado izquierdo, incorpora efectos de orden superior entre más de dos elementos.

**2.5 Otros métodos de filtros colaborativos**

Con el tiempo, se han desarrollado varios otros enfoques basados en modelos. Un enfoque popular simple basado en elementos es el algoritmo Slope One (Lemire y Maclachlan 2005). Otra familia de algoritmos se basa en el enfoque de factores latentes mediante la descomposición de matrices (Koren et al. 2009). Más recientemente, el aprendizaje profundo se ha convertido en un método muy popular para la finalización flexible de matrices, la factorización de matrices y la clasificación colaborativa. Zhang, Yao, Sun y Tay (2019) presentan una encuesta exhaustiva. Estos algoritmos están fuera del alcance de este documento.

1. **Evaluación de los algoritmos de recomendación**

La evaluación de los sistemas de recomendación es un tema importante y Herlocker, Konstan, Terveen y Riedl (2004) y Gunawardana y Shani (2009) presentaron revisiones. Por lo general, dada una matriz de calificación , los algoritmos de recomendación se evalúan dividiendo primero los usuarios (filas) en en dos conjuntos . Las filas de correspondientes a los usuarios de entrenamiento se utilizan para entrenar el modelo de recomendación. Entonces cada usuario es visto como un usuario activo. Antes de crear recomendaciones, algunos elementos se retienen del perfil y mide qué tan bien es la calificación pronosticada coincide con el valor retenido o, para los algoritmos N principales, si los elementos en la lista recomendada tienen una calificación alta por parte del usuario. Finalmente, se promedian las medidas de evaluación calculadas para todos los usuarios de la prueba.

Para determinar cómo dividir en podemos usar varios enfoques (Kohavi 1995).

* **Splitting (División):** podemos asignar aleatoriamente una proporción predefinida de los usuarios al conjunto de entrenamiento y todos los demás al conjunto de prueba.
* **Muestreo Bootstrap:** podemos muestrear desde con reemplazo para crear el conjunto de entrenamiento y luego usar los usuarios que no están en el conjunto de entrenamiento como conjunto de prueba. Este procedimiento tiene la ventaja de que para conjuntos de datos más pequeños podemos crear conjuntos de entrenamiento más grandes y todavía quedan usuarios para probar.
* **Validación cruzada de k-capas:** Aquí dividimos en conjuntos (llamados capas) de aproximadamente el mismo tamaño. Luego evaluamos veces, siempre usando un pliegue para probar y todos los demás pliegues para inclinarse. Los resultados se pueden promediar. Este enfoque asegura que cada usuario esté al menos una vez en el conjunto de prueba y el promedio produce resultados más sólidos y estimaciones de error.

|  |  |  |
| --- | --- | --- |
| **actual / predicción** | **negativo** | **Positivo** |
| **negativo** | a | b |
| **positivo** | c | d |

Tabla 2: Matriz de confusión de 2x2

Los elementos retenidos en los datos de prueba se eligen al azar. Breese et al. (1998) introdujo los cuatro protocolos experimentales denominados *Dado 2*, *Dado 5*, *Dado 10* y *Todos menos 1*. Para los protocolos dados para cada usuario, se entregan elementos elegidos al azar al algoritmo de recomendación y los elementos restantes se retienen para su evaluación. Para Todos excepto , el algoritmo obtiene todos los elementos retenidos excepto .

A continuación, analizamos la evaluación de las calificaciones pronosticadas y luego de las listas de recomendaciones de los N principales.

**3.1 Evaluación de las calificaciones pronosticadas**

Una forma típica de evaluar una predicción es calcular la desviación de la predicción del valor real. Esta es la base para el *Error Promedio Medio (MAE)*

Donde es el conjunto de todas las parejas usuario-elemento para las que tenemos una calificación predicha y una calificación conocida que no se usó para aprender el modelo de recomendación.

Otra medida popular es el *error cuadrático medio (RMSE)*

RMSE penaliza los errores más grandes con más fuerza que MAE y, por lo tanto, es adecuado para situaciones en las que los pequeños errores de predicción no son muy importantes.

**3.2 Evaluación de las Top-N recomendaciones**

La lista de los top-N elementos pronosticados y los elementos retenidos que le gustan al usuario (generalmente determinados por un umbral simple en la calificación real) para todos los usuarios de la prueba se pueden agregar en una *matriz de confusión* que se muestra en la tabla 2 (ver Kohavi y Provost (1998)) que corresponde exactamente a los resultados de un experimento estadístico clásico. La matriz de confusión muestra cuántos de los elementos recomendados en las listas top-N (columna predicha positiva;) fueron elementos retenidos y, por lo tanto, recomendaciones correctas (celda ) y cuántos eran potencialmente incorrectos (celda ). La matriz también muestra cuántos de los elementos no recomendados (columna predicha negativa;) deberían haberse recomendado en realidad, ya que representan elementos retenidos (celda ).

De la matriz de confusión se pueden derivar varias medidas de desempeño. Para la tarea de minería de datos de un sistema de recomendación, el rendimiento de un algoritmo depende de su capacidad para aprender patrones significativos en el conjunto de datos. Las medidas de rendimiento utilizadas para evaluar estos algoritmos tienen su origen en el aprendizaje automático. Una medida comúnmente utilizada es la precisión, la fracción de recomendaciones correctas respecto al total de recomendaciones posibles.

Una medida de error común es el error absoluto medio (*MAE*, también llamado desviación absoluta media o *MAD*).

Donde es el numero total de elementos que se pueden recomendar y es el error absoluto de cada elemento. Dado que tratamos con datos , solo puede ser cero (en las celdas y en la matriz de confusión) o uno (en las celdas y ). Para los algoritmos de recomendación de evaluación para calificar datos, a menudo se usa el error cuadrático medio. Para datos 0-1 se reduce a la raíz cuadrada de MAE.

Los sistemas de recomendación ayudan a encontrar elementos de interés del conjunto de todos los elementos disponibles. Esto puede verse como una tarea de recuperación conocida como recuperación de información. Por lo tanto, las medidas de rendimiento de recuperación de información estándar se utilizan con frecuencia para evaluar el rendimiento del recomendador. La precisión y el recuerdo son las medidas más conocidas utilizadas en la recuperación de información (Salton y McGill 1983; van Rijsbergen 1979).

A menudo, se desconoce el número total de recomendaciones *útiles* necesarias para la el recall, ya que habría que inspeccionar toda la colección. Sin embargo, en lugar de las *recomendaciones* *útiles totales* reales, a menudo se utiliza el número total de recomendaciones útiles conocidas. La precisión y la recuperación son propiedades en conflicto, alta precisión significa poca recuperación y viceversa. Para encontrar un equilibrio óptimo entre la precisión y el recall, se puede utilizar una medida de valor único como la *medida-E* (van Rijsbergen 1979). El parámetro controla el trade-off entre la precisión y el recall.

Una medida popular de un solo valor es la medida-F. Se define como la media armónica entre la precisión y el recall.

Es un caso especial de la medida-E con que otorga el mismo peso tanto a la precisión como al recall. En la literatura de evaluación de recomendadores, la medida-F a menudo se denomina medida F1.

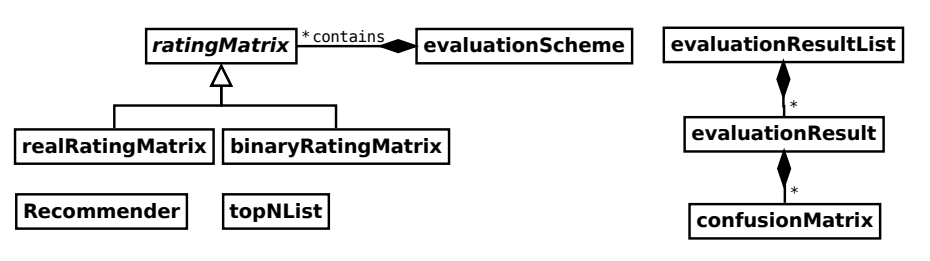


Figura 3: Diagrama de clases UML para el laboratorio de recomendación de paquetes (Fowler 2004).

Otro método utilizado en la literatura para comparar dos clasificadores con diferentes configuraciones de parámetros es la *curva característica operativa del receptor (ROC)*. El método fue desarrollado para la detección de señales y se remonta al modelo Swets (van Rijsbergen 1979). La curva ROC es un gráfico de la probabilidad de detección del sistema (también llamada sensibilidad o tasa de verdaderos positivos TPR, que es equivalente al recall como se define en la ecuación 13) por la probabilidad de falsa alarma (también llamada tasa de falsos positivos FPR o, donde ) con respecto a los parámetros del modelo. Una forma posible de comparar la eficiencia de dos sistemas es comparando el tamaño del área bajo la curva ROC, donde un área más grande indica un mejor rendimiento.

1. **Extracción de los datos de la API de Steam**

Para extraer la data desde Steam existen diferentes API’S algunas exigen el uso de la API key, tal y como se documenta en [Steam Web API Terms of Use](https://steamcommunity.com/dev/apiterms), paquetes como [SteamR](https://github.com/josegallegos07/steamR), tambien permiten extraer la información, utilizando la API Key.

Para este ejecicio se utilizara la **api steampowered** documentada en [Steam\_Web\_API](https://developer.valvesoftware.com/wiki/Steam_Web_API) y en [RJackson/StorefrontAPI](https://wiki.teamfortress.com/wiki/User:RJackson/StorefrontAPI" \l "appdetails).

Para este ejercicio de extraer toda la información se realizará en tres pasos

* **Paso 1.** Primero traer el id y nombre de cada elemento que se encuentra en Steam.
* **Paso 2.** Por cada id de cada elemento de Steam, se traer toda la información relevante y s e filtra solo por videojuegos.
* **Paso 3.** Tiendo cada id de los juegos que sean pagos se buscan las reseñas que cada usuario le asigno al juego y si lo recomendo o no.

**Nota:** La API de Steam solo permite 100.000 llamados por dí, que provienen de los [Términos y condiciones](https://steamcommunity.com/dev/apiterms) de la API de Steam; sin embargo, parece que Steam estima la tasa en períodos más cortos, limitando a alrededor de 200 llamadas por cada 5 minutos. Si se excede el numero de llamadas la API arroja el [**error 429**](https://www.rfc-editor.org/rfc/rfc6585#section-4), esto supone una limitante grande a la hora de extraer la información.

**4.1 Paso 1: Extracción de la información lista de juegos**

Primero se cargan los paquetes necesarios para hacer la extracción y limpieza de la información.

library(dplyr)

library(jsonlite)

library(xlsx)

library(lubridate)

library(data.table)

library(kableExtra)

library(crayon)

Utilizando la versión 2 del api steampowered se procede a traer el appid (identificador único del elemento de Steam) y name (nombre del videojuego) de cada elemento, filtro por aquellos elementos que tienen no tienen nombre, y verifico la cantidad de elementos traídos.

**Nota:** Cada día se agregan nuevos elementos alrededor de 50 a 200 elementos nuevos.

*## Lista de videojuegos*

lista\_juegos <- as.data.frame(fromJSON("http://api.steampowered.com/ISteamApps/GetAppList/v0002")$applist) %>% rename(appid = apps.appid, name = apps.name) %>% arrange(appid) %>% filter(name != "")

print(dim(lista\_juegos))

## [1] 149777 2

Como vemos hay un total de 149.777 juegos, e imprimimos los 10 primero objetos con las columnas appid y names:

|  |  |
| --- | --- |
| a**ppid** | **name** |
| 5 | Dedicated Server |
| 7 | Steam Client |
| 8 | winui2 |
| 10 | Counter-Strike |
| 20 | Team Fortress Classic |
| 30 | Day of Defeat |
| 40 | Deathmatch Classic |
| 50 | Half-Life: Opposing Force |
| 60 | Ricochet |
| 70 | Half-Life |

Tabla 3: Encabezado listado de juegos extraído de la Api de Steam.

Como se observa en la Tabla 3, en esta primera parte solo se trae el listado de los video juegos alojados en Steam, ya que la url <http://api.steampowered.com/ISteamApps/GetAppList/v0002>, solo ofrece un json con salida de estas dos columnas se procederá en el paso 2 a traer la información de cada juego.

**4.2 Paso 2: Obtener la información del videojuego**

Para extraer la información relevante de cada videojuego se accede a la segunda Api del tipo GET <http://store.steampowered.com/api/appdetails?appids=10&cc=us&l=en>, donde para este ejemplo se puso appids=10, es decir tráigame la información relevante del juego *Counter-Strike*. La Api además permite ingresar otros parámetros para hacer el request, estos son:

* cc (Opcional) Código del país para devolver los valores de moneda apropiados.
* l (Opcional) Indicar el idioma, por defecto toma el nombre del idioma en inglés.

Para este caso se ingresa cc=us y l=en, es decir tráigame el precio de los juegos en dólares americanos $USD y el lenguaje en inglés, estos sirven por ejemplo para el género del juego, que indicaría Action, fantasy en vez de acción y fantasía.

A continuación, imprimo los 5 primeros enlaces

*## Creo los enlaces*

links <- paste0("http://store.steampowered.com/api/appdetails?appids=",lista\_juegos$appid,"&cc=us&l=en")

head(links, 5)

## [1] "http://store.steampowered.com/api/appdetails?appids=5&cc=us&l=en"

## [2] "http://store.steampowered.com/api/appdetails?appids=7&cc=us&l=en"

## [3] "http://store.steampowered.com/api/appdetails?appids=8&cc=us&l=en"

## [4] "http://store.steampowered.com/api/appdetails?appids=10&cc=us&l=en"

## [5] "http://store.steampowered.com/api/appdetails?appids=20&cc=us&l=en"

Para la api  [GET http://store.steampowered.com/api/appdetails/](https://wiki.teamfortress.com/wiki/User:RJackson/StorefrontAPI#appdetails), la salida que arroja se encuentra documentada, en [User:RJackson/StorefrontAPI - Official TF2 Wiki | Official Team Fortress Wiki](https://wiki.teamfortress.com/wiki/User:RJackson/StorefrontAPI#appdetails)

*## Creo mi funcion para obtener la lista de juegos*

get\_game <- **function**(url){

df\_games <- tryCatch(df\_games <- fromJSON(url), error = **function**(mensaje) {

message("Se ha generado un error")

list(`1` = data.frame(success = FALSE))}

)

**if**(df\_games[[1]]$success){

res <- df\_games[[1]]$data

**if**(res$type != "game"){df = NULL} **else** {

df = data.frame(type = res$type, name = res$name, steam\_appid = res$steam\_appid, required\_age = as.integer(res$required\_age), is\_free = res$is\_free, supported\_languages = gsub(pattern = "<br>", replacement = " ",

x = gsub(pattern = "<strong>", replacement = "",x = gsub(pattern = "\\/", replacement = "", x = gsub(pattern = "\\\*", replacement = "", x = ifelse(is.null(res$supported\_languages), NA\_character\_, res$supported\_languages))))),

header\_image = res$header\_image, developers = paste0(res$developers, collapse = ";"), publisher = res$publisher,

price = ifelse(is.null(res$price\_overview$initial), NA\_real\_, res$price\_overview$initial) / 100,

metacritic\_score = ifelse(is.null(res$metacritic$score), NA\_real\_, res$metacritic$score),

categories = paste0(res$categories$description, collapse = ";"), genres = paste0(res$genres$description, collapse = ";"),

recommendations = ifelse(is.null(res$recommendations$total), NA\_integer\_, res$recommendations$total),

release\_date = lubridate::dmy(res$release\_date$date))

}

} **else** {df = NULL}

return(list(df = df, status = df\_games[[1]]$success))

}

**Nota:** tenga en cuenta que el api al tener restricción de llamadas por ciclo, se enviaran en lotes de 200 url’s y también solo se puede realizar peticiones cada 5 minutos, por lo que agregaremos la funcion Sys.sleep(210), que lo que hace es agregar un tiempo de espera de 210 segundos esto es 3M 30S, que sumado al tiempo que demora en hacer las 200 peticiones 1M 30S nos da en conjunto los 5 minutos aproximadamente.

Como tenemos en total 149.777 y debemos particionarlo en lotes de 200 se tiene la siguiente operación:

n <- ceiling(nrow(lista\_juegos) / 200)

print(n)

## [1] 749

my\_list <- split(1:nrow(lista\_juegos), rep(1:n), rep(200, n))

Para este ejercicio cambiaremos el valor de n solo para traer una muestra pequeña, ya que el proceso es bastante demorado, pondremos por defecto , es decir que me traiga las 5 primeras listas y cada una de ellas tiene 200 urls, lo que equivale a  traer 100 juegos, sin embargo, debemos descontar los request que fallen. Al final esta información se ira guardando en un data frame llamado df\_juegos.RDS.

**Nota:** No todos los appid tiene información, observe un ejemplo <http://store.steampowered.com/api/appdetails?appids=5&cc=us&l=en>, ya que estos id no corresponde todos a juegos sino también a dlc abreviatura de downloadable content (contenido descargable).

Ahora creamos un bucle que vaya iterando en cada una de las páginas, y como se observa en la siguiente salida, el primer llamado a la Api es exitoso, sin embargo el segundo falla ya que no es un videojuego sino el Steam Client como se observa en la Tabla 3.

n <- 5 *#Borrar esta linea*

df\_juegos\_total = NULL

contador <- 0

val <- system.time({

**for**(j **in** 1:n){

df\_juegos = NULL

**for**(i **in** my\_list[[j]]){

result <- get\_game(url = links[i])

df\_juegos <- bind\_rows(result$df, df\_juegos)

contador = contador + 1

print(paste("fila:",contador,"-","Estatus:", ifelse(result$status,"succes","fail"),"-","app id:", i))

}

cat(green("finaliza lista",j, "\n"))

df\_juegos\_total <- bind\_rows(df\_juegos\_total, df\_juegos)

saveRDS(object = df\_juegos\_total, file = "df\_juegos.RDS")

Sys.sleep(210)

beepr::beep(2)

}

beepr::beep(8)

})

## [1] "fila: 1 - Estatus: fail - app id: 1"

## [1] "fila: 2 - Estatus: succes - app id: 750"

## [1] "fila: 3 - Estatus: succes - app id: 1499"

## [1] "fila: 4 - Estatus: succes - app id: 2248"

## [1] "fila: 5 - Estatus: fail - app id: 2997"

El proceso es bastante demorado en tan solo traer la información de estos 1.000 juegos demora alrededor de 20 minutos.

lubridate::seconds\_to\_period(round(val[3]))

## [1] "21M 12S"

kbl(head(df\_juegos\_total, 10)) %>% kable\_styling(bootstrap\_options = c("striped","hoover"), full\_width = F)

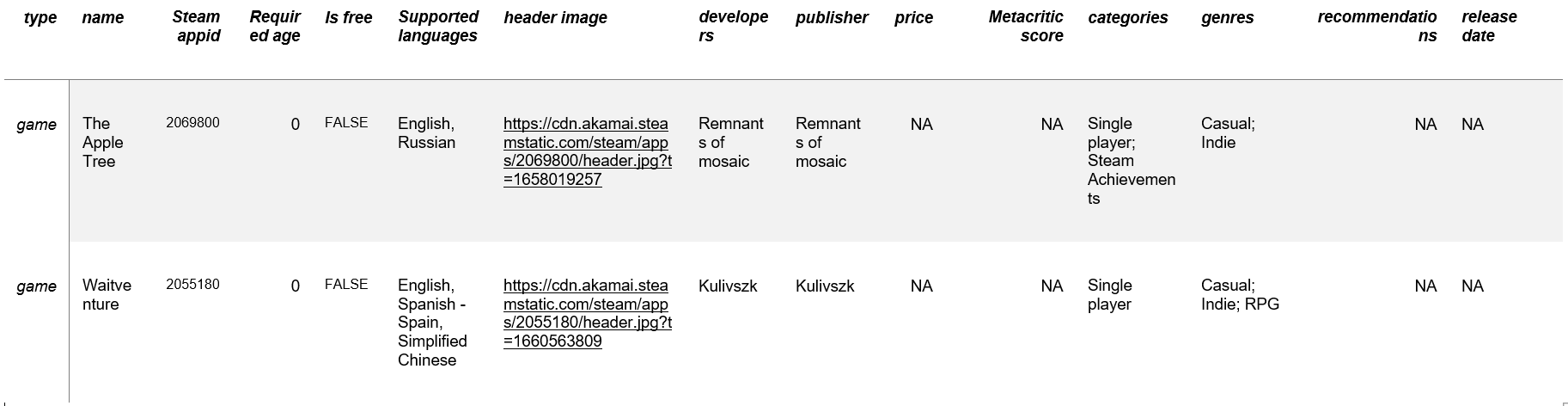
La salida de la tabla es la siguiente:

Tabla 4: Información de cada elemento traído de la Api de Steam.

**4.3 Paso 3: Obtener la información de los usuarios**

Limpiamos el entorno y cargamos la base de juegos que se descargó en el paso 2. Filtramos por los juegos que son pagos, removemos en caso de que haya algún duplicado, y se filtra por aquellos juegos que tengan como mínimo 100 calificaciones.

**Nota:** adicional mente podemos filtrar por aquellos juegos que sean con una fecha de lanzamiento en específico como por ejemplo la última década.

df\_juegos <- as.data.table(distinct(readRDS(file = "df\_juegos.RDS"))) *# Leo el archivo con los juegos*

df\_juegos <- df\_juegos[is\_free == FALSE & !is.na(recommendations)] *# Filtro por aquellos juegos pagos y que tengan recomendaciones*

df\_juegos <- df\_juegos[order(metacritic\_score), head(.SD, 1), by=steam\_appid] *# Retiro duplicados*

*df*\_juegos <- df\_juegos[release\_date >= '2012-01-01'] *# Filtro por juegos de la ultima decada*

df\_juegos <- df\_juegos[recommendations >= 100 & recommendations <= 5000] *# Filtro los juegos con numero de recomendaciones entre 100 y 5000*

Los parámetros de salida de este api se encuentran documentados en [User Reviews - Get List](https://partner.steamgames.com/doc/store/getreviews). Que es una lista con formato JSON de las reseñas que coinciden con los parámetros ingresados, que son:

Respuesta:

* **success:** - 1 si la consulta fue exitosa
* **query\_summary:** - Devuelto en la primera solicitud
  + **num\_reviews:** el número de revisiones devueltas en esta respuesta
  + **review\_score:** - La puntuación de la revisión
  + **review\_score\_desc:** - La descripción de la puntuación de la revisión
  + **total\_positive:** - Número total de críticas positivas
  + **total\_negative:** - Número total de críticas negativas
  + **total\_reviews:** número total de revisiones que coinciden con los parámetros de consulta.
* cursor: el valor para pasar a la siguiente solicitud como el cursor para recuperar el siguiente lote de revisiones
* reseñas
  + **recommendationid:** el ID único de la recomendación.
  + **autor**
    - **steamid:** - el SteamID del usuario
    - **num\_games\_owned:** - número de juegos propiedad del usuario
    - **num\_reviews:** - número de reseñas escritas por el usuario
    - **playtime\_forever:** - tiempo de reproducción de por vida rastreado en esta aplicación
    - **playtime\_last\_two\_weeks:** tiempo de reproducción registrado en las últimas dos semanas para esta aplicación
    - **playtime\_at\_review:** - tiempo de juego cuando se escribió la reseña
    - **last\_played:** - hora de la última vez que el usuario jugó
  + **language:** idioma que el usuario indicó al escribir la reseña
  + **review:** - texto de revisión escrita
  + **timestamp\_created:** - fecha en que se creó la reseña (marca de tiempo de Unix)
  + **timestamp\_updated:** - fecha en la que se actualizó la reseña por última vez (marca de tiempo de Unix)
  + **voted\_up:** - verdadero significa que fue una recomendación positiva
  + **votes\_up:** - el número de usuarios que encontraron útil esta reseña
  + **votes\_funny:** - el número de usuarios a los que les pareció graciosa esta reseña
  + **weighted\_vote\_score:** - puntuación de ayuda
  + **comment\_count:** - número de comentarios publicados en esta revisión
  + **steam\_purchase:** verdadero si el usuario compró el juego en Steam
  + **receive\_for\_free:** verdadero si el usuario marcó una casilla que dice que obtuvo la aplicación de forma gratuita
  + **written\_during\_early\_access:** verdadero si el usuario publicó esta reseña mientras el juego estaba en acceso anticipado
  + **developer\_response:** texto de la respuesta del desarrollador, si corresponde
  + **timestamp\_dev\_responded:** marca de tiempo de Unix de cuándo respondió el desarrollador, si corresponde

get\_users <- **function**(steam\_appid, conteo = 0){

df\_reseñas <- NULL

url <- paste0("https://store.steampowered.com/appreviews/",steam\_appid,"?json=1&filter=updated&language=all&purchase\_type=steam&num\_per\_page=100&cursor=\*")

result <- fromJSON(url)

conteo <- conteo + 1

print(paste("Request:",conteo))

**if**((conteo %% 200) == 0) Sys.sleep(210)

*#result$success == 1 & result$query\_summary$num\_reviews > 0*

df\_general <- data.frame(appid = steam\_appid,

review\_score = result$query\_summary$review\_score,

total\_positive = result$query\_summary$total\_positive,

total\_negative = result$query\_summary$total\_negative,

total\_reviews = result$query\_summary$total\_reviews)

reseñas <- as.data.table(result$reviews)

reseñas <- mutate(.data = reseñas,

author.last\_played = as\_datetime(author.last\_played),

timestamp\_created = as\_datetime(timestamp\_created),

timestamp\_updated = as\_datetime(timestamp\_updated),

appid = steam\_appid)

df\_reseñas <- bind\_rows(df\_reseñas, reseñas)

*#print(paste("Filas reseñas:",nrow(df\_reseñas)))*

cursor <- gsub(pattern = "[+]", replacement = "%2B", x = result$cursor)

n <- ifelse(result$query\_summary$total\_reviews %% 100 == 0, floor((result$query\_summary$total\_reviews-1) / 100), floor(result$query\_summary$total\_reviews / 100))

*# ====================================================================*

**for**(i **in** 1:n){

result <- fromJSON(paste0("https://store.steampowered.com/appreviews/",steam\_appid,"?json=1&filter=updated&language=all&&purchase\_type=steam&num\_per\_page=100&cursor=",cursor))

result$success

result$query\_summary$num\_reviews

conteo <- conteo + 1

print(paste("Request:",conteo))

**if**((conteo %% 200) == 0) Sys.sleep(210)

reseñas <- as.data.table(result$reviews)

reseñas <- mutate(.data = reseñas,

author.last\_played = as\_datetime(author.last\_played),

timestamp\_created = as\_datetime(timestamp\_created),

timestamp\_updated = as\_datetime(timestamp\_updated),

appid = steam\_appid)

df\_reseñas <- bind\_rows(df\_reseñas, reseñas)

*#print(paste0("Filas reseñas: ",nrow(df\_reseñas)))*

cursor <- gsub(pattern = "[+]", replacement = "%2B", x = result$cursor)

}

return(list(df\_general = df\_general, df\_reseñas = df\_reseñas, conteo = conteo))

}

Aplico un bucle para cada uno de los id de los juegos, y obtener la información de los usuarios junto con las recomendaciones que ha dado cada uno de estos.

lubridate::seconds\_to\_period(round(val2[3]))

## [1] "8M 58S"

kbl(head(df\_users\_total, 10)) %>% kable\_styling(bootstrap\_options = c("striped","hoover"), full\_width = F)

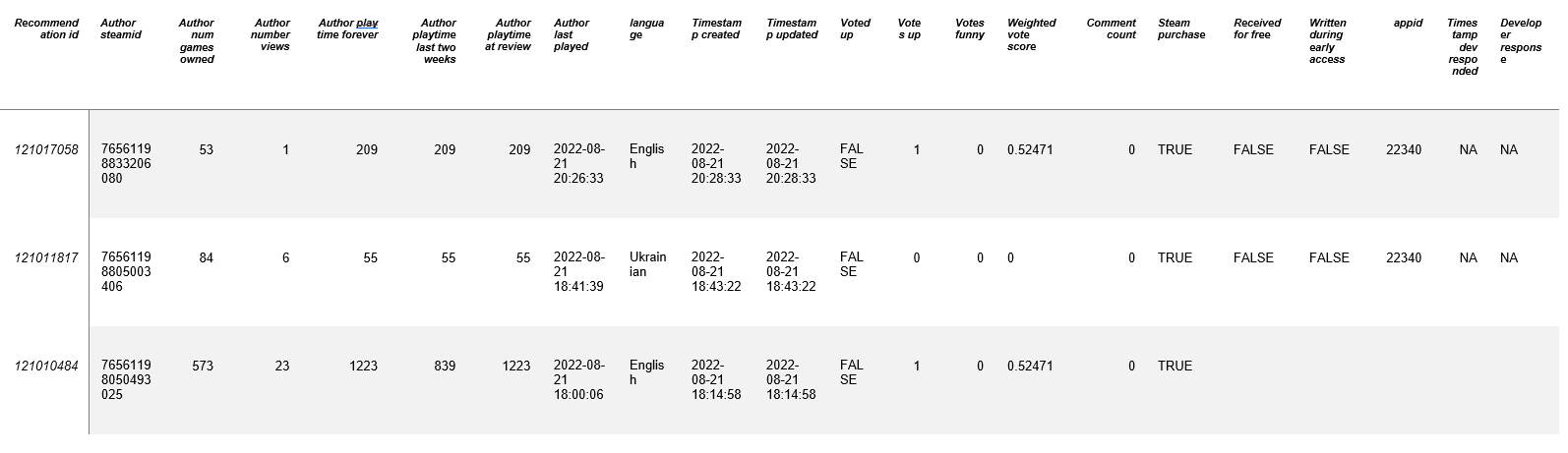


Tabla 5: Información de los usuarios y calificación de cada uno de los videojuegos.

1. **Aplicación del Sistema de Recomendación a los datos de Steam**

Después de extraer la información de los videojuegos y usuarios de la API de steam, se filtra por aquellos juegos de la última década es decir con fecha de lanzamiento de mínimo el año 2012 y que tengan entre 100 reseñas y un máximo de 1.000, esta restricción se puso debido a que la Api de Steam solo permite 100.000 llamadas por día o lotes 200 llamadas cada 5 minutos, que para juegos con 500 K reseñas se necesitaría varias horas o incluso días para extraer toda esta información.

Para la base de videojuegos se tiene un total de 3.776 juegos únicos, con la siguiente estructura.

|  | | | | | |
| --- | --- | --- | --- | --- | --- |
| appid | name | header\_image | developers | releasedate | genres |
| 238870 | Citadels | https://cdn.akamai.steamstatic.com/steam/apps/238870/header.jpg?t=1592555951 | Games Distillery s.r.o. | 2013-07-25 | Action;Strategy |
| 262190 | Zombeer | https://cdn.akamai.steamstatic.com/steam/apps/262190/header.jpg?t=1582878998 | Moonbite Games;PadaOne Games | 2015-01-30 | Action;Indie |
| 289360 | Konung 2 | https://cdn.akamai.steamstatic.com/steam/apps/289360/header.jpg?t=1656591573 | Fulqrum Publishing | 2014-04-23 | RPG |
| 290490 | The Flock | https://cdn.akamai.steamstatic.com/steam/apps/290490/header.jpg?t=1467753758 | Vogelsap | 2015-08-21 | Action;Indie |
| 217100 | Dementium II HD | https://cdn.akamai.steamstatic.com/steam/apps/217100/header.jpg?t=1474895574 | Memetic Games | 2013-12-17 | Action;Adventure;Indie |
| 692840 | Underworld Ascendant | https://cdn.akamai.steamstatic.com/steam/apps/692840/header.jpg?t=1623195185 | OtherSide Entertainment | 2018-11-15 | Action;Adventure;Indie;RPG |

Tabla 6: Base de los videojuegos.

Para la base de usuarios, aunque se tiene mucha más información traida de la API, para la metodología que se va a aplicar de filtros colaborativos solo se trae si voto positiva o negativamente cada uno de los juegos, por lo que las columnas solo son 3 appi (id del videojuego), el steamid (el id del usuario) y voted\_up (con TRUE y FALSE), con lo cual se aplicara un sistema de recomendación con respuesta binaria. Se tiene un total de 1.267.798 usuarios, sin embargo, solo nos quedaremos con aquellos usuarios que hayan realizado al menos 10 reseñas, ya que así lo exige el algoritmo que se va aplicar.

| appid | author.steamid | voted\_up |
| --- | --- | --- |
| 1313 | 76561197960914766 | TRUE |
| 1313 | 76561197962062015 | TRUE |
| 1313 | 76561197962462916 | FALSE |
| 1313 | 76561197962750090 | TRUE |
| 1313 | 76561197963002630 | TRUE |
| 1313 | 76561197963885198 | TRUE |

Tabla 7: Base de los usuarios.

Las dos bases se fusionan haciendo un joint, para obtener una base final que contiene el total de la información, juegos y usuarios. A continuación, se hace un pequeño análisis descriptivo de algunas de las variables de interés.

Gráfico, Gráfico de barras

Descripción generada automáticamente

Figura 4: Numero de votos positivos y negativos de los juegos.

Como se observa en la figura 4, el 77% de las calificaciones del listado de juegos son positivas, mientras que el 22% son negativas es decir no recomiendan el videojuego.

En la figura 5, se realizan dos histogramas de ratings por usuario a la izquierda, indica el numero de calificaciones que ha hecho cada usuario, como vemos la mayoría se concentra en 5 calificaciones o menos. Para encontrar recomendaciones con sentido será necesario quedarnos con los usuarios de 10 calificaciones o más. Por otra parte, el histograma de la derecha muestra que para las películas todas recibieron al menos 100 reseñas y un máximo de 1000, concentrándose la mayoría entre 100 y 400 calificaciones. Por lo que no es necesario descartar ningún video juego.

Gráfico, Histograma

Descripción generada automáticamente Gráfico, Histograma

Descripción generada automáticamente

Figura 5: Número de ratings por usuario y por juego.

A continuacion se muestra el top 5 videojuegos con mejor calificacion, en primer lugar esta Little Mouse’s Encyclpedia con un 99% de votos positivos, mientras que el peor valorado es Race To Mars con solu un 6% de votos positivos.



Figura 6: Top 5 de los videojuegos con mejor rating.

En cuanto a los juegos por el género, se tiene que el 26% de ellos son del tipo indie, seguido de acción con un 16% y aventura con un 15%. Mientras que Photo Editing, Game Development y Software Training. Son de los que menos hay representando menos del 1%, del total.

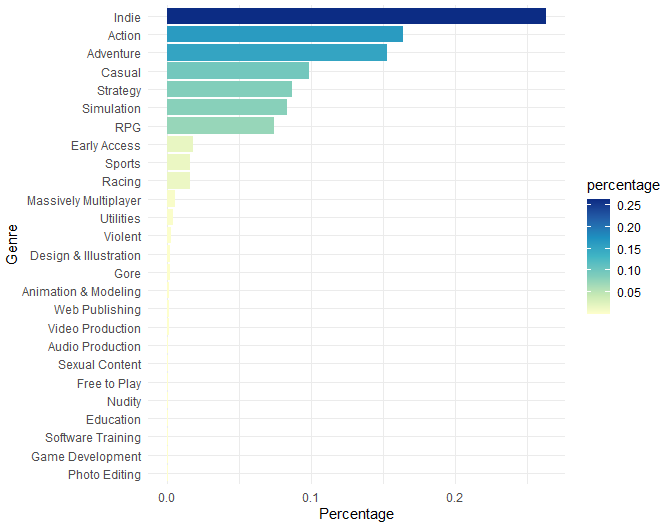


Figura 7: Porcentaje de juegos por género.

Para el ajustar el modelo se utilizó el paquete de R **recommenderlab**, cuyo repositorio es <https://github.com/mhahsler/recommenderlab> . además para aplicar cualquier algoritmo es necesario convertir los datos en una matriz dispersa, al aplicarla a nuestros datos se obtuvo el objeto binaryRatingMatrix del siguiente tamaño:

4377 (usuarios) x 3776 (peliculas) rating matrix of class ‘binaryRatingMatrix’ with 91905 ratings.

Además, para el esquema de entrenamiento, se utilizó la función evaluationScheme del paquete recommenderlab, con el parámetro given 10, que restringe a que los usuarios hayan calificado mínimo 10 video juegos, se particiono el conjunto de datos en dos partes, entrenamiento un 80% y test 20%, además de aplicar validación cruzada de 5 capas para evitar el sobreajuste del modelo.

Se aplicaron 7 algoritmos diferentes estos son: aleatorio como punto de partida para comparar las recomendaciones generadas por los demás algoritmos, popular para recomendar a los usuarios los juegos mas populares que no han sido calificados por los usuarios, el altering las square, reglas de asociación con un soporte = 0.001, confianza = 0.2, maxlen = 2. Para filtro colaborativo basado en usuarios y filtro colaborativo basado en elementos, se utilizó el índice de Jaccard de la ecuación 7 como medida de similitud. No fue posible aplicar el algoritmo de descomposición en valores singulares, pues esta metodología no esta implementada para calificaciones binarias.

En la figura 8, se muestra el resultado obtenido después de aplicar cada uno de los algoritmos, donde el n indica el numero de recomendaciones que se darían, es decir 5 o 10 videojuegos. como se esperaba el aleatorio es el de peor rendimiento en cuanto a precisión y recall, con valores del 0.2% y 0.1%. Mientras que reglas de asociación es de mejor performance con un 14% de precisión (para 5 juegos) y hasta un 12% de recall si se recomendaran 10 juegos. En cuanto a los algoritmos de recomendación por filtros colaborativos de usuarios y elementos, el rendimiento es parecido, un porco mejor el filtrado por ítems, lo que es de esperarse pues los videojuegos tienen todos al menos 100 reseñas, mientras que los usuarios un mínimo de 10. El tiempo de ejecución para estos 7 modelos fue de 1 Hora y 34 minutos.

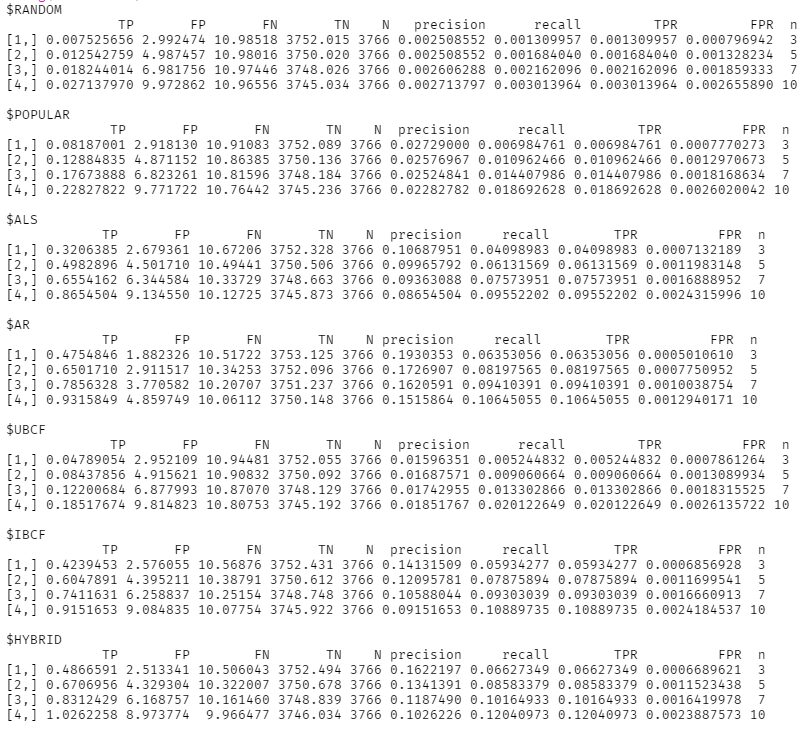


Figura 8: Resultado al aplicar los diferentes algoritmos de recomendación.

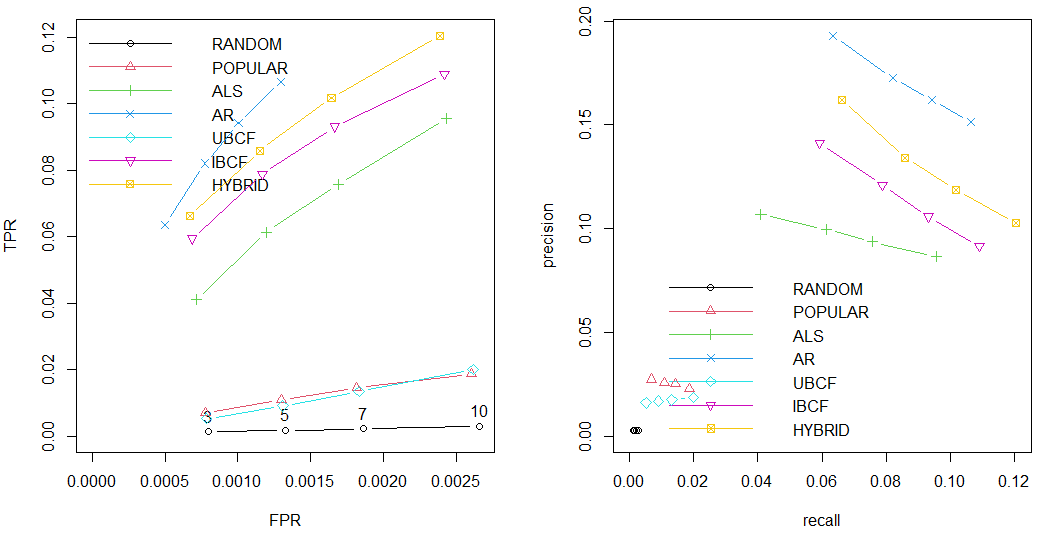


Figura 8: Curva ROC (izquierda) y precisión vs recall (derecha).

Para mejorar el performance del modelo, se utilizará como modelo definitivo y final un esquema de recomendación hibrido. Este sistema utiliza un concepto similar al stacked de machine learning, donde se pueden fusionar varios modelos débiles para obtener uno mas robusto. Para este se utilizaron, los 4 que mejor funcionaron asignado a cada uno pesos diferentes según los resultados obtenidos en la figura 8, popular (5%), reglas de asociación (60%), filtros colaborativos por usuario (10%) y filtros colaborativos por elementos (25%), donde el peso de los modelos debe sumar el 100%. El modelo hibrido tarda cerca de 6 minutos en entrenarse, con 3500 usuarios que compone la data de entrenamiento. Después de entrenarse se guarda en un binario para su posterior despliegue.

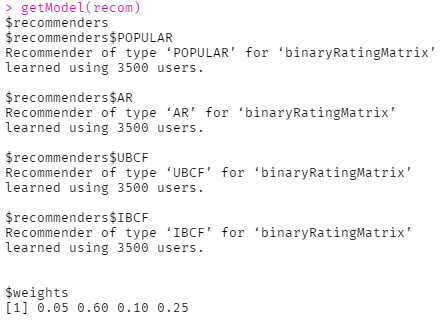


Figura 9: Modelo hibrido con sus respectivos pesos.

Predicción de ejemplo se utiliza al usuario con id 76561197960284889, que ha calificado 17 todos positivamente, para este usuario se recomiendan los siguientes 5 juego utilizando el modelo hibrido.

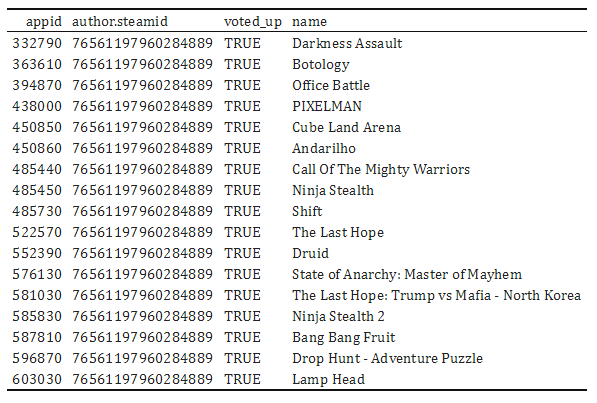


Tabla 8. Calificaciones dadas por el usuario con id 76561197960284889 a los video juegos.

Para este usuario las recomendaciones son:

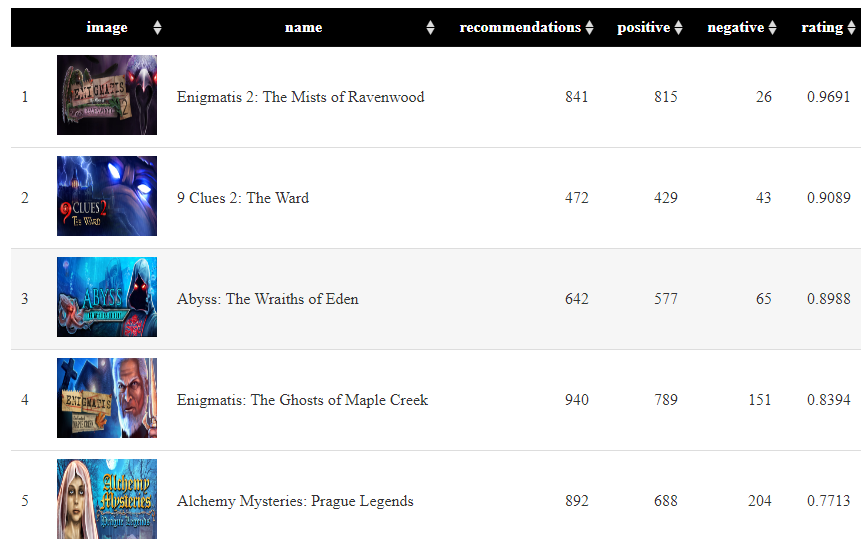


Figura 10: Top 5 recomendaciones para el usuario, con id 76561197960284889.

1. **Despliegue modelo de recomendación**

Para desplegar el modelo de recomendación y ser utilizado en tiempo real se puede desplegar como una Api Restful y utilizar diferentes servicios en la nube, como Google Cloud Platform, Amazon Web Service o Azure. Por facilidad se utilizará r Shiny, ya que su interfaz es amigable y permite al usuario interactuar con los diferentes componentes. Se puede desplegar gratuitamente en [shinyapps.io](https://www.shinyapps.io/), por lo que se hace en este servicio en el siguiente enlace

<https://rafael-eduardo-diaz.shinyapps.io/recommender_systems/>

1. **Conclusiones**

En este documento, se describió la teoría de los principales sistemas de recomendación que utilizan filtros colaborativos, además se mostró como extraer la información de Steam desde las diferentes Api’s públicas que están disponibles en la web, el software utilizado fue R y el paquete **recomenderlab**, que está especialmente orientado al desarrollo y prueba de algoritmos de recomendación. El paquete permite crear esquemas de evaluación siguiendo métodos aceptados y luego usarlos para evaluar y comparar algoritmos de recomendación.

Se realizo una aplicación a los datos de Steam utilizando varios de los algoritmos estándar y se compararon entre si, donde en una primera fase, el algoritmo reglas de asociación fue el que mostro mejor performance, para finalmente utilizar un algoritmo hibrido, que es la fusión de Popular AR, UBCF e ICBF, finalmente el modelo se desplegó utilizando el framework r shiny.

1. **Referencias**

Hahsler, M. (2022). recommenderlab: An R Framework for Developing and Testing Recommendation Algorithms. *arXiv preprint arXiv:2205.12371*.

1. **Anexos**

Se adjunta el código utilizado, para realizar los modelos de recomendación y el análisis descriptivo.

## Cargo mis paquetes

library(data.table)

library(dplyr)

library(feather)

library(stringr)

library(ggplot2)

library(DT)

library(recommenderlab)

library(kableExtra)

## Cargo las bases de datos de juegos

df\_juegos <- as.data.table(read\_feather(path = "data/df\_juegos.feather"))

df\_juegos <- df\_juegos[,.(appid = steam\_appid, name, header\_image, developers, release\_date, genres)]

head(df\_juegos) %>%

kbl(caption = "Base 1. Información de los videojuegos") %>%

kable\_classic(full\_width = F, html\_font = "Cambria")

## Cargo la base de datos de usuarios

df\_users <- as.data.table(read\_feather(path = "data/df\_users.feather"))

df\_users <- df\_users[order(appid, author.steamid),.(appid, author.steamid, voted\_up)]

head(df\_users) %>%

kbl(caption = "Base 2. Información de los usuarios") %>%

kable\_classic(full\_width = F, html\_font = "Cambria")

## Uno ambos data

df\_total <- left\_join(x = df\_users, y = df\_juegos, by = "appid")

rm(df\_users, df\_juegos); invisible(gc())

length(unique(df\_total$author.steamid)) #817226 usuarios

df\_total <- left\_join(x = df\_total,

y = df\_total[,.(recommendations = .N, positive = sum(voted\_up), negative = sum(voted\_up == FALSE), rating = round(sum(voted\_up) / .N, 4)), by = appid],

by = "appid")

dumies <- fastDummies::dummy\_columns(.data = distinct(df\_total[,.(appid, genres)])[,2], select\_columns = "genres", split = ";")[,-1]

colnames(dumies) <- gsub(pattern = "genres\_", replacement = "", x = colnames(dumies))

dumies <- bind\_cols(distinct(df\_total[,.(appid)]), dumies)

head(dumies,3)

res <- as.data.frame(str\_split\_fixed(string = df\_total$genres, pattern = ";", n = 9)); df\_total$genres <- NULL

df\_total <- left\_join(x = df\_total, y = dumies, by = "appid")

## Algunos descriptivos

df\_total %>% ggplot(aes(x = voted\_up, fill = factor(voted\_up))) +

geom\_bar(color = "grey20") + scale\_fill\_brewer(palette = "YlGnBu") + guides(fill = FALSE) + theme\_bw()

table(df\_total$voted\_up)

df\_total %>%

group\_by(author.steamid) %>%

summarize(number\_of\_ratings\_per\_user = n()) %>%

ggplot(aes(number\_of\_ratings\_per\_user)) +

geom\_bar(fill = "cadetblue3", color = "grey20") + coord\_cartesian(c(0, 15))

df\_total %>%

group\_by(appid) %>%

summarize(number\_of\_ratings\_per\_game = n()) %>%

ggplot(aes(number\_of\_ratings\_per\_game)) +

geom\_bar(fill = "orange", color = "grey20", width = 1)

tabla1 <- df\_total %>%

select(header\_image, name, recommendations, positive, negative, rating) %>% distinct() %>%

mutate(image = paste0('<img src="', header\_image, '" width="100" height="80"></img>')) %>%

arrange(-rating) %>%

#top\_n(5,wt = rating) %>%

select(image, name, recommendations, positive, negative, rating)

datatable(data = tabla1, class = "nowrap hover row-border", rownames = TRUE,

escape = FALSE, options = list(dom = 't',scrollX = FALSE, autoWidth = TRUE,

initComplete = JS("function(settings, json) {","$(this.api().table().header()).css({'background-color': '#000', 'color': '#fff'});","}")))

generos <- as.data.frame(sort(prop.table(table(c(res[,1], res[,2], res[,3], res[,4], res[,5], res[,6], res[,7], res[,8], res[,9]))[-1])))

colnames(generos) <- c("title", "percentage")

ggplot(data=generos, aes(x=title, y=percentage, fill = percentage)) + geom\_bar(stat="identity") +

coord\_flip() + scale\_fill\_distiller(palette = 'YlGnBu', direction = 1) + labs(y = 'Percentage', x = 'Genre') + theme\_minimal()

sort(prop.table(table(c(res[,1], res[,2], res[,3], res[,4], res[,5], res[,6], res[,7], res[,8], res[,9]))[-1]))

df\_total[,.(n = .N), by = appid][order(-n)] # Peliculas con más de 100 reseñas

df\_total[,.(n = .N), by = author.steamid][order(-n)]

steamRatings <- as(df\_total[,.(author.steamid, appid, voted\_up = as.numeric(voted\_up))],"realRatingMatrix") # 817226 (usuarios) x 3776 (peliculas) x 1267743 (ratings)

steamRatings\_binarize <- binarize(steamRatings, minRating=1) # 817226 (usuarios) x 3776 (peliculas) x 986289 (ratings)

## Filtro por aquellos usuarios con almenos 10 reseñas

steamRatings\_binarize <- steamRatings\_binarize[rowCounts(steamRatings\_binarize) > 10] # 4377 (usuarios) x 3776 (peliculas) x 91905 (ratings)

set.seed(456) # Creador del esquema de evaluacion con un train del 0.9, validacion cruzada de 10 capas

esquema <- evaluationScheme(steamRatings\_binarize, method='cross-validation', k=5, given=10, train = 0.8)

# 1) Artículos populares (POPULAR) es un algoritmo no personalizado que recomienda a todos los usuarios los artículos más

# populares que aún no han calificado.

# 2) Elementos elegidos aleatoriamente (ALEATORIO) crea recomendaciones aleatorias que se pueden utilizar como referencia

# para la evaluación del algoritmo de recomendación.

# 3) El filtrado colaborativo basado en el usuario (UBCF) predice las calificaciones agregando las calificaciones de los usuarios que

# tienen un historial de calificación similar al del usuario activo.

# 4) El filtrado colaborativo basado en elementos (IBCF) utiliza la similitud de elemento a elemento en función de las calificaciones

# de los usuarios para encontrar elementos similares a los elementos que le gustan al usuario activo.

algoritmos <- list(RANDOM = list(name="RANDOM", param=NULL),

POPULAR = list(name="POPULAR", param=NULL),

ALS = list(name="ALS"),

AR = list(name="AR", param = list(supp = 0.001, conf = 0.2, maxlen = 2)),

UBCF = list(name="UBCF", param=list(method="Jaccard")),

IBCF = list(name="IBCF", param=list(method="Jaccard")))

system.time(resultados <- evaluate(x = esquema, method = algoritmos, n = c(5,10)))

lubridate::seconds\_to\_period(4003) # 46 M 20S

avg(resultados)

par(mfrow = c(1,2))

plot(resultados, legend="topleft", annotate=T)

plot(resultados,"prec/rec")

#HybridRecommender

algoritmos\_final <- list(POPULAR = list(name="POPULAR", param=NULL),

AR = list(name="AR", param = list(supp = 0.001, conf = 0.2, maxlen = 2)),

UBCF = list(name="UBCF", param=list(method="Jaccard")),

IBCF = list(name="IBCF", param=list(method="Jaccard")))

system.time({recom <- Recommender(getData(esquema, "train"), method = "HYBRID",

parameter = list(recommenders = algoritmos\_final, weights = c(.05, .6, .1, .25)))})

lubridate::seconds\_to\_period(370) #6M 10S

getModel(recom)

system.time(prediccion\_top <- predict(object = recom, newdata = getData(esquema, "known"), n = 5, type="topNList"))

prediccion <- as(prediccion\_top, "list");lubridate::seconds\_to\_period(209) #3M 29S

distinct(df\_total[,.(appid, name)])[appid %in% prediccion[[1]]]

round(calcPredictionAccuracy(prediccion\_top, getData(esquema, "unknown"), given = 10), 5)

cosa <- calcPredictionAccuracy(prediccion\_top, getData(esquema, "unknown"), given = 10, byUser = TRUE)

round(apply(cosa, 2, mean), 5)

system.time(prediccion\_total <- as(predict(object = recom, newdata = steamRatings\_binarize, n = 5, type="topNList"), "list"))

length(prediccion\_total)

df\_steam\_ratings <- as.data.table(getData.frame(from = steamRatings\_binarize))

df\_predict\_user <- distinct(df\_steam\_ratings[,.(user)])

df\_predict\_user <- left\_join(x = data.frame(appid = unlist(prediccion\_total), author.steamid = rep(x = df\_predict\_user$user, each = 5)) %>% mutate(appid = as.integer(appid)),

y = left\_join(x = df\_juegos, y = distinct(df\_total[,.(appid, recommendations, positive, negative, rating)]), by = "appid"),

by = "appid")

datatable(as.data.frame(t(df\_predict\_user[1:5,-2]))[,1:2])