UNIVERSIDAD DEL BÍO BÍO
FACULTAD DE CIENCIAS EMPRESARIALES
DEPARTAMENTO DE SISTEMAS DE INFORMACIÓN

UNIVERSIDAD DEL BÍO-BÍO

Desarrollo, validación e integración de una extensión a un framework para la evaluación de recomendaciones conscientes del contexto temporal.

Memoria para Optar al Título de

Ingeniero de Ejecución en Computación e Informática.

Alumnos: Carlos Palma Valenzuela.
Matías Valdés Garrido.

Profesor guía: Dr. Pedro Campos Soto.

02/10/2015

Agradecimientos.

En primer lugar quiero agradecer a mis padres, de quienes estoy tremendamente orgulloso, gracias por la educación, por lograr que nunca me faltara nada ni a mí ni a mi hermana, y lo más importante, gracias por los valores que me dieron y lo que pude extraer de ellos quizás sin que se dieran cuenta, gracias por todo su esfuerzo, que vale mucho más que un cartón con la palabra "Ingeniería".

Un agradecimiento muy especial a mi abuelo Carlos Valenzuela, que falleció unos meses atrás. La persona a la cual más me parezco. Me hubiera gustado que me viera titulado, pero tardé más de la cuenta.

Gracias a mi novia Nicol por ser mi principal pilar de apoyo y motivación para terminar la carrera, vencer la distancia y lograr al fin estar juntos. A mi Tío Carlos por alimentar desde pequeño mi curiosidad. A mi mejor amigo Eric Román por estos 20 años de amistad, a Juan Félix que más que primo es un hermano. A Francisco Muñoz, amigo y compañero de departamento. A mi amiga Fran por todas esas horas de conversación. A Favio, Carlos, Diego y toda "La Familia" por darme siempre apoyo. A mis amigos de Los Ángeles, el Huevo, Pacheco, Javi, Willy, Burgos y Paji por todos los años que fuimos compañeros y los recuerdos creamos. A mis compañeros de carrera, Ilich, Gerard, Seba, Daniel, Carlos, Pablo, Julio y Andy por toda esta etapa universitaria que atravesamos juntos, por hacer de la sala de clases un lugar agradable que de seguro más de una vez extrañaremos. Gracias también a Pedro Campos, mi profesor guía por apoyarme durante este proyecto de tesis y en los ramos donde fui su alumno. También a Matías, por ser un gran amigo, compañero de proyectos y tesis, estoy seguro que será un gran profesional.

Gracias a todas las personas que directa o indirectamente ayudaron en mi formación como profesional y como persona, no habría podido alcanzar mis metas sin ustedes.

"Lo que el árbol tiene de florido vive de lo que tiene sepultado"

-LPDA.

Carlos Palma Valenzuela.

Quiero comenzar agradeciendo a mi familia, en especial a mi madre por el esfuerzo realizado durante todos estos años para darme lo mejor, por la educación que entregó, lo cual me ha ayudado a llegar donde estoy lo cual me hace sentirme orgulloso de ella, a mis abuelos Eugenia y Rolando, los cuales formaron parte importante en mi crianza desde pequeño, a mi padre por su ayuda y palabras de apoyo, motivándome para superarme siempre, a mis hermanos, tíos, primos, y amigos, por el apoyo incondicional y por la constante preocupación en cada etapa importante.

Agradezco a Constanza, la madre de mi hija, la cual me ha acompañado de forma incondicional entregándome su cariño, amor y apoyo durante toda mi etapa universitaria junto a nuestra hija Maite, quien me da fuerzas todos los días y es mi motivación para lograr mis metas, para darle lo mejor y lograr ser una familia.

Dar gracias a mis compañeros a lo largo de la carrera, con los cuales hemos compartido buenos momentos tanto dentro como fuera de las salas de clases, en especial a mi compañero de tesis Carlos, quien ha sido un gran amigo dentro de la carrera y aprendí a conocer a una gran persona y no me cabe duda que será un gran profesional. También agradezco a nuestro profesor guía el Dr. Pedro Campos por la posibilidad de trabajar en este tema, por el apoyo durante el desarrollo de nuestra tesis y los conocimientos entregados en este trabajo y en el ramo que fui su alumno.

“El que se doble permanecerá entero;
El que se incline se erguirá;
El que sufra el desgaste se renovará;
El que abarque poco adquirirá el conocimiento seguro
El que abarque mucho caerá en la duda.”
Tao – Tê Ching.

Matías Valdés Garrido.

Resumen.

En este trabajo se desarrollan, validan e integran a un framework existente una serie de métodos de evaluación de Sistemas de Recomendación que toman en cuenta el contexto, particularmente el contexto temporal. Se comienza explicando qué es un Sistema de Recomendación, sus clasificaciones y técnicas de recomendación, para luego integrar el concepto de Contexto.

Posteriormente se revisan los principales conceptos relacionados con la evaluación de Sistemas de Recomendación, así como también diferentes protocolos de evaluación existentes, tareas de recomendación y metodologías de evaluación tiempo-dependientes y tiempo-independientes presentes en la literatura.

Para desarrollar los métodos de evaluación a integrar, se toman como base los conceptos y clases utilizados en el framework a extender. Se describen los paquetes y clases principales que conforman este framework, y luego se describe el diseño e implementación de los nuevos métodos, centrados en la realización de validaciones cruzadas, las que permiten tener una mejor idea sobre la variabilidad de los resultados de evaluación.

Se ejecutan una serie de experimentos utilizando las nuevas validaciones cruzadas integradas al framework, con el fin de comprobar su importancia, así como también la repercusión que tiene el contexto a la hora de realizar evaluaciones.

En los resultados de los experimentos se observaron diferencias cercanas al 30% en las mediciones hechas en las diferentes validaciones cruzadas con diversos métodos probados, así como también se determinó que la consideración del contexto temporal en la evaluación influye en los resultados de las métricas aplicadas.

Tabla de contenido

1. Introducción.

Las personas a menudo tienen la necesidad de seleccionar un servicio o producto (ítem) entre varias alternativas sin tener conocimiento exacto de cada uno de ellos. Muchas veces, la elección final depende del conocimiento pasado acerca de un ítem o de las recomendaciones de otras personas. Hoy en día, esta tarea resulta más complicada debido al aumento de información que se puede encontrar en la Web. Lo mismo sucede en la vida cotidiana, por ejemplo, al comprar en una gran tienda; existen una gran cantidad de pasillos con diferentes productos, y puede tomar una gran cantidad de tiempo recorrer cada uno de los pasillos para encontrar el producto indicado, o incluso sin llegar a encontrarlo. Lo mismo sucede, aunque a mayor escala, al realizar una búsqueda en Internet, por lo que cada vez se hace más difícil gestionar la excesiva cantidad de información.

Los Sistemas de Recomendación son herramientas de software y técnicas que proporcionan sugerencias de ítems que pueden ser interesantes para un usuario. Estos sistemas se han ido consolidando en el tiempo como potentes herramientas que ayudan a disminuir la sobrecarga de información a la que se enfrentan las personas en procesos de búsqueda a través de sitios Web. Las sugerencias están relacionadas con diferentes procesos en la toma de decisiones, usando distintas técnicas para identificar el ítem de mayor preferencia o que mejor satisface las necesidades del usuario.

En general, un Sistema de Recomendación se centra en un dominio en específico (por ejemplo, películas, música, o noticias), por lo tanto, su interfaz gráfica, y el núcleo de su técnica de recomendación usada para generar recomendaciones están adaptadas para ofrecer sugerencias útiles y eficaces en dicho dominio [1].

El objetivo de los Sistemas de Recomendación (SR) es ayudar al usuario a encontrar un ítem en específico, a través de la recopilación de información. En general, trabajan sugiriendo ítems que deberían ser los más atractivos a los usuarios en función de sus preferencias personales pasadas. Con el tiempo se ha buscado la forma de mejorar la precisión de la predicción de preferencias, con lo cual se ha incorporado el contexto (por ejemplo, ubicación, tiempo, clima, si se encuentra acompañado, el estado de ánimo, y el dispositivo desde el que se encuentra

conectado), demostrando que la información de contexto es una fuente de información muy valiosa para mejorar la calidad de las recomendaciones.

Dentro de las dimensiones contextuales existentes, la información de tiempo puede ser considerada como una de las más útiles. Facilita el seguimiento de la evolución de las preferencias del usuario [2] lo que permite por ejemplo, determinar la periodicidad en los hábitos del usuario e intereses, lo cual es un aporte clave para muchos SR sensibles al contexto (CARS) [3].

Debido entre otros motivos, a las diferentes fuentes de información que pueden explotar, se han propuesto variados algoritmos de recomendación. Para elegir el mejor algoritmo para un sistema de recomendación, usualmente se realizan experimentos, comparando el rendimiento de un número de algoritmos candidatos. En estas evaluaciones, normalmente se aplican una o más métricas de evaluación, lo cual nos ofrece una clasificación de los algoritmos candidatos.

La evaluación de los sistemas de recomendación, y en específico de los algoritmos utilizados en ellos, muchas veces se ve dificultada por varias razones. Por ejemplo, muchas veces se comparan algoritmos ideados para diferentes dominios de aplicación. También, pueden existir diferentes objetivos al momento de evaluar. Al considerar además la posibilidad de utilizar diferentes fuentes de datos, y en particular el contexto temporal, hace que la evaluación sea aún más compleja. Por ello, resulta de gran utilidad contar con un marco de trabajo (framework) que permita establecer claramente la forma en que se realiza esta evaluación, es decir, el protocolo de evaluación. Actualmente se cuenta con un framework para estos efectos. Sin embargo, dicho framework no aborda completamente una importante familia de métodos utilizados en evaluación: validaciones cruzadas. La validación cruzada repite la ejecución de los algoritmos de recomendación utilizando diferentes datos en cada iteración, lo cual permite obtener información sobre la variabilidad de los resultados en la evaluación de un sistema de recomendación.

Debido a lo anterior, en este proyecto de título se plantean los siguientes objetivos:

Objetivo General

Desarrollar, validar e integrar una extensión a un framework existente de evaluación de Sistemas de Recomendación, que permita realizar pruebas a distintos algoritmos de recomendación conscientes del contexto, utilizando diferentes métodos de validación cruzada.

Objetivos específicos

Analizar, diseñar e implementar un conjunto de clases que provea la funcionalidad necesaria para implementar diferentes metodologías de evaluación de recomendaciones conscientes del contexto, basadas en el uso de validación cruzada.

Probar el conjunto de clases implementadas, integrándolas a una librería de evaluación de algoritmos de recomendación existentes.

Diseñar una aplicación que haga uso del framework extendido, incluyendo una interfaz gráfica de usuario de fácil uso para la ejecución de experimentos sobre conjuntos de datos reales, que permita validar la adecuada integración del conjunto de clases implementado.

Ejecutar experimentos de evaluación de recomendaciones utilizando el framework extendido, para validar la implementación realizada.

El resto del trabajo se encuentra organizado de la siguiente manera: En el capítulo 2 se describen los tipos de sistema de recomendación y se clasifica la incorporación del contexto; en el capítulo 3 se identifican los principales conceptos involucrados en la evaluación de sistemas de recomendación que consideran el contexto temporal, y que inciden en los métodos de validación cruzada aplicables; en el capítulo 4 se describen los paquetes y clases principales que conforman el framework de evaluación a extender; en el capítulo 5 se identifican las nuevas propuesta de evaluación y se describe la implementación de los nuevos métodos en el framework; en el capítulo 6 se describe una aplicación que utiliza la funcionalidad del framework, validando la integración de los métodos de validación cruzada implementados, y que provee una interfaz gráfica para la configuración y ejecución de experimentos; en el capítulo 7 se presentan los

resultados obtenidos una vez realizados los experimentos con los distintos métodos de evaluación, según las métricas de evaluación obtenidas. Finalmente, en el capítulo 8 se presentan las conclusiones de este trabajo.

2. Marco teórico.

Los sistemas de recomendación son una herramienta muy popular hoy en día en sitios web de comercio electrónico y comunidades de investigación. Estos sistemas constan de una variedad de propiedades que pueden influenciar la experiencia del usuario, tales como precisión, robustez, escalabilidad, etc.

En los últimos años se ha observado una sobrecarga de información en internet y cada vez es más difícil gestionar esta excesiva cantidad a la que diariamente se enfrentan los usuarios, lo cual es un problema al momento de encontrar la información más útil, dependiendo de sus necesidades, volviéndose una tarea bastante compleja. Esta situación ha llevado a muchos sitios web a implementar sistemas de recomendación, ayudando a filtrar la información recuperada, a través de distintas técnicas para identificar aquellos ítems que mejor satisfacen las preferencias o necesidades de los usuarios.

En general los sistemas de recomendación se encargan de orientar a los usuarios, generando una selección de ítems de acuerdo a gustos y preferencias que han manifestado previamente. Para ello aplican diversas técnicas estadísticas y de descubrimiento de conocimiento (Machine Learning). Un buen sistema de recomendación es aquel que mejor se adapta a las necesidades del usuario, evitando recomendaciones que no sean de su interés, por ejemplo, en un sistema de arriendo de películas online, donde las películas más vistas por un usuario son de ciencia ficción, una buena recomendación probablemente sería una película del mismo género que el usuario no conozca.

Actualmente, hay una gran cantidad de sitios web que utilizan sistemas de recomendación. Uno de los sitios más destacados es Amazon[1] el cual tiene millones de usuario e ítems registrados, adaptándose a cada usuario según sus preferencias (intentando simular una tienda para cada usuario, con los artículos que necesita) a medida que va interactuando con el sitio web. Con estos datos es capaz de ofrecer al usuario productos que le puedan interesar y que no sea el usuario el que tenga que consultar los miles de productos que tiene Amazon. Otro sitio que utiliza

[1] Sitio de ventas por Internet Amazon. Disponible en http://www.amazon.com

sistemas de recomendación es YouTube[2], el cual hace sus recomendaciones según los últimos vídeos vistos por un usuario y recomienda videos similares, basándose en el contenido o en los "tags" característicos de los vídeos.

Si bien estos sistemas se basan en modelos matemáticos y estadísticos, lo que intentan simular es lo que hacen comúnmente las personas al momento de elegir algún servicio o producto: "pedir consejos o recomendaciones a terceros que tengan gustos similares", o "seleccionar ítems con características similares a algunos ítems que haya elegido anteriormente o que se parezca al que busco inicialmente". Dando lugar a dos principales "ramas" de los sistemas de recomendación: Los basados en contenido y los colaborativos.

2.1 Técnicas de recomendación.

Según la literatura existen varios tipos de recomendaciones, por ejemplo, según el tipo de datos y los métodos que se van a utilizar para generar las recomendaciones. Burke [4] distingue cuatro técnicas principales de recomendación: **Técnicas basadas en contenido (CB)**, que sugieren al usuario objetivo (para el cual se generan las recomendaciones) ítems similares a los preferidos en el pasado por él; **técnicas de filtrado colaborativo (FC)**, que sugieren ítems preferidos por usuarios con gustos similares a aquellos del usuario objetivo, **técnicas demográficas**, que exploran la demografía de los usuarios para generar recomendaciones de ítems, y **técnicas basadas en el conocimiento**, que explotan el conocimiento de un dominio en específico acerca de ítems para recomendar. También, es posibles distinguir **recomendadores híbridos**, que combinan dos o más de las técnicas mencionadas anteriormente con el fin de aminorar algunas de sus limitaciones.

En general, todas estas técnicas buscan sugerir ítems que sean los más atractivos para un usuario de acuerdo a sus preferencias. La preferencia que un usuario tiene por un ítem comúnmente se representa con el concepto de "rating" o valoración. De esta forma, el problema de generar recomendaciones se puede representar matemáticamente a través de una función F

[2] Sitio de streaming de video Youtube, disponible en http://www.youtube.com

que calcula un rating predicho $\hat{r}_{u,i}$ para un rating desconocido $r_{u,i}$ que el usuario u ∈ **U** asignaría al ítem i ∈ **I**:

$$\mathbf{F: U \times I \rightarrow R}$$

Donde R denota un conjunto totalmente ordenado.

Las preferencias pasadas de los usuarios, en las que se basan los SR para generar recomendaciones, comúnmente se recolectan a partir de la retroalimentación que proporcionan los usuarios respecto de los ítems que consumen o compran. La información de retroalimentación se suelen clasificar como explícita o implícita.

La retroalimentación explícita corresponde a valoraciones que el usuario realiza de forma expresa, por ejemplo asignar 5 estrellas o realizar una indicación de “me gusta” o “no me gusta”. La retroalimentación implícita, por otro lado, se recolecta de forma automática, a partir del comportamiento normal del usuario, sin pedir una indicación explícita de preferencia. Por esto, se requiere realizar una interpretación de la retroalimentación implícita para determinar la preferencia del usuario. Por ejemplo, la acción de comprar un producto se puede interpretar como una preferencia del usuario hacia dicho producto.

También existen algunos sistemas de recomendación que incorporan el uso de información de contexto asociado a las preferencias de los usuarios, por ejemplo, la hora, el tiempo, estado del ánimo, en compañía de quién se encuentra el usuario, etc. Sin embargo a veces es difícil contar con esta información, ya que muchos usuarios no están dispuestos a entregarla por preocupaciones de privacidad.

Los sistemas de recomendación que explotan cualquiera de los tipos de información anteriores son llamados **Sistemas de recomendación conscientes del contexto.** Ampliando el problema mencionado anteriormente, Adomavicius et al (2005) [5] plantea un modelo genérico incorporando dimensiones adicionales de información contextual **C** en la función **F**:

$$\mathbf{F: U \times I \times C \rightarrow R}$$

Entre las dimensiones contextuales existentes, podemos encontrar las dimensiones de tiempo, por ejemplo, la hora del día, día de la semana, estación del año, las cuales son más fáciles de obtener, ya que cualquier sistema puede registrar el momento asociado a un rating o consumo de un ítem. Estos tipos de sistemas de recomendación (conscientes del tiempo) se pueden considerar con un tipo especializado de sistemas de recomendación conscientes del contexto.

La principal característica de estos sistemas es proporcionar distintas recomendaciones dependiendo del tiempo asociado a la recomendación objetivo (el momento en que se desea consumir el ítem), basándose en aquellas preferencias expresadas por el usuario en contextos similares. Así la formulación de la predicción del rating dependientes del contexto puede ser personalizada considerando la dimensión del contexto temporal **T**:

$$\mathbf{F: U \times I \times T \rightarrow R}$$

Donde T puede ser representado de varias maneras, dado que te no se puede considerar con un valor fijo.

2.1.1 Clasificación de los sistemas de recomendación.

Los sistemas de recomendación se pueden clasificar según su funcionamiento y la forma de hacer recomendaciones (estimar los ratings no puntuados). Dentro de la literatura nos podemos encontrar con diferentes clasificaciones [1, 6], como lo muestra la figura 1.

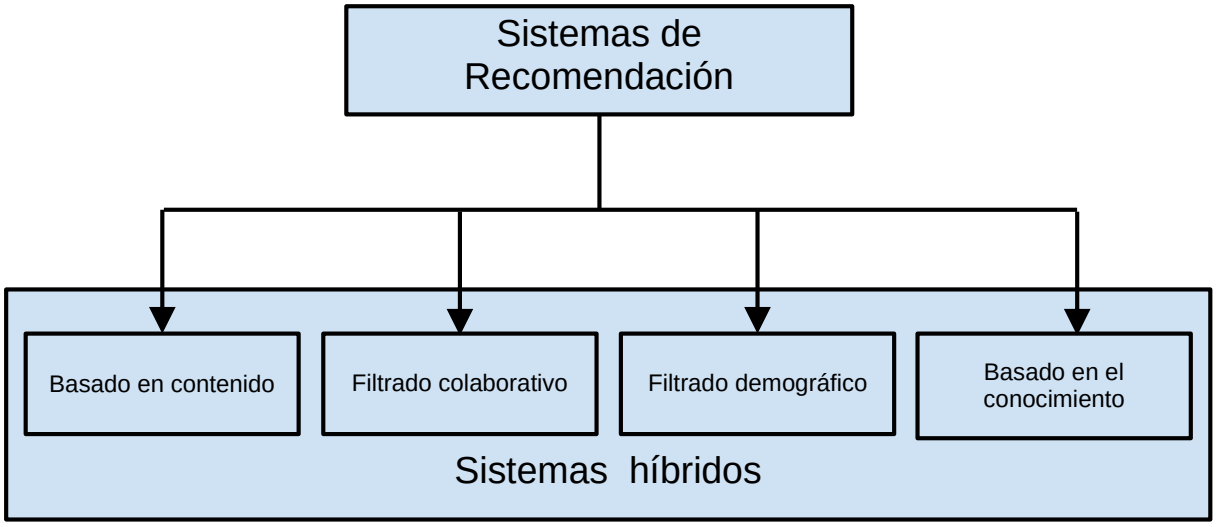


Figura 1. Clasificación de los sistemas de recomendación.

2.1.1.1 Sistemas de recomendación basados en contenido.

Es aquel que basa sus recomendaciones en un perfil creado a partir del análisis del contenido (descripciones) de los ítems que el mismo usuario ha comprado, utilizado o visto en el pasado. El objetivo de este método es obtener características de los ítems no conocidos por el usuario y compararlo con su perfil para predecir sus preferencias. Por lo tanto, lo que se pretende es recomendar ítems cuyo contenido sea muy similar a aquellos de los que se sabe son del gusto del usuario, es decir, las que son parte de su perfil.

El filtro basado en contenido fue utilizado por mucho tiempo, hasta la llegada del filtro colaborativo. Uno de los problemas de los recomendadores basados en contenido es la “sobre-especialización”. La sobre-especialización se da al reducir las recomendaciones a unos contenidos muy similares sin tener en cuenta la posible arbitrariedad de los gustos e intereses de los usuarios. Una solución parcial es con la incorporación de algoritmos de aleatoriedad.

Otro problema de estos sistemas basados en contenido, es el ingreso de un nuevo usuario, ya que las recomendaciones se basan en ítems valorados con anterioridad, y en este caso no hay rating previos con los cuales comparar.

2.1.1.2 Sistemas de recomendación basados en filtrado colaborativo.

Son aquellos que se basan en la similitud entre usuarios. Es decir, recomienda ítems que son del gusto de otros usuarios con intereses similares. Estos algoritmos se pueden encuadrar dentro de dos categorías: los algoritmos basados en memoria y basados en modelo.

El objetivo es sugerir nuevos ítems a un usuario basándose en sus preferencias anteriores y las de personas que tengan un historial similar de ratings. Existen dos formas de recuperar estas valoraciones. Una es de forma explícita, es decir el usuario asigna una puntuación que será un valor numérico. La segunda forma es recuperar las valoraciones implícitamente, extrayendo la información necesaria de las acciones del usuario.

El filtrado colaborativo basado en memoria, corresponde a aquellos algoritmos que utilizan todos los datos de ítems y usuarios con historial de valoraciones similar a la del usuario objetivo, para generar predicciones. La desventaja de este método es que se necesita un mínimo de usuarios con un mínimo de ratings, incluido el usuario al cual queremos realizar predicciones. Uno de los ejemplos más conocidos de este tipo, es el algoritmo de vecinos más cercano (nearest neighbors) [4].

El filtrado colaborativo basado en modelos desarrolla primero un modelo de los ratings del usuario. Se calculan el valor esperado para cada ítem en función de los ratings anteriores. Para ello se utilizan distintas técnicas de aprendizaje tales como clustering o redes neuronales como las Redes de Funciones de Base Radial (RBFN). Este tipo de filtrado colaborativo suele ser más rápido que el anterior, pero necesita de un proceso de entrenamiento intensivo.

Al igual que los sistemas basados en contenidos, el filtro colaborativo también tiene problemas al momento de la incorporación de un nuevo usuario, que al igual que en el caso anterior, al no tener valoraciones de ítems, el sistema no podrá realizar recomendaciones hasta que su perfil esté lo suficientemente completo para encontrar su propia vecindad. Un problema similar ocurre con el ingreso de un nuevo ítem, que al no tener puntuaciones por ningún usuario, no será parte de ninguna nueva recomendación.

2.1.1.3 Sistemas de recomendación basados en filtrado demográfico.

Este tipo de sistemas recomienda ítems considerando el perfil demográfico del usuario, escogiendo aquellos que son similares para conformar una vecindad, no solo en base a sus calificaciones y preferencias, si no también cuando dichos usuarios pertenecen al mismo segmento demográfico.

Uno de los problemas de estos sistemas de recomendación es lo difícil que puede ser la captura datos demográficos “necesarios”, ya que muchas veces los usuarios suelen ocultar su información personal.

2.1.1.4 Sistemas de recomendación basados en el conocimiento.

Son aquellos que recomiendan ítems basados en el conocimiento de un dominio en específico acerca de cómo ciertas características de ítems satisfacen las necesidades y preferencias de los usuarios.

Los sistemas basados en el conocimiento más importante son los basados en casos. En estos sistemas una función se encarga de medir cuanto las necesidades del usuario (descripción del problema) coinciden con las recomendaciones (soluciones del problema).

Los sistemas basados en el conocimiento tienden a funcionar mejor que otros en el inicio de su implementación, pero si no están equipados con componentes de aprendizaje pueden ser superados por otros métodos que pueden aprovechar los registros de la interacción usuario/máquina (como en el CF).

2.1.1.5 Los sistemas híbridos.

Son aquellos que combinan los otros métodos ya mencionados. Un sistema híbrido es aquel que puede combinar diferentes técnicas, con el fin de utilizar las ventajas de alguna de ellas para solucionar las desventajas de otra.

En la actualidad, se buscan conseguir más sistemas híbridos, los cuales interactúen con todas las técnicas de recomendación vistas, y sean capaces de mejorar su precisión lo más posible al momento de entregar recomendaciones al usuario objetivo.

2.2 Sistemas de recomendación conscientes del contexto.

Como hemos visto anteriormente en los distintos tipos de Sistemas de Recomendaciones, existe un problema que consiste en sugerir ítems que sean los más atractivos para un usuario de acuerdo a sus preferencias. Sin embargo, en muchas aplicaciones, tales como recomendar paquetes de vacaciones, contenido personalizado en un sitio web, diversos productos en una tienda en línea, o películas, puede que no sea suficiente considerar sólo los usuarios y los ítems, sino que también es importante incorporar la información contextual del escenario de la decisión del usuario en el proceso de recomendación. Por lo tanto, la predicción exacta de las preferencias del consumidor depende, sin duda, del grado en que la información contextual relevante se incorpora en un método de recomendación.

Existen muchas definiciones de contexto a través de diversas disciplinas e incluso dentro de los sub-campos específicos de estas disciplinas. Esto no es sorprendente, dada la complejidad y el carácter multifacético del concepto. De acuerdo con Dey (2001) [7], "contexto es cualquier información que se puede utilizar para caracterizar la situación de una entidad" donde en el caso de un RS una entidad puede ser un usuario, un elemento, o una experiencia que el usuario está evaluando [8]. Por lo tanto, cualquier información relacionada a la situación en la que un usuario experimenta un ítem, por ejemplo, ubicación, clima, tiempo, y el estado de ánimo pueden ser considerados como contexto [9].

Un ejemplo clásico desde el punto de vista del uso apropiado del contexto en un proceso de recomendación, es en el dominio del turismo: Donde a un usuario que le guste el esquí, recomendarle un resort de esquí en época de verano, es una recomendación no muy acertada. También se puede dar el caso donde no es lo mismo recomendar paquetes vacacionales para una pareja de recién casados, para una familia con hijos o para un grupo de adolescentes. Por lo tanto, podría ser perjudicial para la confianza del usuario en el sistema, si lo interpretan como una recomendación "fuera de contexto".

Si comparamos RS que utilizan y no utilizan información contextual podemos encontrarnos que los primeros ofrecen más confianza a los usuarios en sus recomendaciones. Además, la confianza afecta el comportamiento de compra, y los RS explotadores de información de contexto aumentan la confianza del usuario y los niveles de ventas [9].

2.2.1 Modelado de Información de contexto en los Sistemas de Recomendación.

El proceso de recomendación en general comienza con la especificación del conjunto inicial de calificaciones que está proporcionado explícitamente por los usuarios o se infiere de forma implícita por el sistema. Dado estas calificaciones iniciales, un sistema de recomendación intenta estimar la función de predicción de rating **F** [1]:

$$\mathbf{F: U \times I \rightarrow R.}$$

Para el par (usuario, ítem) que no han sido valorados aún por los usuarios. Este tipo de sistemas son llamados tradicionales o de dos dimensiones (2D), ya que consideran sólo las dimensiones usuarios e ítems en el proceso de recomendación. Muchos de estos sistemas tradicionales o 2D sólo se preocupan de recomendar ítems para usuarios o usuarios a los ítems, sin tomar en consideración cualquier información contextual adicional, como tiempo, el lugar, la compañía de otras personas. Dado estás situaciones es que nacen los sistemas de recomendación conscientes del contexto (cuya sigla en inglés es CARS), que tratan de modelar y predecir los gustos y preferencias del usuario mediante la incorporación de la información contextual disponible en el proceso de recomendación como categorías adicionales explícitas. Estas preferencias y gustos a largo plazo suelen ser expresados como ratings y se modelan como la función de no sólo los ítems y de usuarios, sino también del contexto. En otras palabras, las predicciones de calificaciones se definen con la función de predicción de ratings **F** como

$$F: U \times I \times C \rightarrow R$$

Donde Contexto especifica la información contextual asociada con la aplicación. En la literatura Adomavicius y Tuzhilin [1] consideran el siguiente ejemplo, para ilustrar este concepto:

Considere una aplicación para recomendar películas a los usuarios, donde los usuarios y las películas se describen como las relaciones que tienen los siguientes atributos:

- Película: el conjunto de todas las películas que se pueden recomendar; se define como Película (MovieID, Title, Length, ReleaseYear, Director, Genre).

- Usuario: las personas a quienes se le recomiendan películas; se define como Usuario (UserID, Name, Address, Age, Gender, Profession).

Además, la información contextual consta de los siguientes tres tipos que también son definidos como relaciones que tienen los siguientes atributos:

- Teatro: las salas de cine que muestran las películas; se define como Teatro (TheaterID, Name, Address, Capacity, City, State, Country).

- Tiempo: el momento en que la película puede ser o ha sido visto; se define como Tiempo (Date, DayOfWeek, TimeOfWeek, Month, Quarter, Year). Para este caso, el atributo DayOfWeek tiene como valores: lun, mar, miér, jue, vie, sáb., Dom. Y el atributo TimeOfWeek tiene como valores "Día de la semana" y "fin de semana".

- Acompañante: representa a una persona o un grupo de personas con las que se puede ver una película. Se define como acompañante (companionType), donde el atributo companionType tiene como valores “alone”, “friends”, “girlfriend/boyfriend”, “family”, “co-workers”, and “others”.

Entonces la calificación otorgada a una película por una persona también depende de dónde y cómo ha visto la película, con quién y en qué momento. Por ejemplo el tipo de película para

recomendar a un estudiante puede variar significativamente dependiendo de si se tiene la intención de ver una película un sábado por la noche con su novia, o un día de semana con sus padres.

El ejemplo anterior, lo que se intenta explicar es que pueden existir diferentes contextos, tales como tiempo, ubicación, acompañante, propósito de la compra, etc... Además, cada tipo contexto puede tener una estructura. De hecho muchas veces se puede observar una estructura jerárquica de la información contextual, que puede ser representada como árboles. Por ejemplo, los tres contextos del caso anterior pueden tener las siguientes jerarquías asociadas con ellos:

Teatro: TheaterID→ City → State → Country; Tiempo: Date → DayOfWeek → TimeOfWeek, Date → Month → Quarter →Year.

Según Adomavicius Tuzhilin [1], además de las dimensiones clásicas de los usuarios y los ítems, dimensiones contextuales adicionales, tales como el tiempo, la ubicación, etc., se pueden incorporar utilizando OLAP-based[3] multidimensional data (MD) model, el cual es muy utilizado en las aplicaciones de almacenamiento de datos en bases de datos. [10].

Por ejemplo, considerando dimensiones de contexto de Usuario (**U**) × Ítem (**I**) × Tiempo (**T**), se puede definir una función $\mathbf{R}$ en el espacio **U** × **I** × **T** que especifica el rating dado por el usuario u ∈ **U** al ítem i ∈ **I** en el tiempo t ∈ **T**, $\mathbf{R}$ (u, i, t).

Se puede graficar lo anterior, considerando que los ratings se almacenan en un cubo multidimensional, tal como la que se muestra en la siguiente figura:

[3] OnLine Analytical Processing (OLAP), representa un enfoque popular para la manipulación y análisis de los datos almacenados en las estructuras de cubos multidimensionales y que es ampliamente utilizado para la ayuda a la decisión.

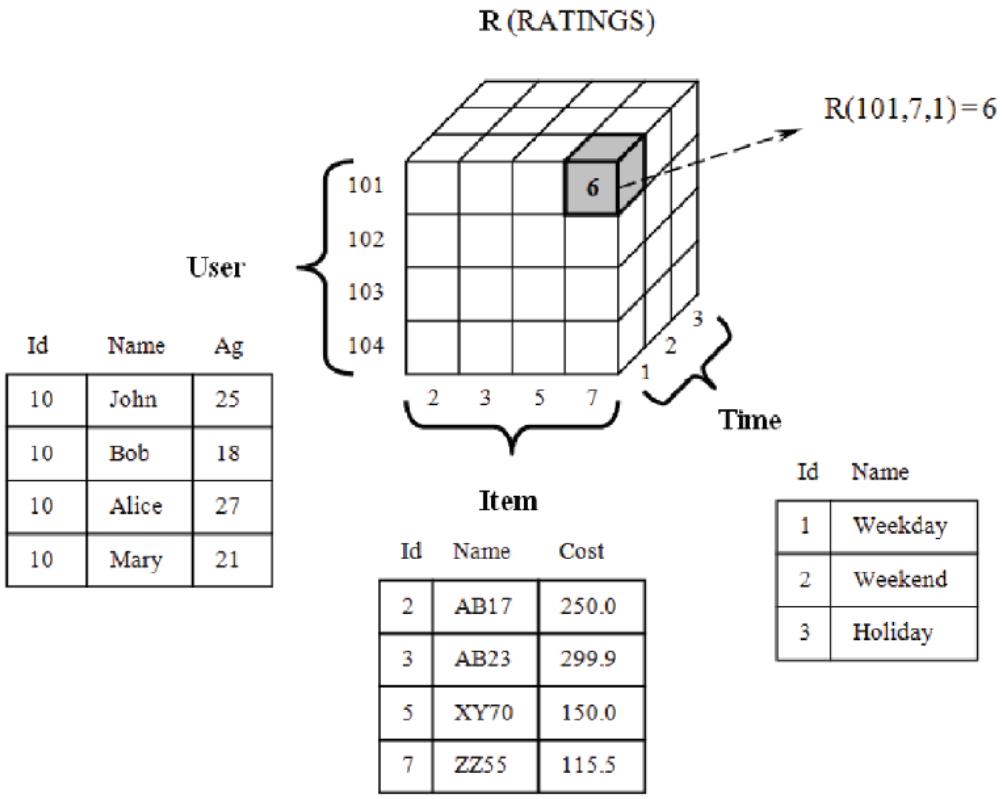


Figura 2. Modelo multidimensional para el espacio de recomendación Usuario × Contexto × Tiempo [6].

El cubo de la figura 2 almacena ratings **R** (u, i, t) para espacio de recomendación **U** × **I** × **T**, donde las tres tablas definen los grupos de Usuario, Ítem y Tiempo asociados a las dimensiones usuario, ítem, y tiempo respectivamente. Por ejemplo, el rating **R** (101, 7 ,1) = 6 en la figura, significa que para el usuario con el UserID 101 y el ítem con el ItemID 7, el rating 6 se especificó durante la semana.

2.2.2 Obteniendo información contextual.

La información contextual puede ser obtenida de distintas maneras, tales como:

- De forma explícita: Esto se refiere a que la aplicación obtenga información contextual directamente del usuario, mediante preguntas directas o provocar esta información a través de otros medios. Por ejemplo, preguntar a través de un formulario o responder algunas preguntas antes de realizar una determinada recomendación.

A veces, obtener información de esta manera, puede generar desconfianza por parte del usuario, ya que se ve en la situación de tener que entregar información “extra” al sistema.

- De forma implícita: Es decir, obteniendo información sin necesidad de consultar al usuario de forma explícita, como lo pueden ser obteniendo la localización en el caso de usar la aplicación a través de un dispositivo móvil o la marca de tiempo de alguna transacción.

- Por inferencia: Esto es inferir el contexto utilizando métodos de minería de datos o estadísticos. Por ejemplo, identificar a una persona que esté viendo canales de televisión en su casa (esposo, esposa, hijo, hija, etc.) no se puede saber de forma explícita para una empresa de televisión por cable; pero se puede inferir mediante los programas de televisión que ha visto y los canales visitados usando métodos de minería de datos [1].

Finalmente, una vez obtenida la información contextual por cualquiera de los métodos anteriores, esta es utilizada por el Sistema de Recomendación para ayudar a generar mejores recomendaciones, buscando la adaptación de las recomendaciones al contexto actual (o deseado) del usuario (contexto objetivo).

2.2.3 Paradigmas para la incorporación de Contexto en Sistemas de Recomendación

Existe una hipótesis (Herlocker y Konstan) [11] de que la inclusión de los conocimientos sobre la tarea del usuario en el algoritmo de recomendación en ciertas aplicaciones puede conducir a mejores recomendaciones [1].

Los diferentes enfoques para el uso de información contextual en el proceso de recomendación pueden ser clasificados en dos grupos: (1) recomendación a través de consultas y búsquedas dirigidas por el contexto, y (2) la recomendación a través de obtención de preferencias y de estimación de contexto. El enfoque de consultas y búsquedas dirigidas por contexto se utiliza a menudo en sistemas de recomendación móviles y turísticos [12, 13, 14]. Los sistemas que utilizan este enfoque suelen utilizar información contextual (obtenida ya sea directamente del usuario,

por ejemplo, mediante la especificación del actual estado de ánimo o interés, o del ambiente, por ejemplo, la obtención de la hora local, el clima, o la ubicación actual) para consultar o buscar un determinado depósito de recursos (por ejemplo, restaurantes) y presentar los mejores recursos que coinciden (por ejemplo, los restaurantes cercanos que están actualmente abiertos) para el usuario [1].

En contraste con el otro método mencionado (donde los sistemas de recomendación utilizan la información de contexto actual y de interés específico del usuario mediante la especificación de estado de ánimo, la obtención de tiempo, el clima, o la ubicación), las técnicas que siguen a este segundo enfoque intentan modelar y aprender las preferencias del usuario, por ejemplo, mediante la observación de las interacciones de éste y otros usuarios con los sistemas o mediante la obtención de información de preferencias del usuario en diversos artículos recomendados previamente.

Para hablar de las técnicas de obtención de preferencias y de estimación de contexto es necesario mencionar los sistemas de recomendación tradicionales 2D, ya que la especificación de tareas para un usuario específico, consiste en una lista de ítems de muestra. En otras palabras, además de las dimensiones de usuarios e ítems estándar, no se utilizan dimensiones contextuales adicionales. La Figura 3, presenta una visión general del proceso de recomendación 2D tradicional, que incluye tres componentes: los datos (de entrada), sistema recomendador 2D (función), y la lista de recomendación (de salida). Teniendo en cuenta que, como se indica en la figura, después de la función de recomendación se define (o construye) en base a los datos disponibles, la lista de recomendación para cualquier usuario *u* dado, normalmente es generada mediante el uso de la función de recomendación sobre el usuario *u*, y todos los artículos candidatos para obtener una calificación predicha para cada uno de los ítems, y luego por la clasificación de todos los ítems en función de su valor de rating predicho.

Figura 3. Componentes generales del proceso de recomendación tradicional [1].

Por otra parte, los sistemas de recomendación conscientes del contexto se construyen basados en el conocimiento parcial del contexto de las preferencias de usuario y por lo general tratan los registros de datos de la forma <usuario, ítem, contexto, rating>, donde cada registro específico incluye no sólo cuánto le ha gustado un ítem en específico a un determinado usuario, sino también la información contextual en la que el artículo fue utilizado o consumido por este usuario (por ejemplo, Contexto = Sábado).

Basándose en la presencia de estos datos contextuales adicionales, surgen varias preguntas importantes [1]: ¿Cómo la información contextual debería reflejarse al modelar las preferencias de usuario? ¿Podemos reutilizar la riqueza de conocimientos en sistemas de recomendación tradicionales (no-contextuales) para generar recomendaciones conscientes del contexto?

En presencia de la información contextual disponible, siguiendo los diagramas de la Figura 4 partimos de la información que tiene la forma U × I × C × R, donde C es la dimensión contextual adicional y terminamos con una lista de recomendaciones contextual i_1, i_2, i_3. . . para cada usuario. Sin embargo, a diferencia del proceso en la Figura 3 (anterior), que no toma en cuenta la información contextual, podemos aplicar la información acerca del contexto c actual (o deseado) en las distintas etapas del proceso de recomendación. Más específicamente, el proceso de recomendación consciente del contexto que se basa en la obtención de preferencias del usuario y la de estimación de contexto puede adoptar una de las tres formas, en base a cuál de los tres componentes del contexto se utiliza, como se muestra en la siguiente figura:

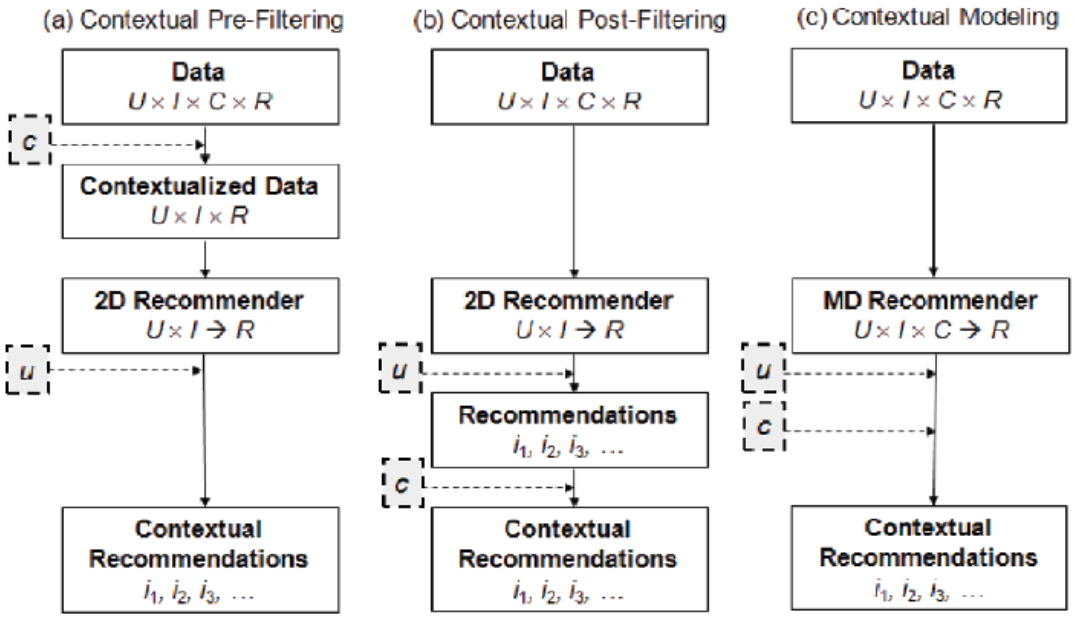


Figura 4. Paradigmas para la incorporación de contexto en los sistemas de recomendación [1].

- Pre-filtrado contextual (Contextual pre-filtering): En este paradigma de recomendación en una primera etapa se explota la información contextual, es decir, se procesan los datos con la información contextual, para obtener un set de datos acorde al contexto objetivo. Luego, se puede utilizar cualquier técnica de recomendación sobre los datos seleccionados (filtrados).

Por ejemplo, en una aplicación que recomendara cines y utilizara la ubicación geográfica como información de contexto, se debe realizar un filtrado de todos los cines de la ciudad o región, para dejar fuera los cines lejanos, y a partir de ese grupo generaría recomendaciones.

- Post-filtrado contextual (Contextual post-filtering): En este paradigma de recomendación, la información contextual se ignora inicialmente, y los ratings son predichos aplicando cualquier técnica de recomendación tradicional sobre el conjunto completo de datos. A continuación, se ajusta el conjunto resultante de recomendaciones (contextualizado) para cada usuario utilizando la información contextual.

Por ejemplo, en una aplicación que recomienda cines, primero se evaluarían recomendaciones de todos los cines del país, y luego se integra la información de contexto, que en este caso es la

ubicación geográfica, descartando los cines más alejados y entregaría los restantes como recomendación.

- Modelamiento Contextual (Contextual Modeling): En este paradigma de recomendación, se incorpora la información contextual directamente en el modelo utilizado para calcular las predicciones de ratings. Mientras que los enfoques pre-filtrado y post-filtrado contextual pueden utilizar las funciones tradicionales de recomendación 2D, el enfoque de modelamiento contextual da origen a funciones de recomendación multidimensionales, que básicamente representan modelos predictivos (construidas usando árbol de decisión, regresión, modelo probabilístico, u otra técnica) o cálculos heurísticos que incorporan información contextual, además de los datos del usuario e ítem, es decir, Rating = R (Usuario, ítem, contexto).[1].

3. Evaluación.

Muchas de las técnicas de recomendación necesitan un largo proceso de entrenamiento del algoritmo para que este pueda entregar buenas recomendaciones. En la literatura podemos encontrar distintos tipos de experimentos que son utilizados con el fin de probar varios recomendadores. Los tipos de experimentos más utilizados corresponden a la evaluación offline, estudio de usuarios y evaluación online. Las evaluaciones offline son las más fáciles de llevar a cabo, ya que no necesitan la interacción de los usuarios con el sistema durante el experimento. En los estudios de usuario, se pide a un grupo pequeño de personas utilizar el sistema en un entorno controlado, para que luego reporten su experiencia. Finalmente, los experimentos que quizás entregan resultados más fiables corresponden a las evaluaciones online, donde el sistema es usado por usuarios reales. Este tipo de evaluación es lo ideal, pero tiene un alto costo asociado.

3.1 Protocolos de evaluación.

Actualmente hay una gran diversidad de protocolos de evaluación de desempeño de sistemas de recomendación. Realizaremos una breve comparación entre dos de ellos, la evaluación online y la evaluación offline, para luego centrarnos en este último.

3.2 Evaluación online.

Se realiza con personas reales probando el sistema, llenando cuestionarios sobre cómo fue su experiencia usando el sistema y recibiendo recomendaciones bajo diferentes configuraciones. La evaluación online puede ser considerada como preferible, ya que es realizada con usuarios reales, pero presenta inconvenientes, por ejemplo, la necesidad de tener una interfaz gráfica para que el usuario interactúe con el sistema, y la necesidad de personas reales que estén dispuestas a probar el sistema, que probablemente cobrarán por su trabajo.

3.3 Evaluación offline.

La evaluación offline se realiza sin usuarios reales, en su lugar se utilizan conjuntos de datos con información de preferencias históricas de usuarios sobre ítems, estos conjuntos de datos serán llamados desde aquí en adelante “Datasets”.

Una importante ventaja de la evaluación offline, es que no necesita una interfaz gráfica para llevarse a cabo, ya que solo se deben probar los algoritmos. Este trabajo se enfocará en este tipo de evaluación.

En primer lugar se necesita un Dataset que llamaremos M, los Ratings de este Dataset serán divididos en dos conjuntos, llamados set de entrenamiento o *training* (Tr) y set de pruebas o *test* (Te), con la condición de que $Tr \cap Te = \emptyset$, este proceso se conoce como “Data-Splitting”. Los Ratings de Te serán ocultados al algoritmo recomendador en evaluación, y los Ratings de Tr serán usados para simular el comportamiento humano para construir el recomendador. Luego, se pedirá al recomendador que sugiera un conjunto de ítems para cada usuario, estos ítems serán comparados con los del set de prueba, para comprobar que tan acertadas son las recomendaciones.

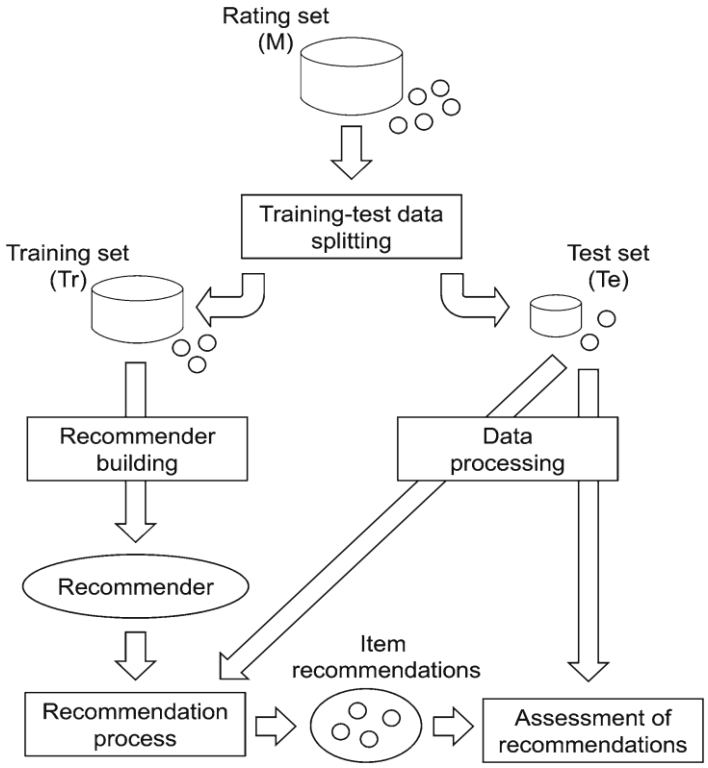


Figura 5. Esquema de las etapas genéricas del protocolo de evaluación offline [9].

3.4 Tareas de recomendación.

En la literatura sobre sistemas de recomendación normalmente es posible distinguir entre dos formas de medir la bondad de las recomendaciones: predicción de rating, a menudo medida en términos del error cuadrático medio (RMSE); y el ranking, que se mide en términos de métricas como la *precision* y *recall*, entre otros.

Recientemente se ha argumentado, sin embargo, que las métricas de precisión de ranking son más adecuados para los propósitos de recomendación, ya que normalmente los sistemas de recomendación presentan un número limitado de los ítems más atractivos para un usuario, en lugar de las predicciones de rating para los ítems individuales [15]. Para ello, en general, un ranking de ítems se construye comparando (y clasificando) predicciones de rating, y los ítems en el ranking Top-N (los N mejores ítems) son considerados como recomendaciones. Las métricas de precisión de ranking miden hasta qué punto la lista de recomendaciones contiene ítems relevantes para los usuarios [9].

En general, las métricas de precisión de predicción de rating se utilizan para evaluar una tarea de predicción de rating, mientras que las métricas de precisión de ranking se utilizan para evaluar una tarea recomendación top-N.

3.4.1 Predicción de rating.

El error cuadrático medio (RMSE) es por lejos la medida de la precisión más popular en la literatura de recomendación [16]. Está asociado comúnmente con el objetivo de la predicción de rating, es decir, predecir el valor de rating que un usuario podría asignar a un ítem que él / ella no ha evaluado aún.

3.4.1.1 Precisión de predicción.

En la base de la mayoría de los sistemas de recomendación se encuentra un motor de predicción. Este motor puede predecir los ratings de los usuarios sobre los ítems (por ejemplo, ratings de películas) o la probabilidad de uso (por ejemplo, la probabilidad de compra).

Se cree que un sistema de recomendación que proporciona predicciones más precisas será preferido por el usuario. Por lo tanto, muchos investigadores se han propuesto encontrar algoritmos que proporcionan mejores predicciones.

La precisión de la predicción es típicamente independiente de la interfaz de usuario, y por lo tanto se puede medir en un experimento offline.

3.4.1.2 Medición de precisión de predicción de ratings.

En algunas aplicaciones, como el popular servicio de streaming de video Netflix, se quiere predecir la calificación que un usuario le daría a un ítem (por ejemplo, en un rango 1 a 5 estrellas). En tales casos, se desea medir la precisión de los ratings predichos por el sistema [1].

El error cuadrático medio (RMSE) es quizás el indicador más utilizado para evaluar la precisión de predicción de ratings. El sistema genera la predicción de rating $\hat{r}_{u,i}$ para un conjunto de pruebas $\mathcal{T}$ de pares usuario-ítem (u, i) para los que se conocen el verdadero rating $r_{u,i}$. Típicamente, $r_{u,i}$ son conocidos porque están ocultas en un experimento offline, o porque se obtuvieron a través de un estudio de usuarios o experimento online. El RMSE entre el rating real y el predichoo está dado por [1]:

$$\mathrm{RMSE} = \sqrt{\frac{1}{|\mathcal{T}|} \sum_{(u,i)\in\mathcal{T}} (\hat{r}_{ui} - r_{ui})^2}$$

El valor absoluto medio (MAE) es otra métrica conocida para medir la precisión de predicción, y está dada por:

$$\mathrm{MAE} = \sqrt{\frac{1}{|\mathcal{T}|} \sum_{(u,i)\in\mathcal{T}} |\hat{r}_{ui} - r_{ui}|}$$

Comparando MAE y RMSE, este último tiende a penalizar los errores más grandes más severamente que otras métricas [1].

El RMSE promedio y MAE promedio sirve para normalizar conjuntos de prueba no balanceados. Por ejemplo, si el conjunto de prueba tiene una distribución no uniforme de ítems, el RMSE o MAE obtenido podría estar fuertemente influenciado por el error en algunos ítems muy frecuentes. Si necesitamos una medida que sea representativo del error de predicción en cualquier ítem, es preferible calcular MAE o RMSE por separado para cada ítem y luego tomar el promedio de todos los ítems.

RMSE y MAE depende sólo de la magnitud de los errores cometidos. En algunas aplicaciones, la semántica de los ratings pueden ser tales que el impacto de un error de predicción no depende sólo de su magnitud. Por ejemplo en una aplicación que clasifique ítems en dos categorías: “Recomendable” o “No recomendable”, dependiendo de si su rating es superior o inferior a 4 (en

una escala de rating de 1 a 5). Si nuestra predicción arroja el valor 3,9 cuando el valor real era 4,1, la magnitud numérica del error sería muy baja (0,2), pero el impacto sería alto, dado que clasificaría el ítem en la categoría incorrecta.

3.4.1.3 Medición de predicción de uso.

Muchos sistemas de recomendación no predicen las preferencias de ítems del usuario, tales como los ratings de películas, pero trata de recomendar a los usuarios ítems que pueden utilizar. En una evaluación offline de predicción de uso, por lo general se tiene un conjunto de datos que consiste en ítems que cada usuario ha utilizado. Luego se selecciona un usuario de prueba, se esconden algunas de sus selecciones, y se pide al recomendador predecir un conjunto de ítems que el usuario va a utilizar. Entonces se tienen cuatro resultados posibles para los ítems recomendados y ocultos, como se muestra en la siguiente tabla:

	Seleccionado	No seleccionado	Total
Relevante	N_{rs}	N_{rn}	N_r
Irrelevante	N_{is}	N_{in}	N_i
Total	N_s	N_n	N

Clasificación del posible resultado de una recomendación de un elemento a un usuario.

En el caso offline, ya que los datos no se recogen durante el proceso de evaluación, nos vemos obligados a asumir que los elementos no utilizados, no habrían de ser usado, aun cuando hubiese sido recomendado - es decir, que son poco interesantes o inútiles para el usuario. Esta hipótesis puede ser falsa, tales como cuando el conjunto de elementos irrelevante, contiene algunos elementos relevantes que el usuario no ha seleccionado. Por ejemplo, un usuario puede no haber utilizado un elemento porque este no tenía conocimiento de su existencia, pero después de la recomendación expuesta ese elemento el usuario puede decidir seleccionarlo.

Podemos contar el número de ejemplos correspondiente a cada celda de la tabla y calcular las siguientes cantidades:

- Precision: $\frac{\mathrm{Nrs}}{Ns}$
- Recall (True Positive Rate): $\frac{Nrs}{Nr}$

Por lo general podemos esperar una compensación entre estas cantidades – al usar listas de recomendación más largas normalmente mejora el recall, pero también es probable que se reduzca la precisión, dado que la precisión representa la probabilidad que un ítem seleccionado sea relevante, mientras que recall representa la probabilidad un ítem relevante sea seleccionado. En aplicaciones donde el número de recomendaciones que se pueden presentar al usuario están pre-ordenadas (dado que los ítems relevantes se encuentran más arriba de la lista y las menos relevantes al final), la medida más útil es la de precisión en N, es decir, en los N primeros ítems de la lista [1].

En otras aplicaciones en las que el número de recomendaciones que se presentan al usuario no está predeterminado, es preferible evaluar algoritmos sobre un rango de longitudes de las listas de recomendaciones, en lugar de utilizar una longitud fija.

3.4.2 Medición de Ranking.

En muchos casos, la aplicación presenta al usuario una lista de recomendaciones. Por ejemplo en Netflix, las pestaña "películas que te encantarán", muestra un conjunto de categorías, y en cada categoría, una lista de películas que el sistema predice que el usuario desea. Estas listas pueden ser largas y el usuario puede tener que seguir "páginas" adicionales hasta que la lista entera sea explorada. En estas aplicaciones, no interesa la predicción de un rating explícito, o la selección de un conjunto de ítems recomendados, como en las secciones anteriores, sino más bien ordenar los elementos de acuerdo a las preferencias del usuario. Esta tarea se conoce normalmente como un ranking de ítems. Hay dos enfoques para medir la exactitud de tales rankings. Se puede tratar de determinar el orden correcto de un conjunto de ítems para cada usuario y medir qué tan cerca un sistema está de este orden correcto, o se puede tratar de medir la utilidad del ranking del sistema para un usuario.

3.4.2.1 Usando un ranking de referencia.

Con el fin de evaluar un algoritmo de ranking con respecto a un ranking de referencia (de un orden correcto), primero es necesario obtener esa referencia. En los casos donde los rating explícitos de usuarios de ítems disponibles, se pueden clasificar los ítems evaluados por orden de ratings decreciente. En los casos en que sólo se tienen datos de uso, puede ser apropiado para construir un ranking de referencia donde los ítems utilizados por el usuario son clasificados por encima de ítems no utilizados. Sin embargo, esto sólo es válido si se sabe que el usuario es consciente de los ítems no utilizados, por lo que se puede inferir que el usuario realmente prefiere los ítems usados a los ítems no utilizados.

3.4.2.2 Ranking basado en utilidad.

La utilidad de cada recomendación es la utilidad del ítem recomendado descontada por un factor que depende de su posición en la lista de recomendaciones. Un ejemplo de una utilidad es la probabilidad de que un usuario podrá observar una recomendación en la posición i en la lista [1]. Generalmente se asume que los usuarios recorren las listas de recomendación desde el principio hasta el final, disminuyendo la utilidad de las recomendaciones en mayor medida hacia el final de la lista. El descuento también se puede interpretar como la probabilidad de que un usuario observaría una recomendación en una posición particular en la lista, mediante la utilidad de la recomendación ya que se observó que sólo depende del ítem recomendable. Según ésta interpretación, la probabilidad de sea observada una posición en particular en la lista de recomendaciones se supone que dependerá sólo de la posición y no en los ítems que se recomiendan.

3.5 Metodologías de evaluación

Algunas etapas de la evaluación de sistemas de recomendación pueden ser implementadas de diferentes formas, lo que conduce a diferencias en las metodologías de los diferentes estudios. Un importante factor en estas diferencias es en el proceso de *Splitting*, donde se separan los ratings en entrenamiento y prueba, especialmente cuando se dispone de información de contexto, como las marcas de tiempo.

En el proceso de *Splitting* entran en juego una serie de condiciones que determinan de qué manera se hará la separación de datos en entrenamiento y prueba, tales como: *Base set condition*, que determina si los *Splits* se harán basados en la matriz de datos completa o en cada usuario, *Rating order condition*, que determina si se ordenan los ratings bajo algún criterio antes de separarlos, *Size condition*, que define cuantos ratings irán a entrenamiento y cuantos a prueba. Estas condiciones se explican a continuación, de acuerdo a lo indicado en [9].

Definiciones generales:

Definición 1: Un *Split* de una matriz de rating M es una partición de M en ratings de entrenamiento (Tr) y ratings de prueba (Te) de tal forma que $Tr \cap Te = \varnothing$.

Definición 2: El procedimiento de *Splitting* es un algoritmo que recibe los parámetros M, B, O, S; donde M es la matriz de ratings, B es una *Base set condition*, O es una *Rating order condition* que define como se ordenarán los ratings y S una *Size condition* que define el tamaño de los set de entrenamiento y prueba. Este algoritmo retorna un Split generado de acuerdo a las condiciones recibidas.

Definición 3: El *Base set*, es el conjunto de datos a partir del cual se generarán los Splits, puede ser la matriz de ratings completa, o parte de ella. Se consideran dos *Base set conditions*, una será basada en usuarios y otra basada en comunidad.

Cuando se usa el enfoque basado en usuario se crea un *Base set* individual para cada usuario, y se crea un *Split* para cada uno de ellos de acuerdo a la *Size condition*. De esta forma, por ejemplo, si se tiene una *Size condition* proporcional de 50% de entrenamiento y 50% de prueba, habría que aplicar esa regla para cada usuario, como lo muestra la figura 6, asegurando además que todos tengan ratings en el set de entrenamiento y en el set de pruebas:

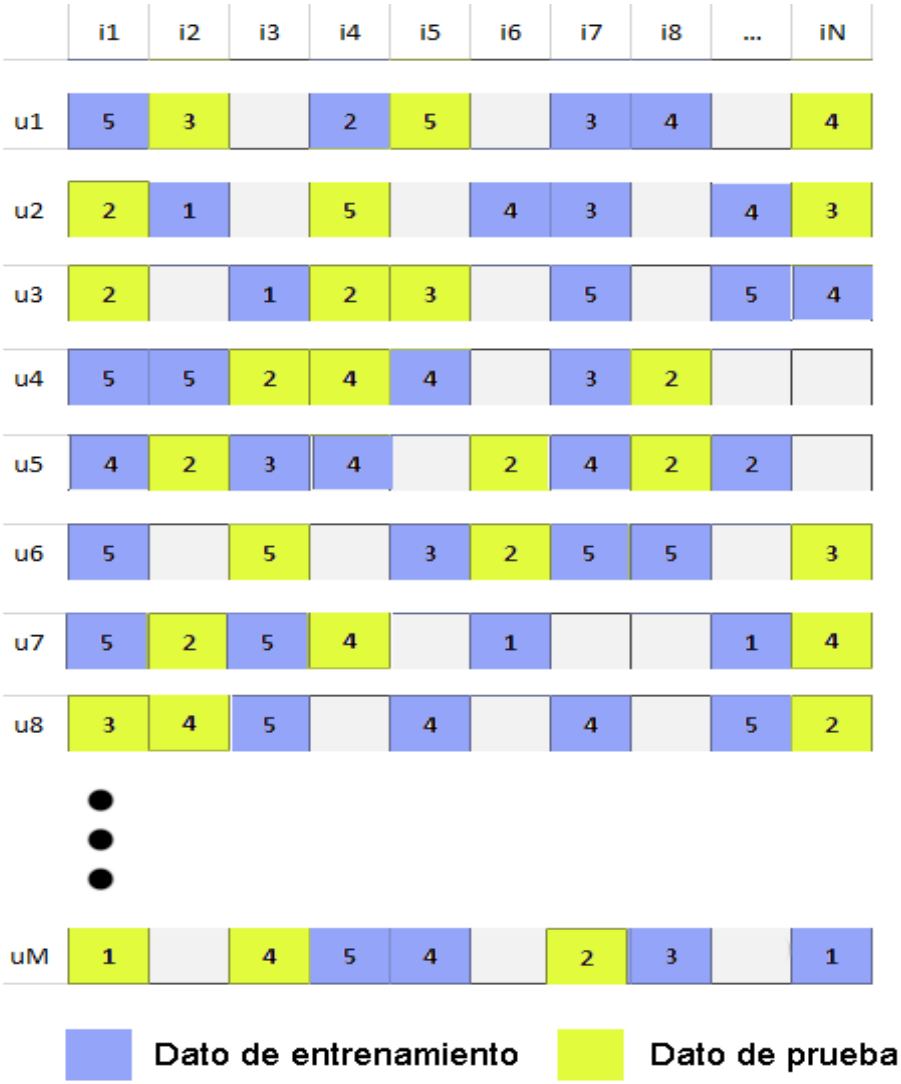


Figura 6. Matriz de datos que representa el enfoque centrado en usuario, de forma aleatoria.

Cuando usamos el enfoque basado en comunidad se toma la matriz de datos como un todo, y sobre ella se aplica la Size condition, por lo que usando el mismo ejemplo anterior de 50% de datos de entrenamiento y 50% de prueba podríamos obtener la siguiente representación:

Figura 7. Matriz de datos que representa el enfoque basado en comunidad.

De esta manera tenemos la misma cantidad de datos para entrenamiento y test que usando el enfoque basado en usuarios, sin embargo, se puede dar la situación de que para ciertos usuarios queden todos o ninguno de sus ratings en el conjunto de prueba (como en el caso del usuario 5), de esta forma, sería imposible evaluar el rendimiento del sistema de recomendación para estos usuarios.

Definición 4: La *Size condition* establece la cantidad de ratings que serán usados para construir el set de entrenamiento y el set de pruebas. Esta condición puede ser de tipo proporcional o fija.

Si se hace de manera proporcional se establece el porcentaje de datos del *Base set* que se usarán para entrenamiento y prueba, por ejemplo usar el 80% del Base set en entrenamiento y el 20% en pruebas.

En el caso de usar una *Base set condition* enfocada en usuarios, este método asegura tener un buen balance de ratings para cada usuario, esta combinación es usada en Zheng and Li (2011) [17].

Si en cambio se elige una *Size condition* fija, se debe indicar el número de ratings que se desea para pruebas y los restantes serán de entrenamiento.

Cuando se usa la combinación de *Size condition* fija y *Base set* enfocada en usuarios, se puede dar el caso de que algunos usuarios tengan muy pocos datos, para estos casos particulares se puede aplicar la *Size condition* proporcional. Por ejemplo, en la "Netflix Prize competition", se estableció como valor fijo el de 9 ratings por usuario para pruebas, y en los casos que el total de ratings del usuario fuera menor a 18, se cambiaba a una *Size condition* proporcional de 50% y 50% [18].

Definición 5: La *Rating order condition* establece el tipo de ordenamiento que se aplicará a los ratings para generar los set de entrenamiento y prueba en un *Split*. En la literatura se identifican dos tipos de *rating order condition* para los sistemas de recomendación basados en contexto temporal (TARS): Los tiempo-independientes y los tiempo-dependientes.

Si se usa la *rating order condition* tiempo-independiente, las marcas de tiempo que podría tener cada rating no serán consideradas, podría quizás usarse otro tipo de ordenamiento donde el tiempo no sea un factor, pero en los TARS en general, si se selecciona esta opción, los ratings serán tomados de forma aleatoria.

La mayor ventaja de esta *order condition* es que se puede aplicar siempre, ya que no necesita que los ratings dispongan de marcas de tiempo, pero una desventaja es que en la creación de sets de entrenamiento y prueba, podrían incluirse en el set de prueba ratings generados después que algunos del set de entrenamiento, lo que en el mundo real sería equivalente a tener información sobre las preferencias futuras del usuario (Campos et al. 2011b) [9].

En la caso de la *rating order condition* tiempo-dependiente, los ratings son ordenados según sus marcas de tiempo, y los *Splits* son generados de manera que todos los ratings del set de prueba tengan fecha superior a los del set de entrenamiento, solucionando así el problema de las preferencias futuras. Una desventaja de este método es que los ratings deben disponer de marcas de tiempo.

3.5.1 Condiciones de validaciones cruzadas.

Al realizar la evaluación de un algoritmo, puede haber variaciones en los resultados obtenidos dependiendo de los valores específicos asignados a los sets de entrenamiento y prueba. Para tener una mejor estimación de los resultados que obtendría un algoritmo sobre un nuevo conjunto de datos, se ha propuesto el repetir los experimentos utilizando cada vez sets de entrenamiento y prueba diferentes (es decir con diferentes *Splits* de datos), pudiendo obtenerse así un resultado promedio. Este proceso es lo que comúnmente se llama “validación cruzada” [21], (Dietterich 1998) [19].

A continuación se explica a grandes rasgos algunos métodos que serán divididos en métodos tiempo-Independientes y Métodos tiempo-Dependientes. En el Capítulo 5 serán vistos en detalle.

3.5.2 Métodos de validación cruzada tiempo-independientes.

Son métodos de validación cruzada en donde no se utilizan las marcas de tiempo de cada evaluación para el ordenamiento de Ratings, aunque se dispongan de ellas.

Repeated sampling: Consiste en repetir el procedimiento de *Split* X veces (donde X es la cantidad de repeticiones del experimento), usando un ordenamiento aleatorio, asegurando que cada *Split* sea diferente al resto. En Gordea y Zanker (2007) [20] se aplica este método.

User resampling: Este método toma subconjuntos de Usuarios en cada repetición y aplica un *Split* sobre ellos. En Zheng and Li (2011) [17] se aplica este método.

X-fold cross validation: En este método, a partir del conjunto de datos M, se crean X subconjuntos (llamados *Folds*), formando así X sets de entrenamiento y sets de prueba. Luego se selecciona uno de ellos como set de prueba, y los restantes (X-1), como sets de entrenamiento. En general los *Folds* son de igual tamaño, por lo tanto, el valor de X determinará el tamaño de los sets de entrenamiento y prueba. En Adomavicius et al. (2005) [5] se aplica este método.

Leave-one-out: Es un caso particular de *X-Fold cross validation*, en donde X = |M|. Un rating en M es considerado como set de prueba, y el resto como set de entrenamiento en cada repetición,

por lo tanto se harán tantas repeticiones, como Ratings se tengan. Este método es el más exhaustivo de los métodos clásicos de validación cruzada [21], y por ello es altamente costoso computacionalmente, por lo que es poco factible en muchas situaciones. En Cremonesi y Turrin (2009) [22] aplica este método.

3.5.3 Métodos de validación cruzada tiempo-dependientes.

Time-dependent resampling: Este método consiste en seleccionar X diferentes conjuntos de ratings y aplicar el procedimiento de *Split* a cada uno de ellos, separando set de entrenamiento y prueba con una variable de tiempo. En Hermann (2010) [23] se usa este método.

Time-dependent users resampling: Este método es similar a *User Resampling*, pero usando una *rating order condition* tiempo-dependiente. X *Splits* son construidos (uno por cada conjunto de usuarios seleccionado). En Cremonesi and Turrin (2010) [24] se usa este método.

Increasing-time window: En este método se define una ventana de tiempo para seleccionar los datos correspondientes al set de entrenamiento y una ventana para el set de pruebas. La ventana de pruebas se mantendrá fija, pero la de entrenamiento irá aumentando en cada *Split*. En Lathia et al. (2009) [25] se usa este método.

Fixed-time window: En este método se define una ventana de tiempo para entrenamiento y otra para prueba, las cuales irán avanzando de manera fija en cada Split. En Pradel et al. (2011) [26] se usa este método.

4. Descripción del framework a extender.

En este capítulo se describe el framework de evaluación que será extendido con los métodos de validación cruzada mencionados en el capítulo anterior. Dado que este framework implementa las condiciones de evaluación descritas en el capítulo 3, facilita la implementación de los métodos de validación cruzada antes presentados.

Cabe mencionar que este proyecto, particularmente en lo que corresponde a evaluación de TARS, se basa en el paper *Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols* [9] e incorporando otras librerías de sistemas de recomendación como las de Apache Mahout[4].

El código fuente del framework se encuentra desarrollado en el lenguaje Java, donde sus principales clases contienen distintos algoritmos de recomendación y métodos de evaluación, clases encargadas de construir recomendadores y metodologías de evaluación, definir las variables para cada algoritmo implementado, y generar la visualización de los resultados.

4.1 yaREF.

EL paquete yaREF (*yet another Recommender Evaluation Framework*) es el paquete principal del framework, y contiene los principales paquetes (módulos o librerías) utilizados en el framework, donde estas librerías pasan a ser una dependencia, y son sub-módulos de yaREF (Ver figura 8). A continuación se describe cada una de las librerías que componen el framework, así como sus paquetes y clases principales.

[4] Librería de algoritmos de recomendación Apache Mahout, disponible en: http://mahout.apache.org

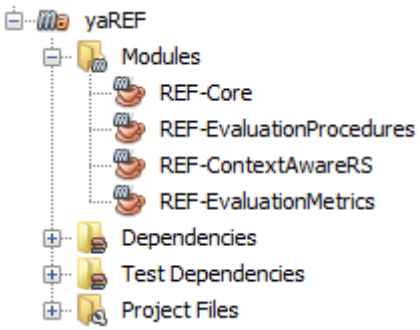


Figura 8. Estructura de yaREF.

4.1.1 REF-Core.

Este módulo está compuesto por dos paquetes principales (ver figura 9). Se puede considerar el paquete *core* el principal del proyecto yaREF, dado que sus clases son las encargadas de asignarle un valor al contexto, definir el tipo de contexto, asi como comparar contextos.

Figura 9. Paquetes principales del proyecto REF-Core.

La estructura del paquete *core* está compuesto por otros 5 paquetes (ver figura 10), en el cual *context* contiene clases encargadas asignarle un valor y un nombre al contexto, obtener el valor y el nombre de un contexto, comparar rating dado un listado de preferencias, comparar contextos, y definir si un contexto es vacío. En el paquete *model* podemos encontrar varias interfaces con distintos métodos definidos, los cuales son implementados en las clases dentro del paquete *impl*. El paquete *impl* lo podemos encontrar en el paquete *model*, y las clases que contienen implementan las interfaces del *impl*. Los métodos de estas clases se encargan de obtener el contexto, usuario, ítem, valor (de un contexto) dado un listado de preferencias tanto implícitas como explícitas. En el paquete *similarity* encontramos una interface que define dos métodos los cuales son implementados en la clase *PearsonWeightedSimilarity* encargados de obtener la similitud de ítems, y usuarios.

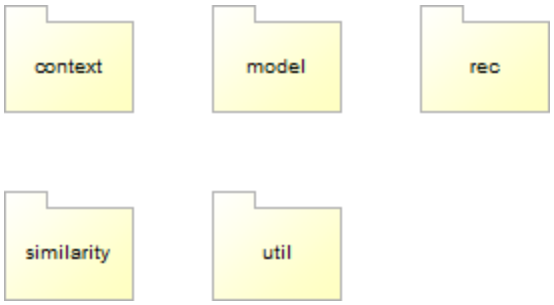


Figura 10. Contenido paquete core.

El paquete *context* está compuesto por las interface CategoricalContextDefinitionIF, ContextIF, ContinuousTimeContextIF, y las clases CategoricalContext, ContextContainer, ContextDefinition, ContinuousTimeContext, EmptyContext, RatingPreferenceComparator (ver figura 11). Donde la clase ContextDefinition contiene los métodos setName que se encarga de asignar un nombre a un contexto, y addValue que se encarga de asignar un valor a un contexto, e implementa los métodos de la interface CategoricalContextDefinitionIF y el método Comparable. Por otra parte, las clases ContextContainer, EmptyContext, y CategoricalContext implementan la interface ContextIF, mientras que la clase ContinuousTimeContext implementa el método de la interface ContinuousTimeContextIF. La clase RatingPreferenceComparator implementa la interface ContinuousTimeContextIF, y contiene el método *compare*, que se encarga de comparar las marca de tiempo de dos preferencias de usuario.

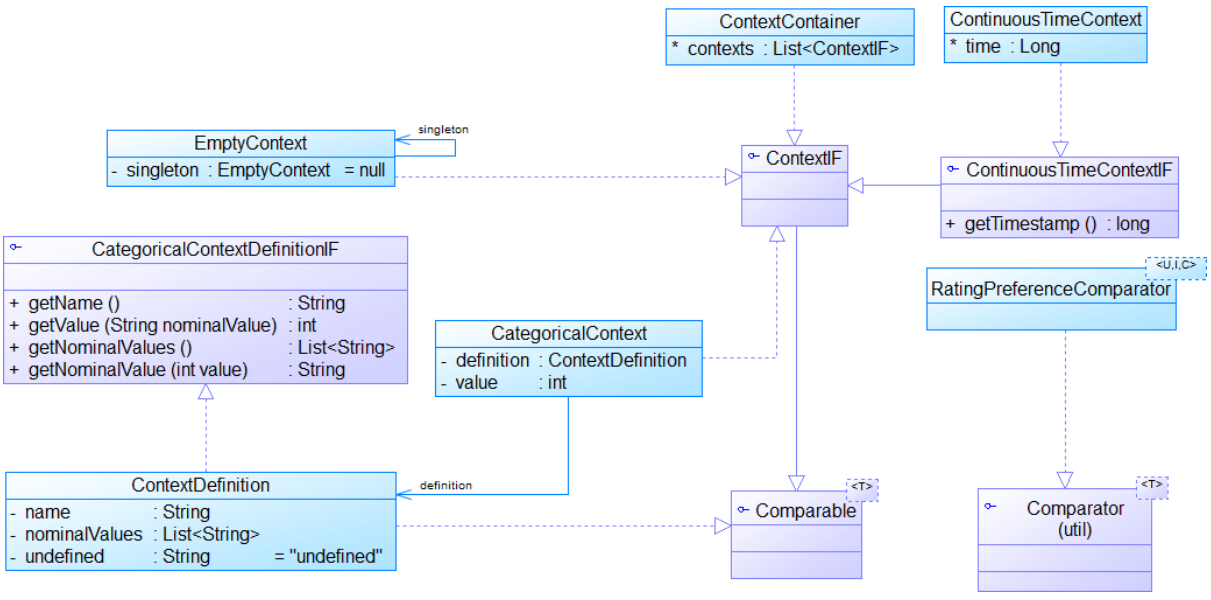


Figura 11. Diagrama de clases paquete context.

Como podemos ver en la figura 10, en el módulo REF-Core también encontramos el paquete *split* el cual está compuesto por el paquete *impl* y la interface SplitIF, que tiene dos métodos definidos (ver figura 12). Los métodos definidos en SplitIF son implementados en la clase Split que se encuentra dentro del paquete *impl*.

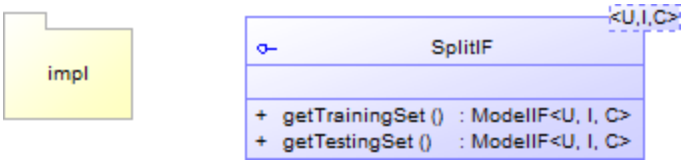


Figura 12. Contenido paquete split.

La clase Split encarga de implementar los métodos definidos en la interface SplitIF, donde el método *getTrainingSet()* se encarga de obtener un subconjunto de datos de training, mientras que *getTestingSet()* se encarga de obtener un subconjunto de datos de test, los cuales son utilizados al momento de realizar el split[5] de los datos para realizar una evaluación.

[5] Un Split de una matriz de rating M es una partición de M en ratings de entrenamiento (Tr) y ratings de prueba (Te) de tal forma que Tr ∩ Te = ∅.

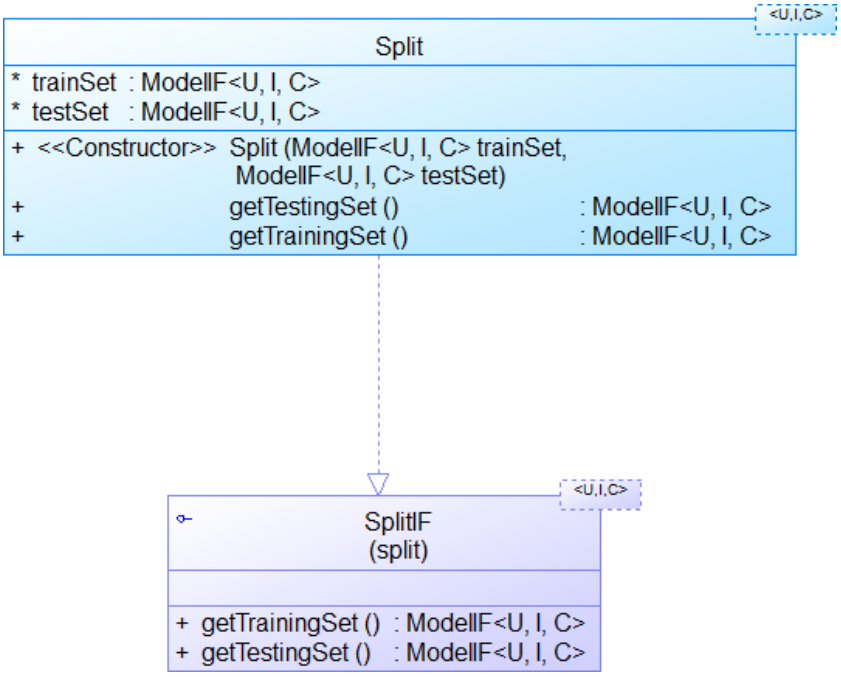


Figura 13. Diagrama de clases del paquete impl.

4.1.2 REF-EvaluationProcedures.

Este proyecto contiene métodos de evaluación y clases encargadas de filtrar los datos que utilizaremos (usuarios, ítems, ratings). REF-EvaluationProcedures se compone de 4 paquetes; *filter*, *nonpersonalized*, *rank*, *split*. Donde las clases del paquete *filter* se encargan de asignar el rating mínimo por usuario, rating mínimo por ítem, valor mínimo de rating, y añadir a una lista usuario, ítem, valor, y contexto, si se conocen la preferencia del ítem. Las clases del paquete *nonpersonalized* se encargan tanto de remover ítems calificados para training (removeTrainingRatedItemsBy), obtener un conjunto relevante y no relevante de ítems dado una comunidad de ítems de training y test. Mientras que el paquete *split* se compone de 4 paquetes y una interface DatasetSplitterIF que define un método split (Ver figura 14). Cada uno de estos paquetes contiene distintas interfaces y clases, que ayudan a construir las metodologías de evaluación dado un: *base set condition*, *size condition*, *rating orden* condition.

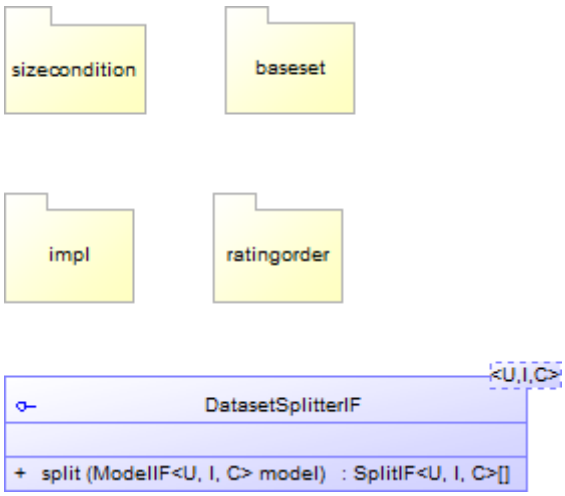


Figura 14. Contenido paquete split (REF-EvaluationProcedures).

Los paquetes *sizecondition*, *baseset*, y *ratingorder* son muy importante para poder construir los métodos de evaluación. En el paquete *sizecondition* encontramos dos interfaces; SizeConditionIF que contiene definido un método, y SizeConditionDateBasedIF que contiene dos métodos definidos, y 3 clases; SCFixed, SCProportion, y SCTimeProportion (ver figura 15), las cuales implementan la interface SizeConditionIF, que nos permiten calcular el número de ratings que serán asignados a training y test, en donde cada clase realiza la distribución a partir de una condición de tamaño. En donde la clase SCFixed establece que un número fijo de los ratings se utilizan como datos de prueba, mientras que SCProportion establece una proporción de los ratings serán utilizados como datos de prueba, y los ratings restantes se utilizan como datos de entrenamiento, y en la clase SCTimeProportion utiliza una condición de tiempo dependiente, lo cual establece un umbral de tiempo que se utiliza para la distribución de los datos para cada conjunto de training y test.

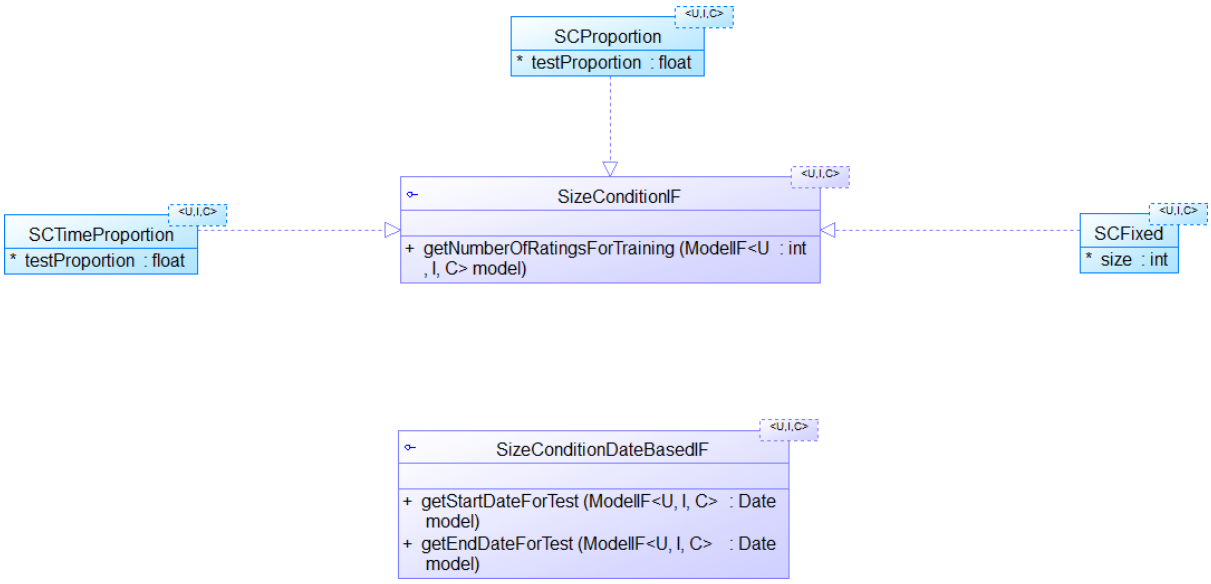


Figura 15. Diagrama de clases paquete sizecondition.

El paquete *baseset* contiene la interface BaseSetGeneratorIF que tiene definido el método getBaseSets y las clases BSCommunity y BSUser, ambas implementan el método definido en la interface (ver figura 16). Estas clases consideran dos enfoques de *base condition*. El primero corresponde a la condición centrada en comunidad[6] (BSCommunity). Cuando se aplica esta condición, algunos usuarios pueden tener todos o ninguno de sus rating en el conjunto de training. Este problema se debe a las grandes diferencias en los patrones de ratings entre los usuarios, donde algunos usuarios poseen más ratings que otros, y también diferentes distribuciones de rating lo largo del tiempo. El segundo enfoque corresponde a la condición centrada en usuario[7] (BSUser), mediante la realización de la división independientemente de los ratings de cada usuario, se puede asegurar que todos los usuarios tendrán ratings, tanto en los conjuntos de training y test.

[6] Un único conjunto de datos base con todas las preferencias en M se utiliza. Cuando se aplica esta condición, algunos usuarios pueden tener todas o ninguna de sus preferencias en el conjunto de prueba.

[7] Un conjunto de datos base M_u se construye con las preferencias de cada usuario u. Mediante la realización de la división independiente de los ratings de cada usuario, se puede asegurar que todos los usuarios tendrán ratings, tanto en los conjuntos de entrenamiento como de prueba.

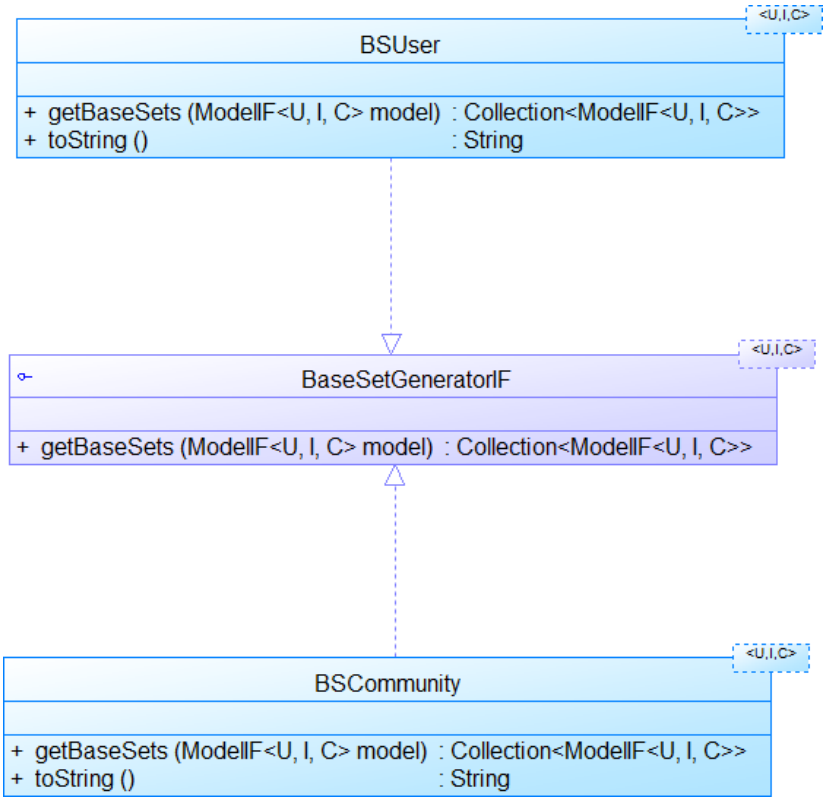


Figura 16. Diagrama de clases paquete baseset.

El paquete *ratingorder* contiene la interface RatingOrderIF que tiene definido el método getOrderedRatings, y las clases RORandom y ROTime, ambas implementan el método definido en la interface (ver figura 17). Estas clases establecen el tipo de ordenamiento que se aplicará en la generación de la secuencia de ratings. Este ordenamiento de los ratings se puede hacer de manera aleatoria (RORandom) u ordenada por tiempo (ROTime).

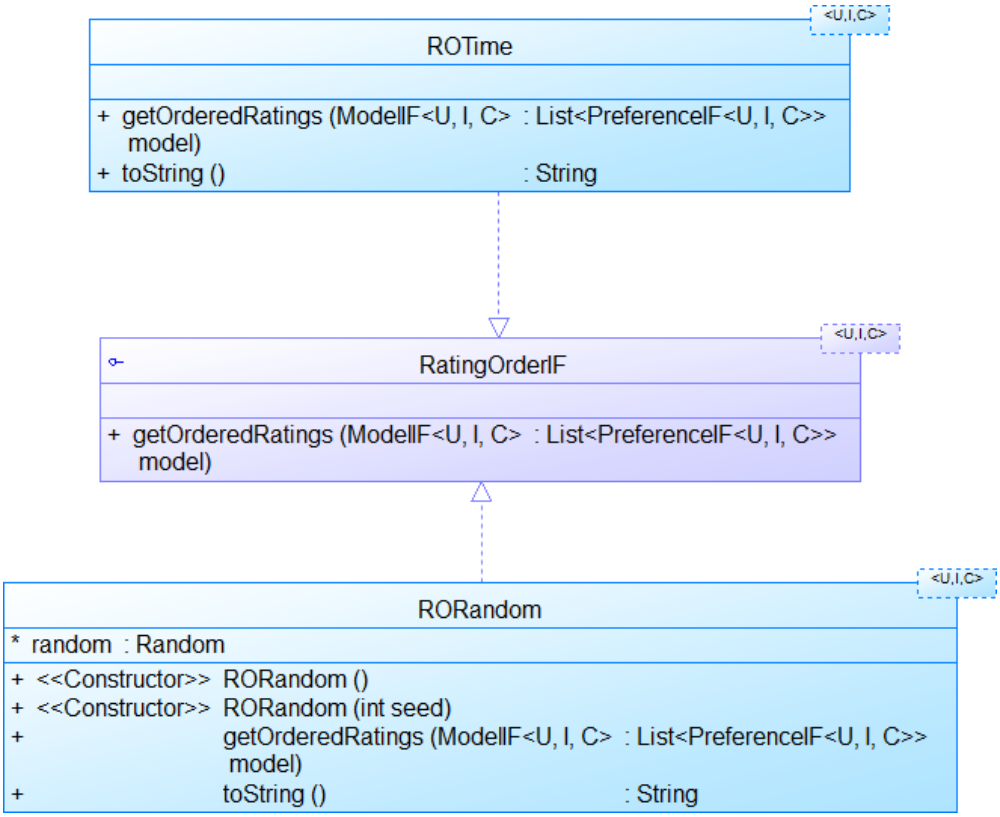


Figura 17. Diagrama de clases paquete ratingorder.

El paquete *impl* es el encargado de contener las clases con los métodos de evaluación, incluyendo validación cruzada. Estas clases implementan la interface DatasetSplitterIF la cual tiene definido el método split, que se encargar de generar los splits de datos obteniendo los conjuntos de training y test. Las clases que posee este paquete son DatasetSplitterBuilder, DatasetSplitter_Holdout, DatasetSplitter_RepeatedSampling, DatasetSplitter_Xfold_CV. Donde la clase DatasetSplitterBuilder es la encarga de construir los distintos métodos de validación cruzada (ver figura 18) dada una *base set condition*, una *rating order condition*, y una *size condition*.

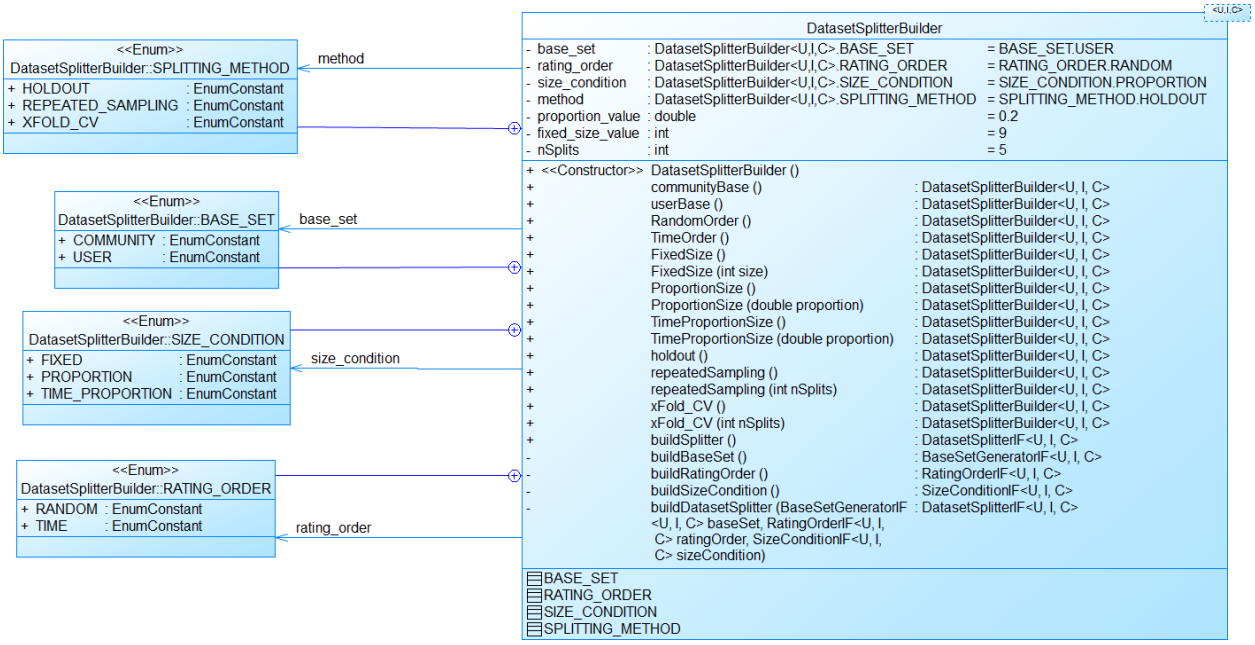


Figura 18. Diagrama de clases paquete impl (REF-EvaluationProcedures).

La clase DatasetSplitter_Holdout implementa la interface DatasetSplitterIF, que se encarga de generar los *splits* con los conjunto de training y test a través del método *getHoldoutSplit* (ver imagen 19), y en particular, se pueden construir distintos tipos de métodos de validación cruzada, según losparámetros reciba la clase DatasetSplitter_Holdout, ya que puede recibir un base set *Community-centered* o *User-centered*, un rating order *Random* o *Time-dependent*, una size condition *Proportion-based*, *Fixed* o *Time-based*.

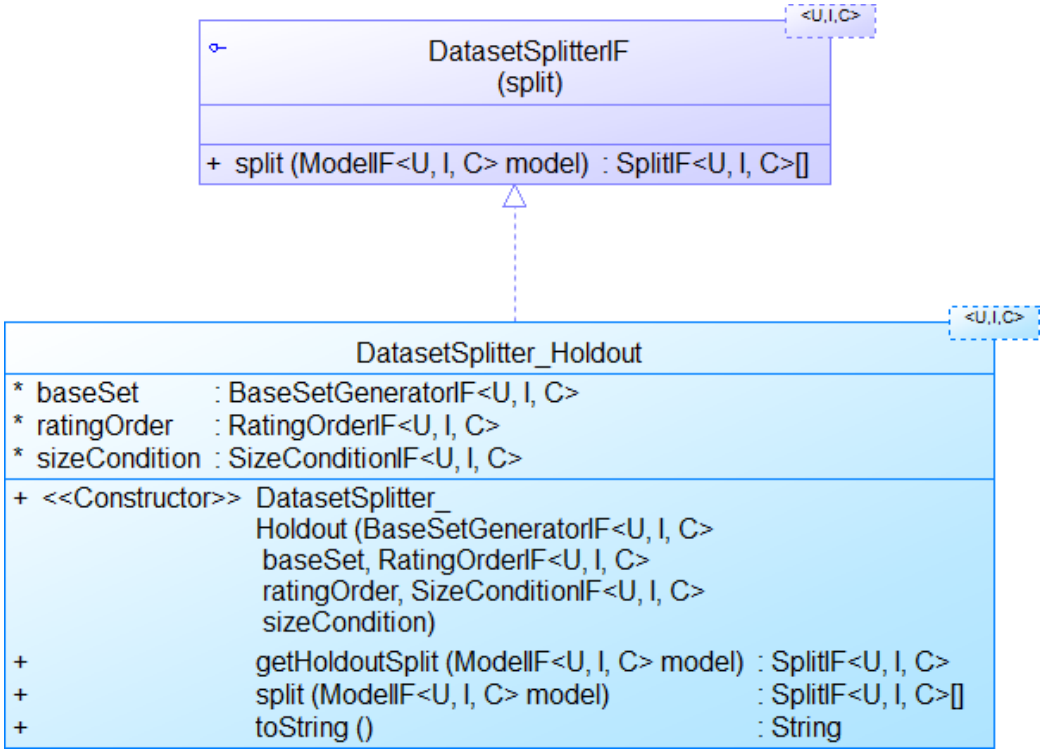


Figura 19. Diagrama de clases DatasetSplitter_Holdout.

La clase DatasetSplitter_Xfold_CV, se base en el método de validación cruzada *X-fold*, formando así X conjuntos de training y test. Luego se selecciona uno de ellos como conjunto de test, y los restantes (X-1), como conjunto de entrenamiento. Esta clase implementa la interface DatasetSplitterIF, que se encarga de generar los *split* con el conjunto de training y test correspondiente (ver figura 20).

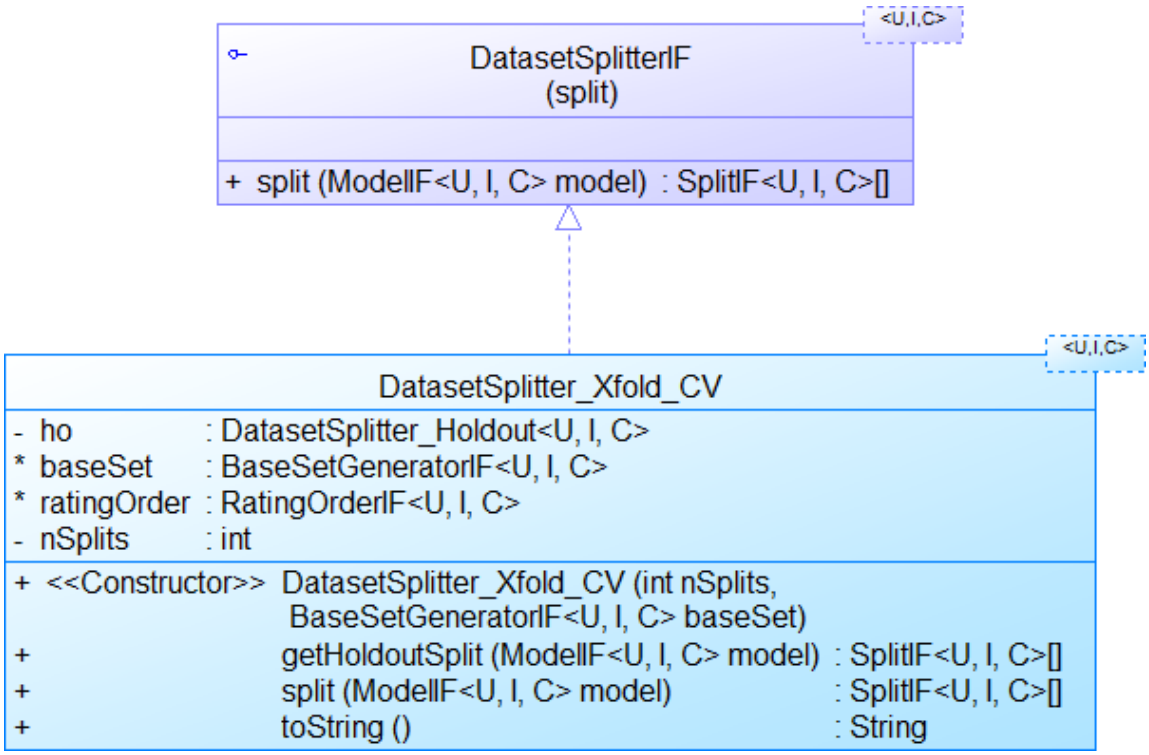


Figura 20. Diagrama de clases DatasetSplitter_Xfold_CV.

La clase DatasetSplitter_RepeatedSampling, se base en el método de validación cruzada Repeated Sampling, el cual construye varios conjuntos de training y test a partir de una cantidad de iteraciones dada la variable *nSplit*. Esta clase implementa la interface DatasetSplitterIF, que se encarga de generar los split con el conjunto de training y test correspondiente (ver figura 21).

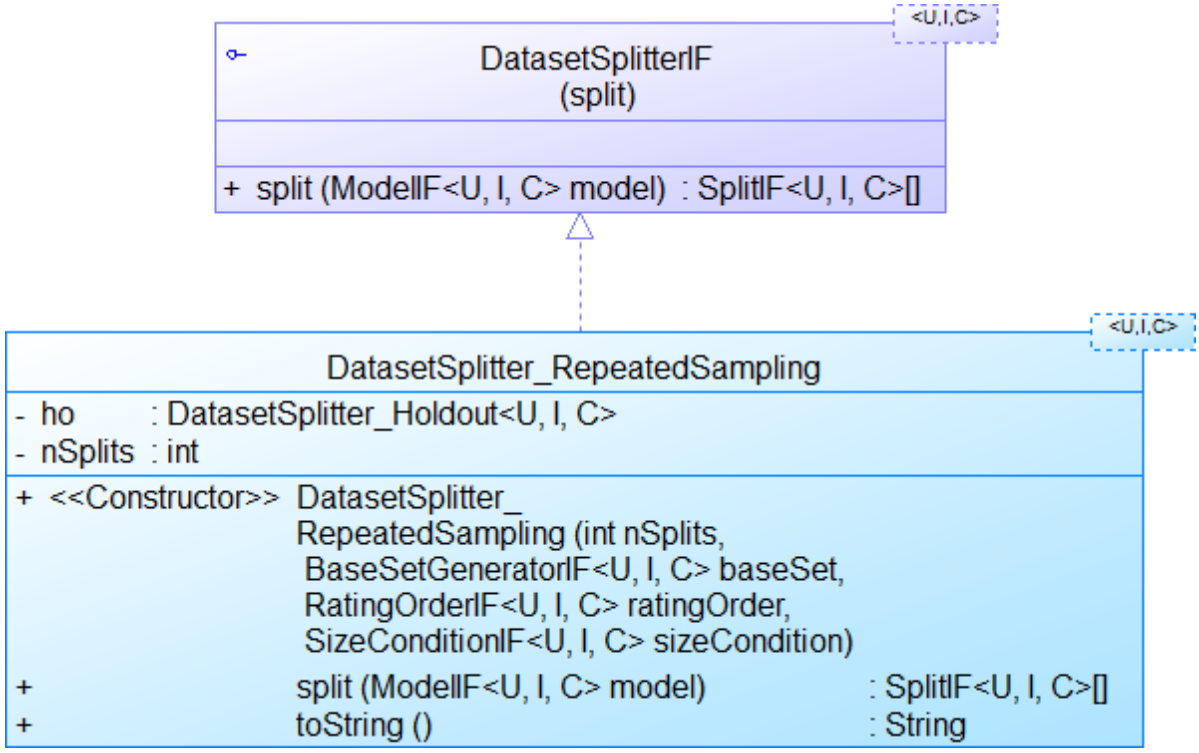


Figura 21. Diagrama de clases DatasetSplitter_RepeatedSampling.

4.1.3 REF-ContextAwareRS.

Este proyecto contiene una gran cantidad de métodos de recomendación (está librería se centra en el método de filtrado colaborativo), modelado de la información contextual (ver capítulo 2), y la clases que se encargan de realizar el proceso de aprendizaje requerido por los métodos de recomendación. Esta librería se compone de tres paquetes principales (ver figura 22). El contenido de estos paquetes se base en distintas clases, las cuales son una adaptación de las clases que podemos encontrar en librerías como Apache Mahout.

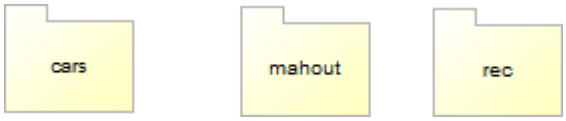


Figura 22. Contenido proyecto ContextAwareRS.

En el paquete *cars* podemos encontrar una nueva distribución de paquetes (ver figura 23) los cuales se encargan de contener las clases e interfaces con los distintos métodos de recomendación, factorización de matrices, modelado de la información contextual, y todo lo necesario para crear un recomendador.

Figura 23. Contenido paquete cars.

En el paquete *contextualfiltering* podemos encontrar las clases con los métodos de recomendación. Sin embargo el paquete que se encarga incluir las clases para construir todos estos métodos, es *recommender*, donde la clase que permite construir el recomendador es NeighborBasedRecommenderBuilder, esta clase implementa el método de la interface RecommenderBuilderIF (ver imagen 24). Los recomendadores se crean conforme a la técnica de recomendación de filtrado colaborativo (CF), ya que han sido ampliamente utilizados para aprovechar el contexto (y, sobre todo, temporal) de la información asociada a los ratings de los usuarios. Siguiendo la clasificación común de los sistemas CF, distinguimos enfoques entre basados en heurísticas (o basados en memoria) y basados en modelos. Según Campos et al. 2013 [9], los algoritmos basados en heurísticos esencialmente calculan las predicciones de ratings directamente de todas los ratings conocidos por medio de una expresión matemática

particular a partir de toda la colección de los perfiles de usuario por medio de ciertas heurísticas. Los enfoques basados en modelo, por otro lado, aprenden un modelo predictivo de la colección de ratings conocidos, esto requiere un proceso de aprendizaje previo en el que se construye el modelo, pero a partir de entonces el modelo genera directamente predicciones de ratings, lo que conduce a una respuesta más rápida a la hora de generar recomendaciones.

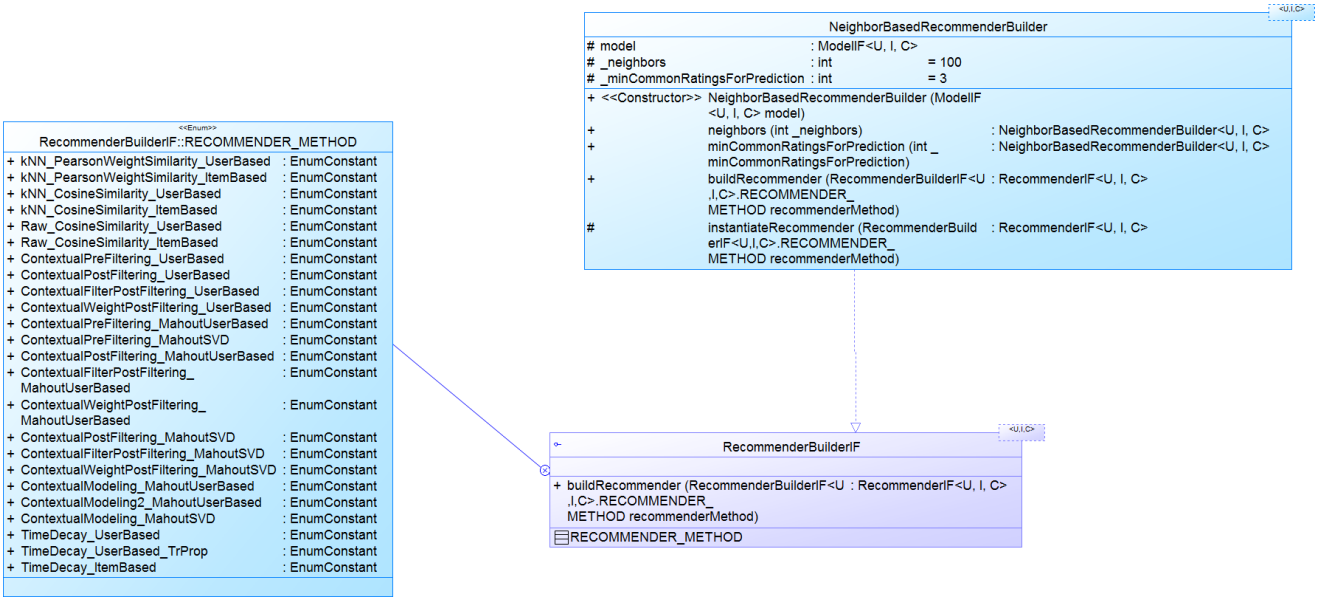


Figura 24. Diagrama de las clases que permiten construir un recomendador.

4.1.4 REF-EvaluationMetrics.

El proyecto REF-EvaluationMetrics contiene diferentes métricas de error, que miden qué tan bien un sistema de recomendación puede predecir los ratings de ítems en particular. REF-EvaluationMetrics incluye tres paquetes principales, varias interfaces y clases (ver figura 25). Las clases que se encuentran en este paquete se encargan de obtener los resultados, y el resultado promedio, de las distintas métricas a través de sus métodos e implementando las interfaces que se encuentran en REF-EvaluationMetrics.

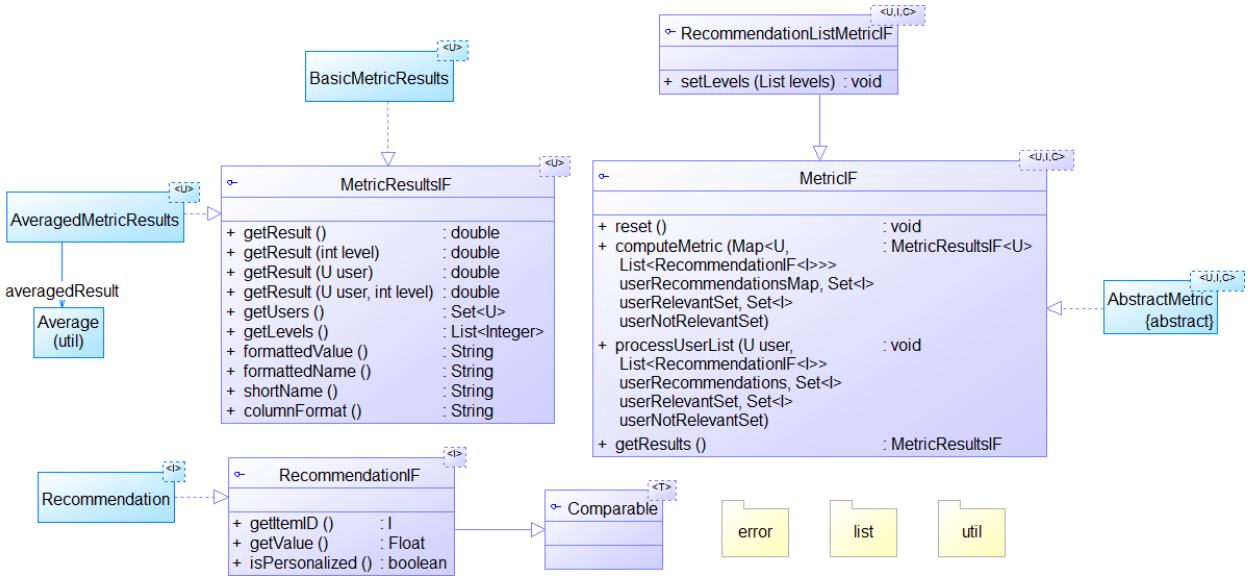


Figura 25. Contenido proyecto REF-EvaluationMetrics.

En el paquete *error* se incluyen las clases encargadas de realizar las tareas de métricas error, tales como: RMSE, MAE, TestCoverage, y otras las cuales se encuentran personalizadas (ver figura 26). Estas clases heredan de los métodos la clase AbstractErrorMetric e implementan la interface MetricIF, con lo cual los métodos implementados realizan el cálculo correspondiente para cada métrica. También, este paquete posee la clase ErrorMetricBuilder, la cual se encarga de construir las distintas métricas de error a partir del listado de métricas.

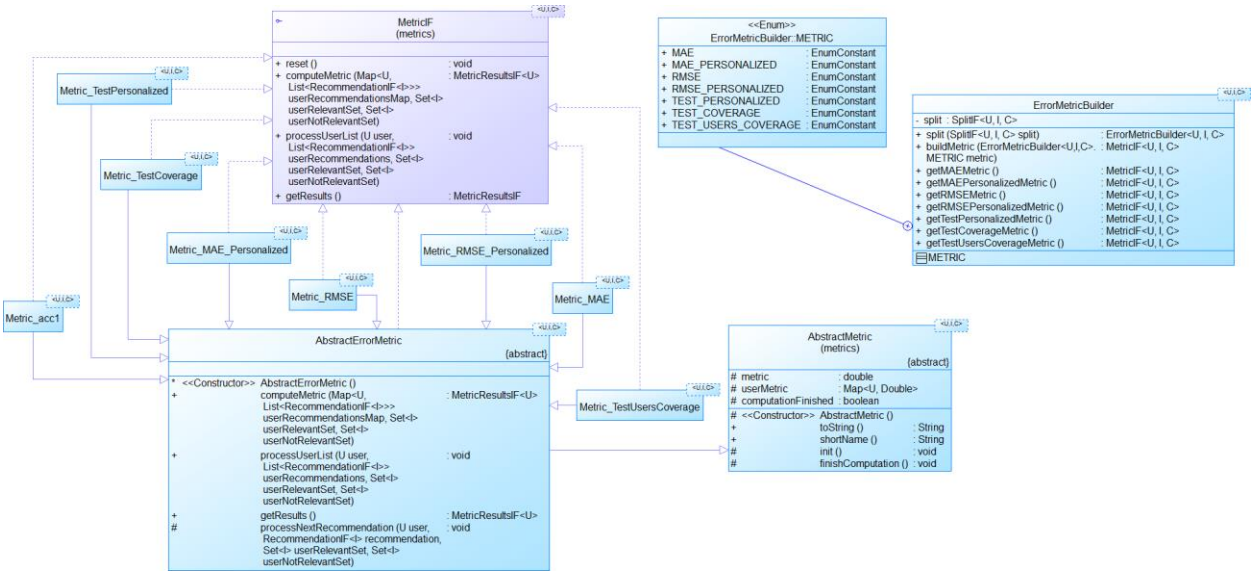


Figura 26. Diagrama de clases paquete error.

Por otra parte, el paquete *list* contiene interfaces y clases encargadas de tareas de predicción, tales como predicción de precisión y recall. En el paquete *util* encontramos la clase getSimilarity la cual nos permite comparar dos ítems.

4.2 Paquete Experiments.

Este paquete contiene clases que ayudan a configurar los métodos de evaluación y recomendadores implementados, con los cuales a través del ajuste de distintos parámetros es posible ejecutar diferentes experimentos de evaluación de algoritmos de recomendación, utilizando diferentes condiciones de evaluación. Este paquete tiene dependencias de los módulos descritos anteriormente.

La clase Experiment_Main (ver figura 27), es la que permite configurar los distintos parámetros que ayudan dar forma a las distintas combinaciones de metodologías de evaluación y métodos de recomendación. Una vez definido los parámetros y métodos a utilizar, el software ejecuta el proceso de evaluación, entregando los resultados en archivos tanto de texto y como comprimidos (.rar).

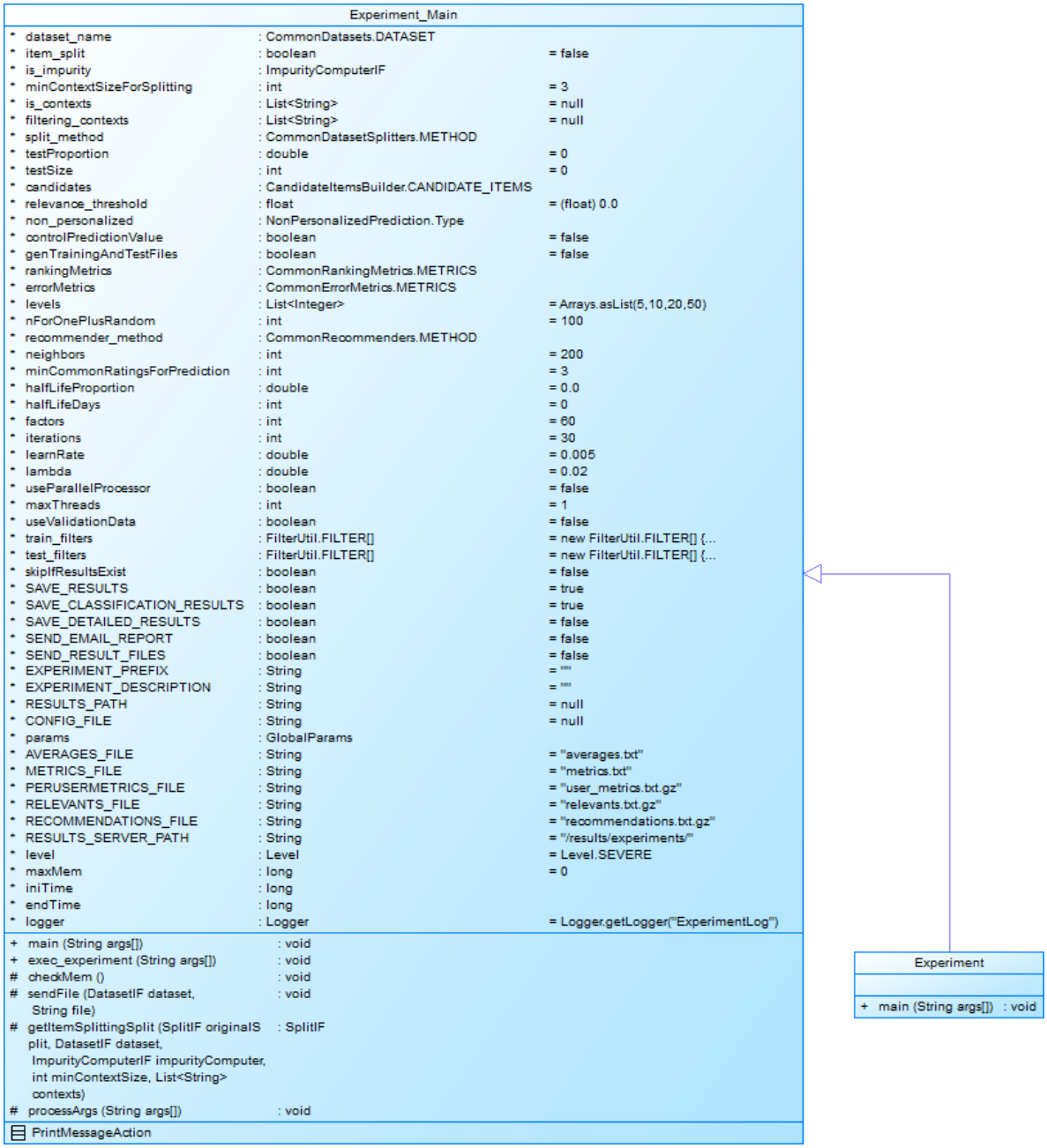


Figura 27. Diagrama de clases de Experiments_main.

5. Diseño e implementación de métodos de validación cruzada.

En este capítulo se describe la propuesta de extensión del framework descrito anteriormente (Capitulo 4) para ampliar los métodos de validación cruzada incluidos en él. La inclusión de nuevos métodos de validación cruzada permitirá realizar una gran cantidad de experimentos adicionales, pudiendo comparar los resultados y medir el rendimiento de los distintos algoritmos de recomendación utilizando estos métodos más robustos de evaluación, como se discutió en el capítulo 3.

El posible clasificar el conjunto de los nuevos métodos implementados y los que ya contaba la librería como métodos Tiempo-Independientes (Capitulo 3.5.2) y Tiempo-Dependientes (Capitulo 3.5.3), ya que alguna de las clases creadas incorporan el contexto en la conformación de los conjuntos de entrenamiento y prueba.

5.1 Extensión de la librería.

Lo que pretende esta extensión de la librería, es poder contar con una mayor cantidad de metodologías de evaluación para medir el desempeño de los recomendadores implementados.

La configuración de estos métodos es la misma que para los que ya se encontraban implementados, haciendo más fácil su implementación y configuración. A continuación se presentan los nuevos métodos de validación cruzada, con una descripción de éstos, y un pseudocódigo de los algoritmos desarrollados.

Las principales funciones presentes en los pseudocódigos serán descritas a continuación:

- getPreferences: Obtiene una lista con las preferencias a partir de un conjunto de datos.
- getUsers: Obtiene un listado con los ID’s de usuarios de un conjunto de datos.
- getNumberOfRatingsForTraining: Obtiene el número de ratings que serán utilizados para Training.
- getSize: Obtiene el tamaño de un conjunto de datos.
- addPreference: Añade una preferencia a un conjunto de entrenamiento o prueba.

- Shuffle: Mezcla el orden de los elementos en un lista.
- getDate: Obtiene la fecha de una preferencia.
- getMinDate: Obtiene la fecha mínima de un conjunto de datos.
- getMaxDate: Obtiene la fecha máxima de un conjunto de datos.
- getDifference: Obtiene la diferencia de tiempo entre dos fechas.
- Before: Determina si una fecha A está antes que una fecha B.
- After: Determina si una fecha A está después que una fecha B.

5.1.1 RepeatSampling

Es una validación cruzada tiempo-independiente en la cual se repetirá el proceso de *Split* X veces (dónde X es la cantidad de veces a repetir la evaluación con diferentes conjuntos de training y test). En cada *Split* se generaran los set de entrenamiento y prueba de manera aleatoria de acuerdo a la *Size condition* dada.

Esquema gráfico del Split

El siguiente esquema gráfico muestra un ejemplo del método *Repeat Sampling* de 3 *Splits*, en donde se separan de forma aleatoria los datos de entrenamiento y prueba según una *Size condition.* Como se puede ver en la imagen, en cada *Split* se eligen diferentes ratings para training y test, de forma aleatoria.

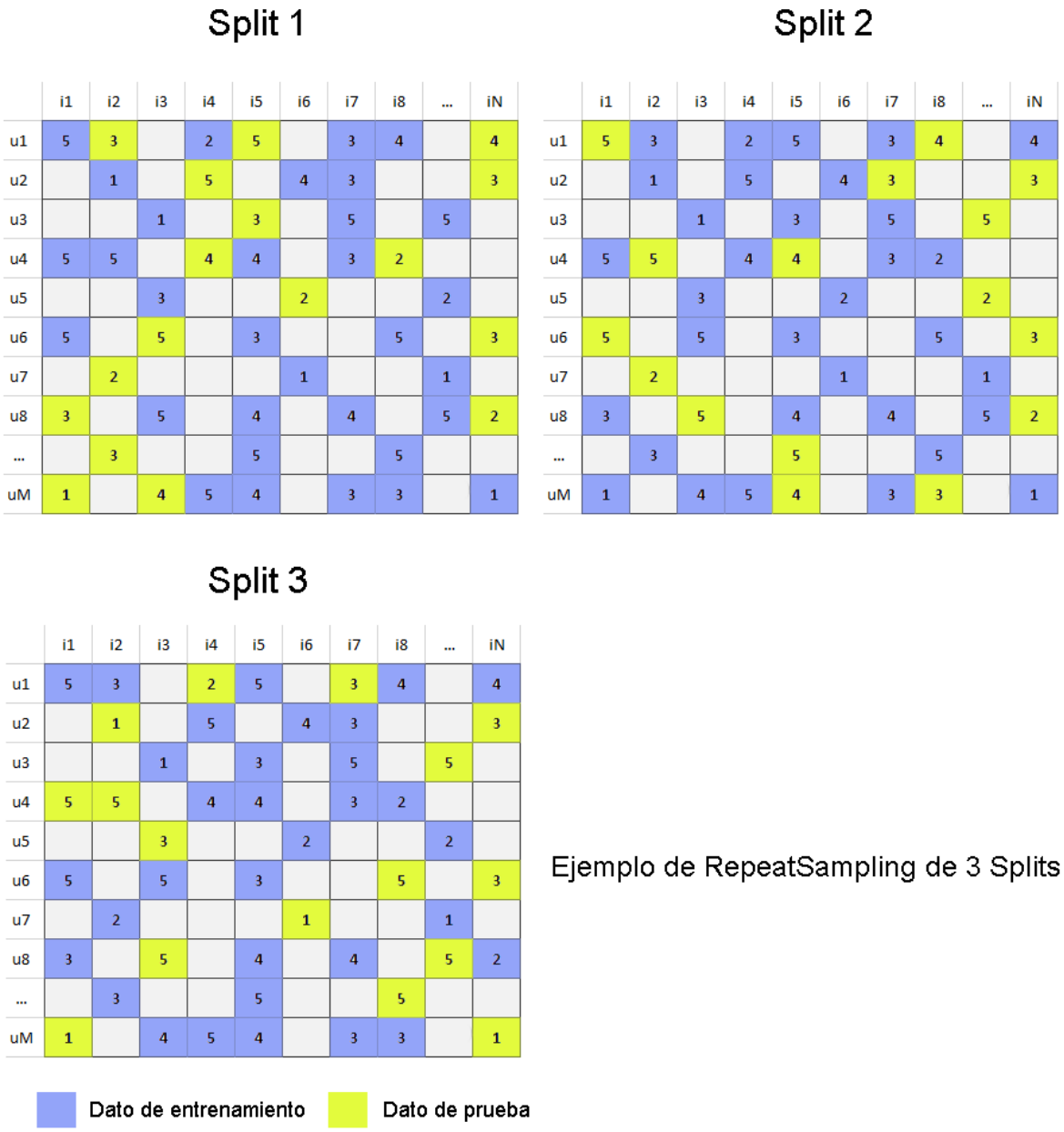


Figura 28. Ejemplo del procedimiento de Split con el método RepeatSampling

5.1.1.1 Pseudocódigo RepeatSampling

El siguiente pseudocódigo muestra los pasos en un procedimiento de *splitting* del método de validación cruzada RepeatSampling. En primer lugar se indica el nombre del método y los parámetros que recibe (línea 1). En este caso el parámetro es un entero, el cual representa la

cantidad de iteraciones que se realizaran en el procedimiento de *splitting* (línea 4) y un double que representa la proporción de datos que serán usados para conformar el set de pruebas. Luego se obtiene la lista de preferencias del data set (línea 4), esta lista será desordenada de forma aleatoria en cada *Split* (línea 7), y se obtiene el número de ratings para training (línea 8). Después eso, y a partir de la *size condition*, se construyen los conjuntos de training y test (líneas 13 y 15). Considerando el tamaño de la *size condition*, las primeras preferencias se asignan al conjunto de training, mientras que las restantes se asignan al conjunto de test.

```
1   RepeatSampling (int SampleNumber, double proportionTest){
2
3   SplitIF split[nSplit];
4   List preferences = getPreferences(M);
5   For(int i=0 ; i<SampleNumber ; i++)
6   {
7       Shuffle(preferences);
8       Int size = preferences.Size * (1 - proportionTest);
9       Int cont = 0;
10      Foreach(preferences)
11      {
12          If(cont <= size)
13              trainData.add(preferences(cont));
14          Else
15              testData.add(preferences(cont));
16          cont++;
17      }
18  split[i] = new Split(trainData, testData)
19  }
20  Return split;
21  }
```

5.1.2 UserResampling

Es una validación cruzada tiempo-independiente en la cual se realiza una selección al azar de N usuarios de los cuales se tomarán todos sus ratings. Una vez obtenidos dichos ratings, serán separados en ratings para entrenamiento y prueba. Esta separación estará definida por una *size condition* proporcional que se aplicará a cada usuario.

Este proceso se repetirá X veces.

Esquema gráfico del Split

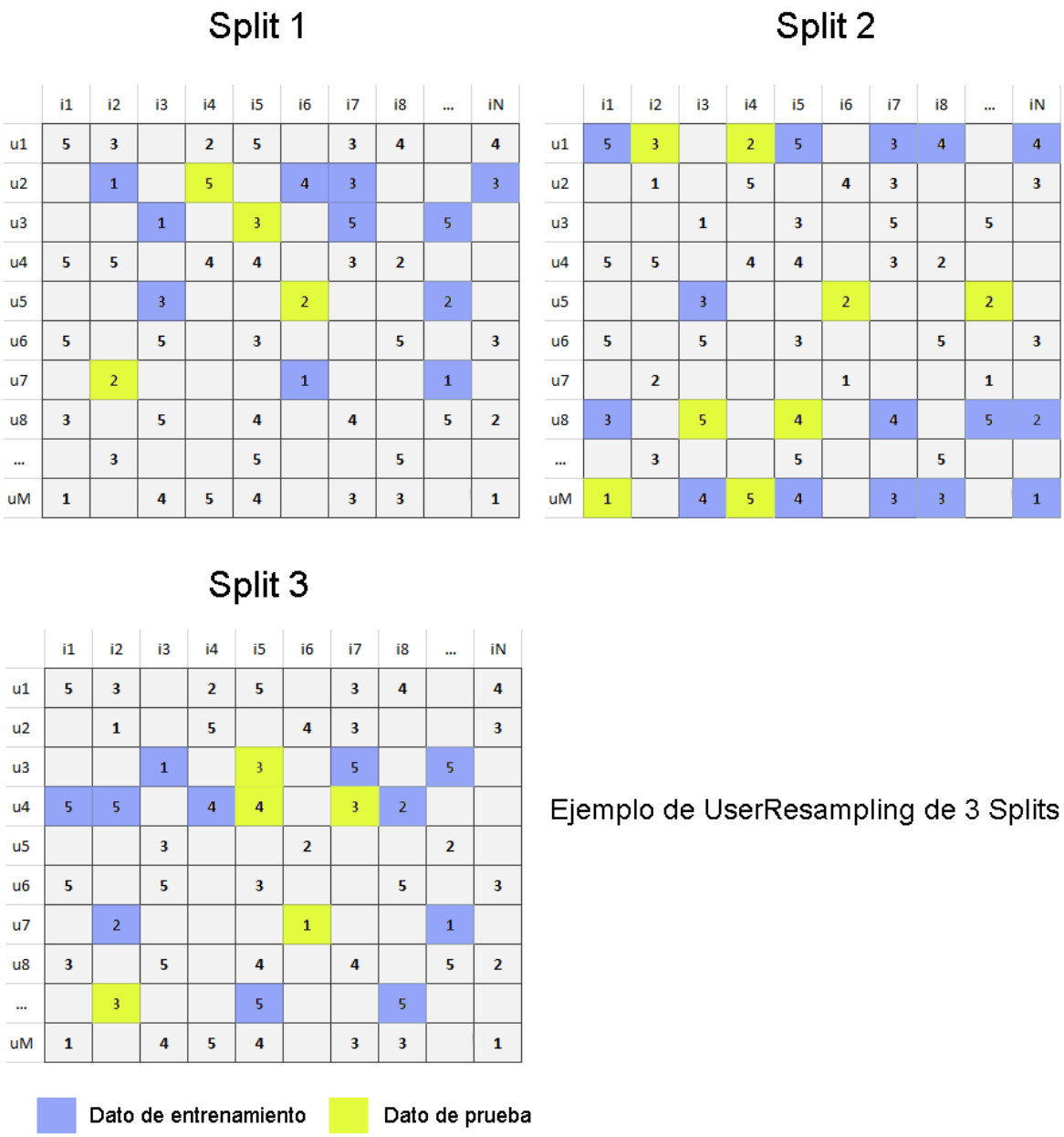


Figura 29. Ejemplo del procedimiento de Split con el método UserResampling

5.1.2.1 Pseudocódigo UserResampling

El siguiente pseudocódigo muestra los pasos en un procedimiento de *splitting* del método de validación cruzada UserResampling. En primer lugar se indica el nombre del método y los parámetros que recibe (línea 1). En este caso los parámetros son dos de tipo enteros, los cuales representan la cantidad de iteraciones que se realizarán en el procedimiento de *splitting* (línea 5), la cantidad de usuarios que se seleccionarán en cada iteración, y un parámetro tipo double, el cual representa la proporción que se utilizará en la distribución de los datos en el conjunto de training y test (línea 16). Luego se obtiene una colección con preferencias de un usuario *i* (línea 10), dado una colección de usuarios, y se obtiene el total de ratings almacenados en la colección de preferencias (línea 11). Después eso, y a partir de la size condition, se construyen los conjuntos de training y test (líneas 17 y 19). Considerando el tamaño de la *size condition*, las primeras preferencias se asignan al conjunto de training, mientras que las restantes se asignan al conjunto de test.

```
1   UserResampling (int SampleNumber, int SampleSize, double proportionTest){
2
3   SplitIF split[SampleNumber];
4   Collection users = getUsers(M);
5   For (int sample=0; sample < SampleNumber; sample ++)
6   {
7       Shuffle(users);
8       For (i=0 ; i < sampleSize ; i++)
9       {
10          Collection UserPreferences = getPrefeferences(users(i));
11          int size = getSize(UserPreferences);
12          int proportion = (1 - proportionTest) * size;
13          int cont = 0;
14          Foreach (UserPreferences)
15          {
16              IF(cont < proportion)
17                  trainData.add(UserPreferences(cont));
18              Else
19                  testData.add(UserPreferences(cont));
20          cont++;
21          }
22  }
23  split[sample] = new Split(trainData, testData)
24  }
25  Return split;
26  }
```

5.1.3 LeaveOneOut

Es una validación cruzada en la cual se toma la primera preferencia del *BaseSet* y de deja ésta únicamente como set de prueba, mientras que todas las restantes serán usadas para entrenamiento. Una vez realizado este *Split*, se pasa a la siguiente preferencia, dejando todas las restantes como set de entrenamiento, y así sucesivamente hasta recorrer toda la matriz de datos.

En esta validación el set de pruebas siempre está compuesto únicamente por un solo rating, y la cantidad de *Splits* será igual a la cantidad de ratings que posea el *BaseSet*, por lo cual es un método tremendamente costoso computacionalmente.

Esquema gráfico del Split

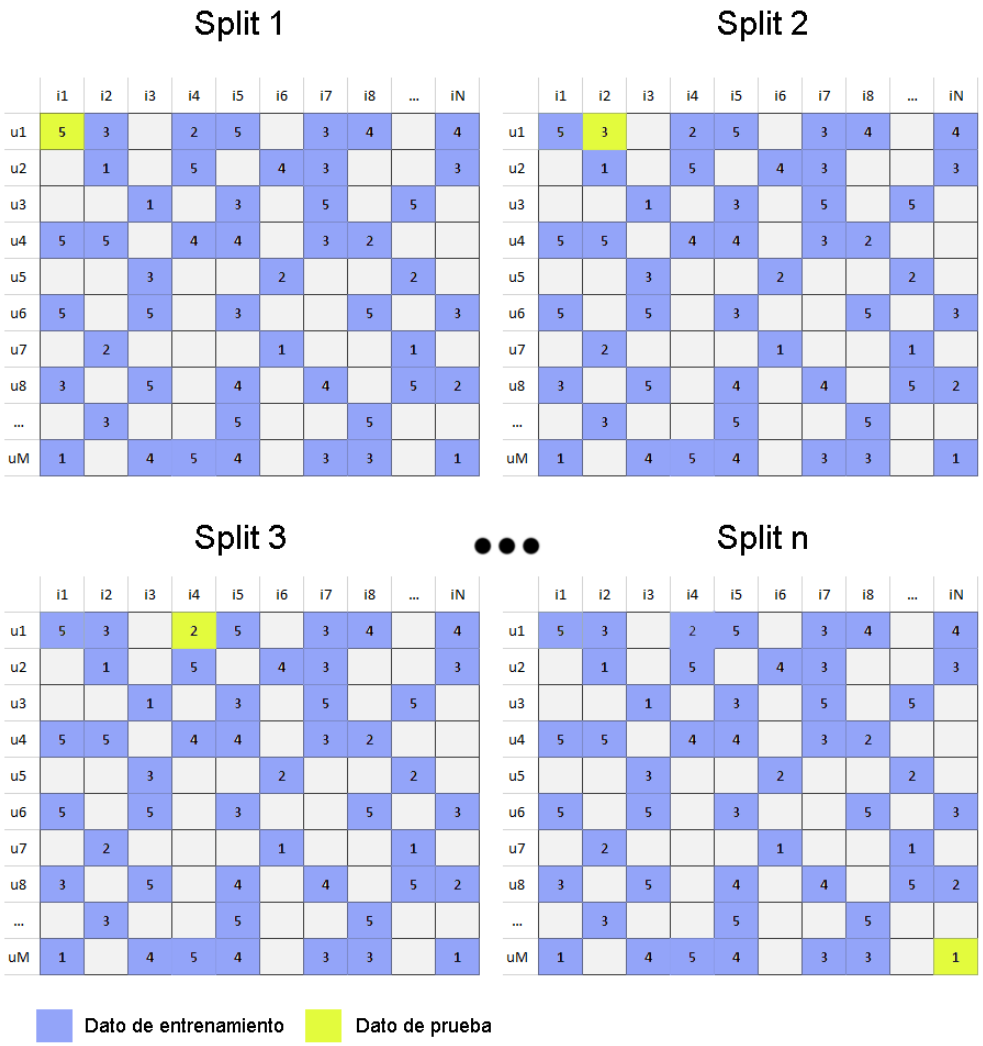


Figura 30. Ejemplo del proceso de Split del método LeaveOneOut.

5.1.3.1 Pseudocódigo LeaveOneOut

El siguiente pseudocódigo muestra los pasos en un procedimiento de *splitting* del método de validación cruzada LeaveOneOut. En primer lugar se indica el nombre del método (línea 1). A continuación se obtienen todas las preferencias del modelo (línea 3) y se determina su tamaño (línea 4).. Se recorre el listado de preferencias con los ciclos de las líneas 6 y 9, y se construyen los conjuntos de training y test (líneas 12 y 14). Considerando el tamaño del iterador, la primera preferencia que es la que se asigna al conjunto de test, y a medida que el iterador avanza se

elige un nueva preferencia para el conjunto de test, mientras que las restantes se asignan al conjunto de training.

```
LeaveOneOut (){

List preferences = getPreferences(M);
Int preferencesSize = getSize(preferences);
SplitIF split[preferencesSize];
For(int i=0 ; i < preferencesSize ; i++)
{
    Int cont = 0;
    Foreach(preferences)
    {
        IF(i == cont)
            testData.add(preferences(cont));
        Else
            trainData.add(preferences(cont));
        cont++;
    }
split[i] = new Split(trainData, testData)
}
Return split;
}
```

5.1.4 Time-Dependent Resampling

Es una validación cruzada en la cual se repite el proceso de *Split* X veces a diferentes conjuntos de preferencias tomados de la matriz de datos. En cada proceso de *Split* se separarán los datos de entrenamiento y prueba por una condición de tiempo.

Esquema gráfico del Split

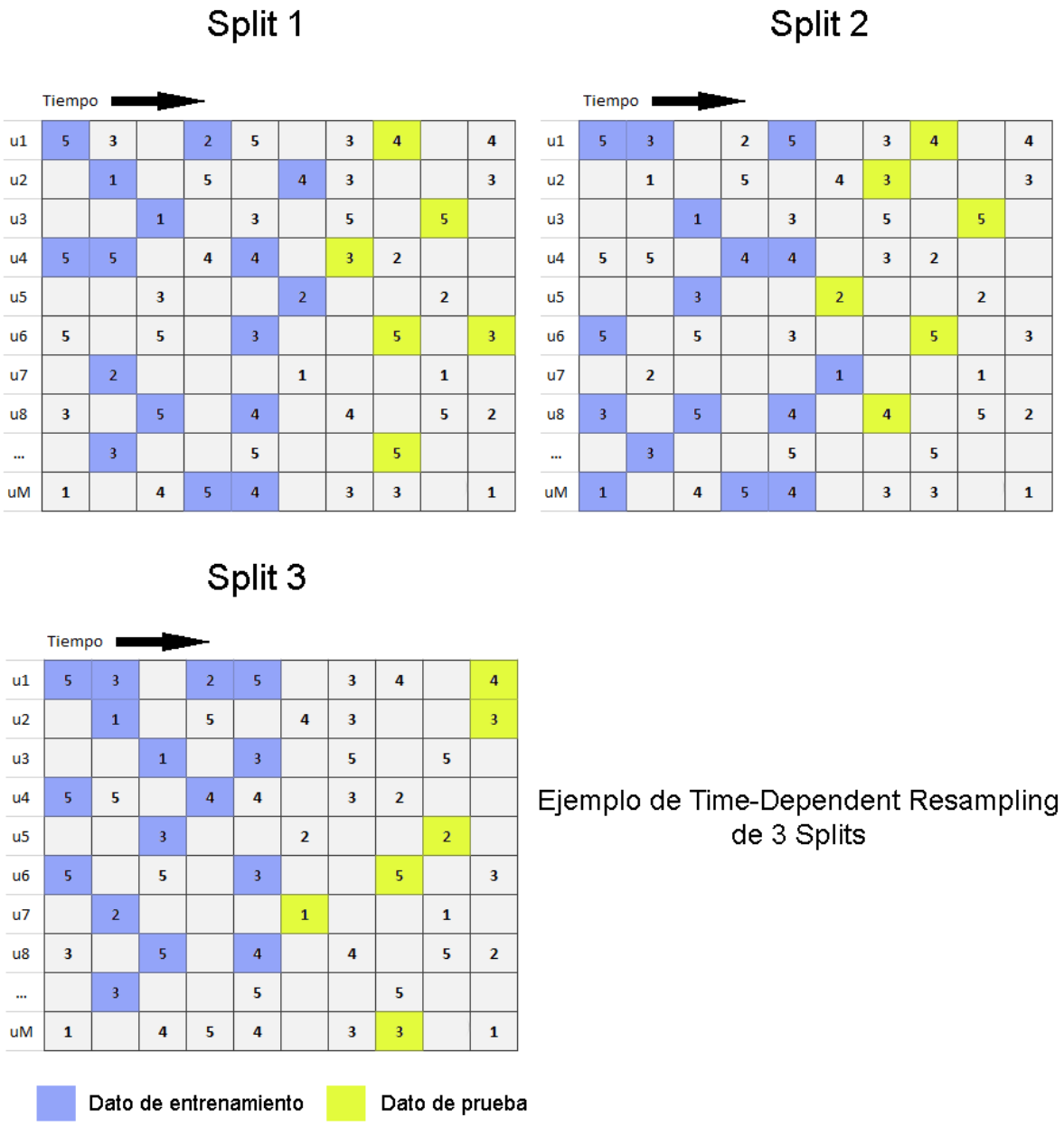


Figura 31. Ejemplo del procedimiento de Split con el método Time-Dependent Resampling (los ítems están ordenados por marca de tiempo).

5.1.4.1 Pseudocódigo Time-Dependent Resampling

El siguiente pseudocódigo muestra los pasos en un procedimiento de *splitting* del método de validación cruzada Time-Dependent Resampling. En primer lugar se indica el nombre del método y los parámetros que recibe (línea 1). En este caso los parámetros son dos enteros, uno representa la cantidad de iteraciones que se realizaran en el procedimiento de *splitting*,el segundo representa la cantidad de preferencias que se usará en cada *Split* (menor a la cantidad total de ratings en el dataset), y uno tipo date, el cual representa el umbral de tiempo al momento de realizar la distribución de los datos en los conjunto de training y test (línea 11). En la línea 4 se obtiene una lista con las preferencias del set de datos, esta lista será desordenada de manera aleatoria en la línea 7 para obtener distintos conjuntos de datos en cada *Split.* En la línea 10 se obtiene la fecha de cada preferencia y a partir del umbral de tiempo, se construyen los conjuntos de training y test (líneas 12 y 14). Considerando el umbral de tiempo, las preferencias con fecha menor al umbral de tiempo se asignan al conjunto de training, mientras que las restantes se asignan al conjunto de test.

```
1  Time-dependent Resampling (int SampleNumber, int SampleSize, Date dateThreshold){
2
3  SplitIF split[SampleNumber];
4  List preferences = getPreferences(M);
5  For (int sample=0; sample < SampleNumber; sample ++)
6  {
7      Shuffle(preferences);
8      For (i=0 ; i < sampleSize ; i++)
9      {
10         Date date = getDate(preference(i))
11         IF (date < dateThreshold)
12             trainData.add(preferences(i));
13         Else
14             testData.add(preferences(i));
15     }
16 split[sample] = new Split(trainData, testData)
17 }
18 Return split;
19 }
```

5.1.5 Time-Dependent UserResampling

Es una validación cruzada tiempo-dependiente en la cual se realiza una selección al azar de N usuarios de los cuales se tomarán todos sus ratings. Una vez obtenidos dichos ratings, serán separados en ratings para entrenamiento y prueba. Esta separación estará definida por una condición de tiempo establecida, que se aplicará a cada usuario.

Este proceso se repetirá X veces.

Esquema gráfico del Split

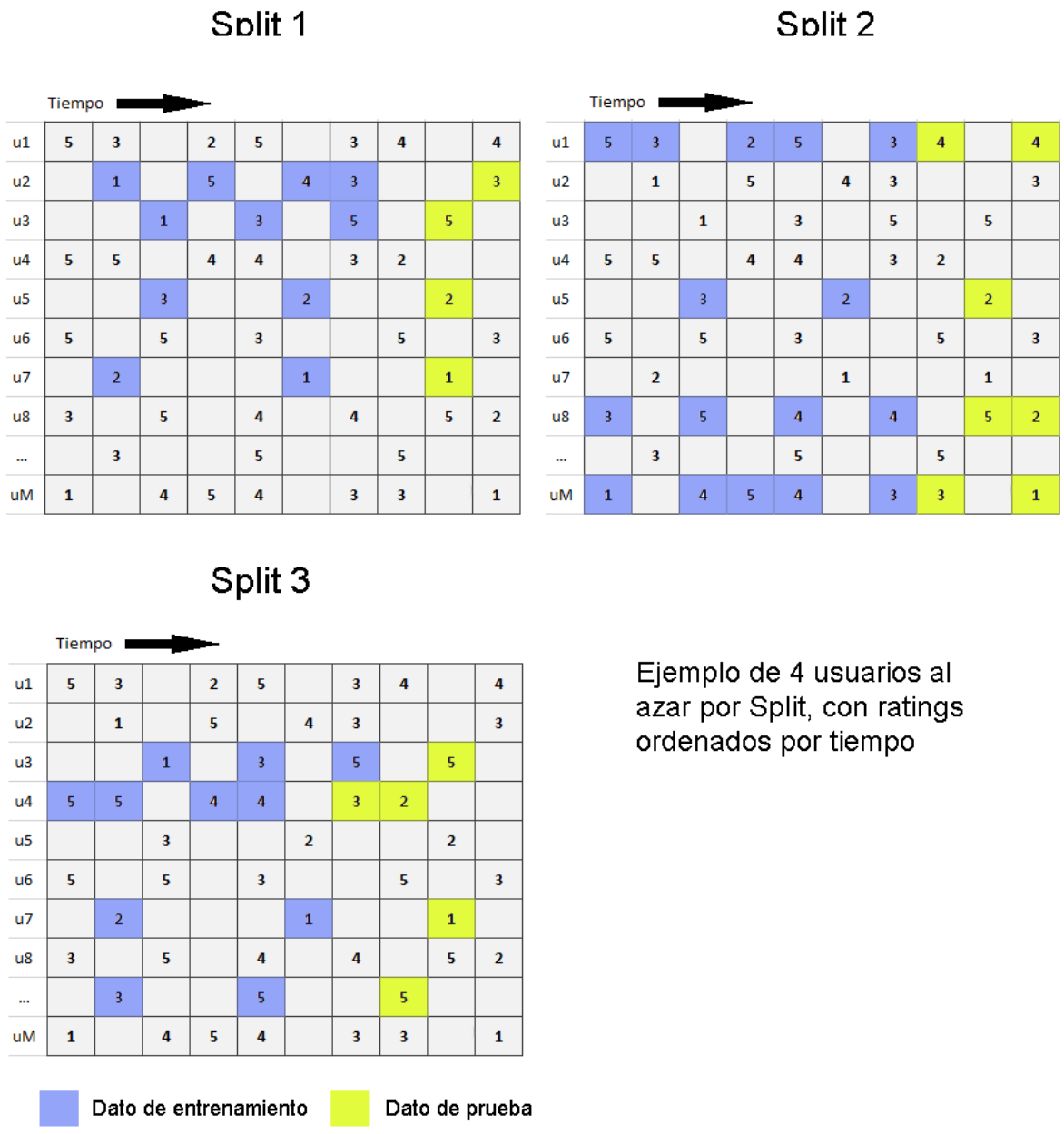


Figura 32**.** Ejemplo del procedimiento de Split con el método Time-Dependent UserResampling (los ítems están ordenados por marca de tiempo)

5.1.5.1 Pseudocódigo Time-Dependent UserResampling

El siguiente pseudocódigo muestra los pasos en un procedimiento de *splitting* del método de validación cruzada Time-Dependent UserResampling. En primer lugar se indica el nombre del método y los parámetros que recibe (línea 1). En este caso los parámetros son dos enteros, los cuales representan la cantidad de iteraciones que se realizaran en el procedimiento de *splitting* (línea 5) y el número de usuarios que se utilizaran en cada *split* (línea 8), y uno tipo Date el cual representa el umbral de tiempo al momento de realizar la distribución de los datos en los conjunto de training y test (línea 14). Luego se obtiene la lista completa de preferencias de un usuario *i* del data set (línea 10), esta lista de usuarios es modificada en cada *Split* gracias a la función *Shuffle* (línea 7), y se obtienen las fechas de las preferencias (línea 13). Después eso, y a partir del umbral de tiempo, se construyen los conjuntos de training y test (líneas 15 y 17). Considerando el umbral de tiempo, las preferencias con fecha menor al umbral de tiempo se asignan al conjunto de training, mientras que las restantes se asignan al conjunto de test.

```
1  TimeDependentUserResampling (int SampleNumber, int SampleSize, Date  dateThreshold){
2
3  SplitIF split[SampleNumber];
4  Collection users(M);
5  For (int sample=0; sample < SampleNumber; sample ++)
6  {
7      Shuffle(users);
8      For (i=0 ; i < sampleSize ; i++)
9      {
10         Collection userPreferences = getPrefeferences(users(i));
11         For (j=0 ; j < userPreferences.Size ; j++)
12         {
13             Date date = getDate(userPreferences(j))
14             IF(date < dateThreshold)
15                 trainData.add(userPreferences(j));
16             Else
17                 testData.add(userPreferences(j));
18 }
19 }
20 split[sample] = new Split(trainData, testData)
21 }
22 Return split;
23 }
```

5.1.6 Increasing time window

Es una validación cruzada en la cual se definen dos ventanas de tiempo, una para entrenamiento y otra para pruebas (por ejemplo: 4 días para entrenamiento y 2 días para prueba).

El procedimiento que sigue será explicado de manera gráfica a continuación

Esquema gráfico del Split

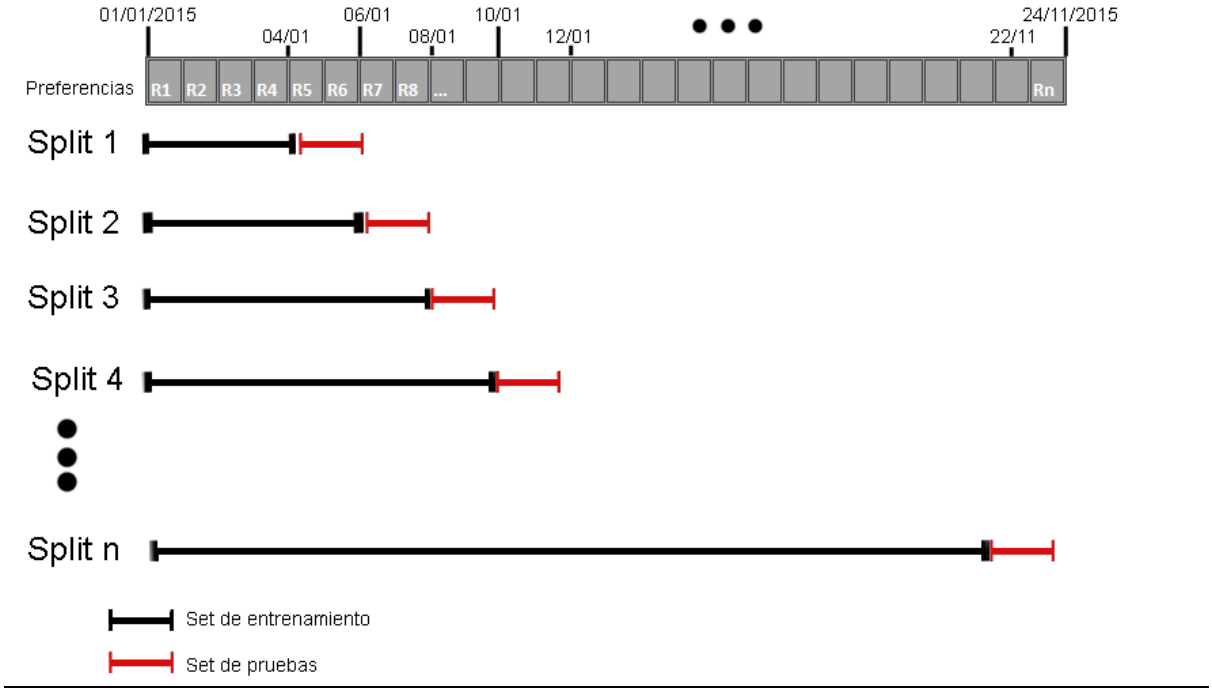


Figura 35. Ejemplo del procedimiento de Split con el método Increasing time window

5.1.6.1 Pseudocódigo Increasing time window.

El siguiente pseudocódigo muestra los pasos en un procedimiento de *splitting* del método de validación cruzada Increasing time window. En primer lugar se indica el nombre del método y los parámetros que recibe (línea 1), en este caso los parámetros son dos enteros, los cuales representan el tamaño de las ventanas de tiempo iniciales, expresadas en días. Luego se obtiene el listado de preferencias del data set (línea 3), posteriormente se obtienen las fechas mínimas y máximas de las preferencias (línea 4 y 5), a continuación se calcula la diferencia en días, entre la fecha mínima y máxima (línea 6), con la diferencia calculada y el tamaño de las ventanas de tiempo iniciales, se calcula el número de ciclos necesarios para cubrir el procedimiento de *splitting* (línea 7), finalmente se calcula los límites iniciales de las ventanas de tiempo (línea 9 y 10). Después, a partir de los límites calculados se construyen los conjuntos de training y test (línea 18 y 20). Una vez construido un *split*, se recalcula los umbrales de tiempo para la distribución de datos del siguiente conjunto de training y test (línea 22 y 23).

```
1   IncreasingTimeWindow (int valueTr, int valueTe){
2
3   List preferences = getPreferences(M);
4   Date minDate = getMinDate(preferences);
5   Date maxDate = getMaxDate(preferences);
6   int difference = getDifference(minDate, maxDate);
7   int cycles = ((difference – valueTr) / valueTe) + 1;
8   SplitIF split[cycles];
9   Date limitTr = minDate + valueTr;
10  Date limitTe = limitTr + valueTe;
11
12  For(int i = 0 ; i < cycles ; i++)
13  {
14      For(j = 0 ; j < preferences.Size ; j++)
15      {
16          Date date = getDate(preference(j));
17          If(date.before(limitTr))
18              trainData.add(preference(j));
19          If(date.after(limitTr) && date.before(limitTe))
20              testData.add(preference(j));
21      }
22  limitTr = limitTr + valueTe;
23  limitTe = limitTe + valueTe;
24  split[i] = new Split(trainData, testData)
25  }
26  Return split;
27  }
```

5.1.7 FixedTimeWindow

Es una validación cruzada en la cual se definen dos ventanas de tiempo, una para entrenamiento y otra para pruebas (por ejemplo: 4 días para entrenamiento y 2 días para prueba). Estas ventanas van avanzando a través del tiempo, manteniendo su tamaño.

El procedimiento que sigue será explicado de manera gráfica a continuación:

Esquema gráfico del Split

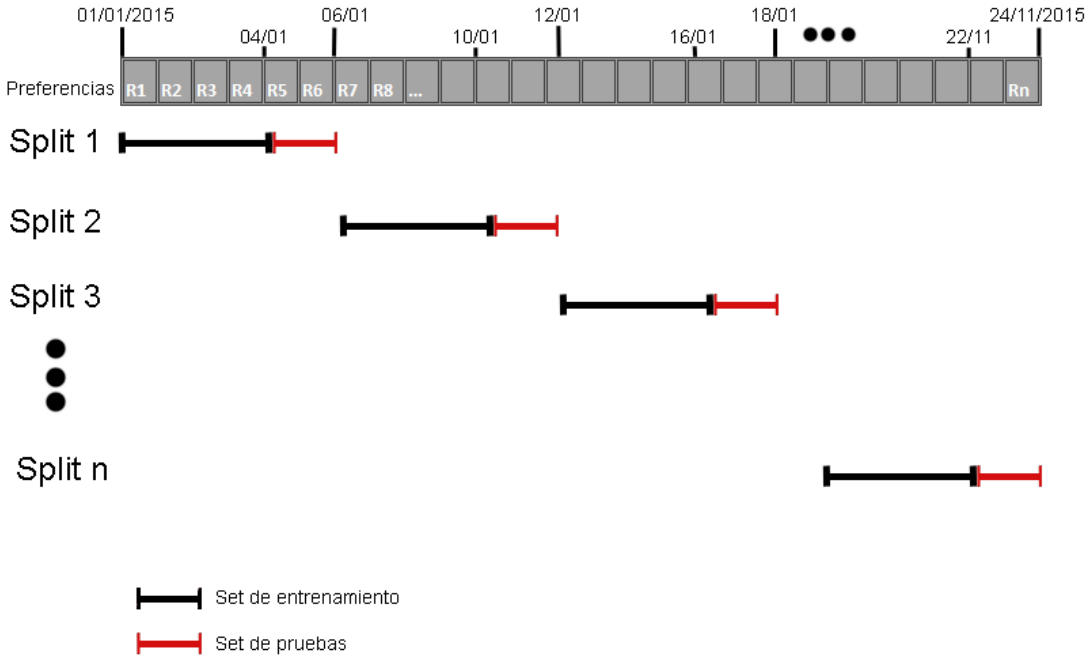


Figura 36. Ejemplo de procedimiento de Split FixedTimeWindows. Ventana de entrenamiento: 4 días, ventana de pruebas: 2 días.

5.1.7.1 Pseudocódigo Fixed time window

El siguiente pseudocódigo muestra los pasos en un procedimiento de *splitting* del método de validación cruzada Fixed time window. En primer lugar se indica el nombre del método y los parámetros que recibe (línea 1), en este caso los parámetros son dos enteros, los cuales representan el tamaño de las ventanas de tiempo iniciales, expresadas en días. Luego se obtiene el listado de preferencias del set de datos (línea 3), posteriormente se obtienen las fechas mínimas y máximas de las preferencias (línea 4 y 5), a continuación se calcula la diferencia en días, entre la fecha mínima y máxima (línea 6), con la diferencia calculada y el tamaño de las ventanas de tiempo iniciales, se calcula el número de ciclos necesarios para cubrir el procedimiento de *splitting* (línea 7), finalmente se calcula los límites iniciales de las ventanas de tiempo (línea 9 y 10). Después, a partir de los límites calculados se construyen los conjuntos de training y test (línea 18 y 20). Una vez construido un *split*, se recalcula los umbrales de tiempo para la distribución de datos del siguiente conjunto de training y test (línea 22 y 23).

```
1   FixedTimeWindow (int valueTr, int valueTe){
2
3   List preferences = getPreferences(M);
4   Date minDate = getMinDate(preferences);
5   Date maxDate = getMaxDate(preferences);
6   int difference = getDifference(minDate, maxDate);
7   int cycles = difference / (valueTr + valueTe);
8   SplitIF split[cycles];
9   Date limitTr = minDate + valueTr;
10  Date limitTe = limitTr + valueTe;
11
12  For(int i=0 ; i<cycles ; i++)
13  {
14      For(j = 0 ; j < preferences.Size ; j++)
15      {
16          date = getDate(preference(j));
17          If(date.before(limitTr))
18              trainData.add(preference(j));
19          If(date.after(limitTr) && date.before(limitTe))
20              testData.add(preference(j));
21      }
22  limitTr = limitTe;
23  limitTe = limitTe + valueTe;
24  split[i] = new Split(trainData, testData)
25  }
26  Return split;
27  }
```

5.2 Integración al framework existente.

En esta sección se presentan los diagramas de clases con las nuevas clases creadas con los métodos de validación cruzada implementados, y su integración con las clases existentes en el framework. El siguiente diagrama (ver figura 33) muestra todas las clases y la interface DatasetSplitterIF que contiene un método, el cual es implementado de distinta forma por cada clase. Esta interface es la encargada de generar los *split* de datos, con los conjuntos de entrenamiento y prueba correspondiente. Cada clase implementa este método de acuerdo a lo descrito en la sección anterior.

Existen métodos que incorporan el contexto al momento de generar los *split* de datos. Este es el caso de las clases DatasetSplitter_TimeDependentUserResampling, DatasetSplitter_IncreasingTimeWindow, DatasetSplitter_TimeDependent, DataSetSplitter_FixedTimeWindows. Estas son las clases que forman partes de los métodos tiempo dependiente.

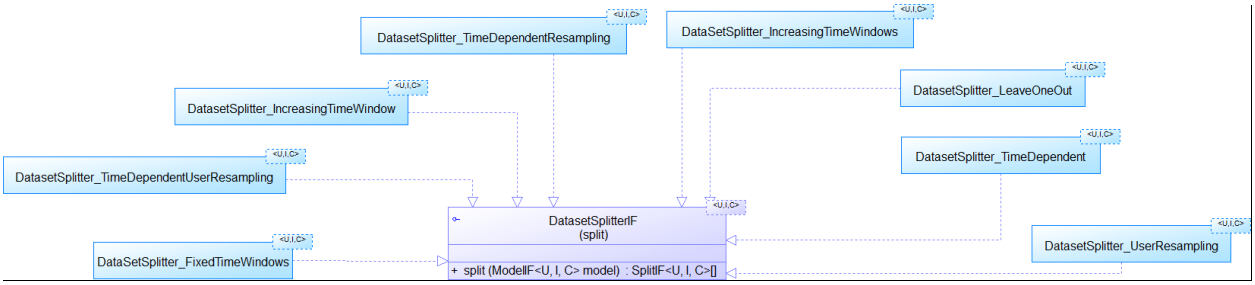


Figura 33. Diagrama de clases con las nuevas clases implementadas y las que ya se encontraban en la librería.

6. Ejemplo de uso del framework extendido: Desarrollo de una aplicación con interfaz gráfica para ejecución de experimentos con uso de validación cruzada.

Una vez completada la extensión propuesta al framework de evaluación de algoritmos de recomendación con la implementación de las nuevas propuestas basadas en los métodos de validación cruzada, se ha desarrollado una aplicación para ejecución de experimentos que hace uso de la funcionalidad de dicho framework. Esta aplicación consiste esencialmente de una interfaz gráfica para realizar el proceso de evaluación de algoritmos de recomendación consciente del tiempo, que utiliza los diferentes métodos de evaluación implementados en el framework, incluyendo métodos de validación cruzada. Con esto, se comprueba la adecuada integración de la nueva funcionalidad al framework. Y de paso, se extiende adicionalmente el framework con esta aplicación gráfica que facilitará su uso. La implementación se hizo en lenguaje java, y utiliza adicionalmente el framework de componentes gráficos Swing provisto por Java. Los componentes que se utilizan en el desarrollo de esta interfaz son: JFrame, JPanel, JLabel, JTextField, JButton, JCheckBox, JComboBox.

Como referencia, se utilizará un conjunto de datos de preferencias de usuarios (ratings) de películas, recolectados por el grupo de investigación GroupLens de la University of Minnesota (grouplens.org) a partir de la aplicación de recomendación de películas MovieLens. El conjunto de datos es llamado MovieLens 100k Dataset[8].

Este dataset está compuesto por un conjunto de archivos:

- u.data: Datos de ratings (incluye id_usuario, id_ítem, valor rating, marca de tiempo)
- u.user: Datos de usuarios (incluye id_usuario, edad, genero, ocupación, codigo postal)
- u.item: Datos de ítems (incluye id_película, título, fecha de estreno, fecha de estreno en video, URL IMDb, géneros).

[8] Conjunto de ratings de películas MovieLens 100k Dataset, disponible en: http://www.grouplens.org/datasets/movielens/

6.1 Configuración inicial.

La interfaz gráfica de la aplicación consiste en dos ventanas con distintas opciones. Antes de comenzar con la primera ventana de opciones, es necesario una configuración previa, la cual para este ejemplo consiste en la configuración de los archivos *paramsMovieLens100k.txt* y *global.config*.

El archivo paramsMovieLens100k.txt es el archivo de configuración del dataset a utilizar, y es el encargado de contener la ruta del directorio con los archivos de datos que se utilizarán para realizar experimentos (figura 34).

```
sets_path=/path/to/dataset_file/
trainFile=u.data
```

Figura 34. Contenido archivo de texto.

Mientras tanto, el archivo global.config contiene la ruta donde se encuentran los archivos de configuración de los diferentes datasets disponibles, en este caso paramsMovieLens100k.txt (ver figura 35). Este archivo es utilizado por el framework para determinar desde donde obtener los archivos de configuración de datasets.

```
BASE_DATASET_CONFIG_FILE_PATH=path/to/file.txt/
```

Figura 35. Contenido archivo global.config.

Estos archivos son requeridos por el framework para su adecuado funcionamiento.

Al finalizar la configuración de los archivos, se habilita la ventana principal de la aplicación, la cual permite seleccionar el dataset con el que se realizará el experimento, las tareas de recomendación, ya sea predicción de rating y/o recomendación Top-N, seleccionar el tipo de validación cruzada y las condiciones según el método escogido, y la elección entre realizar el cálculo de recomendaciones y/o generar archivo con los *splits* generados, con lo que se deberá indicar la ruta donde se quieren guardar los archivos con los resultados generados.

La siguiente ventana tiene como opciones elegir el algoritmo con el cual se realizará el cálculo de las recomendaciones (en caso de haber elegido esta opción), las métricas de predicción de rating; MAE y RMSE, y las métricas de recomendación Top-N; Recall y Precision. Una vez seleccionada cada una de estas opciones se puede ejecutar el experimento, con lo cual aparecerá un mensaje mientras se esté ejecutando y al momento de finalizar mostrará otro mensaje confirmando que el experimento se ha realizado con éxito, además de la ruta con los resultados obtenidos.

6.2 Configuración de un experimento.

Al iniciar el framework a través de la interfaz gráfica, se debe comenzar con la configuración de los archivos *paramsMovieLens* y *global.config*, seleccionando la opción *Archivo → Configurar archivos*, en la parte superior izquierda de la barra de menú (ver figura 36). La configuración de los archivos es importante para comenzar a utilizar las demás opciones, dado que si no se configuran adecuadamente no es posible realizar un experimento.

Figura 36. Ventana principal del framework.

Una vez seleccionada la opción *Configurar archivos*, se abrirá la ventana que se contiene la opción de seleccionar el dataset con el que se desea trabajar (ver figura 37), con lo cual se creará el archivo *paramsMovieLens100k.txt* que se encarga de contener la ruta y el nombre del dataset seleccionado. Luego es necesario indicar la ruta en la cual se quiere guardar el archivo que se necesita crear, con esto se puede pasar a seleccionar la ruta donde se creará el archivo *global.config*, que tiene la ruta del archivo *paramsMovieLens100k.txt*.

Figura 37. Ventana configuración de archivos.

Si ya encuentran configurados archivos *paramsMovieLens* y *global.config* (como se mostró anteriormente), solamente se deberá señalar la ruta del archivo *global.config*, seleccionando la opción *Archivo → Leer Configuración.* Una vez seleccionada la opción se mostrará una ventana (ver figura 38), en la cual se debe seleccionar la ruta archivo *global.config.* Una vez obtenida la ruta del archivo, se deberá seleccionar Aceptar para comenzar la configuración del experimento.

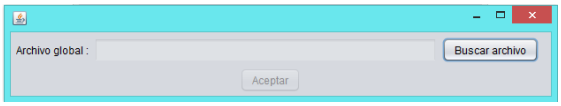


Figura 38. Ventana para seleccionar ruta archivo *global.config.*

Luego de seleccionar el archivo y las rutas donde se crearán los archivos, se deberá seleccionar Aceptar, con lo que se habilitarán las opciones de la ventana principal de la aplicación, basadas en las opciones de ejecución que provee el framework de evaluación (ver figura 38). La primera opción indica el dataset que se utilizará para realizar el experimento, en este caso MovieLens100k, luego se habilitarán las opciones de las tareas de recomendación, donde es posible elegir Predicción de rating y/o Recomendación Top-N.

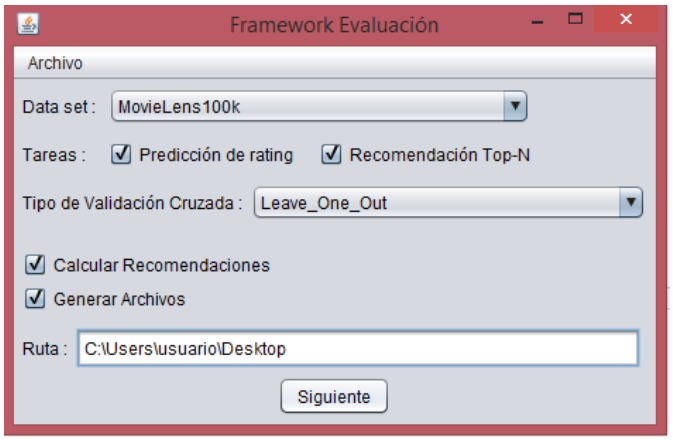


Figura 38. Configuración ventana principal del framework.

Una vez seleccionadas las tareas de recomendación, se habilitará la lista desplegable con los distintos de métodos de validación cruzada. Cada uno de estos métodos tienen asociados diferentes parámetros, los cuales permiten realizan el *split* de datos. Los parámetros existentes son: *testProportion*, cuyo valor es un número decimal, con el cuál se realizará la distribución de datos de training y test de forma proporcional dado el valor (porcentaje) ingresado, *testSize*, cuyo valor es un número entero que representa un valor fijo (cantidad) de los datos para el conjunto de training, en el caso de los métodos RepeatSampling y UserResampling el valor de *testSize* representa el número de iteraciones en los splits de datos, *sampleSize*, cuyo valor es un número entero, él se utiliza en el método RepeatSampling, y que indica la cantidad de usuarios que se utilizaran en cada iteración de los split, las variables *valueTraining* y *valueTest*, que son los parámetros que utilizan los métodos FixedTimeWindows e IncreasingTimeWindows, y cuyos valores son números enteros y que representan la ventana de tiempo en días para la distribución de datos, por último el valor *date* que representa el umbral para tiempo para separar los datos de training y test, la cual es utilizada en los método TimeDependentResamplig, TimeDependentUserResampling. Cada uno de estos campos se encuentran validados, ya sea para ingresar sólo números decimales o enteros según corresponda, además se incluye un placeholder (campo de ayuda) que indica el valor que se debe ingresar.

Cuando se ha escogido el método de validación cruzada e ingresado sus parámetros correspondientes, se debe seleccionar si se desean generar los archivos con los resultados del experimento a realizar. Si se selecciona la opción *Generar Archivos*, el programa solicitará buscar el directorio en el cual se guardarán los archivos generados. Finalmente, al seleccionar aceptar,

se pasa a la siguiente pantalla de configuración (ver figura 39), que consiste en una lista desplegable con los algoritmos de recomendación que deseamos utilizar.

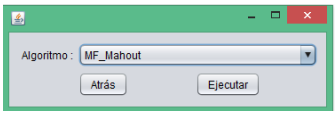


Figura 39. Segunda ventada de configuración.

Un vez configurada las opciones, se debepresionar el botón *Ejecutar*, con lo cual, la aplicación ejecutará el experimento con todas las configuraciones previas, utilizando la funcionalidad provista por el framework, y mostrando un mensaje indicando que el experimento se está ejecutando (ver figura 40).

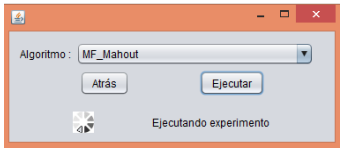


Figura 40. Ejecución de un experimento desde la interfaz gráfica de la aplicación desarrollada.

Una vez presionado el botón *Ejecutar*, la interfaz de manera interna se encarga de crea un objeto de la clase Experiments (esta clase se encuentra en el proyecto Experiments, dentro del paquete experiments), con lo cual se pueden asignan valores a las variables que permiten configurar un experimento. Estos valores son los que se obtienen de las distintas opciones configuradas, tales como el dataset que se utilizará para el experimento, cuyo valor se obtiene de la lista desplegable que se muestra en la pantalla principal de configuración, y es asignado a la variable *dataset_name* de tipo *CommonDatasets.DATASET*. Luego en la opción de tareas de recomendación, a través de la selección del checkbox Predicción de rating y/o Recomendación Top-N se asignarán a las variables *rankingMetrics* y *errorMetrics* las cuales son de tipo *CommonRankingMetrics* y *CommonErrorMetrics* el valor de *None* en caso que se no sean seleccionada una de estas opciones y *All* en el caso contrario, después la selección del método de validación cruzada a utilizar, se le asigna el valor obtenido desde la lista desplegable, a la variable *split_method* de tipo *CommonDatasetSplitters.METHOD*, con lo cual existen campos asociados a cada método los cuales son asignados a las variables *testProportion* de tipo *Double*,

testSize, *sampleSize, valueTraining, valueTest* de tipo *int* y date de tipo String, según el dato correspondiente a cada método, mientras que la opción Generar archivo nos permite asignar la ruta obtenida a la variable *RESULTS_PATH* de tipo *String*. Luego en la siguiente ventana, dada la selección del algoritmo de recomendación a través de la lista desplegable, el valor obtenido es asignado a la variable *recommender_method* de tipo CommonRecommenders.METHOD.

Con estas configuraciones, se comienza con el proceso de creación de subconjuntos de datos de training y test según el método de validación cruzada seleccionado, el cual se construye en la clase *CommonDatasetSplitters* (ubicado en el proyecto Experiments, en el paquete experiments), donde cada método validación cruzada es construido a partir los métodos de la clase *DatasetSplitterBuilder* y que además recibe como parámetro los valores obtenido en las variables asociada a cada validación cruzada. Una vez construido los subconjuntos de datos de training y test, el algoritmo de recomendación comenzará con el proceso de predicción de rating para un determinado usuario. Una vez terminado el proceso de predicción con los datos en el conjunto de training, se procede a comparar los resultados con los datos en el conjunto test, realizando los cálculos para obtener un resultado estandarizado de acuerdo a la diferencia entre las predicciones y el valor de rating real. Luego de finalizar el experimento se mostrará un mensaje de finalización la ruta en la cual se encuentran los resultados obtenidos (ver figura 41).

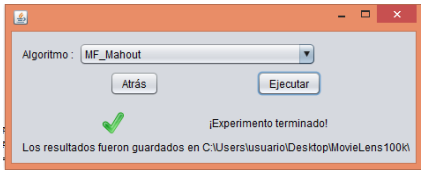


Figura 41. Finalización de experimento.

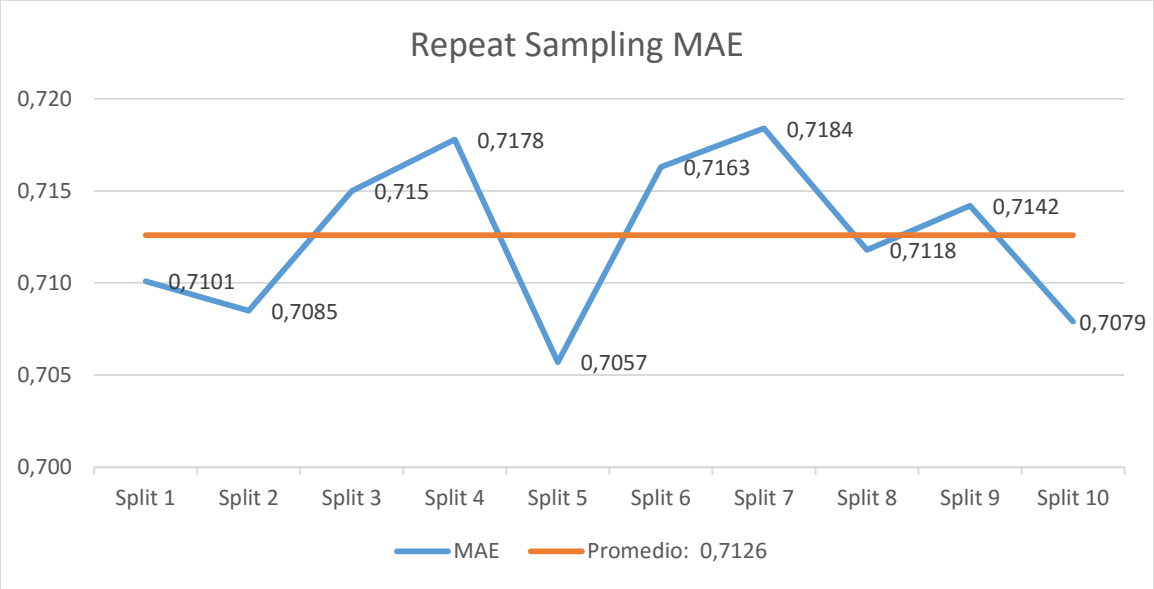
7. Experimentos.

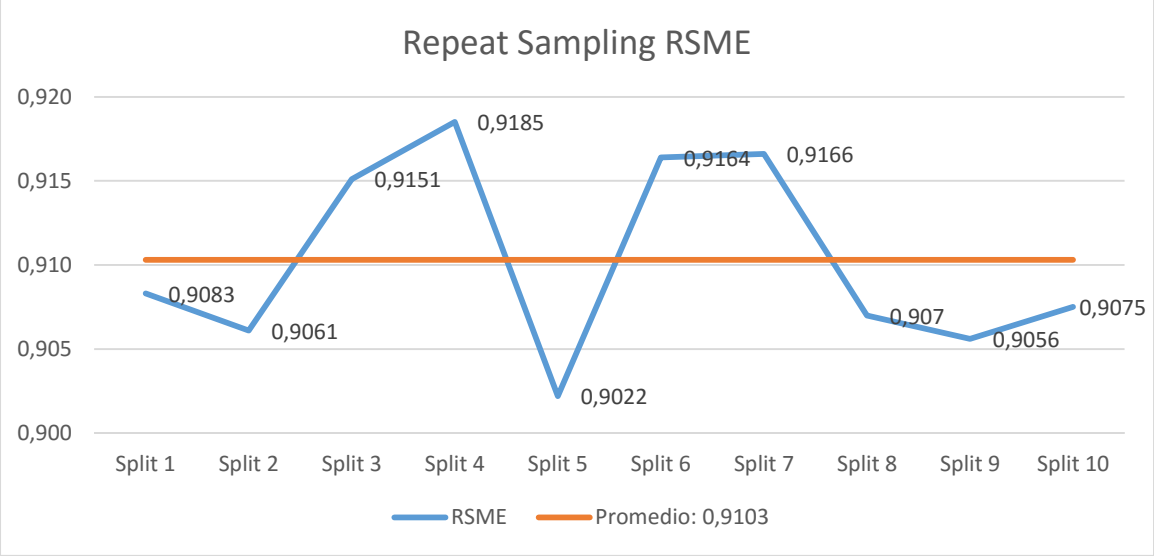
En este capítulo se presenta la serie de experimentos con que se probaron y validaron los métodos de validación cruzada desarrollados en este proyecto. Los experimentos buscan encontrar el MAE y RSME para distintos conjuntos de datos, medir sus variaciones entre cada *Split* y calcular un resultado estandarizado que sea más confiable y que refleje el rendimiento real del algoritmo de recomendación. Además se realizará un análisis sobre la influencia del contexto en los experimentos.
Detalles a considerar:

- El algoritmo de recomendación utilizado es Factorización de Matrices implementado en la librería Apache Mahout (clase SVDRecommender) con los valores de parámetros asignados por defecto en la librería.
- El set de datos utilizado pertenece a MovieLens (MovieLens 100k). Consta de 100.000 ratings distribuidos entre 943 usuarios y 1.682 ítems (películas). Estos ratings fueron asignados entre el 20 de Septiembre de 1997 y el 22 de Abril de 1998.
- Los ratings no están distribuidos de manera uniforme a lo largo del tiempo.
- El equipo donde se realizaron los experimentos posee un procesador Intel Core i5 4200m 2.5 Ghz – 3.1 Ghz de 2 núcleos físicos, 8 GB de memoria Ram y Windows 7 Ultimate SO.
- Dado que para cada validación cruzada se utilizan distintas variables, los resultados de los experimentos se mostrarán en diversos tipos de gráficos.

7.1 Experimento 1: Repeat Sampling.

Repeat sampling trabaja con el total de datos, por lo cual en cada repetición serán usados los mismos ratings, lo que cambia es su distribución entre datos de entrenamiento y prueba. Estos cambios de distribución son los que generan variaciones en los resultados de cada Split.

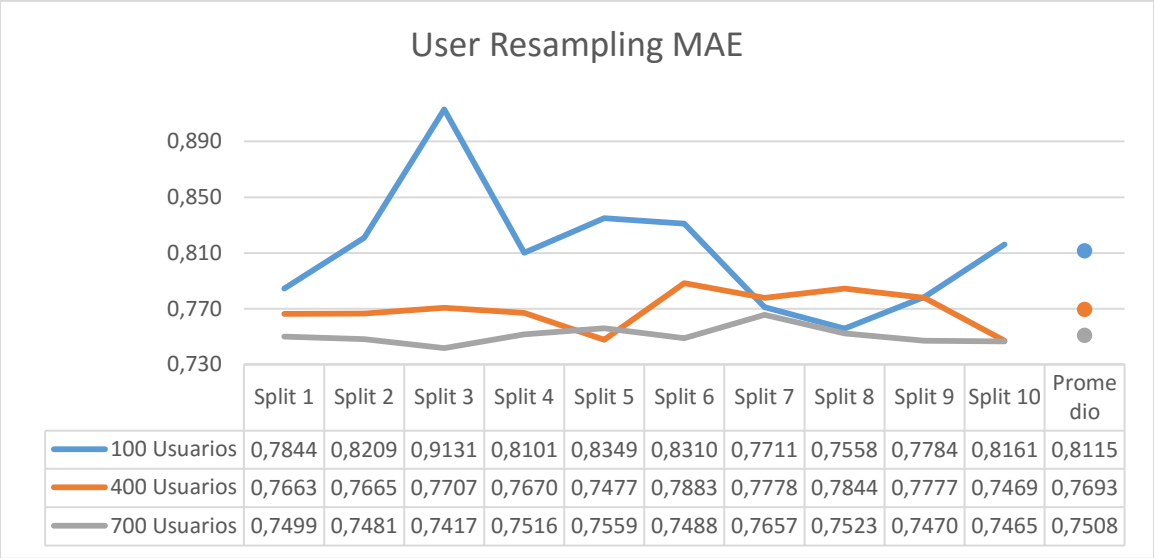




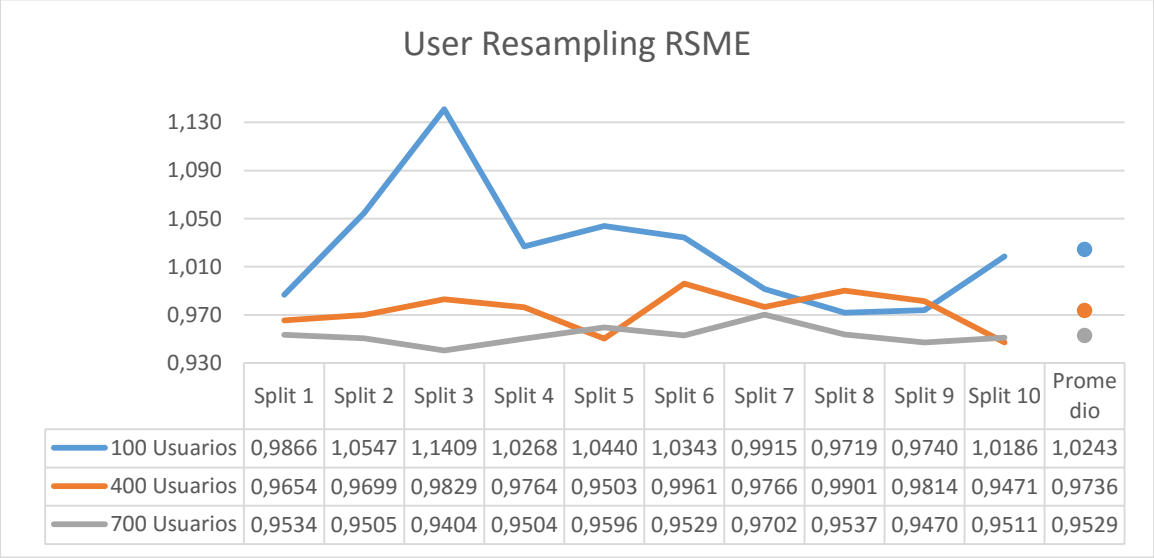
Este experimento se realizó con 10 Splits, obteniendo en cada uno pequeñas variaciones. La diferencia de resultado más grade tanto en MAE como en RSME fue entre el Split 4 (el valor más alto) y el Split 5 (el valor más bajo). El resultado de la validación cruzada fue un valor promedio de 0,7126 para MAE y 0,9103 para RSME.

7.2 Experimento 2: User Resampling.

La validación cruzada *User Resampling* trabaja tomando usuarios al azar y separando todos sus ratings en datos de entrenamiento y de prueba. Para este experimento el factor principal es el número de usuarios que se probará en cada Split. Se realizó el experimento para 100, 400 y 700 usuarios.



	Split 1	Split 2	Split 3	Split 4	Split 5	Split 6	Split 7	Split 8	Split 9	Split 10	Promedio
100 Usuarios	0,7844	0,8209	0,9131	0,8101	0,8349	0,8310	0,7711	0,7558	0,7784	0,8161	0,8115
400 Usuarios	0,7663	0,7665	0,7707	0,7670	0,7477	0,7883	0,7778	0,7844	0,7777	0,7469	0,7693
700 Usuarios	0,7499	0,7481	0,7417	0,7516	0,7559	0,7488	0,7657	0,7523	0,7470	0,7465	0,7508



	Split 1	Split 2	Split 3	Split 4	Split 5	Split 6	Split 7	Split 8	Split 9	Split 10	Promedio
100 Usuarios	0,9866	1,0547	1,1409	1,0268	1,0440	1,0343	0,9915	0,9719	0,9740	1,0186	1,0243
400 Usuarios	0,9654	0,9699	0,9829	0,9764	0,9503	0,9961	0,9766	0,9901	0,9814	0,9471	0,9736
700 Usuarios	0,9534	0,9505	0,9404	0,9504	0,9596	0,9529	0,9702	0,9537	0,9470	0,9511	0,9529

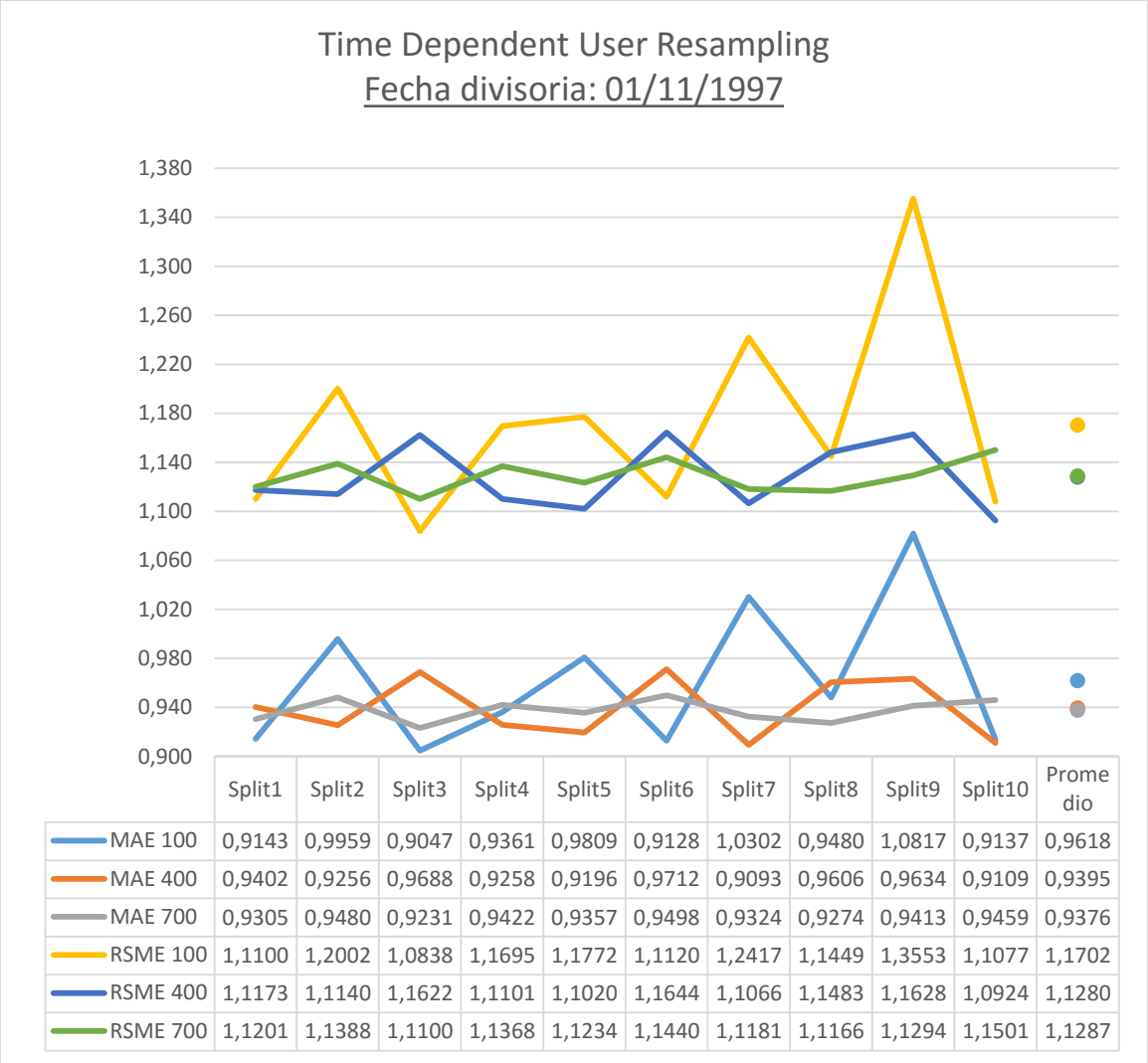
En este experimento se obtuvieron variaciones importantes en los resultados de cada Split. Se puede observar que estás diferencias se hacen menores a mayor número de usuarios. Al realizar

el experimento con 700 usuarios se obtuvieron variaciones mucho menor a las de las pruebas con 100 usuarios.

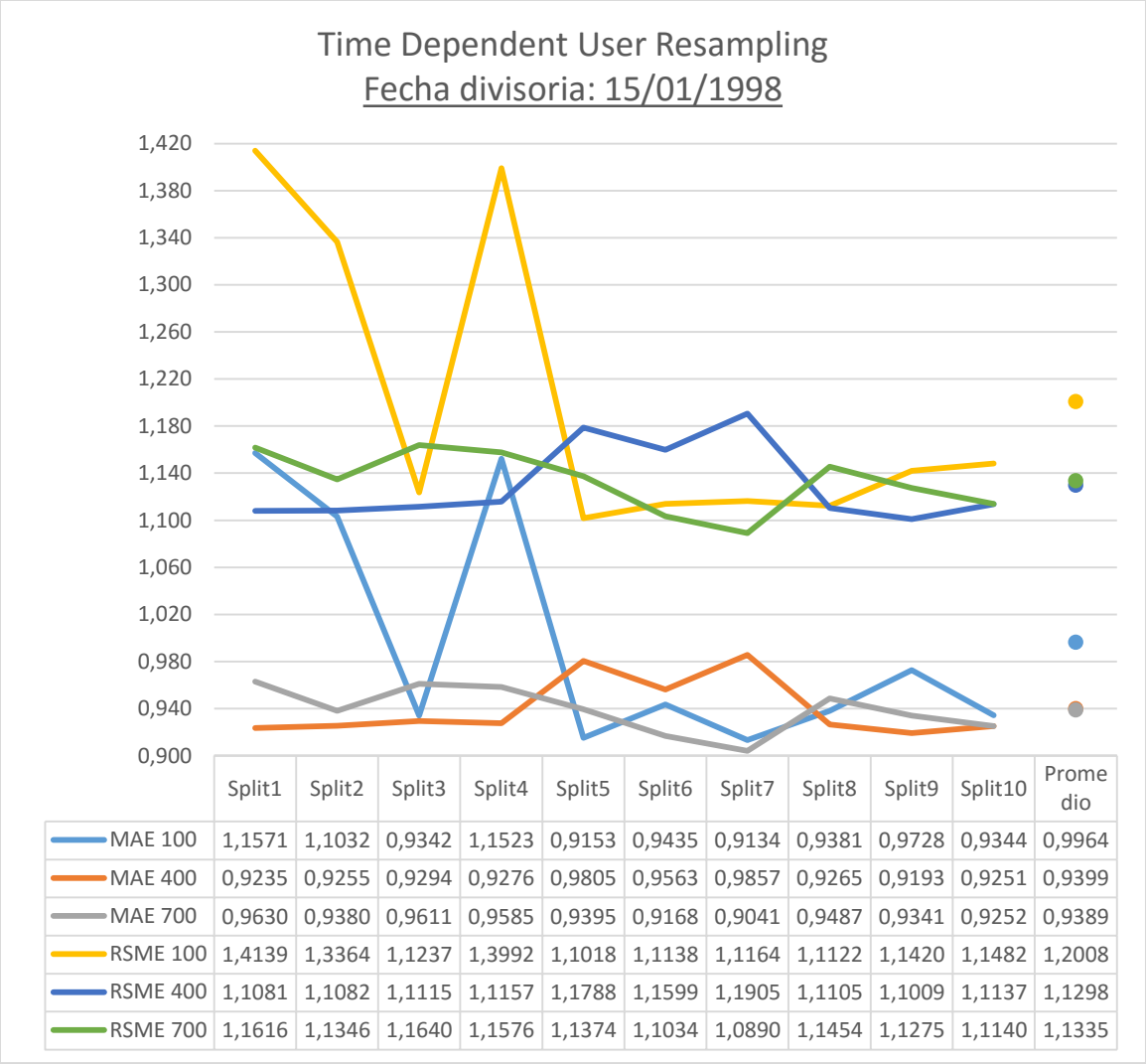
7.3 Experimento 3: Time Dependent User Resampling.

En este experimento se toma un conjunto de usuarios al azar, al igual que en *User Resampling*, pero en este caso sus ratings son separados entre entrenamiento y prueba por una condición de tiempo dada.

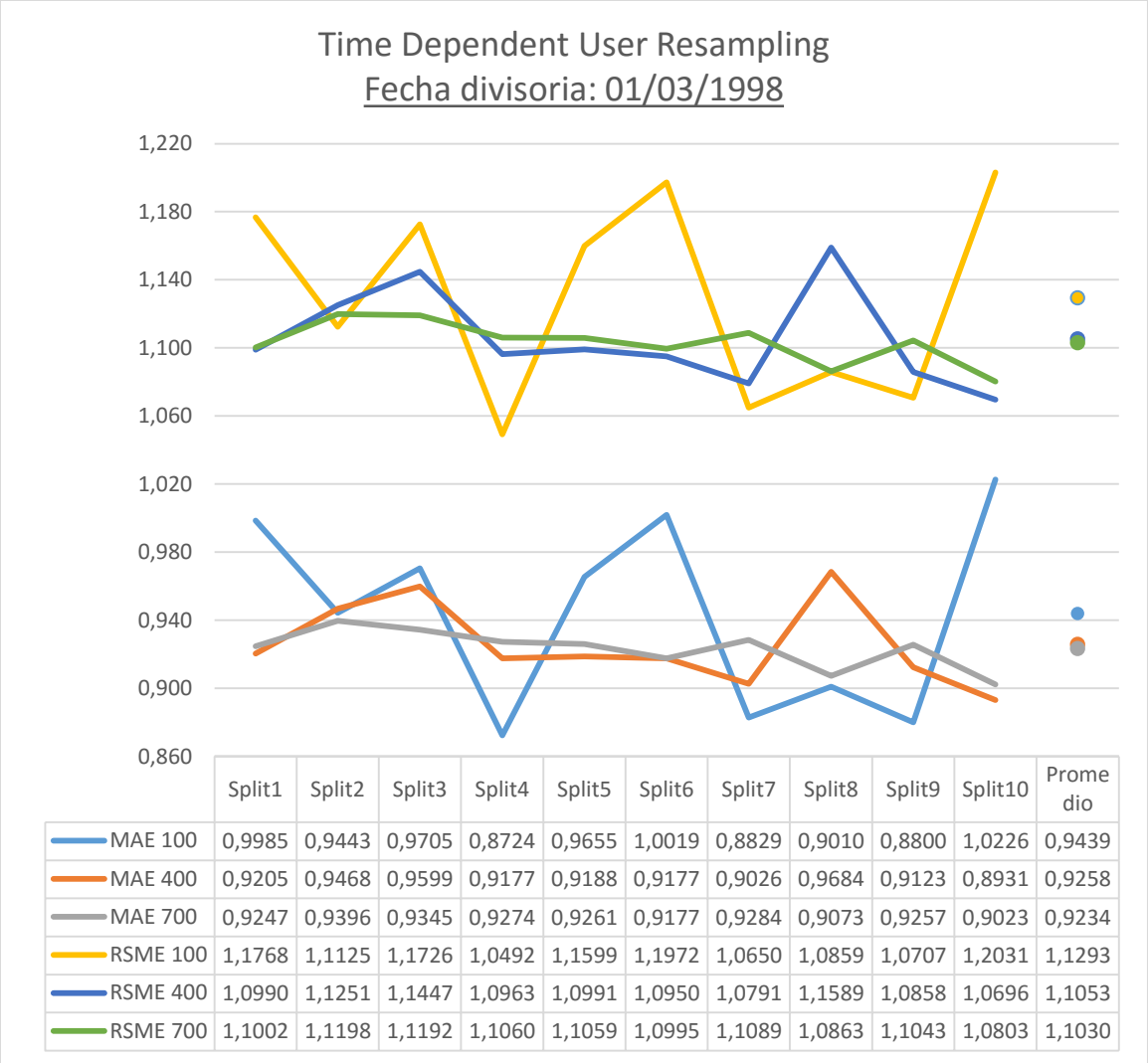
Los principales factores en esta validación cruzada son la cantidad de usuarios que compondrá cada Split y la condición de tiempo que separará sus ratings. En este experimento se utilizaron muestras de 100, 400 y 700 usuarios, y además 3 condiciones de tiempo diferentes para separar sus ratings: el 01/11/1997, el 15/01/1998 y el 01/03/1998.



	Split1	Split2	Split3	Split4	Split5	Split6	Split7	Split8	Split9	Split10	Promedio
MAE 100	0,9143	0,9959	0,9047	0,9361	0,9809	0,9128	1,0302	0,9480	1,0817	0,9137	0,9618
MAE 400	0,9402	0,9256	0,9688	0,9258	0,9196	0,9712	0,9093	0,9606	0,9634	0,9109	0,9395
MAE 700	0,9305	0,9480	0,9231	0,9422	0,9357	0,9498	0,9324	0,9274	0,9413	0,9459	0,9376
RSME 100	1,1100	1,2002	1,0838	1,1695	1,1772	1,1120	1,2417	1,1449	1,3553	1,1077	1,1702
RSME 400	1,1173	1,1140	1,1622	1,1101	1,1020	1,1644	1,1066	1,1483	1,1628	1,0924	1,1280
RSME 700	1,1201	1,1388	1,1100	1,1368	1,1234	1,1440	1,1181	1,1166	1,1294	1,1501	1,1287



	Split1	Split2	Split3	Split4	Split5	Split6	Split7	Split8	Split9	Split10	Promedio
MAE 100	1,1571	1,1032	0,9342	1,1523	0,9153	0,9435	0,9134	0,9381	0,9728	0,9344	0,9964
MAE 400	0,9235	0,9255	0,9294	0,9276	0,9805	0,9563	0,9857	0,9265	0,9193	0,9251	0,9399
MAE 700	0,9630	0,9380	0,9611	0,9585	0,9395	0,9168	0,9041	0,9487	0,9341	0,9252	0,9389
RSME 100	1,4139	1,3364	1,1237	1,3992	1,1018	1,1138	1,1164	1,1122	1,1420	1,1482	1,2008
RSME 400	1,1081	1,1082	1,1115	1,1157	1,1788	1,1599	1,1905	1,1105	1,1009	1,1137	1,1298
RSME 700	1,1616	1,1346	1,1640	1,1576	1,1374	1,1034	1,0890	1,1454	1,1275	1,1140	1,1335



	Split1	Split2	Split3	Split4	Split5	Split6	Split7	Split8	Split9	Split10	Promedio
MAE 100	0,9985	0,9443	0,9705	0,8724	0,9655	1,0019	0,8829	0,9010	0,8800	1,0226	0,9439
MAE 400	0,9205	0,9468	0,9599	0,9177	0,9188	0,9177	0,9026	0,9684	0,9123	0,8931	0,9258
MAE 700	0,9247	0,9396	0,9345	0,9274	0,9261	0,9177	0,9284	0,9073	0,9257	0,9023	0,9234
RSME 100	1,1768	1,1125	1,1726	1,0492	1,1599	1,1972	1,0650	1,0859	1,0707	1,2031	1,1293
RSME 400	1,0990	1,1251	1,1447	1,0963	1,0991	1,0950	1,0791	1,1589	1,0858	1,0696	1,1053
RSME 700	1,1002	1,1198	1,1192	1,1060	1,1059	1,0995	1,1089	1,0863	1,1043	1,0803	1,1030

Los resultados obtenidos en cada *Split* fueron muy dispares, se pueden observar grandes diferencias, sobre todo cuando se experimenta con Splits de 100 usuarios.

Al igual que en los experimentos de *User Resampling*, se puede observar que a mayor cantidad de usuarios, menor es la variación de resultados, esto se dio en las 3 pruebas realizadas con diferentes fechas de corte para la separación de ratings en entrenamiento y prueba.

Los resultados para los experimentos con 100, 400 y 700 usuarios cambiaron abruptamente al variar la fecha, lo que demuestra que ésta es un factor determinante. Las grandes variaciones se pueden deber a que los ratings no están distribuidos de manera uniforme en el tiempo, por lo cual hay “cúmulos” de datos en algunas semanas, por lo cual no es difícil que queden todos o

ninguno de los ratings de un usuario en el conjunto de entrenamiento o prueba, lo que puede provocar un alza en el valor de MAE y RSME.

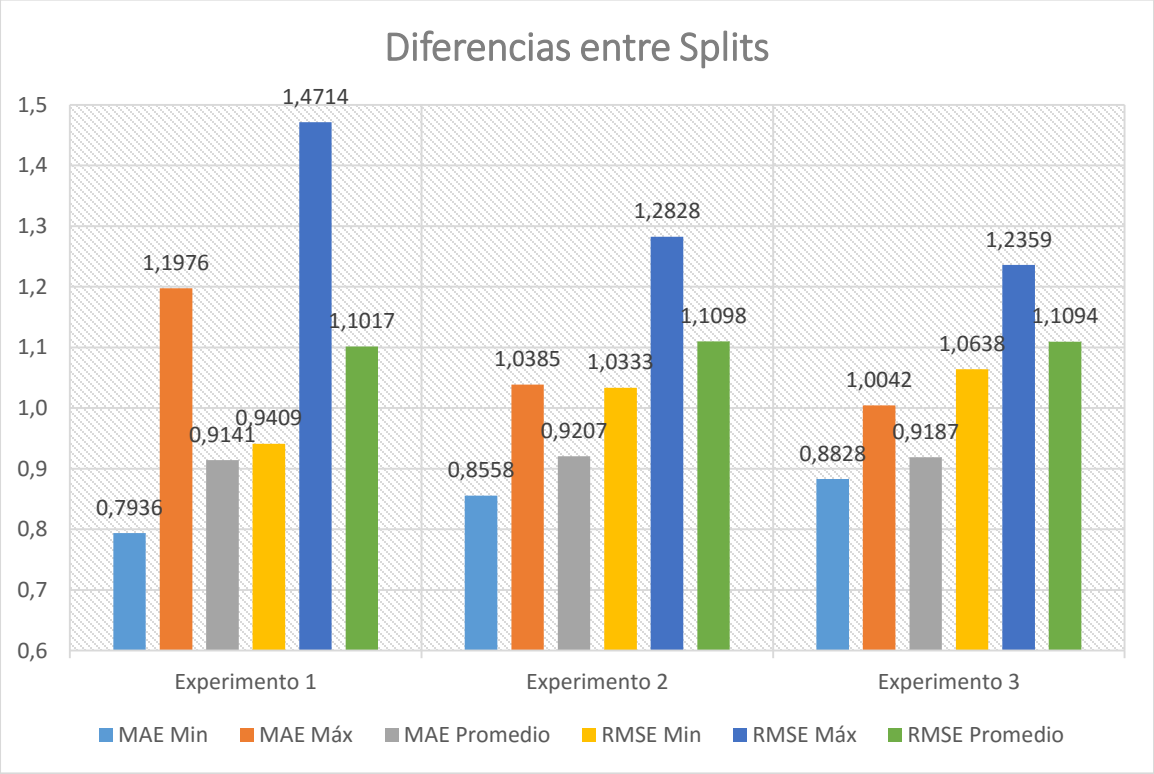
7.4 Experimento 4: Increasing Time Window.

Para esta validación cruzada se realizaron 3 experimentos, cada uno con valores iniciales diferentes para las ventanas de tiempo de entrenamiento y prueba.

Los valores iniciales de las ventanas de tiempo fueron 14 días de prueba y 7 de entrenamiento para el primer experimento, 21 días de prueba y 14 de entrenamiento para el segundo, y 28 días de pruebas y 21 de entrenamiento para el tercero.

Los resultados con cada *Split* fueron los siguientes:

	Experimento 1		Experimento 2		Experimento 3	
# Split	MAE	RSME	MAE	RSME	MAE	RSME
1	0,9045	1,0866	0,9260	1,0988	0,8891	1,0752
2	0,9558	1,1300	0,8856	1,0768	0,9159	1,0992
3	0,8632	1,0399	0,9252	1,1113	0,8828	1,0638
4	0,8595	1,0479	0,8558	1,0333	0,9241	1,1195
5	0,9052	1,0974	0,8977	1,0816	1,0042	1,2359
6	0,9644	1,1678	0,9590	1,1626	0,8880	1,0676
7	0,8728	1,0321	0,8929	1,0795	0,9303	1,1356
8	0,8499	1,0275	1,0385	1,2828	0,9274	1,1007
9	0,8575	1,0352	0,9544	1,1474	0,9070	1,0873
10	0,9056	1,0938	0,8979	1,0847		
11	0,8777	1,0481	0,8973	1,0971		
12	0,9444	1,1477	0,9024	1,0679		
13	0,9648	1,1684	0,9092	1,0833		
14	0,8805	1,0632	0,9475	1,1297		
15	0,9107	1,1148				
16	1,1976	1,4714				
17	0,9074	1,1034				
18	0,9351	1,1309				
19	0,9877	1,1763				
20	0,7936	0,9409				
21	0,9731	1,1851				
22	0,9257	1,1177				
23	0,8493	1,0623				
24	0,9253	1,0868				
25	0,8910	1,0582				
26	0,9260	1,0993				
27	0,8371	1,0119				
28	0,9651	1,1578				
29	0,8796	1,0481				
Promedio	0,9141	1,1017	0,9207	1,1098	0,9187	1,1094



Como puede observarse, hubo grandes variaciones entre los *Splits*. En el Experimento 1 encontramos la diferencia de resultados más grande de los 3 experimentos, entre los Splits 20 y 16, siendo este primero el de menor resultado con MAE 0,7936 y RSME 0,9409; y el segundo el de mayor resultado con MAE 1,1976 y RSME 1,4714.

Entre más pequeñas son las ventanas, más altos son los valores de MAE y RSME máximos, lo que puede deberse a que tiene menos ratings para comparar y realizar las predicciones, esto podría cambiar al tener un dataset con mayor densidad de ratings en el tiempo.

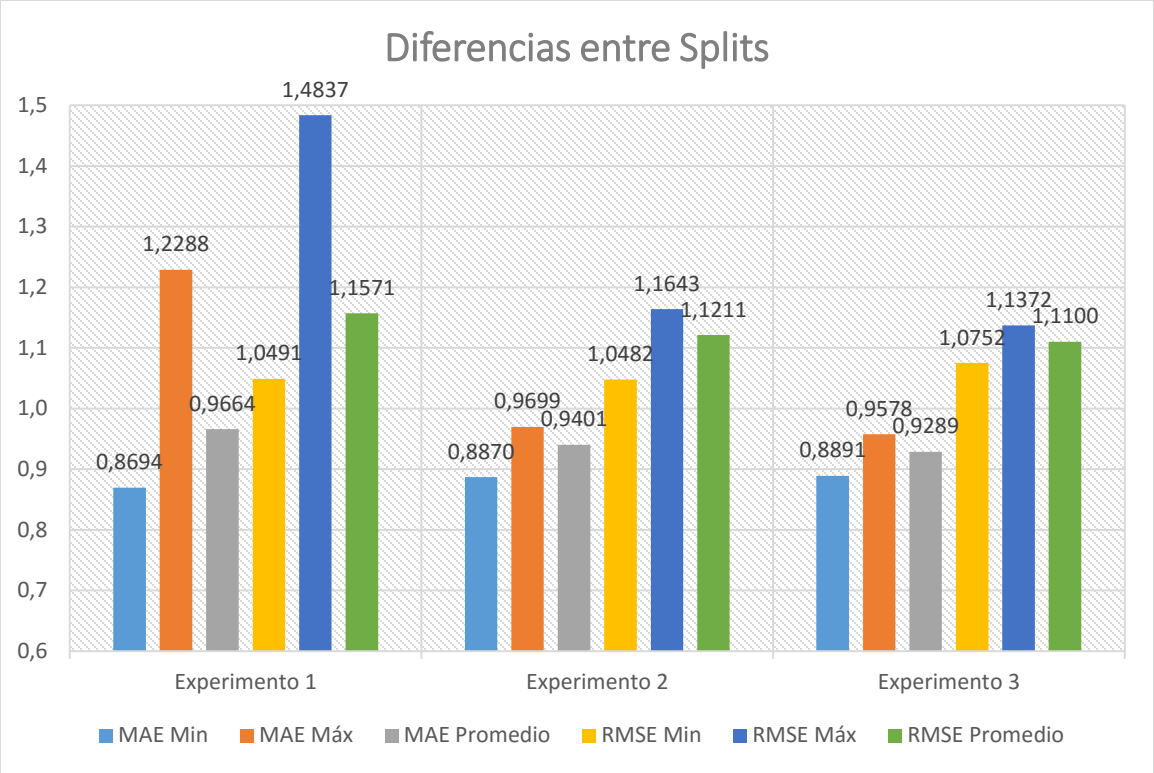
7.5 Experimento 5: Fixed Time Window.

Para esta validación cruzada se realizaron 3 experimentos, cada uno con valores iniciales diferentes para las ventanas de tiempo de entrenamiento y prueba.

Al igual que en los experimentos de Increasing Time Window, los valores de las ventanas de tiempo fueron 14 días de prueba y 7 de entrenamiento para el primer experimento, 21 días de prueba y 14 de entrenamiento para el segundo, y 28 días de pruebas y 21 de entrenamiento para el tercero, con la diferencia de que los tamaños de las ventanas se mantienen fijos en este método.

Los resultados de los Splits fueron los siguientes:

	Experimento 1		Experimento 2		Experimento 3	
# Split	MAE	RSME	MAE	RSME	MAE	RSME
1	0,9046	1,0866	0,9261	1,0988	0,8891	1,0752
2	0,8694	1,0732	0,8870	1,0482	0,9127	1,0995
3	0,8914	1,0491	0,9626	1,1643	0,9559	1,1372
4	0,9275	1,1246	0,9442	1,1271	0,9578	1,1282
5	0,9776	1,1793	0,9512	1,1412		
6	1,2288	1,4837	0,9699	1,1467		
7	0,9983	1,1756				
8	0,9672	1,1558				
9	0,8970	1,0656				
10	1,0026	1,1770				
Promedio	0,9664	1,1571	0,9401	1,1211	0,9289	1,1100



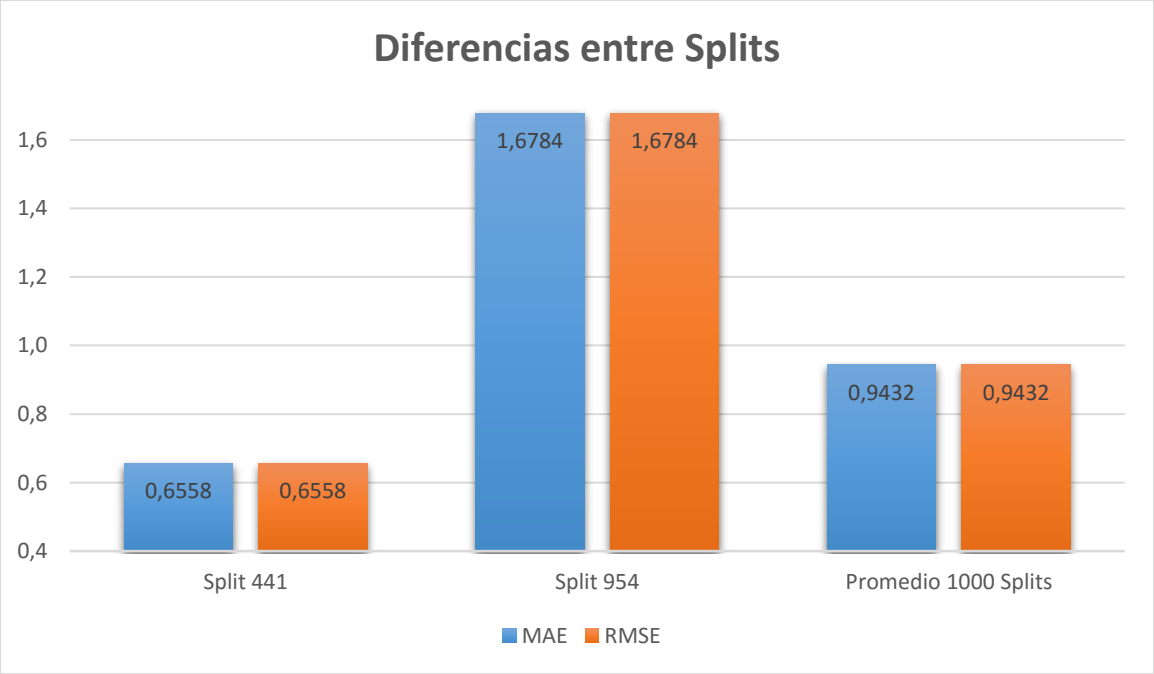
Los resultados obtenidos se parecen bastante a los de *Increasing Time Window*, hubo grandes variaciones entre los *Splits*. En el Experimento 1 se encuentra la diferencia de resultados más grande de los 3 experimentos, entre los *Splits* 2 y 6, siendo este primero el de menor resultado con MAE 0,8694 y RSME 1,0491; y el segundo el de mayor resultado con MAE 1,2288 y RSME 1,4837.

Al igual que en *Increasing Time Window*, entre más pequeñas son las ventanas, más altos son los valores de MAE y RSME máximos, lo que puede deberse a que tiene menos ratings para comparar y realizar las predicciones, esto podría cambiar al tener un dataset con mayor densidad de ratings en el tiempo.

7.6 Experimento 6: Leave One Out.

Dada la complejidad computacional de este experimento, solo se pudo realizar con un set de datos de 1000 Ratings, 250 usuarios y 550 ítems.

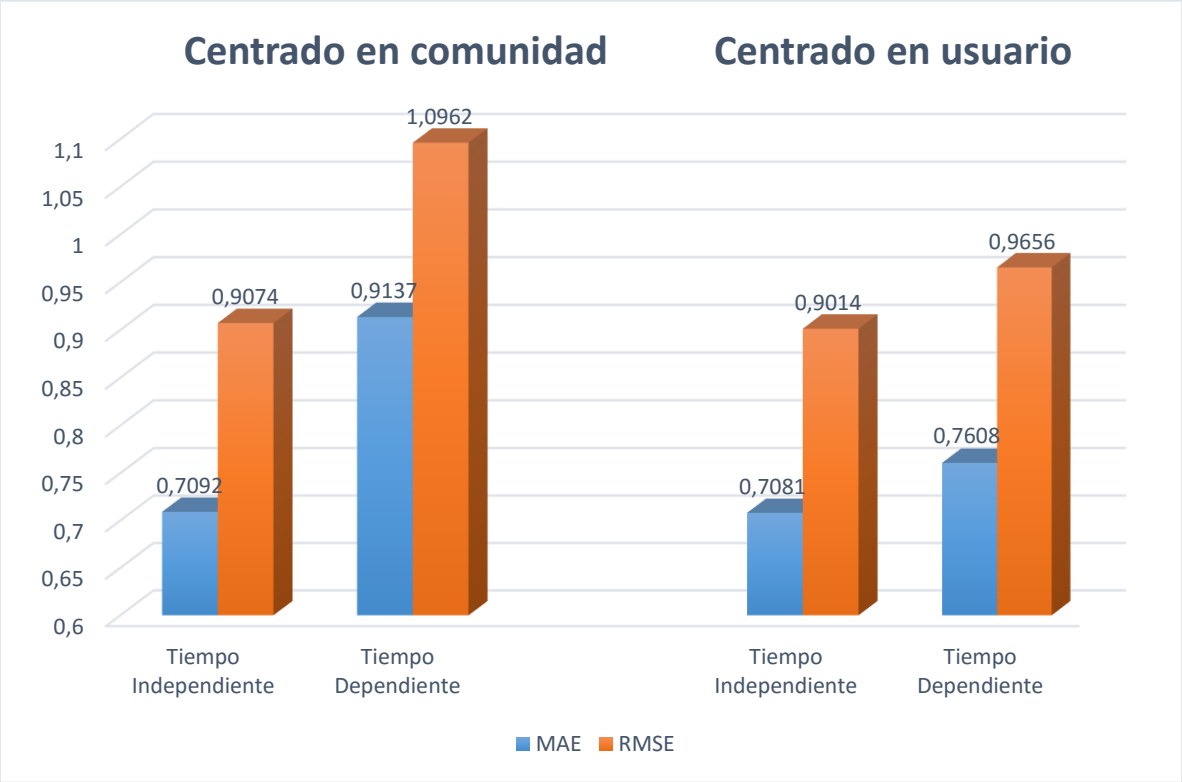
En el experimento se pueden observar *Splits* con índices de error muy bajos, como el *Split* 441, así como *Splits* con índices muy altos, como el *Split* 954. Finalmente, el promedio de los 1000 *Splits* realizados, nos entrega información más estandarizada y confiable.



7.7 Experimento 7: Prueba de contexto.

El siguiente experimento busca medir la repercusión de utilizar o no información de contexto (marcas de tiempo) a la hora de evaluar un algoritmo de recomendación, se probará utilizando el enfoque centrado en comunidad y el centrado en usuario. Ambos enfoques serán probados de forma tiempo dependientes (ordenando los ratings por marcas de tiempo) y tiempo independiente (sin ordenar los ratings por marcas de tiempo).

Para los experimentos de centrados en comunidad la *Size condition* es de 90% de ratings para entrenamiento y *10% para pruebas*, mientras que para el experimento centrado en usuario la *Size condition* será de 5 ratings por usuario para pruebas y los restantes para entrenamiento.



Tanto en el experimento centrado en comunidad como en el centrado en usuario se obtuvieron valores más altos al momento de ordenar los ratings por marcas de tiempo. Los experimentos tiempo independientes y tiempo dependientes se realizaron repetidas veces y bajo las mismas condiciones para asegurar que el ordenamiento temporal fuera el único factor que variara los resultados.

Los valores de error MAE y RSME más alto al momento de incorporar contexto podrían deberse a que si no se ordenan los ratings, pueden quedar en el set de entrenamiento ratings con fecha posterior a algunos del set de pruebas, lo que podría considerarse como información futura al momento de entrenar, algo que no es posible de obtener en el mundo real, lo cual puede desvirtuar la simulación del comportamiento humano que se realiza al evaluar de forma offline.

8. Conclusiones del Proyecto.

Los sistemas de recomendación están cada vez más presentes en las plataformas informáticas, por ello es de suma importancia tener la capacidad de medir con precisión el rendimiento de sus algoritmos.

En este Proyecto de Título se desarrolló, integró y validó una extensión enfocada en métodos de validación cruzada para un framework de evaluación de sistemas de recomendación consciente del contexto. Adicionalmente se desarrolló una aplicación con interfaz gráfica de usuario que permite realizar experimentos para medir la precisión de un algoritmo de recomendación, otorgando al usuario la posibilidad de establecer a gusto y de manera simple diversos tipos de validaciones cruzadas presentes en la literatura. Cabe señalar que el desarrollo de esta aplicación permite demostrar la adecuada integración de los métodos de validación cruzada implementados al framework existente.

Se realizó una serie de experimentos con el fin de comprobar la importancia del proceso de validación cruzada en la evaluación de un algoritmo de recomendación mediante los criterios MAE y RSME, además de la incidencia del factor “Contexto” en los resultados de dichos experimentos.

Todas las validaciones cruzadas mostraron, en menor o mayor medida, variaciones en los valores de MAE y RSME de cada *Split*. En los experimentos con validaciones que no usan el total de ratings del Dataset se encontraron las mayores variaciones de resultados, con diferencias cercanas al 30% como en los casos de *Increasing time window* y *Fixed time window*; estas variaciones tienden a disminuir cuando agregamos más usuarios y ratings a los *Splits*, esto se debe a que al realizar el proceso de *Splitting* con muy pocos usuarios y/o ratings el set de entrenamiento será muy reducido, por lo cual no tendrá suficientes datos para realizar una buena predicción, entregando así, en algunos casos altos índices de error, esto es algo que hay que tener en cuenta cuando se trabaja con Datasets con ratings distribuidos de manera muy irregular en el tiempo, ya que no resultaría difícil que en ciertos *Splits* algunos usuarios quedaran con todos o ninguno de sus ratings en el set de entrenamiento, dificultando así una buena predicción.

Cuando se utilizó el ordenamiento de ratings aleatorio se obtuvieron índices MAE y RSME más bajos que cuando fueron ordenados por marcas de tiempo, esto puede deberse a que en el ordenamiento aleatorio pueden quedar en el set de entrenamiento datos posteriores a los del set de pruebas, esto haría que el algoritmo tenga información sobre preferencias futuras al momento de entrenarse, lo cual le daría una ventaja, ya que las personas suelen cambiar de gustos a lo largo del tiempo, pero no hay que olvidar que la evaluación offline busca simular el comportamiento de un usuario humano y dicho usuario no posee información sobre sus gustos futuros.

Por todo lo anteriormente expuesto, en vista de las grandes variaciones que pueden haber a la hora de medir el rendimiento de un algoritmo en un experimento y de hacerlo con precisión, concluimos que las validaciones cruzadas son de gran importancia ya que nos dan un valor más estandarizado y confiable del rendimiento real de un algoritmo de recomendación, y que el contexto tiene una fuerte influencia en dichos resultados.

Con el Framework desarrollado en este proyecto esperamos brindan una herramienta completa y de fácil uso para la evaluación de algoritmos de recomendación.

Referencias.

[1] Adomavicius, G., Tuzhilin, A.: Context Aware Recommender Systems. In Ricci, F., Rokach, L., Shapira, B., Kantor, P. B., eds. Recommender Systems Handbook. Springer. ISBN 978-0-387-85820-3 (2011).

[2] Xiang, L., Yuan, Q., Zhao, S., Chen, L., Zhang, X., Yang, Q., Sun, J.: Temporal recommendation on graphs via long- and short-term preference fusion. Sixteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 723–732, Washington (2010).

[3] Campos, P.G., Díez, F., Bellogín, A.: Temporal rating habits: a valuable tool for rater differentiation. Second Workshop on Context-Aware Movie Recommendation, pp. 29–35, Chicago (2011).

[4] Burke, R.: Hybrid web recommender systems. In: Brusilovsky, P., Kobsa, A., Nejdl,W. (eds.) The Adaptive Web, pp. 377–408. Springer, Berlin (2007).

[5] Adomavicius, G., Sankaranarayanan, R., Sen, S., Tuzhilin, A.: Incorporating contextual information in recommender systems using a multidimensional approach. ACM Trans. Inf. Syst. 23(1), 103–145 (2005).

[6] Gorgoglione, M., Paniello, U., Tuzhilin, A.: The effect of context-aware recommendations on customer purchasing behavior and trust'. Fifth ACM Conference on Recommender Systems, pp 85–92, Chicago (2011).

[7] Dey, A.K.: Understanding and using context. Pers. Ubiquitous Comput. 5(1), 4–7 (2001).

[8] Baltrunas, L.: Context-aware collaborative filtering recommender systems. PhD Dissertation, Free University of Bozen-Bolzano (2011).

[9] Campos, P.G., Díez, F., Cantador, I.: Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols. User Modeling and User-Adapted Interaction 24(1-2), pp. 67-119 (2013).

[10] Ramakrishnan, R., and Gehrke, J., *Database Management Systems*. USA: McGraw Hill Companies (2000).

[11] Herlocker, J.L., and Konstan, J.A., Content-independent task-focused recommendation. IEEE Internet Computing, pp- 40–47 (2001).

[12] Koenigstein, N., Dror, G., Koren, Y.: Yahoo! Music recommendations: modeling music ratings with temporal dynamics and item taxonomy'. Fifth ACM Conference on Recommender Systems, pp. 165–172, Chicago (2011).

[13] Cena, F., Console, L., Gena, C., Goy, A., Levi, G., Modeo, S., and Torre, I., Integrating heterogeneous adaptation techniques to build a flexible and usable mobile tourist guide. *AICommunications*, 19(4), pp. 369–384 (2006).

[14] Van Setten, M., Pokraev, S., and Koolwaaij, J., Context-aware recommendations in the mobile tourist application compass. In Nejdl, W., and De, P.,Bra, editors, *Adaptive Hypermedia*, pp. 235–244. Springer Verlag (2004).

[15] Konstan, J.A., Riedl, J.: Recommender systems: from algorithms to user experience. User Model. User- Adapt. Interact. 22, pp. 101–123 (2012).

[16] Steck, H., Evaluation of Recommendations: Rating-Prediction and Ranking, Seventh ACM Conference on Recommender Systems, pp. 213-220, Hong-Kong (2013).

[17] Zheng, N., Li, Q.: A recommender system based on tag and time information for social tagging systems. Expert Syst. Appl. 38(4), pp. 4575–4587 (2011).

[18] Bennet, J., Lanning, S.: The Netflix Prize. KDD Cup and Workshop, San Jose (2007).

[19] Dietterich, T.G.: Approximate statistical tests for comparing supervised classification learning algorithms. Neural Comput. 10(7), pp. 1895–1923 (1998).

[20] Gordea, S., Zanker, M.: Time filtering for better recommendations with small and sparse rating matrices. Eighth International Conference on Web Information Systems Engineering, pp. 171–183, Nancy (2007).

[21] Arlot, S., Celisse, A.: A survey of cross-validation procedures for model selection. Stat. Surv. 4, pp. 40–79 (2010).

[22] Cremonesi, P., Turrin, R.: Analysis of cold-start recommendations in IPTV systems. Third ACM Conference on Recommender Systems, pp. 233–236, New York (2009).

[23] Hermann, C.: Time-based recommendations for lecture materials. 2010 World Conference on Educational Multimedia, Hypermedia and Telecommunications, pp. 1028–1033, Toronto (2010).

[24] Cremonesi, P., Turrin, R.: Time-evolution of IPTV recommender systems. Eighth International Interactive Conference on Interactive TV & Video, pp. 105–114, Tampere (2010).

[25] Lathia, N., Hailes, S., Capra, L.: Temporal collaborative filtering with adaptive neighbourhoods. Thirty-Second International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 796–797, Boston (2009).

[26] Pradel, B., Sean, S., Delporte, J., Guérif, S., Rouveirol, C., Usunier, N., Fogelman-Soulié, F., Dufau-Joel, F.: A case study in a recommender system based on purchase data. Seventeenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 377–385, San Diego (2011).