UNIVERSITAT DE BARCELONA

Trabajo de final de grado

GRADO DE INFORMÁTICA

Facultad de Matemáticas e Informática
Universidad de Barcelona

MEJORA DE LAS RECOMENDACIONES EN ALGORITMOS CONVERSACIONALES BASADOS EN EXPERIENCIAS

Autor: Fernando Torralba Barrabés

Directora:	Dra. Maria Salamó Llorente
Realitzado en:	Departamento de Matemáticas e Informática
Barcelona,	30 de junio de 2016

Abstract

In recent years there has been an exponential growth of the new technologies. The great expansion of internet and easy access to the network has increased the number of internet data and behold the context of the problem: the difficulty of finding the right information.Nowadays million of users access for internet searching products and information, cause of this it's necessary the use of recommenders which allow users to find out what they are looking for. At the same time, most interactive applications of recommenders are a key business factor for those who sell their products through the network, enabling them to increase their incomes if they use a recommender system suitable.

This project is focused on recommender systems based on critics (critiquing), which allow the user to indicate by a feedback, the characteristics of the product you are looking for in order to provide products according to their needs and preferences.

Continuing the methodology Critiquing, a study of standard algorithms (STD) and incremental (IC) will be performed, as well as those belonging to the category Experience-based Critiquing: EBC, HAC, HGR, HOR and Graph based, all based on Unit Critiquing. In addition, the algorithms previously mentioned will be implemented to be used by users through Compound Critiques. These allow the user to provide simultaneously a feedback about multiples characteristics of the product. This fact differentiates them from Unit Critiques, which only provide information on a single characteristic. In this project an analysis algorithms will be held Unit Critiques vs. Compound Critiques. The result of this analysis will allow to reflect optimization that occurs in a recommender process if Compound Critiques are used.

Resumen

En los últimos años se ha producido un crecimiento exponencial de las nuevas tecnologías. La gran expansión de internet y el fácil acceso a la red ha incrementado el número de datos en internet y he aquí el marco del problema: la dificultad para encontrar la información adecuada en el momento adecuado. Hoy en día millones de usuarios acceden a internet en busca de información y productos, por ello es necesario el uso de sistemas de recomendación que permitan al usuario encontrar aquello que se está buscando. Al mismo tiempo, la mayoría de aplicaciones interactivas de recomendación son un factor de negocio clave para aquellas personas que venden sus productos a través de la red, permitiéndoles aumentar sus ingresos si usan un sistema de recomendación adecuado.

Este proyecto está enfocado en los sistemas de recomendación basados en críticas (*critiquing*), que permiten al usuario indicar mediante un feedback las características del producto que está buscando con el objetivo de ofrecerle unos productos de acuerdo a sus necesidades y preferencias.

Continuando con la metodología Critiquing, se realizará un estudio de los algoritmos estándar (STD) e incremental (IC), así como los pertenecientes a la categoría *Experience-based Critiquing*: EBC, HAC, HGR, HOR y Graph Based, todos ellos basados en *Unit Critiquing*. Además, se implementarán los algoritmos anteriormente nombrados para poder ser utilizados con un *feedback* basado en *Compound Critiques*. Éstas permitirán al usuario proporcionar feedback simultáneamente a múltiples características del producto. Este hecho las diferencia de las *Unit Critiques*, que únicamente permiten definir las preferencias sobre una sola crítica. En este proyecto se realizará un análisis de algoritmos de *Unit Critique vs. Compound Critique*. El resultado de este análisis permitirá reflejar la optimización que se produce en un proceso de recomendación si se utilizan Compound Critiques.

Resum

Durant els últims anys s'ha produït un creixement exponencial de les noves tecnologies. La gran expansió d'Internet i el fàcil accés a la xarxa ha incrementat el nombre de dades, i es aquí on tenim el marc del problema: la dificultat per trobar la informació adequada en el moment adequat. Avui en dia milions d'usuaris accedeixen a Internet per buscar informació i productes, per això és necessari l'ús de sistemes de recomanació que permetin a l'usuari trobar allò que està cercant. La majoria d'aplicacions interactives de recomanació són un factor de negoci clau per aquelles empreses que venen els seus productes a través de la xarxa, augmentant els seus ingressos si utilitzen un sistema de recomanació adequat.

Aquest projecte està enfocat als sistemes de recomanació basats en crítiques (*critiquing*), que permeten a l'usuari indicar mitjanÁant un *feedback* les característiques del producte que estan buscant amb l'objectiu d'optimitzar la cerca i oferir-li uns productes d'acord amb les seves necessitats i preferències.

Continuant amb la metodologia Critiquing, es realitzarà un estudi dels algoritmes estàndards (STD) e incrementals (IC), així com les pertinents a la categoria Experience-based Critiquing: EBC, HAC, HGR, HOR i Graph Based, tots ells basats en *Unit Critiquing*. A més, s'implementaran els algoritmes nombrats anteriorment per poder ser utilitzats per usuaris mitjançant *Compound Critiques*. Aquestes permetran a l'usuari proporcionar *feedback* simultàniament sobre les múltiples característiques del producte. Aquest fet el diferencia de las *Unit Critiques*, que únicament ofereixen informació sobre una sola característica. En aquest projecte es realitzarà una anàlisi d'algoritmes d'*Unit Critique vs. Compound Critique*. El resultat d'aquesta anàlisi permetrà reflectir l'optimització que es produeix en un procés de recomanació si s'utilitzen *Compound Critiques*.

Agradecimientos

En primer lugar, dar las gracias a la Dra. Maria Salamó por brindarme la oportunidad de realizar este proyecto y continuar con su trabajo, además de las facilidades que me ha ofrecido en los momentos complicados de éste. Agradecer también a David Contreras el apoyo ofrecido en aspectos más técnicos del proyecto, ya que también está directamente involucrado en el proyecto realizando su doctorado sobre este mismo tema.

Me gustaría mostrar mi agradecimiento a todas aquellas personas que han creído en mi, sobre todo a mi familia, en especial a mis padres, a mis abuelos y a mi novia, que me han apoyado, que me han animado en los momentos complicados y que han sufrido conmigo en todos estos años de formación.

Índice

1. Introducción **1**

1.1. Motivación del trabajo realizado 1

1.2. Objetivos 1

1.3. Estructura de la Memoria 2

2. Estado del arte **3**

2.1. Taxonomía de los sistemas de recomendación 3

2.1.1. Filtrado Colaborativo 4

2.1.2. Filtrado basado en contenido 4

2.2. Recomendadores Conversacionales 5

2.3. Critiquing 6

2.3.1. Standard Critiquing 7

2.3.2. Incremental Critiquing 8

2.3.3. Experience-based 9

2.4. Experience-Based Critiquing 9

2.5. History-Aware Critiquing-Based 11

2.6. History-Guided Conversational Recommendation 12

2.7. History Opinion Recommender 14

2.8. Minería en los senderos de usuario 16

2.9. Collaborative Compound Critiques 18

3. Análisis y diseño **21**

3.1. Paquetes Utilizados 21

3.2. Simulador 23

3.3. Queries 25

3.4. Críticas 26

4. Implementación **28**

4.1. QueryEngine 30

4.2. PreCaseRec 30

4.3. RecCases 30

4.4. RecQueries 35

4.5. Clustering 35

4.5.1. Recomendar al usuario la clase mayoritaria 37

4.5.2. Potencia la clase mayoritaria 37

4.5.3. Potencia la clase minoritaria 38

5. Evaluación y análisis de resultados **39**

5.1. Metodología de los experimentos 39

5.2. Comparativa de los algoritmos con los diferentes tipos de queries 40

5.3. Análisis del beneficio obtenido en cuanto a IC 43

5.4. Progresión de los algoritmos 45

5.5. Progresión de los algoritmos según el tamaño de la base de casos 47

5.6. Resultados Clustering 48

6. Análisis de costes **51**

6.1. Tiempo invertido 51

6.2. Cálculo de coste global 52

7. Conclusiones y trabajo futuro **54**

1. Introducción

En este capítulo se va a introducir cual es el marco del problema, la motivación que me ha llevado ha realizar este trabajo de final de grado, así como los objetivos que se esperan conseguir en el mismo y la estructura que va a seguir la memoria del trabajo.

1.1. Motivación del trabajo realizado

La motivación principal que me ha llevado a realizar este proyecto radica en la optimización de búsqueda de datos, ya que actualmente se está incrementando la cantidad de datos existentes y es necesario tener y conocer métodos que permitan localizar rápidamente lo que se está buscando.

Además la posibilidad de utilizar grandes cantidades de datos; así como la adaptación a un proyecto ya existente; añadían una motivación extra en el proyecto, ya que a la hora de comenzar a trabajar en un proyecto en el mercado laboral, posiblemente ya esté comenzado, por lo que será necesario saber adaptarse.

Otra punto a favor reside en el lenguaje de programación usado. Por una parte JAVA me ha permitido una adaptación más rápida al proyecto ya que es el lenguaje que más se ha utilizado a lo largo de estos años de estudio en la universidad. Por otro lado, la utilización de XML me ha permitido asentar más las bases de este lenguaje que casi no conocía, además de aportar facilidad para la declaración de los datos que se utilizan en el proceso de recomendación.

1.2. Objetivos

El objetivo general del proyecto es mejorar el proceso de recomendación de los algoritmos basados en críticas. A partir de este objetivo general, se definen los objetivos específicos del mismo.

Objetivo específico 1: realizar un estudio de los algoritmos de recomendación conversacionales basados en *critiquing* mediante la aplicación de críticas unitarias.

Objetivo específico 2: implementar nuevos algoritmos basados en *experience-based* (EBC, HAC, HGR y HOR) mediante críticas compuestas y algoritmos basados en *Clustering*.

Objetivo específico 3: analizar mediante un simulador el rendimiento de los algoritmos implementados.

Objetivo específico 4: obtener conclusiones para corroborar si se produce una

optimización en el proceso de recomendación al utilizar *Compound Critiques* respecto a *Unit Critiques*.

1.3. Estructura de la Memoria

En esta sección se detalla como está dividida la memoria del proyecto que se ha realizado.

El primer capítulo está formado por la introducción del proyecto, el marco del problema actual y los objetivos del proyecto.

En el segundo capítulo se realiza el estado del arte de los diferentes algoritmos y técnicas existentes en el ámbito de los sistemas de recomendación.

En el tercer capítulo se procederá a detallar el análisis y diseño de la plataforma sobre la que está implementada el proyecto, así como de los lenguajes utilizados y el tipo de elementos más característicos utilizados en un proceso de recomendación.

En el cuarto capítulo se explica la implementación realizada, especificación de las nuevas modificaciones que se han realizado para los algoritmos en estudio, los nuevos paquetes implementados y el proceso de ejecución que llevan a cabo cada uno de ellos.

En el quinto capítulo se analizan los resultados obtenidos de la ejecución anterior, mediante la comparación de un algoritmo con otro dependiendo del conocimiento previo del usuario, del número de usuarios que se ven envueltos en las técnicas de experience-based, y de si la ejecución se realiza con Críticas Unitarias o Compuestas.

En el sexto capítulo, se procede a realizar un estudio de los costes económicos, incluyendo una valoración de los diferentes ámbitos que engloban el proyecto.

En el séptimo y último capítulo, se incluyen las conclusiones obtenidas tras la realización del proyecto, los objetivos logrados y una posible ampliación futura.

2. Estado del arte

Durante los últimos años y debido al gran contenido de información hoy en día en internet, han cobrado gran importancia los sistemas de recomendación, los cuales proporcionan información o productos que pueden ser de gran interés para el usuario.

Un sistema de recomendación[1] es un sistema inteligente que proporciona a los usuarios una serie de sugerencias personalizadas (recomendaciones) sobre un determinado tipo de elementos (ítems). Estos sistemas estudian las preferencias de cada usuario y mediante un procesamiento de los datos, encuentran un subconjunto de ítems que puedan resultar de interés para éste.

2.1. Taxonomía de los sistemas de recomendación

La comunidad científica de los sistemas de recomendación ha explorado un diverso espacio de diferentes técnicas de recomendación: basadas en filtrado colaborativo (*Collaborative Filtering*[2]) y basadas en filtrado por contenido (*Content-Based*[6]). Véase Figura 1. El filtrado colaborativo se fundamenta en los perfiles de usuario basados en calificaciones para generar recomendaciones de usuarios similares. El filtrado basado en contenido depende de la disponibilidad de conocimiento del producto, por lo general en forma de descripciones detalladas, para hacer recomendaciones.

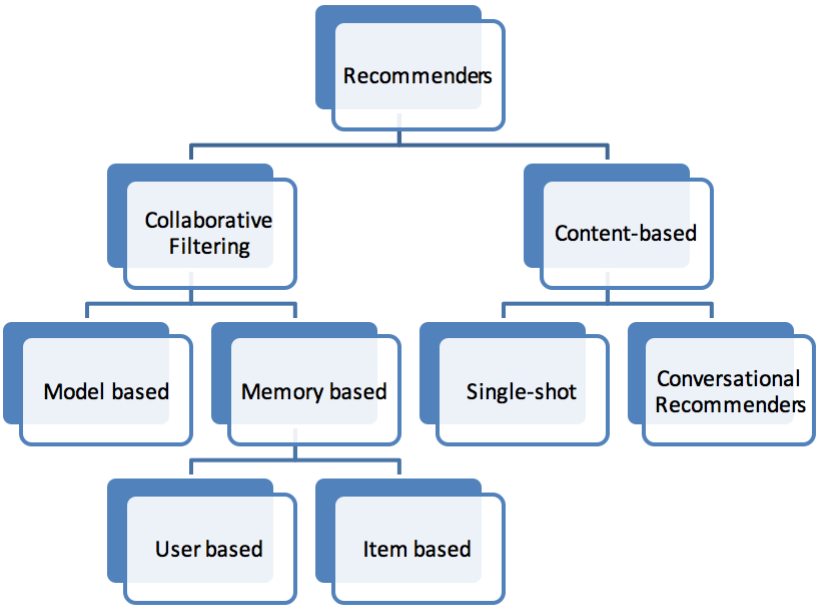


Figura 1: Clasificación de las Recomendaciones

En la Figura 1 se puede ver que los algoritmos basados en filtrado colaborativo se dividen en *model based* y *memory based*, la cual está formada también por los de tipo *user based* e *item based*. Los basados en filtrado por contenido se dividen en *Single-Shot* y en *Conversational Recommenders*, sobre los que se enfoca este proyecto.

2.1.1. Filtrado Colaborativo

El objetivo de los algoritmos basados en filtrado colaborativo[2][6] es sugerir nuevos ítems o predecir la utilidad de un determinado ítem al usuario basándose en opiniones del usuario y las opiniones de usuarios afines a éste. Normalmente se tiene una lista de usuarios y una lista de productos, en los que el usuario ha expresado sus opiniones. Las opiniones pueden ser dadas de forma explícita mediante una calificación o implícitamente por registros de compra, registro de hipervínculo web, etc.

Dentro de este tipo de sistemas de recomendación se han desarrollado diversas técnicas como *Memory-based* que utilizan toda la matriz de datos (usuarios e ítems) con sus calificaciones (*ratings*) para generar una predicción. Por otro lado los *Model-based* proporcionan recomendaciones de ítems generando primero un modelo de usuario. Estos algoritmos tienden a calcular el valor esperado de una predicción de un usuario, dada las calificaciones de otros ítems. La construcción del modelo se lleva a cabo mediante algoritmos de aprendizaje automático. *User based* utiliza técnicas estadísticas para encontrar un conjunto de usuarios, con una historia similar, es decir, o bien los diferentes tipos de artículos que prefieren son similares o bien compran un conjunto de ítems similares. Una vez obtenido el conjunto similar se utilizan técnicas como Nearest-Neighbor para combinar las preferencias y realizar una recomendación. *Item based* se basa en el cálculo de la similitud que existe entre varios ítems para determinar la mejor recomendación.

2.1.2. Filtrado basado en contenido

El filtrado basado en contenido[6] forma parte de un conjunto de técnicas de recomendación basadas en contenido (realiza filtraciones cognitivas), es decir, recomienda ítems basándose en una comparación entre el contenido (características) de los ítems y el perfil del usuario.

Diferentes algoritmos han ido desarrollándose dentro de este ámbito. La mayoría de sistemas utiliza una técnica denominada *singleShot*, que consiste en producir una única lista de recomendaciones para el usuario. Esta estrategia ha sido muy efectiva a lo largo de los años cuando los productos que debían recomendarse eran sencillos, como películas, libros, música, etc, pero se ha demostrado que no es tan adecuada para productos más complejos[1], por lo que

[1]Complejo: productos o ítems que el usuario no sabe exactamente que comprar debido al gran contenido de posibilidades o al gran número de características de un producto

debido a le necesidad de un enfoque más complejo, los investigadores desarrollaron una estrategia denominada *Recomendadores Conversacionales*.

2.2. Recomendadores Conversacionales

En este nuevo enfoque, los usuarios participan en un diálogo de recomendación [1], recibiendo recomendaciones y ofreciendo una serie de retroalimentaciones (basadas en críticas) sobre esas recomendaciones que le han sido presentadas, permitiendo al recomendador refinar la búsqueda y ofrecer un conjunto de productos más similares al que busca el usuario. Básicamente podemos resumirlos como aquellos sistemas que guían al usuario a través del espacio de productos, haciéndole a su vez sugerencias de productos y provocando *feedback*[2]. De entre algunos de los recomendadores conversacionales, destacan por encima del resto:

- **Value Elicitation**[8]: Esta metodología se basa en un valor específico que el usuario introduce, el cuál permite hacer una búsqueda muy efectiva respecto a una característica, por ejemplo, precio = 500 euros. La desventaja de este enfoque es que el usuario debe tener un amplio conocimiento acerca de las características del producto o ítem a buscar, sino este sistema es ineficaz.
- **Ratings-based feedback**[8]: A diferencia del enfoque anterior, *Ratings-based feedback* no precisa un gran conocimiento previo del usuario acerca de las características del producto que busca. Los usuarios asignan una calificación sencilla, por ejemplo, 3 estrellas de 5 indicando la satisfacción del producto recomendado. En este enfoque el usuario no introduce una retroalimentación acerca de las características del producto.
- **Preference-based feedback**[8]: es un caso especial, en el que en lugar de clasificar un determinado conjunto de recomendaciones, el usuario simplemente indica su recomendación preferida, es decir, elige la mejor opción de entre las posibles. Es un enfoque de muy bajo coste de retroalimentación por parte del usuario, que además requiere un mínimo conocimiento de dominio, únicamente la habilidad de distinguir lo que es una buena de una mala recomendación. Ahora bien, está claramente limitada en el contenido de la información.
- **Critiquing**[1][2][5]: forma parte de los recomendadores conversacionales y es una forma común de feedback del tipo "Muéstrame más como ésto pero...", donde los usuarios proporcionan unas limitadas características acotando el espacio de valor de una característica, es decir, si un usuario desea un producto barato, entonces basará sus críticas en la característica precio, indicando precio más bajo.

En la Figura 2 se puede observar los diferentes enfoques en los que se dividen los Recomendadores Conversacionales: *Value Elicitation, Critiquing,*

[2]Feedback, retroalimentación: método de control de sistemas, en el cual los resultados obtenidos de una tarea o actividad son reintroducidos nuevamente en el sistema con el objetivo de realizar las modificaciones necesarias

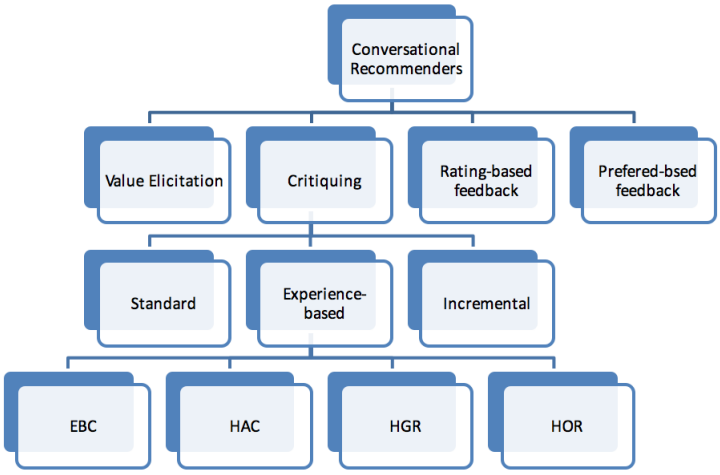


Figura 2: Clasificación de los Recomendadores conversacionales

Rating-based feedback, Prefered-based feedback. En el caso de *Critiquing*, éste se divide en los algoritmos *Standard, Incremental* y la familia de los *experience-based* formada por los algoritmos *Experience-Based Critiquing (EBC), History-Aware Critiquing-Based (HAC), History-Guided Conversational Recommendation (HGR), History Opinion Recommender (HOR).*

2.3. Critiquing

En la última década se ha producido un gran avance en cuanto a la investigación en el terreno de los sistemas de recomendación basados en *Critiquing*[1][2][5], la cual ha sido influenciada en gran medida por las necesidades cambiantes de los usuarios y la variedad de productos cada vez más complejos, así como las ventajas que ofrece: es simple de implementar y es apropiado para usuarios que no tienen mucho conocimiento del producto que quieren comprar.

Durante los últimos años, una variedad de algoritmos de recomendación basados en críticas han sido propuestos con el objetivo de demostrar su potencial y mejorar el proceso de recomendación.

El proceso de los recomendadores basados en *Critiquing* sigue el siguiente ciclo durante el proceso de recomendación: 1) Se le pregunta al usuario acerca de sus preferencias sobre uno o varios productos, y éste informa de algunas preferencias sobre algunas determinadas características. 2) Posteriormente, el sistema devuelve uno o varios productos de acuerdo con las preferencias iniciales. 3) El usuario puede seleccionar un ítem y terminar la sesión, pero en la mayoría de los casos el usuario no está conforme con el primer producto ofrecido, por lo que realiza una crítica sobre éste producto, reportando así una retroalimentación. 4) Estas críticas sirven al recomendador para actualizar las recomendaciones y recomendar otros productos en otros ciclos que se ajusten a las preferencias

indicadas por el usuario.

El funcionamiento del Critiquing está sustentado por diferentes formas de presentar el feedback del usuario: *Unit Critiquing*, el usuario está realizando una crítica sobre una característica del producto en cada ciclo, por lo que esta crítica actúa como filtro sobre los ítems a analizar, p.ej. restaurantes más baratos (se está realizando un filtro por precio, por lo que el recomendador mostrará el conjunto de restaurantes que más se aproxime al filtro marcado), y *Compound Critiques*, que permiten al usuario filtrar por un colección de *Unit Critiquing*. Cabe destacar que *Compound Critiques* son dinámicamente generadas durante cada ciclo de recomendación, es decir, no las construye el usuario. Se ha demostrado a través de diversos estudios que la utilización de *Compound Critiques* por parte de los usuarios conlleva una mejora del beneficio reduciendo la longitud de las sesiones en un 20-50 por ciento comparado con sistemas que sólo usan *Unit Critiques*.

2.3.1. Standard Critiquing

El algoritmo de *Standard Critiquing*[5][11] corresponde a una de las técnicas más simples de Critiquing. Consiste en aplicar el proceso de critiquing descrito en el apartado anterior con una particularidad, el usuario acepta el ítem recomendado en el primer ciclo (ver fase *Retrieve* en la Figura 3), es decir, el usuario introduce sus preferencias acerca de un producto, el sistema genera una serie de ítems que concuerden con las preferencias introducidas, y el usuario puede aceptar uno de esos elementos como producto final, o bien puede realizar una crítica sobre esos productos (ver acción de *feedback* en la fase *Reuse* y obtener una nueva recomendación de productos, (ver fase *Revise* en la Figura 3) en la que se revisa el producto por parte del recomendador y se actualiza el nuevo producto a recomendar.

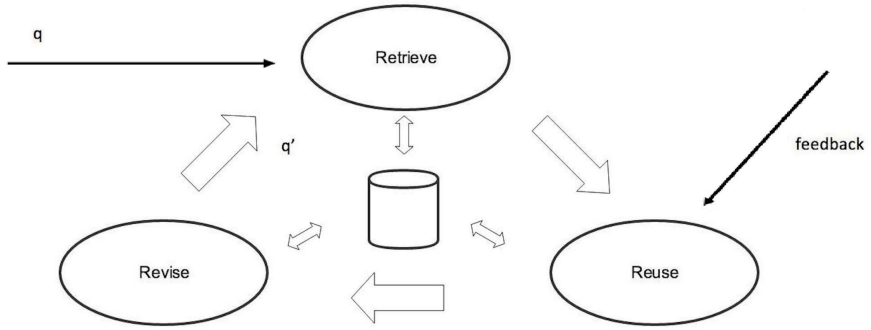


Figura 3: Ciclo del algoritmo Standar Critiquing

2.3.2. Incremental Critiquing

Uno de los problemas principales del enfoque anterior, el *Standard critiquing* es que durante el ciclo de recomendación actual, no tiene en consideración las críticas realizas por el usuario anteriormente en la sesión. Ésta es una de las grandes mejoras que incorpora esta nueva técnica, ya que muchos usuarios no tienen claras sus preferencias al principio de la sesión, por lo que a medida que avanza ésta, los usuarios van refinando su búsqueda en cuanto a las características del producto. *Incremental Critiquing*[3][4][5] considera las críticas realizadas en cada ciclo de recomendación, además permite solventar el problema de la incompatibilidad entre las críticas. Por ejemplo, si un usuario está buscando una cámara, basa su crítica de esta sesión en el precio y busca una más barata que 300 euros, pero en ciclos posteriores refina su búsqueda y desea cámaras más caras que 300 euros, ésto genera una inconsistencia. Incremental critiquing solventa este problema mediante la incorporación de un *Modelo de usuario*, que almacena un conjunto de críticas realizadas por el usuario durante la sesión *(*$U = u_1, u_2, ...u_n$*)* donde u_i es una *Unit Critique*. La estrategia IC, actualiza el modelo de usuario dando prioridad a la nueva crítica y eliminando las inconsistencias anteriores que ésta genere (véase fase *updateModel* Figura 4).

$$Compat(c', U) = \frac{\sum_{\forall_i} satisfies(U_i, c')}{|U|} \tag{1}$$

$$IQual(c', c, U) = \alpha * Compat(c', U) + (1 - \alpha) * Sim(c', c) \tag{2}$$

Además IC ordena los casos dependiendo de la similitud que éstos tienen con el caso de recomendación, por lo que se contabiliza un factor de compatibilidad (véase Ecuación 1) para cada candidato. Esta compatibilidad es el porcentaje de críticas en el modelo de usuario que satisface el nuevo caso, devuelve 1 cuando satisface la crítica y 0 en caso contrario. Esta compatibilidad es combinada con la similitud del candidato *c'* al caso de recomendación *c*, con el objetivo de obtener la *Quality*, véase Ecuación 2. Por defecto alpha = 0.75 en la ecuación 2. Finalmente, ésta se utiliza para ordenar los diferentes casos con el objetivo de priorizarlos para la siguiente recomendación.

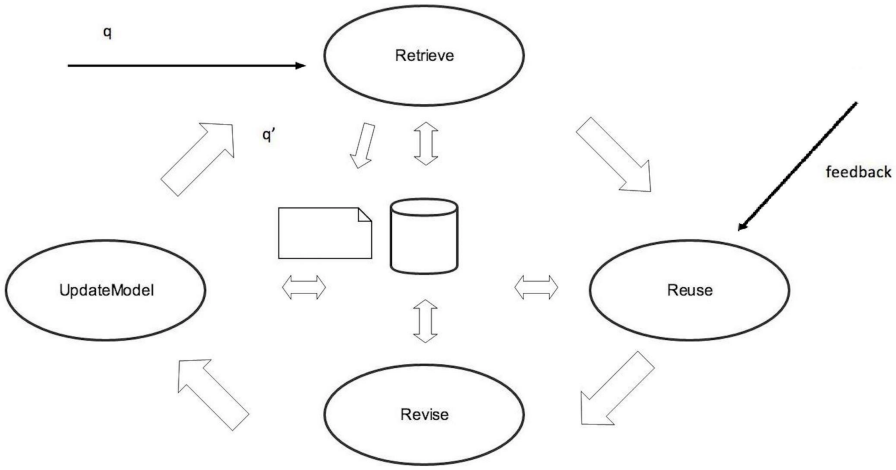


Figura 4: Ciclo del algoritmo Incremental Critiquing

2.3.3. Experience-based

Este tipo de algoritmos de recomendación están inspirados en ideas que provienen del razonamiento basado en casos (CBR), el cuál propone aprovechar un nuevo tipo de recurso durante el proceso de recomendación, las críticas de los usuarios en sesiones anteriores. *Experience-based*[6] se basa en el principio de las experiencias exitosas, es decir las experiencias previas que han derivado en una compra, poseen patrones que pueden ayudar a acotar el proceso de crítica para usuarios futuros.

Normalmente los usuarios en un sistema de recomendación de critiquing-based producen una larga colección de sesiones de críticas. Estas sesiones reflejan la experiencia de estos usuarios y capturan el conocimiento útil sobre sus preferencias. Experience-based utiliza esta información en el proceso de recomendación, pero como se ha nombrado anteriormente, únicamente de las experiencias exitosas.

A continuación se detallan los algoritmos más conocidos en este tipo de recomendadores.

2.4. Experience-Based Critiquing

Experience-based Critiquing (EBC)[6] se basa en el critiquing convencial reutilizando sesiones pasadas para guiar al usuario en el proceso de crítica. La idea principal de actuación de este enfoque es recomendar al usuario uno de los ítems que en situaciones pasadas han derivado en compra o aceptación en un escenario similar, en lugar de recomendar el ítem más similar a la recomendación actual. Este recomendador se orienta en 3 pasos:

1. **Construcción de casos de experiencia**: Antes de utilizar casos base, obtenidos de la experiencia de usuarios previos, es necesario construir estos casos base por medio de sesiones anteriores de estos usuarios. Este proceso no se basa únicamente en extraer críticas de recomendaciones pasadas, ya que en muchas ocasiones los usuarios pueden cambiar de opinión acerca del producto que buscan en medio de una sesión. Por tanto se enfoca la crítica trabajando hacia atrás a través de la sesión comenzando por la última crítica antes de la aceptación. Si la crítica actual entra en conflicto con la crítica menos reciente, ésta se elimina. Tras ésto, se obtiene un conjunto de críticas que representan los límites de las preferencias del usuario con respecto a las características criticadas. Cuando este proceso concluye, se obtiene el caso final (producto comprado en esa sesión), el cual será el candidato de esta experiencia en particular.

2. **Identificación de sesiones de crítica relevantes**: Cuando un usuario aplica una crítica c_i sobre un ítem recomendado r_m, se utilizará la actual sesión de crítica c_1,...,c_m como una *query* q_T, sobre el caso base de sesiones de crítica pasadas, con el objetivo de identificar un conjunto de sesiones relevantes (son aquellas que tienen una cierta superposisición con la *query* actual, basada en una métrica particular de superposición, véase Ecuación 3 y 4).

$$OverlapScore(q_T, s_i) = [\sum_{c_i \,\epsilon\, q_T} \sum_{c_j \,\epsilon\, s_i} match(c_i, c_j)]^2 \tag{3}$$

$$S^{REL} = RelevantSessions(q_T, S) = \{s_i \,\epsilon\, S : OverlapScore(q_T, s_i) > t\} \tag{4}$$

3. **Ordenación de candidatos a recomendar**: Cada sesión relevante consiste en una secuencia de críticas que previamente han dirigido a un usuario a un final exitoso, es decir, a encontrar el producto de su agrado. El caso de recomendación final obtenido de las sesiones de experiencia con el *OverlapScore* mayor será una buena recomendación para el usuario. Cabe destacar, que el caso de recomendación obtenido r_T será compatible con la última crítica realizada por el usuario, y habrá sido seleccionada como una recomendación final anteriormente, ver ecuación 5.

$$r_T = Recommend(S^{REL}) = argmax_{\forall s_i \,\epsilon\, S^{REL}}\{OverlapScore(q_T, s_i)\} \tag{5}$$

En resumen, la técnica de *experience-based critiquing* recomienda un ítem que ha sido frecuentemente aceptado por usuarios pasados en condiciones similares. Normalmente, el ítem recomendado no tiene porque ser el ítem más similar al ítem criticado, permitiendo al sistema de recomendación avanzar mucho más rápido en cuanto al acotamiento del proceso de selección de ítems y con suerte mejorar la eficiencia general de la recomendación.

2.5. History-Aware Critiquing-Based

Tras demostrar que el algoritmo EBC[6] lograba una mejora considerable en reducir las longitudes de las sesiones respecto a STD[5][11], se continúa con la investigación de nuevos enfoques en busca de mayores mejoras en el rendimiento.

El punto de partida para estas nuevas mejoras, parte de la demostración de que EBC no ha sido tan eficaz respecto a IC[3][4] y tiende a producir sesiones más largas que éste, por ello se propone un nuevo enfoque de EBC que consiga mejorar la eficiencia sobre STD e IC. El enfoque anterior, EBC, se basaba en la similitud de los patrones de crítica. Ahora bien, algunas veces los resultados que son recomendados al usuario no son los que él esperaba. Para comprender mejor este problema, supongamos que el usuario busca un restaurante indio y basa sus críticas en un precio reducido. EBC, compararía las críticas del usuario más antiguas con cada una de las críticas del caso base con el objetivo de identificar un conjunto de sesiones relevantes y generar así una recomendación. Esta recomendación irá acorde con sus críticas, pero quizás no cumpla con sus expectativas, es decir, es posible que se le presente al usuario un restaurante Francés. Por lo que EBC pone especial atención en los patrones de crítica pero no en el tipo de ítem que se le recomendará al usuario.

Teniendo en cuenta la problemática anteriormente planteada, el algoritmo *history-aware critiquing-based (HAC)*[7] propone añadir un nuevo componente al algoritmo EBC que permita tener en cuenta también la similitud entre los ítems cuando se seleccionan sesiones relevantes de recomendación. Durante el curso de una sesión de recomendación las necesidades del usuario son refinadas mediante las críticas que éste hace hacia los ítems que se le van recomendando. Cada sesión de recomendación s se presenta como una secuencia de pares de críticas s=p_1,p_2,...,p_n donde p_i=(r_i,c_i), en la que r_i representa la recomendación y c_i representa la crítica que se aplica. Cada crítica c_i es representada como una tripla, (f_i,v_i,$type_i$), donde f_i es la característica, v_i es el valor de f_i, y $type_i$ el tipo de crítica (<, >, =, <>). Cuando un usuario realiza una crítica sobre una recomendación, se generan un conjunto de pares, p_1,p_2,...,p_m que es usado como query q_T sobre el caso base de pasadas sesiones de recomendación, con el objetivo de identificar un conjunto de sesiones relevantes.

$$averageSimilarity(q_T, s) = \frac{\sum_{r_i \,\epsilon\, q_T} \sum_{r_j \,\epsilon\, s} sim(r_i, r_j)}{|q_T| * |s|} \tag{6}$$

En la Ecuación 6 se define el promedio de similitud entre los ítems de recomendación que se generan en la actual sesión de recomendación y los ítems en pasadas sesiones de recomendación en q_T. Sim(r_i,r_j) calcula la similitud entre dos ítems, (r_i, ítem que se recomienda en la sesión actual y r_j que se recomienda en otra sesión pasada).

$$Quality(q_T, s) = \alpha * averageSimilarity(q_T, s) + (1 - \alpha) * OverlapScore(q_T, s) \tag{7}$$

Se introduce en la Ecuación 7, una métrica que combina el promedio de la Ecuación 6 y la crítica Overlap, donde los pesos relativos de la similitud y el overlap son controladas por un factor α.

$$S^{REL} = RelevantSessions(q_T, S) = \{s \epsilon S : Quality(q_T, s) > t\} \tag{8}$$

La Ecuación 8 permite identificar un conjunto de sesiones relevantes de recomendación de un conjunto de sesiones pasadas S. Aunque se controla, la mínima calidad requerida es controlada con un threshold t, aproximadamente los valores de t y α variarán entre 0.8 y 0.75 respectivamente.

$$r_T = Recommend(S^{REL}) = argmax_{\forall_s \epsilon S^{REL}} \{Quality(q_T, s)\} \tag{9}$$

Finalmente la Ecuación 9 define como se aceptan recomendaciones de sesiones pasadas que tienen la mayor puntuación de calidad y es presentada como una nueva recomendación, r_T en la actual sesión de recomendación.

2.6. History-Guided Conversational Recommendation

Continuando con el estudio del desarrollo de los enfoques basados en critiquing, *History-Guided Conversational Recommendation (HGR)*[8] es capaz de utilizar los pares de recomendación (ítem y crítica) o las críticas sólo en la sesión de recomendación actual para predecir las recomendaciones de ítems más probables y, por tanto, acortar las sesiones de recomendación que a veces son más prolongadas en otros enfoques.

El punto de partida está inspirado en el razonamiento basado en casos, esta nueva técnica intenta aprovechar las experiencias de otros usuarios durante el proceso de crítica. Al igual que en HAC[7], este nuevo enfoque se basará en aquellas sesiones que hayan resultado ser exitosas, es decir, que se haya aceptado la recomendación. Esta metodología asume que existen patrones de recomendación en las críticas, los cuales nos ayudan en los procesos futuros de recomendación. Se procede a describir HGR con el que se consiguen grandes mejoras de eficiencia en comparación con IC y HAC.

Durante una sesión de recomendación los usuarios van refinando sus ideas de lo que buscan, lo que conlleva cambios en las características de búsqueda. En esta metodología se define la sesión s_i como una secuencia de pares de recomendación-crítica, donde r_i representa la recomendación y c_i la crítica

que el usuario aplica a la recomendación. C_i está representada por una tripla *(f_i,v_i,$type_i$)*, en la que f_i es la característica perteneciente a r_i que la crítica aplica, v_i es el valor de f_i, y $type_i$ es el tipo de crítica que se aplica (<,>,=,<>).

Como se ha comentado en algoritmos de recomendación anteriores orientados a critiquing-based, en el sistema hay usuarios que producen sesiones de crítica *($S = s_1$,...,s_k)*, las cuales reflejan la experiencia de esos usuarios.

Para el resto del desarrollo se asumirá que la sesión aceptada es la *solución* del caso y los pares de crítica que le siguen serán la *especificación* del caso.

HGR extiende el critiquing conversacional reutilizando sesiones pasadas para guiar el proceso de crítica. En lugar de recomendarle un ítem que sea más similar a sus preferencias, se le recomienda un ítem que haya sido aceptado por otros usuarios pasados en sesiones anteriores. A continuación, se dividirá el proceso de recomendación en tres partes, para comprender mejor su funcionamiento.

1. **Identificación de sesiones de crítica relevantes**: Cuando un usuario aplica una crítica c_i hacia un ítem recomendado r_m, se usará la sesión parcial actual *(r_1,c_1),...(r_m,c_m)*, como una *query* q_T, sobre el caso base de sesiones de críticas pasadas, para identificar sesiones relevantes. Por ello, se propone el uso de la Ecuación 10 que calcula el número de pares de recomendación de q_T que están presentes en la actual sesión.

$$OverlapScore(q_T, s_i) = \sum_{(r_i,c_i)\,\epsilon\,q_T} \sum_{(r_j,c_j)\,\epsilon\,s_i} match((r_i, c_i), (r_j, c_j)) \quad (10)$$

   Si no hay sesiones relevantes y no hay candidatos a recomendar se utiliza la Ecuación 11 que calcula sólo el número de críticas en q_T que están en la sesión actual.

$$OverlapScore(q_T, s_i) = \sum_{c_i\,\epsilon\,q_T} \sum_{c_j\,\epsilon\,s_i} match(c_i, c_j) \quad (11)$$

   Finalmente se usa la Ecuación 12 para obtener las sesiones relevantes. Cabe destacar que en el caso de no existir sesiones relevantes y por tanto candidatos a recomendar, se pasa al IC y se devuelve el ítem más similar a la recomendación actual y compatible con las críticas del modelo de usuario.

$$S^{REL} = RelevantSessions(q_T, S) = \{s_i\,\epsilon\,S : OverlapScore(q_T, s_i) > t\} \quad (12)$$

2. **Ordenación de candidatos a recomendar**: Las sesiones relevantes (SREL) han llevado previamente a usuarios hacia un resultado exitoso. Cada sesión relevante termina con una recomendación r_F, la cual genera un candidato para la siguiente recomendación. Por lo que se da preferencia a las recomendaciones que han sido aceptadas en sesiones pasadas

mediante un ranking de puntuaciones según su *RecScore* (ver Ecuación 13).

$$RecScore(r_F, q_T, S^{REL}) = \sum_{\{\forall s_i \epsilon S^{REL} : r_F = terminal(s_i)\}} OverlapScore(q_T, s_i) \quad (13)$$

3. **Filtrado de candidatos con conflictos**: No hay garantía de que estos candidatos sean compatibles con las críticas del usuario, por lo que en el último paso se eliminarán aquellos candidatos a recomendar que sean incompatibles con las críticas del usuario. Debido a que muchos usuarios cambian sus preferencias durante una sesión, se editan las críticas del usuario, comenzando por la última crítica, la más reciente. Si una crítica entra en conflicto con una más reciente, la más antigua se elimina. Una vez terminado este proceso, los ítems que no satisfagan las críticas del usuario serán eliminados de la lista de candidatos a recomendar, por lo que los candidatos que queden en la primera parte de esta lista, serán recomendados al usuario.

2.7. History Opinion Recommender

Hasta ahora, ningún algoritmo de recomendación integraba las propuestas hechas en la literatura. *History Opinion Recommender (HOR)*[9] integra esas propuestas previas en los últimos recomendadores desarrollados, y a la vez incluye un nuevo tipo de información extraída de los usuarios, que corresponde a sus opiniones expresadas en diferentes sitios web. Para una mayor comprensión del algoritmo que se va a desarrollar se definen las siguientes estructuras:

- **Base de casos**: es un conjunto de productos para recomendar, $CB=p_1,...,p_n$, donde p_i es un producto y el conjunto de características que describe cada producto se define como $F = f_1,...,f_m$
- **Modelo de usuario**: el modelo de usuario U es un conjunto de pares de críticas de recomendación, $U=u_1,...,u_k$, en la que $ui=(r_i,c_i)$ donde r_i representa el producto recomendado y c_i representa la crítica.
- **Base de sesiones**: es un conjunto de sesiones de crítica pasadas de otros usuarios, $SB=s_i,...,s_l$ donde s_i es una secuencia de pares de recomendación (ri,ci), y cada sesión termina con un producto terminal, $term_s i$
- **Base de opiniones**: es un conjunto de opiniones de usuario definidas como $OB=o_1,...,o_n$, donde o_i es el ratio de opiniones del producto obtenido de los usuarios.
- **Crítica**: una crítica ci está representada por una tripla $(f_i,type_i,v_i)$, donde f_i es la característica en una recomendación, $type_i$, es el tipo de $ci(<,>,<>)$, y v_i es el valor de f_i.

El proceso de recomendación se inicia cuando se recibe la *query* p_q inicial junto con la base de casos CB, la base de sesiones SB, la base de opiniones OP. Tras ésto, se comienza con un ciclo que tiene 4 fases: *Recommend Item, User Review, Query Revise, Update model*, y se repite hasta que el usuario acepta la recomendación.

En la primera fase *RecommendedItem*, se elige un nuevo producto a recomendar p_r, basado en la *query* actual p_q, en las críticas de sesiones pasadas de usuarios SB, en otras opiniones de usuario, y previos pares de recomendación. El procedimiento *RecommendedItem* usa la actual sesión de crítica *(r_1,c_1)...(r_m,c_m)*, guardada en el modelo de usuario U, sobre las sesiones pasadas guardadas en SB, con el objetivo de encontrar *Sesiones Relevantes*. Primeramente, se calcula el número de pares de recomendación en el modelo de usuario, que están presentes en cada sesión pasada s_i (ver Ecuación 14 y 15).

$$OverlapPair(U, s_i) = \sum_{\forall (r_i,c_i)\, \epsilon\, U} \sum_{\forall (r_j,c_j)\, \epsilon\, s_i} matchPair((r_i, c_i), (r_j, c_j)) \quad (14)$$

$$matchPair((r_i, c_i), (r_j, c_j)) = \begin{cases} 1 \;\; if\, (r_i = r_j)\, and\, (c_i = c_j), \\ 0 \;\; otherwise \end{cases} \quad (15)$$

Segundo, si no hay sesiones relevantes, se calcula el número de críticas que hay en el modelo de usuario, que están presentes en cada sesión pasada s_i, (ver Ecuaciones 16 y 17).

$$OverlapCritique(U, s_i) = \sum_{\forall (c_i)\, \epsilon\, U} \sum_{\forall (c_j)\, \epsilon\, s_i} matchCritique(c_i, c_j) \quad (16)$$

$$matchCritique(c_i, c_j) = \begin{cases} 1 \;\; if\, c_i = c_j, \\ 0 \;\; otherwise \end{cases} \quad (17)$$

En el cálculo de sesiones relevantes se obtiene un conjunto de sesiones *SREL*, en las que cada sesión termina aceptando el producto recomendado *term(s_i)*, y cada sesión contiene también su *Overlapscore*, (calculado previamente en Ecuación 14 y 16). Cuando *SREL* está vacío HOR recomienda productos aceptados de estas sesiones relevantes.

$$RecScore(U, s_i, o_j) = \alpha * OverlapScore(U, s_i) + (1 - \alpha) * o_j \quad (18)$$

RecScore (véase Ecuación 18), es el promedio de *OverlapScore* y la clasificación de opiniones o_j perteneciente a OB del producto p_j de la sesión s_i. La principal diferencia de HOR con HGR, es que éste último sólo usa el *overlapScore* de las sesiones relevantes para ordenar los productos.

Posteriormente se procede a filtrar los candidatos y eliminar aquellos que no satisfacen el modelo de usuario U y finalmente se recomienda el producto con mayor puntuación en la clasificación. Otros algoritmos de *experience-based*, cuando no existen sesiones relevantes consideran utilizar el enfoque *Standard* o *Incremental*, en cambio este algoritmo propone utilizar un algoritmo incremental modificado. Para ésto, se obtiene la última crítica del modelo U y se filtran todos los productos de CB que satisfagan esta crítica, y si no hay críticas se considera toda la base de casos. Posteriormente, se ordenan todos los candidatos basándose en la medida de calidad Q, aunque se utiliza una modificación de ésta usando *Reinforcement Learning* y calculando la similitud. Además se aplica un mecanismo para el cálculo de la ponderación en la similitud *(Local Weighting, LW)*, el cuál, está basado en el modelo de usuario U, y permite la detección automática de la importancia de cada característica para el cálculo de la similitud. Es decir, LW prioriza la similitud de aquellas características que no han sido criticadas aún por el usuario. Finalmente, se recomienda el ítem p_r, con la mayor calidad.

En la segunda fase, *UserReview*, el usuario realiza la crítica. HOR recibe la retroalimentación por parte del usuario y elimina la recomendación del producto actual p_r de la base de casos CB. Finalmente, esta fase termina devolviendo la base de casos y la crítica.

En la tercera fase, *QueryRevise*, se basa en la revisión de la query p_q para el siguiente ciclo. Básicamente, la recomendación del producto actual p_r llega a ser la nueva query producto p_q.

En la fase final, *UpdateModel* se guarda el producto recomendado p_r y la crítica actual U_{cq} en el modelo de usuario U. Cabe destacar que al añadir críticas puede hacer inconsistencias con críticas anteriores, por tanto se eliminan aquellas críticas que sean inconsistentes, manteniendo en el modelo de usuario la más reciente.

2.8. Minería en los senderos de usuario

En el contexto de sistemas de recomendación basados en crítica, un *sendero* puede ser definido como la ruta que el usuario sigue a partir de un producto, crítica y elección del producto preferido en cada ciclo hasta que éste llega al producto con el que esta satisfecho.

En la mayoría de los sistemas de recomendación conversacionales, los usuarios dejan senderos a medida que exploran el espacio de productos. Este algoritmo quiere demostrar el potencial de los conocimientos extraídos de dichos senderos con la finalidad de reducir el número de ciclos de interacción necesarios para alcanzar el producto que interesa al usuario.

Cuando los usuarios critican un producto y toman decisiones, evalúan implícitamente utilidades. Por lo tanto, las críticas de los productos de los diferentes senderos se pueden utilizar para inducir un modelo de usuario público, que se

puede utilizar para clasificar las recomendaciones y mejorar la experiencia de usuario. Se proponen 4 métodos para clasificar las recomendaciones basadas en grafos de preferencias. El objetivo es usar grafos de preferencias para inducir un modelo de preferencia sobre los datos y el uso del mismo para clasificar las recomendaciones.

Frecuency based ranking (FR)

Frecuency based ranking[14] es un enfoque básico donde las recomendaciones son clasificadas en base a la frecuencia en la que unos productos son preferidos respecto a otros, aplicando la misma crítica sobre el último conjunto de senderos.

Critique specific PageRank (CPR)

La hipótesis principal del enfoque *Critique specific PageRank*[14] se basa en que: *La utilidad de un elemento es alta a través de una cierta crítica, si esa misma crítica se prefiere sobre elementos de alta utilidad.* Este algoritmo tiene su homólogo en el *PageRank*[14]. El algoritmo de *PageRank* asigna un número real a cada nodo en un grafo con el objetivo de que cuanto mayor es la puntuación *PageRank* de un nodo, más importante es éste. Este algoritmo es usado normalmente en escenarios web para medir la importancia de las páginas web basándose en sus *links*. La utilidad se calcula según la Ecuación 19:

$$U(r_i) = \sum_j U(r_j) * \frac{w_{ji}}{W_j} \ (W_j = \sum_k w_{jk}) \tag{19}$$

donde $U(r_i)$ es la utilidad, $w_j i$ es el peso de la arista de r_j a r_i. Cabe destacar que a diferencia de FR, CPR tiene en cuenta las asociaciones de orden superior entre los ítems en el gráfico de la preferencia.

Restaurant and critique specific Pagerank (RCPR)

En esta sección se propone un método de *PageRank* sensible a cada restaurante. Los grafos de preferencias de críticas están formados específicamente para cada restaurante. Este método está basado en un dominio de *Restaurant*. Aunque hay una importante diferencia en las aristas que se añaden al grafo. En esta metodología los grafos están formados por la agregación de preferencias de todos los senderos. Una vez que éstos están formados, el *PageRank* es usado para facilitar el ordenamiento entre los elementos que pueden a su vez ser utilizados en la clasificación de las recomendaciones.

Fusion of CPR and RCPR

Este método es una combinación lineal de CPR y RCPR, donde el valor

α se determina usando una validación-cruzada. RCPR, es sensible a los elementos criticados. Ahora bien, sufre el problema de la escasez, puede que no haya suficientes senderos específicos de un ítem para favorecer la ordenación. Por lo que la fusión de ambos enfoques utiliza lo mejor de cada uno, usa CPR para suavizar los modelos de preferencias y compensar la escasez de RCPR.

$$Fusion_{score} = \alpha * RCPR_{score} + (1 - \alpha) * CPR_{score} \tag{20}$$

2.9. Collaborative Compound Critiques

Se ha demostrado a lo largo de diferentes experimentos[1][11][16] en los algoritmos existentes que el número de ciclos en una sesión puede disminuir cuando se usan *Compound Critiques*. Además éstas, permiten a los usuarios ver los atributos que son altamente dependientes entre sí. Por ello se muestra en este apartado la manera de incorporar *Compound Critiques* mediante el enfoque *Collaborative Compound Critiques*, el cuál, en lugar de simplemente contar las críticas solapadas entre usuarios, desarrolla una medida basada en la similitud gráfica *(graph-based)* para identificar sesiones similares.

Básicamente, la evolución en el proceso de recomendación se divide en: generación de las críticas compuestas, identificación de sesiones similares, y recomendación de ítem.

Generación de Compound Critiques. Una crítica compuesta denotada como C_i, es un elemento que posee un conjunto de críticas únicas.

$$C_n = (attribute_n, operator_n, value_n) \tag{21}$$

donde $attribute_n$, es una categoría para realizar críticas, $operator_n$ es un elemento en el operador ($=$, $<$, $>$) y $value_n$, es el valor del operador. La generación de críticas compuestas se puede dividir en 2 pasos, primero, un conjunto de atributos frecuentes, son descubiertos mediante la asociación del algoritmo *Apriori*, y que serán seleccionados como candidatos de crítica. Cada candidato se calcula mediante la función (ϕ), ver Ecuación 22:

$$\phi(C_i) = U(C_i) + D(C_i, SC) \tag{22}$$

en la que tiene en cuenta tanto la utilidad $U(C_i)$ del actual candidato a crítica C_i, como la diversidad con las críticas seleccionadas hasta el momento $D(C_i,SC)$. Para el cálculo de la utilidad se define la ecuación 23:

$$U(C_i) = \sum_{n=1}^{|C_i|}(\alpha_n * w(attribute_n)) * \frac{1}{SR(C_i)} \sum_{i \epsilon SR(C_i)}^{|SR(C_i)|} u(i) \tag{23}$$

donde cada crítica única en C_i se asocia con un parámetro α_n, w es el atributo que marca la importancia, y $\frac{1}{SR(C_i)} \sum_{i \epsilon SR(C_i)}^{|SR(C_i)|} u(i)$ es el promedio de la utilidad de los ítems *SR(C_i)* que satisface C_i. Hay que nombrar que cada ítem *u(i)* es calculado en base a la teoría Multi-Attribute Utility Theory. Finalmente los candidatos con mayor (ϕ) serán presentados al usuario como posibles críticas.

Identificación de sesiones similares. En este enfoque, cada sesión de crítica está definida como una secuencia de críticas, y el objetivo es incorporar experiencias pasadas de los usuarios. La identificación de sesiones similares consiste en:

$$Sim(s_i, s_j) = \beta * ItemSim(s_i, s_j) + (1 - \beta) * OverlapScore(s_i, s_j) \tag{24}$$

donde *itemSim(s_i,s_j)* es el promedio de similitud entre todos los ítems en dos sesiones, *OverlapScore(s_i,s_j)* es el número de solapamientos de críticas entre dos sesiones, dónde β es 0.75.

El grafo de sesión para las sesiones de crítica s_k, se denota como G_k, y se representa como dos tuplas:

$$G_k =< V_k, E_k > \ \ (V_k = I_k \cup C_k; E_k = {E_k}^{i \rightarrow c} \cup {E_k}^{i \leftarrow c}) \tag{25}$$

donde el conjunto de vértices $I_k = \cup_{j=1}^{n} i_{j,k}$, denota todos los ítems en s_k, $C_k = \cup_{j=1}^{n} C_{j,k}$, el cual es el conjunto de vértices de todas las críticas contenidas en la sesión, y el conjunto de aristas E_k incluye dos tipos de arista: ${E_k}^{i \rightarrow c}$ desde un vértice ítem hasta un vértice crítica, y ${E_k}^{i \leftarrow c}$ desde un vértice crítica a un vértice ítem. Por tanto, se puede observar como la ecuación presentada anteriormente para el cálculo de la similitud sólo considera los vértices y no las aristas, por lo anterior, se define la Ecuación 26:

$$RSim(G_x, G_y) = \frac{|I_x \cap I_y|}{|I_x \cup I_y|} * \frac{|C_x \cap C_y|}{|C_x \cup C_y|} * \frac{|{E_x}^{i \rightarrow c} \cap {E_y}^{i \rightarrow c}|}{|{E_x}^{i \rightarrow c} \cup {E_y}^{i \rightarrow c}|} * \frac{|{E_x}^{i \leftarrow c} \cap {E_y}^{i \leftarrow c}|}{|{E_x}^{i \leftarrow c} \cup {E_y}^{i \leftarrow c}|} \tag{26}$$

donde G_x y G_y son dos gráficos de sesión para sesiones de crítica s_x y s_y. Para evitar cero resultados, se adopta $e^{RSim(G_x, G_y)}$ como resultado final de similitud.

Recomendación del ítem. En este paso se define el ítem que se va a recomendar, por lo que se consideran como candidatos todos aquellos pertenecientes a las sesiones más similares, y entre éstos, se calcula la relevancia de cada ítem mediante la Ecuación 27:

$$R(i_t, q, S) = argmax\ e^{Rsim(G(q), G({s_k}^{(i_t)}))} \tag{27}$$

q es la actual sesión de crítica, S es el conjunto de sesiones de crítica similares de otros usuarios, $G({s_k}^{(i_t)})$ es el gráfico de sesión que empieza desde el principio hasta el ítem it en la sesión s_k y el máximo valor $e^{RSim(G_q, G({s_k}^{(i_t)}))}$ es utilizado como ítem i_t's del ranking de puntuación. Finalmente, el ítem con el máximo valor será recomendado al usuario.

3. Análisis y diseño

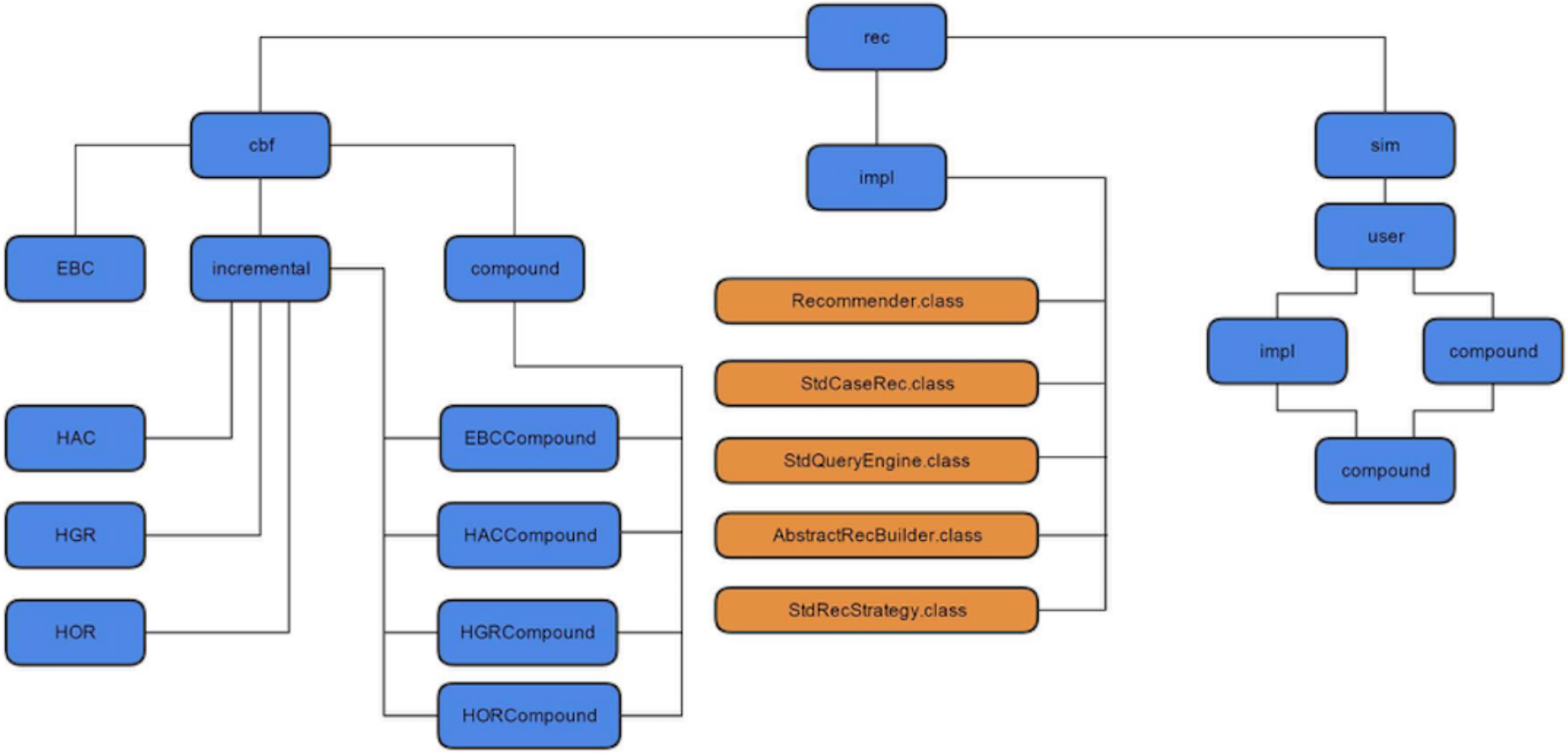
En este capítulo se va a introducir la plataforma sobre la cuál se han implementado los algoritmos desarrollados en este trabajo final de grado. Además, se van a introducir también los diferentes paquetes utilizados para cada algoritmo, cómo se relacionan entre ellos y cómo pueden ejecutarse sobre el simulador para obtener diferentes resultados con los que comparar la eficiencia de estos algoritmos.

A nivel global se utilizan dos programas: *Critique Recommender*, es el programa que incluye los paquetes, tanto de los algoritmos, como los que permiten realizar la simulación y obtener un fichero con las recomendaciones que se han generado, y *Reporter*, que es el programa que nos permite obtener un resultado a partir del fichero generado en el programa anterior, este proceso será explicado más adelante.

Destacar también que el lenguaje principal sobre el que se han desarrollado los algoritmos ha sido JAVA, ya que como se ha comentado en la introducción, partimos de un proyecto en desarrollo con la intención de adaptar nuevos algoritmos a éste. Además de JAVA, se ha utilizado el lenguaje XML para definir los datos que se van a utilizar y facilitárselos al simulador para que éste pueda ejecutarlo. Finalmente para la realización de la parte de *Clustering* se han utilizado varias librerías que permiten la unión del lenguaje JAVA con el lenguaje de análisis de datos R.

3.1. Paquetes Utilizados

Este trabajo de final de grado continúa con un trabajo ya existente, que como ya se ha comentado parte de unos algoritmos ya implementados mediante críticas unitarias. La plataforma actual permite la ejecución de estos algoritmos mediante los paquetes *cbf* (sin considerar *Compound Critiques*), *impl* y *sim*. *Cbf* contiene el algoritmo principal sobre el cuál se desarrollan los demás, *impl* contiene las clases necesarias para la ejecución del algoritmo estándar de recomendación, *sim* contiene las clases necesarias para realizar la simulación, *incremental* contiene lo necesario para la ejecución del algoritmo incremental critiquing y *compound* es el paquete que permite la generación de críticas compuestas. Partiendo de éstos, se han implementado los algoritmos: *EBCCompound*, *HACCompound*, *HGRCompound*, *HORCompound*. Se muestra a continuación un esquema principal con los paquetes que se han utilizado, se muestran también las clases principales del paquete *impl*, en las que todos los algoritmos basaran sus clases.


rec
cbf
impl
sim
EBC
incremental
compound
Recommender.class
user
impl
compound
StdCaseRec.class
HAC
EBCCompound
StdQueryEngine.class
compound
HGR
HACCompound
AbstractRecBuilder.class
HOR
HGRCompound
StdRecStrategy.class
HORCompound

3.2. Simulador

El simulador corresponde una parte de gran importancia en este proyecto ya que nos permite simular el comportamiento del usuario, es decir, nos permite simular sesiones de recomendación en las que un usuario parte de un caso base y realizando críticas según ciertas características que desea, obtiene un producto final. Esta herramienta facilita mucho el trabajo, ya que evita la dependencia de usuarios reales físicos para realizar pruebas.

Una vez definido que es el simulador, es necesario concretar y definir que datos necesita para la ejecución de cualquiera de los algoritmos que han sido desarrollados. Pues bien, como se ha introducido al principio del capítulo, la carga inicial de datos se va realizar mediante lenguaje XML.

Primeramente, el programa se inicia en la clase *TestMain*, mediante la siguiente sentencia, esta clase produce la unión entre ambos lenguajes, JAVA y XML.

$$FileSystemResource\ res\ =\ new\ FileSystemResource(\text{"experimentsIncrementalFernando.xml"})$$

El comando anterior, permite realizar la carga de los datos a nuestro programa y a continuación el objeto *Simulation Factory* guarda los datos extraídos en variables de nuestro sistema y comenzará a realizar las simulaciones.

Los datos se extraen de XML por medio de *Beans*[3] y se guardan en el sistema. Dentro de nuestro proyecto, existen diferentes tipos de bean, los cuales tienen los datos necesarios para realizar la simulación: queries, bases de datos, usuarios, algoritmos, etc.

El bean que indica que elementos se van a utilizar durante la simulación es el *Setup*. Como se puede comprobar en la figura, está dotado de ciertos atributos:

```
<bean class="rec.sim.Setup">
        <property name="label"><value>Proof-q1-IC-CC-SmartPhone</value></property>
        <property name="simUser"><ref bean="compoundUser"/></property>
        <property name="recommender"><ref bean="IC_CC"/></property>
</bean>
```

Figura 5: Bean Setup

- **label** indica el nombre del archivo *log* que va a generar el programa Critique Recommender, el cual va a contener los datos de las sesiones que se han simulado durante la ejecución.

[3]Bean: elemento perteneciente al lenguaje XML que permite almacenar datos en forma legible y de una manera sencilla.

- **simUser**: nos indica el tipo de usuario que se va a utilizar. Si el bean definido hace referencia a *stdUser*, estaremos ante un caso de usuario estándar, el cual utiliza críticas unitarias, o *compoundUser* que se basa en críticas compuestas. Cabe destacar también, que tanto stdUser como compoundUser hacen referencia a otros bean, definidos en el mismo archivo *experimentsIncrementalFernando.xml*.

```
<bean id="stdUser" class="rec.sim.user.impl.StdSimUser">
    <property name="recAcceptor">
        <bean class="rec.sim.user.impl.StdRecAcceptor"/>
    </property>
    <property name="recSelector">
        <bean class="rec.sim.user.impl.StdRecSelector"/>
    </property>
    <property name="querySelector">
        <bean class="rec.sim.user.impl.StdQuerySelector"/>
    </property>
</bean>
```

Figura 6: Bean para el caso de SimUser StdUser

```
<bean id="compoundUser" class="rec.sim.user.impl.StdSimUser" parent="stdUser">
    <property name="querySelector">
        <bean class="rec.sim.user.compound.LowSupportQuerySelector"/>
    </property>
</bean>
```

Figura 7: Bean para el caso de SimUser CompoundUser

- **recommender**: la definición del algoritmo que se va a utilizar funciona también mediante la asignación de un bean ya definido anteriormente en el mismo archivo. En el ejemplo mostrado en la Figura 8, se puede ver como el algoritmo utilizado es el IC_CC que es el id de un bean definido anteriormente, el cual hace referencia al algoritmo Incremental Critiquing mediante Compound Critiques.

```
<bean id="IC_CC" class="rec.impl.Recommender" parent="compound">
    <property name="caseRec">
        <bean class="rec.incremental.IncrementalCaseRec"/>
    </property>
    <property name="recStrategy">
        <bean class="rec.incremental.IncrementalStrategy"/>
    </property>
</bean>
```

Figura 8: Algoritmo Incremental Compound Critiquing

En la Figura 8 se pueden observar los paquetes y las clases que utiliza este algoritmo, aunque entraremos más a fondo en esto cuando expliquemos el proceso que siguen los algoritmos implementados.

Finalmente, el bean de *Simulation Factory* contendrá dos propiedades (*property*):

<property name="casebase"><ref bean="smartphoneCasebase"/></property>
<property name="queries"><ref bean="smartphoneQueries"/></property>

Casebase nos indicará los casos base de los productos existentes en el sistema y *queries* que emula las críticas que realiza el usuario en el proceso de recomendación. Se muestra a continuación un ejemplo de un caso que puede estar incluido en la base de casos CB que se utiliza durante la ejecución:

CASE0 |<Manufacturer>Nokia |<Model>Lumia1520 |<Weight>209.0 |<Size>6.0 |<Storage>32.0 |<Ram>2048.0 |<Resolution>20.0 |<SO>Microsoft Windows Phone 8 Black |<CPU>Quad-core 2.2 GHz Krait 400 |<Price>550.0

3.3. Queries

Cabe destacar que en el proyecto se han utilizado varios archivos queries para realizar las simulaciones. Para cada base de datos, por ejemplo *SmartPhone*, se ha trabajado con los archivos *queries-n.txt*, siendo n el número de características que contiene el producto que el usuario está buscando. Por tanto, a mayor número de características, mayor facilidad y rapidez tendrá el usuario a la hora de buscar y encontrar un producto, y como consecuencia el tiempo de simulación será menor.

Una vez definido ésto, en las simulaciones realizadas se ha partido de n ϵ {1,3,5}. Se entiende pues, que en las pruebas realizadas con queries-1 el usuario tendrá menor conocimiento del producto final que está buscando y con queries-5 encontrará antes el producto que busca.

Los 3 tipos de archivos siguen un mismo patrón, el cual se define al comenzar el archivo. Vamos a ver detenidamente como están formadas las queries para la base de datos de Smartphone:

Manufacturer&Model&Weight&Size&Storage&Ram&Resolution&SO&CPU&Price&
Nominal&Nominal&Numeric&Numeric&Numeric&Numeric&Numeric&Nominal&
Nominal&Numeric

Como se puede observar, entre los símbolos & aparecen diferentes características en relación a la base de datos utilizada. Veamos un tipo de query para cada uno de los archivos que se utilizan:

- **queries-1.txt**:

CASE1674|CASE245|*&GalaxyAceAdvanceS6800&*&*&*&*&*&*&*&*&

- **queries-3.txt**

CASE730|CASE262|*&GalaxyPocketS5300&97.0&2.8&*&*&*&*&*&*&*&

- **queries-5.txt**

CASE191|CASE216|*&GalaxyMusicS6010&*&3.0&*&512.0&3.15&*&850 MHz Cortex-A9&*&

Cada una de las sentencias comienza con el caso inicial, seguido del caso final, y de las características del producto (aquellas que aparecen representadas como * son aquellas que el usuario no conoce al inicio de la simulación).

3.4. Críticas

En esta sección se explican los diferentes tipos de críticas que se van a utilizar en los algoritmos de recomendación que se han analizado e implementado.

Como se ha explicado en los primeros capítulos, los usuarios buscan productos a través de procesos de recomendación. Durante este proceso se van realizando críticas, recordamos que éstas son preferencias direccionales sobre una característica de dicho producto. Dentro del programa, una crítica sigue el patrón:

$$(f_i,\, v_i,\, type_i)$$

Siendo f_i la característica perteneciente al producto que la crítica aplica, v_i es el valor de f_i, y $type_i$ es el tipo de crítica que se aplica ($<$,$>$,$=$,$<>$).

La forma estándar de crítica es una *Unit Critique* en la que el usuario realiza una crítica de una sola característica, y las *Compound Critique* son aquellas que describen un conjunto de Unit Critiques, es decir, permiten al usuario proporcionar un feedback de más de 1 característica con una sola crítica.

Anteriormente se han comentado los diferentes tipos de bean que se definen en el archivo xml encargado de hacer la carga, ahora bien, ¿cómo se indica al simulador si los usuarios deben realizar Unit Critiques o Compound Critiques? Pues bien, nos lo indica el tipo de usuario:

El usuario estándar (stdUser) está definido para realizar un proceso de recomendación mediante la utilización de Unit Critiques, las cuales podrán basarse

en cualquiera de las características que hagan referencia al elemento correspondiente a la base de datos seleccionada, por ejemplo en el caso de SmartPhone, el formato de una crítica unitaria podría ser:

Price 40.0 '>' |

El usuario compoundUser está definido para la utilización de Compound Critiques, las cuáles son un conjunto de críticas unitarias. Cuando se llegue al apartado de la generación de queries se procederá a explicar como se generan éstas. Se muestra a continuación una posible Compound Critique perteneciente a la base de datos de SmartPhone:

{Manufacturer Nokia '<>' |, Weight 209.0 '<' |, Ram 2048 '<' |, }

4. Implementación

En este capítulo se van a explicar los algoritmos implementados. Se ha implementado la versión que usa *Compound Critique* de EBC, HAC, HGR, HOR. Estos nuevos algoritmos se denominan: EBC_CC, HAC_CC, HGR_CC, HOR_CC. Para una mejora comprensión, es necesario entender previamente que proceso siguen los algoritmos y que clases utilizan. Como se ha comentado en el capítulo anterior, la ejecución se llevará a cabo mediante un simulador, por lo que todos los algoritmos deben estar adaptados para poder ser reproducidos por éste.

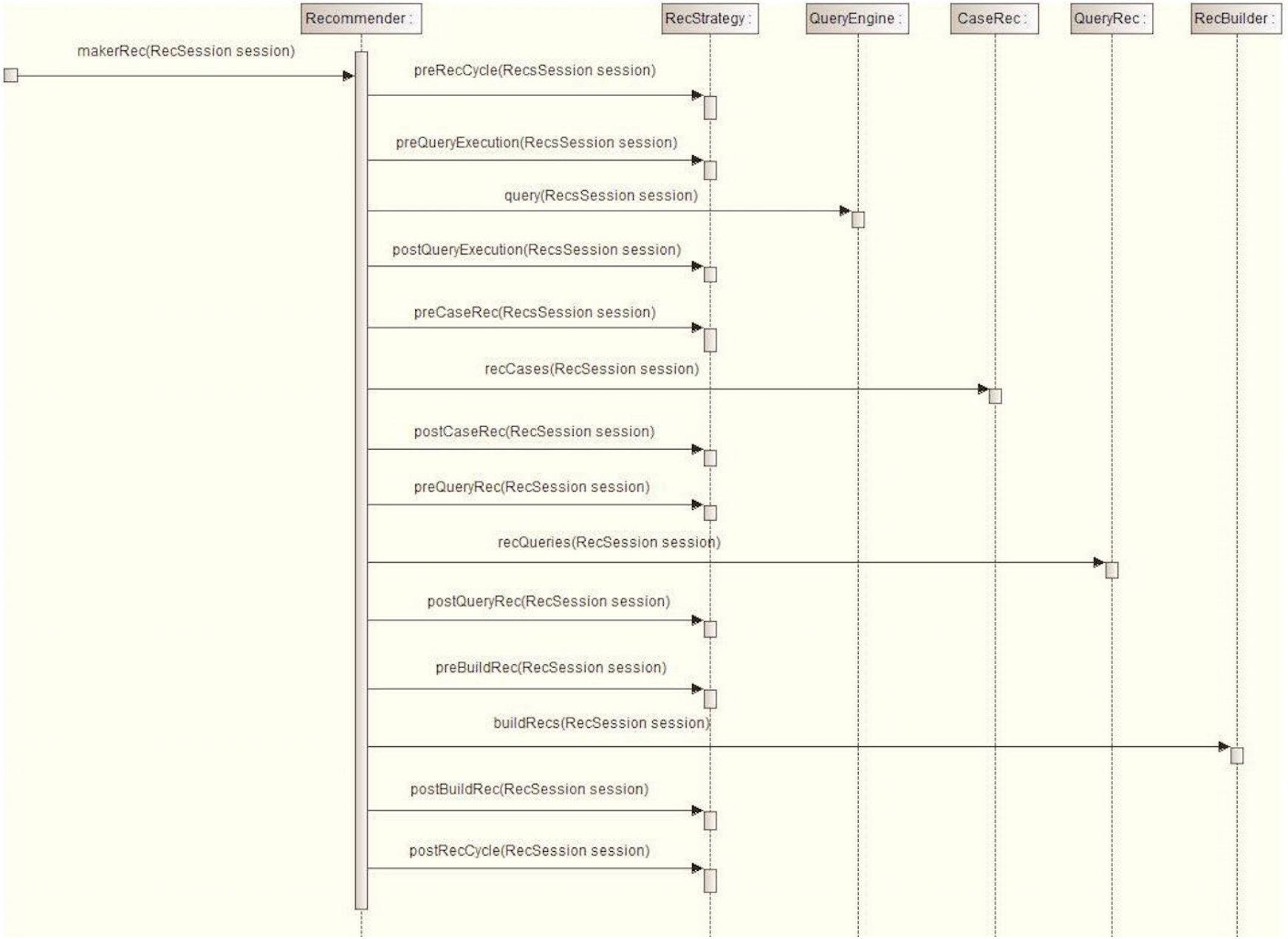
Por ello, cada algoritmo ejecutado seguirá una misma metodología de ejecución que le presenta el recomendador dentro del método principal de éste: *public void makeRec(RecSession session)*. De éste modo, dependiendo del algoritmo que se haya definido para la ejecución dentro del *bean* de configuración de ejecución Setup, se llamará a las clases de un paquete u otro.

Finalmente se implementan 3 técnicas diferentes utilizando *Clustering* sobre el algoritmo Incremental Critiquing con el objetivo de intentar optimizar más el proceso de recomendación.

Todos los algoritmos siguen el patrón de la siguiente Figura, ya que todos siguen la misma estructura que presenta el algoritmo estándar. Antes de comenzar, definiremos unos elementos necesarios:

- **RecSesion**: es la sesión propiamente dicha, en la que el usuario está envuelto en el momento actual. Contiene los elementos RecContext y UserContext.
- **RelevantCases**: son aquellos casos que satisfacen la query actual.
- **RecContext**: contiene información sobre los RelevantCases, los casos que faltan por comprobar, las queries recomendadas para un caso, las recomendaciones que hace el usuario, etc.
- **UserContext**: contiene información relacionada con el usuario actual, la query actual, el número de ciclos que ha ejecutado, si ha encontrado producto, si la sesión actual ha terminado y cual es el ítem seleccionado.

Se procede pues, a la explicación de los diferentes métodos que se utilizan en la simulación y como actúan para los diferentes algoritmos. El siguiente diagrama de secuencia muestra el proceso que realizan todos los algoritmos dentro del recomendador.


Recommender :
RecStrategy :
QueryEngine :
CaseRec :
QueryRec :
RecBuilder :
makerRec(RecSession session)
preRecCycle(RecsSession session)
preQueryExecution(RecsSession session)
query(RecsSession session)
postQueryExecution(RecsSession session)
preCaseRec(RecsSession session)
recCases(RecSession session)
postCaseRec(RecSession session)
preQueryRec(RecSession session)
recQueries(RecSession session)
postQueryRec(RecSession session)
preBuildRec(RecSession session)
buildRecs(RecSession session)
postBuildRec(RecSession session)
postRecCycle(RecSession session)

Una vez se ejecuta el método *makeRec(RecSession session)*, la ejecución recorrerá durante varios ciclos el proceso mostrado en la figura anterior, aunque nos basaremos principalmente en las siguientes órdenes:

```
queryEngine.query(session);
recStrategy.preCaseRec(session);
caseRec.recCases(session);
queryRec.recQueries(session);
```

4.1. QueryEngine

Este método es idéntico para todos los algoritmos y es ejecutado por todos ellos al comenzar la ejecución. Es el encargado de obtener los *RelevantCases* con los que se va a realizar la ejecución. Es decir, obtendrá los casos que son satisfactorios con la query actual, por tanto actuará como filtro para obtener sólo aquellos casos que cumplen las críticas del usuario y los guardará en el RecContext.

4.2. PreCaseRec

Este método permite la gestión del llamado Modelo de Usuario. Recordamos que un *Modelo de usuario*, permite almacenar un conjunto de críticas realizadas por el usuario durante la sesión *($U = u_1, u_2, ...u_n$)* donde u_i es una crítica única. En el caso de la estrategia IC, actualiza el modelo de usuario dando prioridad a la nueva crítica y eliminando las inconsistencias anteriores que ésta genere.

Cabe destacar que todos los algoritmos que siguen el enfoque *incremental critiquing* que vamos a ver utilizan modelo de usuario y por tanto, tendrán sus correspondientes métodos para gestionar el modelo de usuario propio dentro del *PreCaseRec*.

4.3. RecCases

RecCases es el método principal de la clase *caseRec* y permite realizar la recomendación del producto que el usuario está buscando. Es la base del proceso de recomendación, puesto que dependerá completamente del algoritmo en ejecución, y por tanto funcionará de manera diferente para cada uno de ellos. Por ello es necesario explicar cada uno de los algoritmos implementados, y ver que recorrido siguen por las diferentes clases de los paquetes que se han explicado en el capítulo anterior.

Algoritmo Estándar

El algoritmo estándar es el que sigue la metodología más sencilla. Al igual que todos los algoritmos que se van a explicar, primeramente se obtendrán los datos en la clase queryEngine como son los casos relevantes para la actual query. Posteriormente se creará la clase *OrderedSimilarityList* que permitirá calcular la similitud que existe entre la query actual y los casos relevantes comparando las características de cada caso con las de la query actual y finalmente escogerá el producto más similar.

Se muestran en la Figura 9, las diferentes clases que se utilizan para la reproducción del algoritmo estándar.

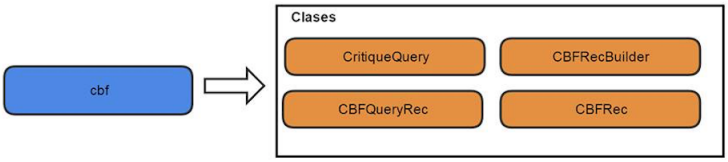


Figura 9: Paquete CBF

Algoritmo Incremental Critiquing

El algoritmo *Incremental Critiquing* se comporta de una manera similar al algoritmo Standard, pero incluyendo un modelo de usuario. Por tanto, una vez obtenidos los datos en queryEngine al igual que en el algoritmos standard, se cargará el modelo de usuario actual y se creará la clase *OrderedSimilarityList*, pero a diferencia del algoritmo anterior, Incremental Critiquing crea el OrderedSimilarityList pasándole un parámetro, en este caso *IncrementalModelSimilarityComputer* que permitirá calcular la compatibilidad, es decir el porcentaje de críticas en el modelo de usuario que satisface el nuevo caso, suponiendo un porcentaje del 75 % esta compatibilidad y del 25 % la similitud, quedando así la *Quality*, que es la que ordenará los diferentes casos con el objetivo de priorizarlos para la siguiente recomendación.

Se muestran a continuación en la Figura 10, las diferentes clases que se utilizan para la reproducción del algoritmo incremental.

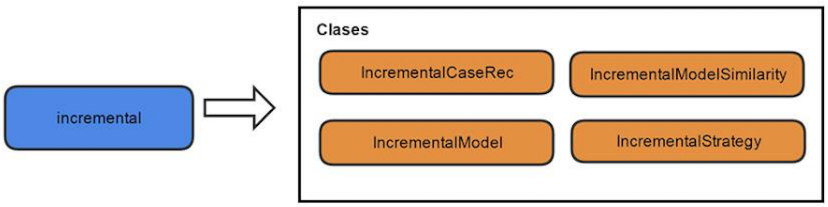


Figura 10: Paquete Incremental

Comenzamos ahora con los algoritmos *Experience-based*. Todos ellos tienen algo en común, y es que todos se basan en sesiones pasadas, por lo que para cada uno de estos algoritmos será necesario cargar estas sesiones al inicio del método recCases, las cuales estarán en un archivo ya generado anteriormente.

Experience-Based Critiquing (EBC)

Podría decirse que el algoritmo EBC es una ampliación del algoritmo STD, pero con varias diferencias. Tras cargar el histórico de sesiones, comenzamos a analizar estas sesiones para ver si han terminado exitosamente, ya que únicamente nos interesan aquellas sesiones que han derivado en compra.

Posteriormente, durante el proceso de recomendación, es necesario puntuar estas sesiones con el objetivo de encontrar las sesiones más relevantes. Para ello en la función *setScores* se realiza una suma de las críticas que se localizan en la sesión actual y la sesión que se quiere puntuar, luego se realizará un filtro y aquellas que menos puntuación tengan serán eliminadas de la lista de sesiones relevantes.

Cabe destacar que la puntuación de la sesión será diferente en el caso de Compound. Este método podría haberse implementado de diversas maneras pero se ha optado por dividir la Compound Critique en varias Unit Critique, de esta forma se comprueba cada una de las críticas con las de la sesión actual con la que se desea puntuar. Si se encuentra la crítica unitaria se suma uno a la valoración de la sesión, por tanto el caso más elevado de ponderación podría darse con una Compound Critique que tenga el numero máximo de críticas que se generen (3) y que cada una de ellas se encuentre en la sesión.

Otro método alternativo de ponderación que se podría haber seguido era puntuar únicamente aquellas sesiones en las que se encontraran todas las Unit Critiques pertenecientes a una Compound, pero se descartó este método ya que si no se encuentran sesiones relevantes EBC pasa a ejecutar el algoritmo STD, que como se ha explicado se basa únicamente en la similitud entre los ítems, por lo que no encontraríamos mejora alguna.

Finalmente, una vez puntuadas todas las sesiones se realiza una ordenación según la puntación de dichas sesiones, se seleccionan los productos terminales (son aquellos productos finales que el usuario ha seleccionado en esa sesión) y se eliminan todos aquellos que no estén dentro de la lista de *relevantCases* calculados previamente. Tras la ordenación, se seleccionará el producto que esté en la cabeza de la lista.

En la Figura 11, se muestran las diferentes clases que se utilizan para la reproducción del algoritmo Experience-based con Compound Critique.

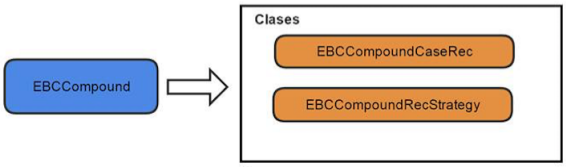


Figura 11: Paquete EBC

History-Aware Critiquing-Based (HAC)

El algoritmo HAC parte de la implementación del EBC, por tanto es necesario realizar la carga de las sesiones al inicio de la ejecución. Como se ha comentado en los primeros capítulos, EBC únicamente comprueba la similitud de las críticas de la sesión actual con la que se está puntuando, por lo que HAC incluye además la comprobación de la similitud del ítem que se va a recomendar.

Primeramente al igual que en EBC se puntúan las sesiones en el método *setQualitySessionsHAC* con el objetivo de localizar las sesiones más relevantes (se aplica el mismo método para puntuar las sesiones que se ha definido en el apartado anterior). Cabe destacar que cuando un usuario realiza una crítica sobre una recomendación, se generan un conjunto de pares de recomendación (crítica, producto recomendado), por lo que además de realizar la ponderación de la sesión se comprueba la similitud del ítem recomendado de las sesiones.

Tras ésto, al igual que en el algoritmo EBC, se procede a ordenar las sesiones según la puntuación final obtenida del resultado del cálculo anterior: *Quality*. Con las sesiones ordenadas, se procede a eliminar todas aquellas sesiones que no aparecen en la lista de los *relevantCases* calculados previamente y se selecciona el producto que esté en la cabeza de la lista.

En el caso en el que no se encuentren suficientes sesiones relevantes, el algoritmo HAC deriva en un algoritmo IC.

Se muestran en la Figura 12 las diferentes clases que se utilizan para la reproducción del algoritmo HAC con Compound Critique.

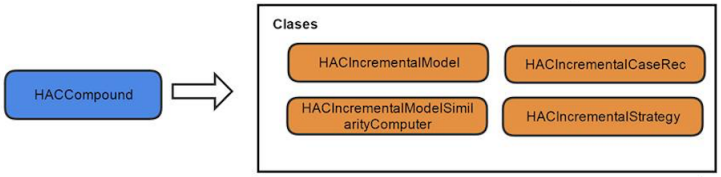


Figura 12: Paquete HAC

History-Guided Conversational Recommendation (HGR)

El algoritmo HGR amplía los requisitos a la hora de buscar sesiones relevantes en cuanto HAC. La principal diferencia con éste radica en que HGR además de calcular la similitud de los pares de recomendación existentes, una sesión relevante será aquella que tenga mas coincidencias entre las parejas de (crítica, ítem) entre la sesión actual y la sesión con la que se compara. Por lo que en el método *setScoreSessionsHGR* se calculará el número de pares de recomendación que coinciden y se asignará esta puntación a la sesión.

Cabe destacar que el algoritmo HGR, en caso de no encontrar sesiones relevantes tras realizar la ponderación anterior, seguirá con el método de puntuación de sesiones utilizado en EBC mediante el *setScoreSessionsEBC*.

Una vez se consigue la lista de sesiones relevantes, se procede a la ordenación de la misma manera que en el algoritmo HAC, por lo que la sesión más importante será la que quede en la primera posición de esa lista.

En la figura 13 se muestran las diferentes clases que se utilizan para la reproducción del algoritmo HGR con Compound Critique.

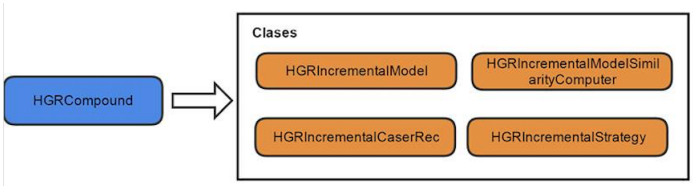


Figura 13: Paquete HGR

History Opinion Recommender (HOR)

El algoritmo HOR parte del algoritmo HGR con alguna particularidad como la utilización de las opiniones realizadas por los usuarios en sesiones anteriores de recomendación.

Primeramente al igual que HGR se comprueba si existen sesiones relevantes con suficientes pares de recomendación (crítica-ítem) para poder ser declarada como sesión relevante, para ello se ponderan las sesiones en el método *setScoreSessionsHGR*, ya que en caso contrario se pasará a seguir el procedimiento descrito en el algoritmo EBC.

Posteriormente, si existen sesiones relevantes, se analiza la lista de sesiones y se pondera cada caso siguiendo la Ecuación 18, en la que se tiene en cuenta además del score de la sesión, las opiniones que han indicado los usuarios acerca del producto, calculando el elemento o_j mediante la función *relSession.getScoreRank()*.

Tras ésto, se ordenan los productos en una lista quedando en primer lugar el elemento de mayor puntuación, y se procede a eliminar aquellos que no sean compatibles con los casos previamente calculados en el modelo de usuario.

En la Figura 14 se muestran a continuación las diferentes clases que se utilizan para la reproducción del algoritmo HOR con Compound Critique.

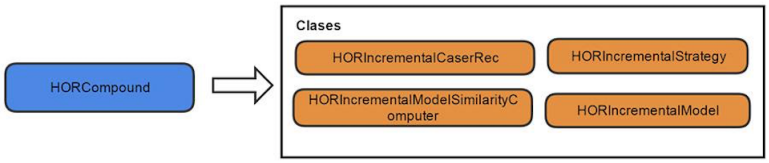


Figura 14: Paquete HOR

4.4. RecQueries

El método *recQueries* pertenece a la clase queryRec y es el encargado de generar las queries con el objetivo de representar el feeedback que genera el usuario tras la sesión. La generación de queries varía dependiendo del tipo de usuario que se haya definido en el bean Setup de configuración de la sesión, ya que si se ha indicado *simUser: stdUser* se generarán críticas unitarias y si es *CompoundUser* las generará compuestas.

En el caso de generación de críticas unitarias, se generará como mínimo una query por cada característica del producto que se está tratando, es decir, si nos basamos en el dataset Smartphone se generará una query para la característica modelo, una para manufacturación, etc. En ocasiones se da el caso de generación de 2 queries por característica ya que si recibe una crítica de tipo Ordinal (precio, capacidad) se generan ambas posibilidades: *Greater than Ordinal*, *Less than Ordinal*.

El caso de generación de críticas compuestas, se puede dividir en 2 pasos: primeramente se realiza una asociación de los atributos frecuentes mediante el algoritmo *Apriori*[1], por lo que estos atributos serán candidatos a crítica, y posteriormente a cada candidato se le asigna una utilidad dependiendo de la diversidad respecto a las críticas hasta el momento.

4.5. Clustering

Las técnicas de *Clustering*[17] consisten en una división de datos en grupos de objetivos similares. Cada grupo, llamado Clúster está formado por objetos similares entre ellos y diferentes de otros grupos. La representación de datos por medio de un número bajo de clusters supone una gran simplificación, aunque a veces puede suponer perdidas de ciertos detalles.

Dentro de los algoritmos de Clustering que existen, se van a utilizar métodos pertenecientes al enfoque *k-means*. El método *k-means*[17] es un método de agrupamiento que tiene como objetivo la partición de un conjunto de n observaciones en k grupos en el que cada observación pertenece al grupo más cercano a la media. Para mejorar la clasificación de los ítems se ha utilizado la técnica de análisis de componentes principales (PCA)[18] que permite enfocarse en las dimensiones más importantes y representativas de los datos.

Para la utilización de *Clustering* dentro de este proyecto se ha modificado la definición de la base de casos para incorporar una clase (*Cluster*) al que pertenece cada caso. Para ejemplificar utilizamos la base de datos de SmartPhone en la cual se siguen los siguientes pasos:

1. Se procede a leer el caso base actual y a construir una matriz con las columnas no nominales de éste, es decir, aquellas columnas con datos numéricos. (véase Figura 15)

```
if (caseBase.equals("Smartphone-Casebase")){
    Clusters = 5;
    r.eval("data<-read.csv(\"dataset/smartphone/newCB.txt\",header=TRUE,sep=\"&\")");
    r.eval("data<-data[-1,]");
    r.eval("Matrix<-subset(data,select=c(3,4,5,6,7,10))");
    r.eval("Matrix<-transform(Matrix, Weight = as.numeric(Weight),"
            + " Size = as.numeric(Size),Storage = as.numeric(Storage),"
            + "Ram = as.numeric(Ram),Resolution = as.numeric(Resolution),"
            + "Price=as.numeric(Price))");
```

Figura 15: Parte del algoritmo para crear la matriz con las columnas del caso base

2. Se define un modelo PCA, PX que consideran las componentes principales de los casos.

```
r.eval("mod<-prcomp(Matrix, center=TRUE,scale=TRUE)");
r.eval("pX <- predict(mod)[,1:2]");
```

3. Se procede a realizar Clustering usando el método *k-means* y el método PX, siendo k=5 mediante la sentencia:

```
r.eval("k_clus<-kmeans(pX,5,nstart=100)");
```

4. Finalmente se crea una Hashmap que nos permita asignar a cada caso un clúster determinado. (véase Figura 16)

Tras la realización de los pasos anteriores, cada caso de la base de casos tendrá un único clúster.

```
REXP rexp3=r.eval("resultClust<-simplify2array(k_clus$cluster)");

int[] matrixClass=rexp3.asIntArray();


casesClass=new HashMap();
int caseNum=firstCase;
for(int j=0;j<matrixClass.length;j++){
      casesClass.put("CASE"+caseNum, matrixClass[j]);
      caseNum++;

 }
```

Figura 16: Parte del algoritmo para asignar un clúster a cada caso

4.5.1. Recomendar al usuario la clase mayoritaria

En este apartado se explica la primera técnica propuesta. El paquete que contiene los elementos necesarios para la ejecución de esta técnica es *rec.incremental.clustering*. Esta nueva técnica se centra únicamente en los casos que tengan el clúster mayoritario, es decir, el clúster que más casos tiene de entre los 1721 casos en la base de Smartphone. Para ello mediante el método *getMajorityClass(Collection casesClust)* de la clase *IncrementalCaseRec* se obtienen todos los casos que pertenecen al clúster mayoritario y se procede a realizar el método *recCases* como hasta ahora.

4.5.2. Potencia la clase mayoritaria

En esta técnica se propone potenciar la clase mayoritaria, es decir, asignar un cierto valor a un caso dependiendo del clúster al que pertenece. El paquete que contiene los elementos necesarios para la ejecución de esta técnica es *rec.incremental.clustering2Mayor*. Para ello, mediante el método *calculateNumberCasesofClaster(Collection cases)* se calcula el número de casos que hay de cada clúster y posteriormente el método *calculatePercentagesClases (HashMap map, int size)* calcula el *customValue* que se va a asignar a cada clúster, el cual se calcula de la siguiente forma:

$$\text{customValue} = \text{número de casos} \ / \ \text{total}$$

Tras el cálculo de este valor para cada clúster, cuando se procede a analizar la *Quality* del algoritmo *Incremental Critiquing* (véase Ecuación 2) se tiene en cuenta el *customValue* calculado, por lo que la nueva Ecuación de ponderación de la *Quality* pasa a ser:

$$IQual(c', c, U) = \alpha * Compat(c', U) + (\beta) * Sim(c', c) + \gamma * customValue \quad (28)$$

Siendo $\alpha + \beta + \gamma = 1$.

4.5.3. Potencia la clase minoritaria

Finalmente la última técnica propuesta se basa en la técnica del apartado anterior pero potenciando la clase minoritaria. Para ello, el método *calculatePercentagesClases (HashMap map, int size)* calcula el *customValue* que se va asignar para cada clúster siendo éste mayor para los casos de la clase minoritaria, (véase Ecuación 28).

5. Evaluación y análisis de resultados

En este capítulo se va a proceder analizar los resultados obtenidos de las ejecuciones de los diferentes algoritmos. Mediante este estudio podremos confirmar si los resultados obtenidos con los algoritmos implementados mejoran a los algoritmos STD e IC. Por tanto, podemos afirmar que esta es una parte muy importante dentro del proyecto realizado.

5.1. Metodología de los experimentos

En los experimentos realizados se ha utilizado la base de datos SMARTPHONE, la cual se detalla en la Figura 17. Para realizar los experimentos se utiliza la metodología *Leave one out*[9] que consiste en seleccionar una muestra de ítems aleatorios de la base de datos y someterla a test.

Base de Casos	# Items	# Caracteristicas Totales	# Características Numericas	# Características Nominales
SMARTPHONE	1721	10	4	6

Figura 17: Estructura de la base de datos SmartPhone

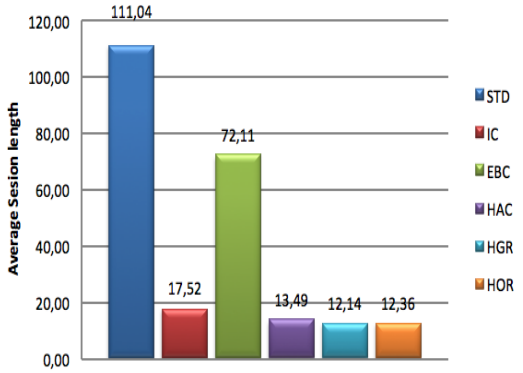
Para realizar la comparativa de los algoritmos y observar si éstos mejoran se utiliza como medida el *average session length (ASL)*, que permite computar el número de ciclos que el usuario ha tardado en finalizar la sesión y seleccionar un producto.

Los experimentos que van a realizarse en este proyecto son los siguientes: *Comparativa de los algoritmos con los diferentes tipos de queries*, permite visualizar el número de ciclos de los algoritmos que se han desarrollado en Compound Critique y compararlos con los existentes en Unit Critique dependiendo del nivel de conocimiento que el usuario tenga al inicio de la sesión. *Análisis del beneficio obtenido en cuanto a IC*, se comparan los algoritmos implementados para observar que beneficio obtiene cada uno de ellos en relación a Incremental Critiquing. *Progresión de los algoritmos*, se analizan los resultados del apartado 5.2 con otro tipo de gráficas para obtener conclusiones diferentes. *Progresión de los algoritmos según Case Base Size*, se analizan los algoritmos implementados para ver como afecta el tamaño del Case Base utilizado al *average session length.*

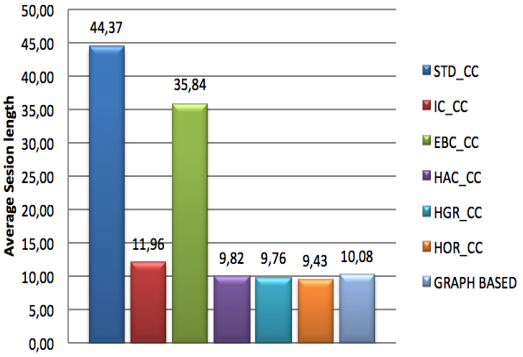
Cabe destacar que en cada una de las pruebas que se realizan, se realiza también la comparativa entre los algoritmos implementados en Compound Critiques con los ya existentes en Unit Critique con el fin de comprobar si se produce mejora en todo momento.

5.2. Comparativa de los algoritmos con los diferentes tipos de queries

Vamos a proceder a analizar como se comportan los diferentes algoritmos dependiendo de la información que tienen al iniciar las sesiones, es decir, que tipo de nivel de conocimiento tienen *(hard, moderate, easy)*. En esta sección se va a realizar la comparativa de los algoritmos que se están tratando con los diferentes tipos de queries.



(a) Unit Critiques q1

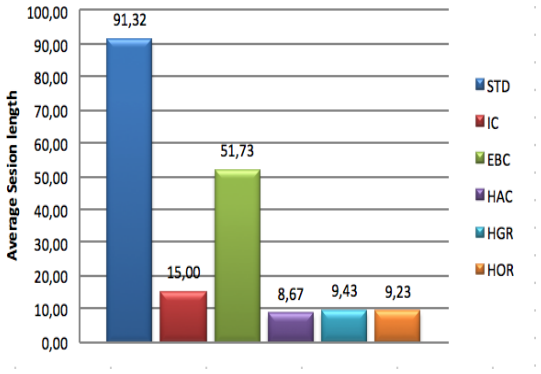


(b) Compound Critiques q1

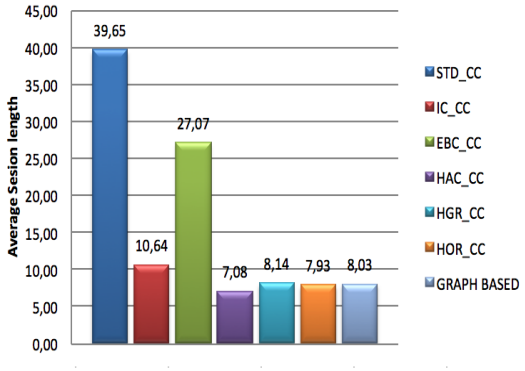
Figura 18: Comparativa algoritmo con q1(hard)

Lo primero que podemos afirmar de la Figura 18 es que mediante la ejecución con q1 *(hard queries)*, se produce una diferencia sustancial si se usa Compound Critique a Unit Critique, ya que como se puede observar, por ejemplo, el algoritmo STD obtiene un ASL de 111,04 con UC y de 44,37 en CC. Cabe destacar la ganancia que se produce de los algoritmos más costosos como son: STD y EBC que consiguen reducir a la mitad el número de ciclos de ejecución al utilizar Compound Critiques.

En la Figura 19, en la ejecución con q3 (*moderate queries*), se puede observar que sigue la misma línea que la ejecución anterior reduciendo considerablemente el ASL. Dentro de la familia de los *experience-based* basados en critiquing, se



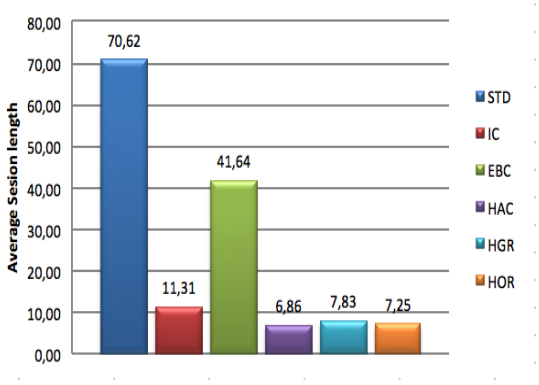
(a) Unit Critiques q3



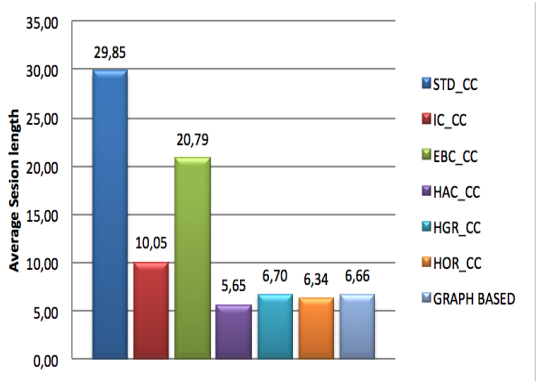
(b) Compound Critiques q3

Figura 19: Comparativa algoritmo con q3 (moderate)

puede observar ciertos cambios que se han producido. En esta ejecución, tanto en (a) como en (b) el algoritmo HOR ha dejado de ser el más optimo con un ASL de 9,23 y 7,93 respectivamente, ocupando HAC este lugar con un ASL de 8,67 y 7,08. Además el algoritmo HGR ha pasado a posicionarse como el peor de estos 3 con un ASL de 9,43 y 8,14 en *Unit Critique* y *Compound Critique* respectivamente.



(a) Unit Critiques q5



(b) Compound Critiques q5

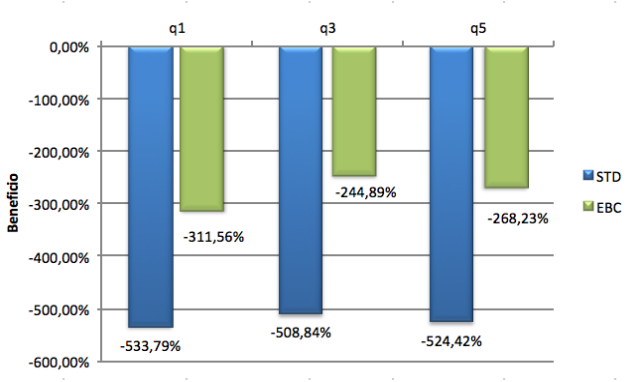
Figura 20: Comparativa algoritmo con q5 (easy)

Finalmente en la Figura 20, la ejecución de q5 *(easy queries)* sigue produciendo mejoría en la utilización de *Compound Critiques* respecto a *Unit Critiques*. Respecto a la puntualización comentada entre HAC y HOR, en este ejecución se puede observar como la mejoría entre HAC y HOR es mayor que en la ejecución de q3, teniendo en cuenta que el número de ciclos ha disminuido y por tanto el porcentaje de la distancia entre ambos es mayor.

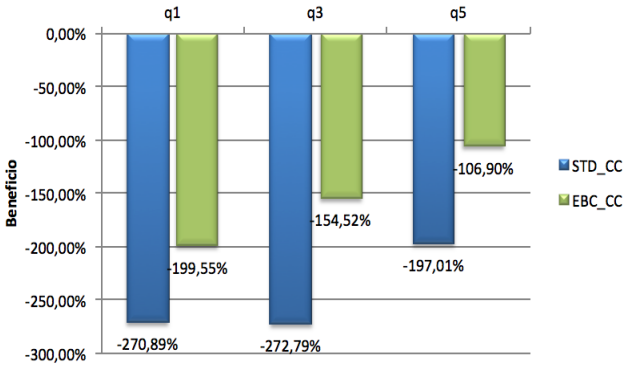
5.3. Análisis del beneficio obtenido en cuanto a IC

En esta sección analizaremos más detalladamente cuanta mejoría se ha obtenido con los diferentes algoritmos en comparación con *Incremental Critiquing*. Como aclaración, se realiza la comparativa en cuanto al algoritmo IC ya que es el primer algoritmo en el que se comienzan a tener en cuenta otros factores y no solamente la comparación de críticas básicas del estándar.

Primeramente se va a analizar la comparativa en cuanto a beneficios con los algoritmos que no pertenecen a la metodología incremental, y posteriormente los que sí.



(a) Unit Critiques

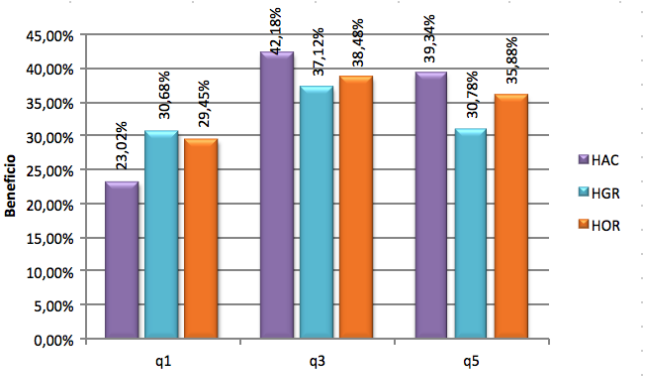


(b) Compound Critiques

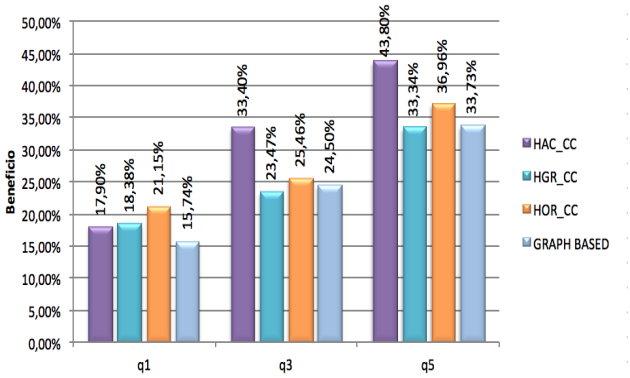
Figura 21: Beneficio obtenido del algoritmo IC vs STD y EBC

La primera observación que llama la atención en la Figura 21 es que los niveles de beneficio son negativos aunque los resultados son correctos, ya que como hemos podido observar en las figuras del apartado anterior el número de ciclos correspondiente al algoritmo IC es mucho menor que los algoritmos STD y EBC. Además la diferencia en cuanto al beneficio entre EBC y STD se reduce cuando se utilizan Compound Critiques ya que en en al apartado (a) la mejora que produce IC sobre el algoritmo STD es aproximadamente de media entre las diferentes queries un 500 % y de 250 % en EBC, siendo solo en el apartado (b)

del 250 % en STD_CC y del 160 % en EBC_CC aproximadamente.



(a) Unit Critiques



(b) Compound Critiques

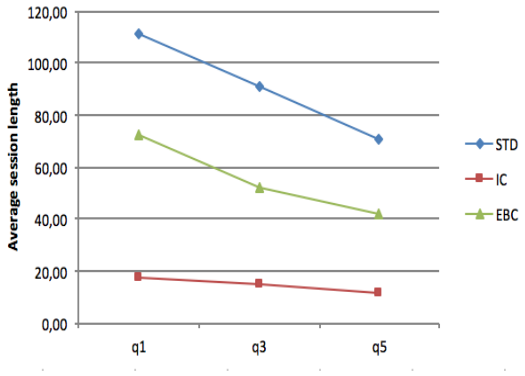
Figura 22: Beneficio obtenido del algoritmo IC vs HAC, HGR y HOR

Como se puede observar en la Figura 22, en cuanto a q1 el aumento del beneficio disminuye en la utilización de *Compound Critiques*, si nos fijamos en la gráfica (b), el único algoritmo que supera un beneficio del 20 % es HOR (al igual que se ha comentado en el apartado anterior, cuando se utiliza q1 HOR es el mejor). En cuanto a q3, HAC es el algoritmo más destacado, ya que a medida que el usuario tiene más conocimiento se posiciona como el mejor. Finalmente en q5 se produce un cambio, y es que HAC supera el 40 % de beneficio en comparación a IC superando de esta forma el beneficio que se produce con *Unit Critiques*, además tanto HGR como HOR obtienen un beneficio mayor respecto a IC con la utilización de CC.

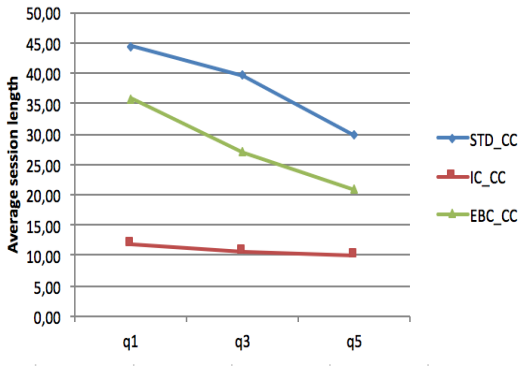
En cuanto a *Graph Based* se puede observar que comienza siendo el algoritmo menos óptimo cuando el usuario tiene un conocimiento menor, pero a medida que éste aumenta en q3 y q5, se posiciona por delante de HGR.

5.4. Progresión de los algoritmos

En esta sección se van a analizar los resultados mediante gráficas de evolución y con ello obtener unas conclusiones finales. Se mostrarán primero los algoritmos STD, IC, EBC y posteriormente HAC, HGR, HOR.



(a) Unit Critiques

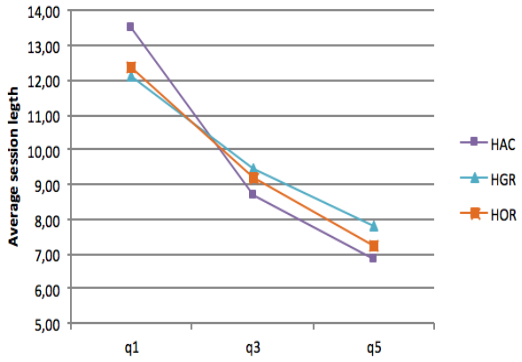


(b) Compound Critiques

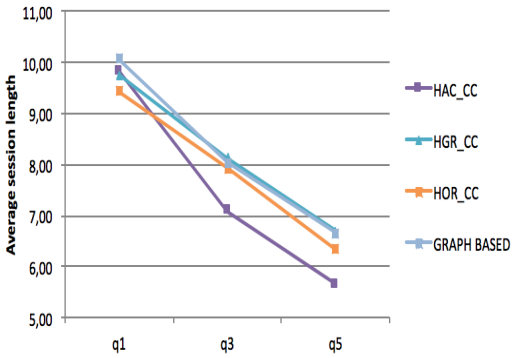
Figura 23: Progresión algoritmo STD, IC, EBC

Mediante las gráficas de las figuras 21 y 22 se puede afirmar que utilizando *Compound Critiques* en los algoritmos se produce una mejoría muy sustancial respecto a utilizar *Unit Critiques*. Como se puede observar en la Figura 23, cuando el usuario tiene un conocimiento inicial más elevado (q5) de las características que desea en el producto final, el número de ciclos disminuye exponencialmente en los algoritmos STD y EBC, ya que el algoritmo incremental se mantiene más lineal.

En cuanto a los algoritmos pertenecientes a *experience-based*, en el enfoque incremental queda claro que el algoritmo más eficiente cuando el usuario tiene menos definidas sus preferencias es el HOR y posteriormente el HGR. A medida que el usuario tiene más definidas sus preferencias el mejor algoritmo pasa a ser HAC, y los demás quedan en un rango muy similar.



(a) Unit Critiques



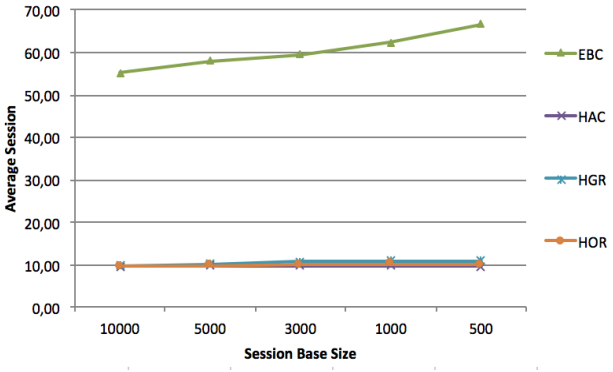
(b) Compound Critiques

Figura 24: Progresión algoritmo IC, HAC, HGR, HOR, GRAPH BASED

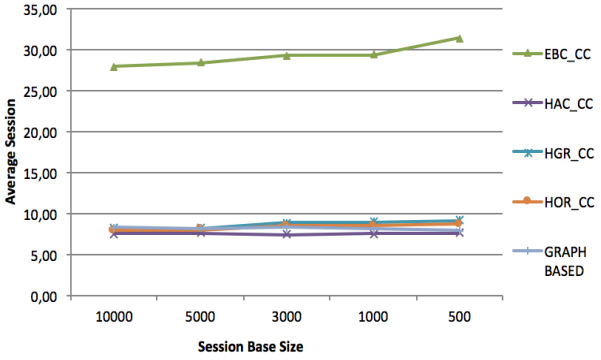
El algoritmo *Graph-based* está por debajo en rendimiento de los algoritmos pertenecientes a *experience-based* (excluyendo EBC), aunque con muy poca diferencia.

5.5. Progresión de los algoritmos según el tamaño de la base de casos

En esta sección se procede a analizar los algoritmos pertenecientes a la metodología Experience-based que son aquellos algoritmos que se basan en la experiencia de usuario, es decir, basan sus críticas en resultados de sesiones pasadas. Como se ha nombrado en capítulos anteriores, este conjunto de sesiones es denominado *histórico (history)* y se construye mediante las sesiones de usuarios anteriores. Se procede a analizar como afecta la medida del *history* en las sesiones actuales, es decir, se procede a analizar como varían los ciclos en la sesión actual dependiendo del tamaño de sesiones históricas que se utiliza.



(a) Unit Critiques



(b) Compound Critiques

Figura 25: Progresión algoritmos dependiendo del histórico utilizado

Como se puede comprobar en la Figura 25, no se visualizan los algoritmos Estándar ni Incremental Critiquing ya que no utilizan ningún History para su ejecución. En cuanto a una visión general de los algoritmos, tanto en la Figura (a) como (b) se produce un aumento de los ciclos de ejecución cuando se utilizan históricos menores, ya que el algoritmo de recomendación dispone de menos información sobre sesiones pasadas y por lo tanto la sesión actual se prolonga.

A nivel específico, en el algoritmo EBC se produce un aumento importante

dependiendo de la utilización de un *History* mayor o menor. Respecto a los pertenecientes al enfoque incremental (HAC, HGR, HOR), se puede observar como se produce también un incremento, aunque no tan significativo como en el algoritmo EBC.

Para concluir el análisis de los resultados, se confirma que la utilización de *Compound Critiques* produce una mejora importante en las sesiones, además el conocimiento previo de los usuarios al inicio de la sesión reduce considerablemente la longitud de ésta. Además se ha comprobado en la última sesión como dependiendo del tamaño del history la sesión actual también varía.

5.6. Resultados Clustering

Se procede a analizar en esta sección los resultados obtenidos en las diferentes técnicas implementadas con el objetivo de encontrar una posible mejora en el *average session length* de éstas nuevas versiones del algoritmo *Incremental Critiquing* en comparación el algoritmo IC basado en *Unit Critiques* que se ha visto durante el proyecto.

Recomendación de la clase mayoritaria

En este apartado se analizan los resultados de la técnica de *Clustering* vista en el apartado 4.5.1. Como se puede ver en la Figura 26 la utilización de los casos del clúster mayoritario no produce ninguna mejora en los resultados del algoritmo *Incremental Critiquing* en ninguna opción en cuanto a nivel de conocimiento inicial se refiere, sino que se produce un empeoramiento muy notable, llegando a ser peor que el algoritmo *Standard Critiquing*.

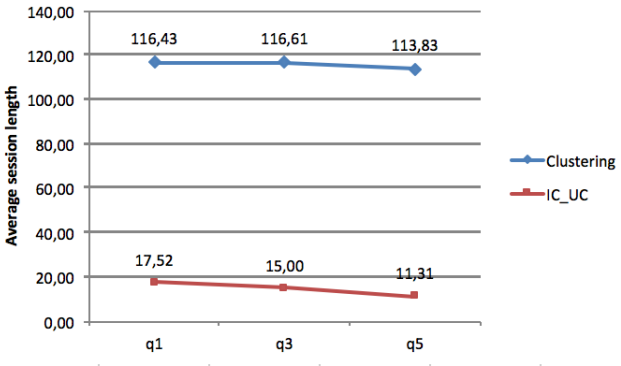


Figura 26: Resultados de la técnica de recomendación de la clase mayoritaria

Potenciación de la clase mayoritaria

En este apartado se analizan los resultados obtenidos de la técnica de *Clustering* vista en el apartado 4.5.2. En la Figura 27, se aprecia un mejora sustancial respecto al algoritmo Incremental Critiquing con los diferentes tipos de conocimiento previo del usuario (q1, q3, q5), siendo en q3 *(moderate)* la mayor ganancia de la técnica implementada con 9,28 de ASL y 15,00 del algoritmo *Incremental Critiquing*.

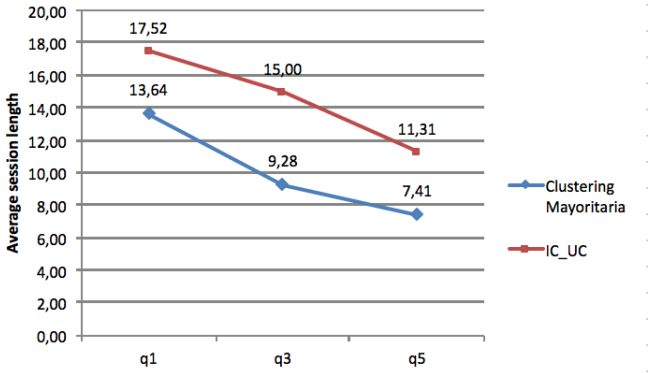


Figura 27: Resultados de la técnica de potenciación de la clase mayoritaria

Potenciación de la clase minoritaria

En este apartado se analizan los resultados obtenidos de la técnica de Clustering vista en el apartado 4.5.3. En la Figura 28, se aprecia un mejora sustancial al igual que en el apartado anterior de la nueva técnica implementada de *Potenciación de la clase minoritaria* respecto al algoritmo *Incremental Critiquing*.

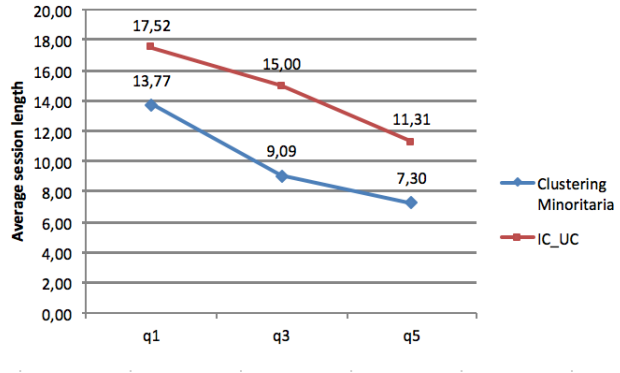


Figura 28: Resultados de la técnica de potenciación de la clase mayoritaria

Para finalizar esta sección podemos extraer varias conclusiones sobre las nuevas técnicas implementadas mediante *Clustering*. Como se puede observar en la

gráfica 26, la técnica de utilización de únicamente los casos pertenecientes al clúster mayoritario no ha mejorado los resultados respecto al algoritmo IC. En cuanto a las técnicas de potenciación de las clases mayoritarias como minoritarias se ha producido una mejora muy similar como se puede ver en las Figuras 27 y 28.

6. Análisis de costes

En este capítulo se procede a analizar el tiempo invertido en cada una de las partes del proyecto realizado, así como una estimación del coste económico. Primeramente se expondrán las partes realizadas y se mostrará mediante un gráfico el porcentaje que representa cada una de ellas sobre el tiempo total invertido. Para finalizar se expondrá una valoración económica más detallada.

6.1. Tiempo invertido

- **Estudio bibliográfico de los algoritmos existentes**: estudio del marco actual referente a los sistemas de recomendación, los diferentes tipos de sistemas existentes así como los algoritmos más referentes de éstos.
- **Reuniones**: sesiones periódicas con la tutora del proyecto para realizar un seguimiento de los objetivos marcados al inicio del proyecto y resolución de dudas que han surgido durante la realización de éste.
- **Análisis**: estudio de los mecanismos existentes referentes al proyecto, como son la plataforma y los diferentes elementos que se necesitan para la ejecución de una simulación.
- **Desarrollo**: implementación de los algoritmos de experience-based critiquing mediante la utilización de Compound Critiques.
- **Análisis de los resultados**: análisis obtenido de la ejecución de las diferentes pruebas con los algoritmos estudiados en el apartado anterior, tanto para Unit Critiques como Compound Critiques.
- **Clustering**: ampliación del proyecto inicial desarrollando técnicas de Clustering con el objetivo de establecer nuevos métodos para acotar la búsqueda de datos.
- **Documentación**: preparación del documento en el que se explica detalladamente todas las acciones realizadas para la realización del proyecto de final de grado.

Se procede a analizar el diagrama siguiente, en el que se puede ver el porcentaje empleado en la realización de las tareas que se han llevado a cabo.

Como se puede comprobar, las partes que más importancia han tenido durante la realización del proyecto han sido el Estudio Bibliográfico y la parte de desarrollo e implementación ya que cada una ha supuesto un 23 % del total (cabe destacar que la parte de Clustering se ha separado pero podría incluirse en este apartado de implementación).

Posteriormente tanto el análisis de los resultados como la creación de la documentación ha supuesto un 16 % del total cada parte, seguido del análisis de la plataforma (12 %), de la parte de Clustering (7 %) y finalmente de las reuniones periódicas que se han realizado.

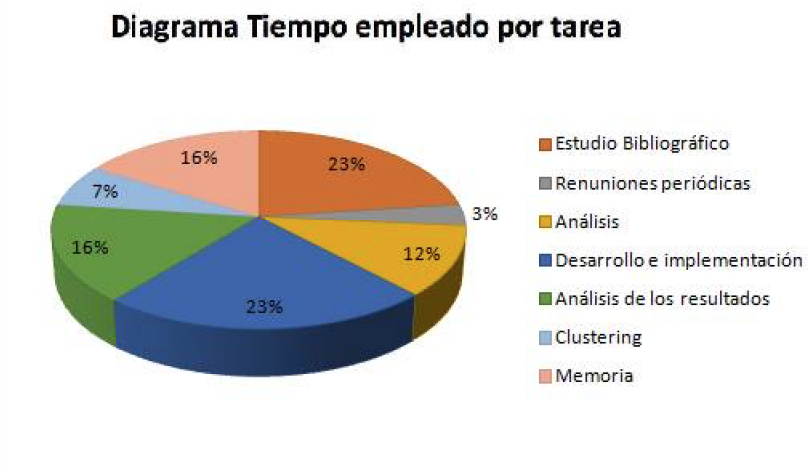


Figura 29: Diagrama de las tareas realizadas

6.2. Cálculo de coste global

En este apartado se va a explicar, en base a los resultados obtenidos en el apartado anterior, el coste que supone la realización del proyecto teniendo en cuenta los costes que implica la realización de cada una de las partes por las que está compuesto.

Tarea Realizada	Estimación de horas	Precio/hora(€/h)	Número de horas invertidas	Coste total
Estudio Bibliográfico	100h	10 €	100h	1.000 €
Renuniones periódicas	10h	10 €	15h	150 €
Análisis	50h	10 €	50h	500 €
Desarrollo e implementación	100h	20 €	100h	2.000 €
Análisis de los resultados	40h	10 €	70h	700 €
Clustering	20h	20 €	30h	600 €
Memoria	80h	15 €	70h	1.050 €
TOTAL	400h		435h	6.000 €

Figura 30: Estimación y coste de las tareas realizadas

De la Figura 30, se puede extraer ciertas conclusiones:

Primero, se puede observar que en la columna de *Estimación de horas* se producen algunas diferencias con las horas reales finales, en el caso de las Reuniones, se ha contabilizado el número de horas de tutoría que se han realizado, estableciendo 1h cada dos semanas, pero finalmente hubo alguna hora extra para solventar ciertos problemas. En cuanto al Análisis de resultados, por ciertos problemas de incompatibilidades con algunos paquetes, el número de horas superó la estimación. Por otra parte, se mejoró la estimación realizada para la creación de la documentación.

En cuanto al *Coste de las horas*, se puede observar que el precio/hora más elevado es el aplicado al ámbito del Desarrollo e implementación, teniendo en

cuenta que éste es el que más horas nos implica en el proyecto, es la tarea que más coste ha supuesto para el proyecto.

Finalmente si realizamos una estimación global: teniendo en cuenta que el coste total del proyecto es de *6000€*, y el número total de horas es de *435h* obtenemos un coste medio de *13,79€/h*.

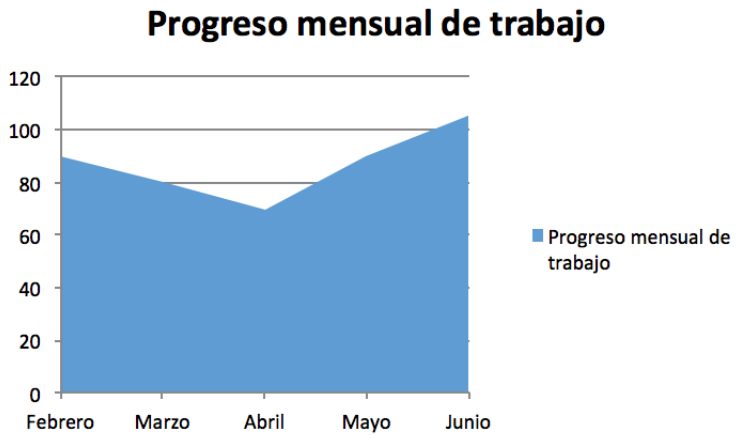


Figura 31: Estimación del progreso del desarrollo del proyecto

El desarrollo del proyecto ha seguido la línea de la estructura de la memoria. Como se puede observar en la Figura 31, gran parte de la realización del estado del arte fue completado en el mes de febrero, durante el mes de marzo se realizó el análisis de la plataforma y se comenzó con la implementación que concluyó a principios de mayo, (se puede observar como en abril se produce un pico en el que se aplicaron menos horas, debido a trabajo externo al proyecto pero referente a la universidad). Una vez obtenidos los resultados, a mediados de mayo, se optó por realizar una ampliación del proyecto aplicando Clustering para obtener nuevas conclusiones, y finalmente terminar con la documentación. (Cabe destacar que la memoria del proyecto se ha ido actualizando a medida que se lograban los objetivos establecidos).

7. Conclusiones y trabajo futuro

El proyecto desarrollado está orientado en el ámbito de los sistemas de recomendación. Partiendo de un estudio de los diferentes tipos de sistemas existentes como son los de tipo filtrado colaborativo y los de tipo basado en contenido, basándose principalmente el proyecto en este último. Dentro de los algoritmos basados en contenido, el trabajo se encaminará hacia el enfoque de los Recomendadores Conversacionales haciendo especial hincapié en Critiquing.

Una vez realizado el análisis de la plataforma, se procede al análisis de los algoritmos de estándar, incremental y los que pertenecen al enfoque Experience-based (EBC, HAC, HGR, HOR) que ya estaban implementados para aceptar Unit Critiques, con el objetivo de comprender como se habían desarrollado para finalmente proceder a la implementación de los algoritmos pertenecientes a Experience-based para poder adaptar Compound Critiques.

Cabe destacar, que no ha sido la tarea de implementación la más costosa del proyecto, aunque va íntegramente relacionado con la tarea del análisis de los algoritmos ya implementados, con el objetivo de comprender que diferencias suponía la adaptación a Compound Critique, la cual si me ha supuesto dificultades en ciertos momentos.

Una vez los algoritmos fueron implementados, se procedió a la ejecución de las diferentes simulaciones tanto de los algoritmos con Unit Critiquing como con Compound Critiquing, con el fin de obtener los resultados de la ejecución. Finalmente, se obtuvieron los resultados que se esperaban en los que los algoritmos implementados sobre Compound Critique mejoraban considerablemente a los que se basaban sobre Unit Critique, quedando finalmente el HOR como el algoritmo más óptimo seguido por el algoritmo HGR. En la sección de resultados se ha podido observar, la comparativa de los diferentes algoritmos y como unos se posicionaban mejor que otros (el caso en que HAC pasaba a ser mejor que HOR) dependiendo del conocimiento previo que los usuarios tenían al iniciar la sesión de simulación.

Respecto a las técnicas implementadas con Clustering, la primera técnica de utilización de los casos de la clase mayoritaria no mejoran los resultados del algoritmo IC. Respecto a las técnicas de potenciación de las clases minoritarias/mayoritarias implementadas se produce una mejora similar en cuanto a IC de ambas técnicas.

En futuras investigaciones un interesante tema a tratar será la búsqueda de métodos más óptimos en cuanto a la generación de pruebas, ya que dependiendo del algoritmo utilizado, los tiempos de ejecución eran demasiado lentos por lo que se podrá enfocar a la optimización de este proceso mediante computación paralela o multihilo.

Como conclusión final comentar que este proyecto ha supuesto un reto para mí, ya que hasta el momento no había realizado nada de este tipo, además ha permitido cumplir los objetivos que se esperaban al principio del proyecto,

tanto los referentes al trabajo de la investigación, implementación y análisis de los resultados como los referentes a la motivación personal que se comentaban al principio de este documento.

Referencias

[1] L. McGinty and J. Reilly. On the Evolution of Critiquing Recommenders, 419-453, 2011.

[2] L. Chen and P. Pu. Critiquing-based recommenders: survey and emerging trends. *User Model User-Adap Inter*, 22:125-150, 2012.

[3] J. Reilly, K. McCarthy, L. McGinty, B. Smyth. Incremental critiquing. Knowledge-Based Systems, 18(4-5):143-151, 2005.

[4] M. Salamó, J. Reilly, L. McGinty, B. Smyth. Improving Incremental Critiquing, 379-388, 2005.

[5] K. McCarthy, L. McGinty, B. Smyth, J. Reilly. On the Evaluation of Dynamic Critiquing: A Large-Scale User Study.

[6] K. McCarthy, Y. Salem, B. Smyth. Experience-Based Critiquing: Reusing Critiquing Experiences to Improve Conversational Recommendation. In *Proceedings of the International Conference on Case Base Reasoning,* 480-494, 2010.

[7] Y. Salem, J. Hong. History-aware Critiquing-based Conversational Recommendation. In *Proceedings of the 22Nd International Conference on WWW Companion,* 63-64, 2013.

[8] Y. Salem, J. Hong, W. Liu. History-guided Conversational Recommendation. In *Proceedings of the 23rd International Conference on WWW Companion,* 999-1004, 2014.

[9] D. Contreras, M. Salamó. Integrating History and User Opinion for Conversational Recommendations, 2014.

[10] L. Chen, P. Pu. Evaluating Critiquing-based Recommender Agents. In *Proceedings of the National Conference on Artificial Intelligence,* 21(1):157-162, 2006.

[11] J. Reilly, K. McCarthy, L. McGinty, B. Smyth. Explanining Compound Critiquing, 24(2):199-220, 2005.

[12] H. Xie, L. Chen, F. Wang. Collaborative Compound Critiquing. In *User Modeling, Adaptation, and Personalization,* 8538:254-265, 2014.

[13] K. McCarthy, J. Reilly, L. McGinty, B. Smyth. On the Dynamic Generation of Compound Critiques in Conversational Recommender Systems. In *Adaptive Hypermedia and Adaptive Web-Based Systems,* 3137:176-184, 2004.

[14] S. Raj Vasudevan, S. Chakraborti. Mining User Trails in Critiquing Based Recommenders. In *Proceedings of the 23rd International Conference on World Wide Web,* 777-780, 2014.

[15] J. Konstan, J. Riedl. Recommender systems: from algorithms to user experience.. In *User Modeling and User-Adapted Interaction,* 22(1-2):101-123, 2012.

[16] J. Reilly, J. Zhang, L. McGinty, P. Pu, B. Smyth. A Comparison of Two Compound Critiquing Systems. In *Proceedings of the 12th International Conference on Intelligent User Interfaces,* 317-320, 2007.

[17] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z. Zhou, M. Steinbach, D. J. Hand, D. Steinberg. Top 10 algorithms in data mining, 2007.

[18] L. Smith. A tutorial on Principal Components Analysis, 2002.