UNIVERSIDAD COMPLUTENSE DE MADRID
FACULTAD DE INFORMÁTICA

TESIS DOCTORAL

Una plataforma basada en conocimiento para la construcción de sistemas recomendadores

MEMORIA PARA OPTAR AL GRADO DE DOCTOR

PRESENTADA POR

José Luis Jorro Aragoneses

Directores

María Belén Díaz Agudo
Juan Antonio Recio García

Madrid

UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA

TESIS DOCTORAL

Una plataforma basada en conocimiento para la construcción de sistemas recomendadores
A knowledge-based platform for building recommender systems

MEMORIA PARA OPTAR AL GRADO DE DOCTOR

PRESENTADA POR

Jose Luis Jorro Aragoneses

DIRECTOR

María Belén Díaz Agudo
Juan Antonio Recio García

Una plataforma basada en conocimiento para la construcción de sistemas recomendadores

– – – – – – – – – – – – – – – – – – – –

A knowledge-based platform for building recommender systems

TESIS DOCTORAL

Jose Luis Jorro Aragoneses

Dirigida por los Doctores

María Belén Díaz Agudo
Juan Antonio Recio García

Facultad de Informática
Universidad Complutense de Madrid
Madrid, 2020

A knowledge-based platform for building recommender systems

Submitted for the degree of Doctor in Computer Science

Jose Luis Jorro Aragoneses

Advisors

María Belén Díaz Agudo
Juan Antonio Recio García

Facultad de Informática
Universidad Complutense de Madrid
Madrid, 2020

Una plataforma basada en conocimiento para la construcción de sistemas recomendadores

Memoria que presenta para optar al título de Doctor en Ingeniería Informática

Jose Luis Jorro Aragoneses

Dirigida por los Doctores

María Belén Díaz Agudo
Juan Antonio Recio García

Facultad de Informática
Universidad Complutense de Madrid
Madrid, 2020

A Irene y Carolina
por apoyarme e ilusionarme

In memory of Miltos Petridis

Agradecimientos

Jamás imaginé que lograría llegar hasta aquí, ¡escribir los agradecimientos de mi tesis doctoral! Si echo la vista atrás, veo cuánto ha cambiado todo en esta etapa de mi vida tanto a nivel profesional como personal. Estos años, he tenido la fortuna de contar con muchísimas personas que me han ayudado para poder completar este trabajo. Y quiero aprovechar la oportunidad para darles las gracias a todos ellos.

A las primeras personas que tengo que agradecer todo su apoyo es a Belén y Juanan. Ellos se embarcaron en la aventura de dirigir mi tesis doctoral y no habría llegado hasta aquí sin su ayuda. Quiero darles las gracias por esta oportunidad que me han dado. Gracias a ellos he conocido un mundo que siempre creí inalcanzable. También quiero agradecerles todo lo que me han aguantado estos años, y haber cuidado de mí este tiempo para poder investigar, cuando en estos años, ser investigador y vivir de ello en España es muy complicado. Muchísimas gracias.

Por otro lado, quiero dar las gracias a todos los compañeros que he conocido durante mi doctorado. Dar las gracias a todos ellos en un párrafo puede que sea la tarea más difícil de esta tesis doctoral. Quiero acordarme de todos los compañeros con los que he convivido en el despacho 411 y en la Facultad. En primer lugar, quiero agradecer a los *"profes"* Raquel, Virginia, Antonio Sánchez, Javier, Manuel, Fede, Susi que me han ayudado a afrontar las distintas etapas del doctorado con sus consejos, además de las conversaciones en el café tan interesantes. En especial quiero dar las gracias a Guille por su ayuda. Gracias a su conocimiento he conseguido mejorar como investigador y ser más crítico con mi trabajo. También quiero dar las gracias a los últimos compañeros con los que he coincidido y que ahora son mis amigos: Alejandro, Lolo, Marta, Marlon, Ismael, Antonio G. Sevilla y todos los demás. Tampoco quiero olvidarme de toda la gente que conocí en mis ocho meses de estancia en Brighton: Stelios, Gharbi, Abdullah, Ade y Mohamed. Ellos me hicieron sentir como en casa y me ayudaron muchísimo en mi adaptación a Inglaterra. Quiero recordar especialmente a Miltos Petridis, una persona que me ayudo mucho en mi estancia, acepto mi primera publicación como investigador para el workshop de UKCBR y que desgraciadamente no ha podido ver este trabajo finalizado.

Quiero dar un agradecimiento más formal a la empresa Sismotur S.L., al proyecto Kazemi Back Health S.L., a los proyectos TIN2014-55006-R y TIN2017-87330-R del Ministerio de Ciencia y Tecnología, y en especial a la Universidad Complutense de Madrid y al Banco Santander por su beca de Formación de Personal Investigador ya que han financiado todo este trabajo.

Uno de los pilares fundamentales en esta etapa ha sido mi familia. Ellos me han acompañado en todo este camino dándome su afecto, apoyándome en los malos momentos y celebrando mis éxitos. Dentro de mi familia, la principal persona que ha hecho que yo llegue hasta aquí es mi madre. Ella me animó a que me presentase a la Selectividad y probase a estudiar la Ingeniería Informática *"para ver si me gustaba..."* Y ahora aquí estoy. También quiero agradecer a mi padre que su orgullo me ha servido de motivación para continuar trabajando duro en mi doctorado. Así como a mis hermanos Nacho y Clara, que me han escuchado siempre, por muy aburrido que fuese lo que les contase. Tampoco me quiero olvidar de Marga que siempre se ha interesado por lo que he investigado y me ayudó muchísimo en mi estancia en Brighton.

Por último, quiero hacer el agradecimiento más especial a la persona que me ha acompañado casi media vida: Irene. Ella ha estado conmigo todos estos años sin pedir nada a cambio. Sin ella, este proceso habría sido infinitamente más difícil. Jamás podré agradecerle su forma de animarme cuando más negro lo veía, escuchar mis presentaciones en inglés o que me acompañase en mi aventura por Brighton. Irene, esta tesis también es tuya. Además, quiero agradecerle que haya traído al mundo a la personita que más quiero en este mundo. Mi pequeña Carolina, llegar a casa y estar contigo es la mejor recompensa, sobre todo, los días que llego agotado. Ahora no lo sabes, pero tu también me has animado en este sprint final.

Seguramente me habré olvidado de alguien y espero que me perdone por el descuido. En definitiva, a todos los que habéis formado parte de mi vida durante todo este tiempo os doy las gracias.

Resumen

En la actualidad Internet es la principal fuente de información que existe debido a la gran cantidad de contenidos digitales que los usuarios están creando. Esto hace que la tarea de buscar información relevante para los usuarios sea bastante compleja. Además, debido a la importancia de atraer usuarios a las distintas aplicaciones web, las empresas cada vez invierten más recursos en mejorar la experiencia de los usuarios en sus aplicaciones. Una de las estrategias más utilizadas para resolver tanto el problema de sobrecarga de información como la mejora de la experiencia de los usuarios es el uso de sistemas recomendadores. Estos sistemas permiten filtrar la información a los usuarios según sus intereses. Debido a la utilidad de estos sistemas, la investigación y el desarrollo de soluciones en esta área han evolucionado mucho desde los primeros trabajos a principios de los años 90. Durante este tiempo, se han ido mejorando los sistemas de recomendación creando nuevas técnicas e incluyendo más información para mejorar los resultados. Este impulso ha hecho que se hayan desarrollado una gran cantidad de frameworks[1] que facilitan la construcción de nuevos sistemas recomendadores. Sin embargo, todos los avances en este campo de investigación hacen que el desarrollo de aplicaciones de recomendación no sea una tarea sencilla. Esto se debe a que, en primer lugar, los desarrolladores deben tener un conocimiento avanzado en las diversas técnicas de recomendación y saber en qué casos se pueden aplicar cada una de esas técnicas. En segundo lugar, deben tener un conocimiento técnico suficiente para desarrollar el sistema recomendador desde cero o, si es el caso, usando un framework externo.

El principal objetivo de esta tesis doctoral es simplificar el proceso de diseño, desarrollo y despliegue de los sistemas recomendadores. La principal contribución en nuestra investigación es la propuesta de una metodología para el desarrollo de sistemas recomendadores a través del uso de componentes. Para ello, hemos realizado un estudio del dominio para comprender las fases del diseño de los sistemas recomendadores y hemos analizado un conjunto de frameworks que facilitan el desarrollo de los sistemas de recomendación. Además, hemos construido un sistema recomendador real como

[1] En esta memoria de tesis doctoral usaremos la palabra *framework* para referirnos a las librerías de desarrollo de software.

caso de estudio para comprender la construcción de este tipo de sistemas y poder evaluar la metodología que se ha propuesto en nuestra investigación.

Como resultado de todo este trabajo, hemos propuesto un modelo donde se representan los distintos elementos que existen en los sistemas recomendadores y cómo se deben componer para construirlos. Este modelo se ha incluido en nuestra metodología de desarrollo de los sistemas recomendadores y permite asistir a los desarrolladores en el proceso de construcción de estos sistemas. A partir de la metodología y el modelo propuestos, se ha desarrollado la plataforma RecoLibry Suite. Esta plataforma contiene un conjunto de herramientas que ayudan a los usuarios en las 3 fases por las que pasa la construcción de los sistemas recomendadores: diseño, desarrollo y despliegue. Todas estas contribuciones han sido evaluadas y los resultados han sido presentados a la comunidad científica con las publicaciones que se aportan en esta tesis doctoral.

Palabras clave: sistemas recomendadores, información contextual, red semántica, arquitectura basada en componentes, metodología de desarrollo.

Abstract

At the present the Internet is the primary source of information because of the large amount of digital content that users create every day. For this reason, searching for relevant information is a quite complicated task. Also, due to the importance of attracting users, companies invest many resources in improving the user experience in their applications. A strategy commonly used to resolve both the problem of information overload and user experience is the use of recommender systems. These systems filter the information based on the users' interests. Research and development in this area have evolved much since the early 1990s, creating new techniques and including more information to improve the recommendations. This evolution is due to the usefulness of these systems. This has also led to the development of frameworks to facilitate the building of recommender systems. However, all of these advances in the recommender systems area mean that nowadays development of such systems is not an easy task. This is because, developers need to have a high knowledge level in the diverse recommender techniques, and they have to know when they can use each technique. Additionaly, they should have enough programming skill level to implement a recommender system from scratch or using an external framework.

The main objective in this Ph. D. thesis is to simplify the processes of designing, developing and deploying recommender systems. The main contribution of our research is the proposal of a methodology to build recommender systems based on components. To do that, first we have carried out a domain study to understand the design steps of recommender systems. Next, we have analyzed the features of a set of frameworks used by developers to build recommender systems. Futhermore, we have developed a real recommender system as a case study to understand the building of these systems, and to evaluate the methodology proposed in our research work.

As a result of our research, we have presented a model to represent the different elements included in recommender systems, and how to use these components to build a system. We have included this model in our development methodology. Thanks to this model, our platform can assist users in the building process of these systems. From the model and the methodology, we have developed the platform RecoLibry Suite. This platform contains a

toolset for aiding users in the three steps to build a recommender system: design, develop, and deploy. All of these contributions have been evaluated, and results have been presented to the scientific community in the publications included in this Ph. D. thesis.

Keywords: recommender systems, contextual information, semantic network, architecture based on components, developing methodology.

Índice

Índice de figuras

Índice de Tablas

Parte I

Contribuciones

Capítulo 1

Introducción

Este capítulo enmarca la investigación de esta tesis doctoral dentro del área de los sistemas recomendadores. El objetivo general de nuestra investigación es el desarrollo de una plataforma que facilite la creación de sistemas recomendadores. Por este motivo, en este capítulo se describen los problemas que encuentran los desarrolladores de este tipo de sistemas y cuál ha sido nuestra planificación de la investigación para solucionar estos problemas. La Sección 1.1 explica las razones por las que consideramos que el trabajo de nuestra investigación es necesario. A continuación, en la Sección 1.2 se exponen el conjunto de objetivos específicos que se han definido a lo largo de nuestra investigación. Por último, la Sección 1.3 muestra un resumen de las contribuciones asociadas a los objetivos específicos.

Para la presentación de esta tesis se ha elegido el formato basado en publicaciones. Todas ellas se encuentran en el Parte III y describen en detalle cada una de las contribuciones que se exponen en esta memoria.

1.1. Motivación

El proceso de desarrollo de un sistema recomendador no es una tarea sencilla. Los desarrolladores tienen que hacer un análisis detallado de la información que usará el sistema recomendador, identificar la técnica que mejor se adapte a esa información y desarrollar el sistema recomendador de cero o elegir uno de los muchos frameworks que existen. Este proceso es cada vez más complejo debido a nuevas técnicas de recomendación y nuevos frameworks que van surgiendo.

Aunque la investigación en los sistemas recomendadores comenzó a principios de los años 90 (Goldberg et al., 1992), fue entre 2000 y 2005 cuando estos sistemas alcanzaron una mayor popularidad. Esta popularidad está ligada al comienzo de la llamada Web 2.0 (O'Reilly, 2009). Los sitios web que aparecían a finales de los años 2000 y principios de 2010 permitían una comunicación bidireccional con los usuarios, es decir, los usuarios podían par-

ticipar activamente añadiendo su propio contenido, por ejemplo, en blogs, wikis, redes sociales, etc.

La Web 2.0 trajo consigo dos factores que impulsaron el desarrollo de sistemas de recomendación: la sobrecarga de información y la facilidad de obtener la opinión de los usuarios. En primer lugar, la cantidad de información que se generaba era cada vez más grande y eso provocaba que, para los usuarios, fuese realmente difícil encontrar información útil. Los sistemas recomendadores sirvieron para mostrar a los usuarios aquellos contenidos que les podían interesar y que eran incapaces de encontrar debido a ese exceso de información que ofrecía la Web. El segundo factor fue que los nuevos sitios Web 2.0 facilitaban la recopilación de opiniones y preferencias de los usuarios. Esto permitió desarrollar sistemas de recomendación basados en la opinión de usuarios, como los *sistemas recomendadores de filtrado colaborativo* (Bobadilla et al., 2009) en los que se recomiendan elementos que otros usuarios con valoraciones similares han preferido. La cantidad de opiniones impulsó la investigación de este tipo de algoritmos, como ocurrió en 2006 cuando Netflix organizó una competición para crear un sistema recomendador a partir de un conjunto de datos con la opinión de sus usuarios (Bennett et al., 2007). Por otro lado, el uso de sistemas de recomendación permitía a los usuarios personalizar su experiencia en sitios web o de comercio electrónico como se explica en Karat et al. (2004). Por ejemplo, los sistemas de recomendación de Netflix, Amazon o Spotify muestran resultados/productos distintos para cada usuario, de forma personalizada.

Otro paradigma de sistemas de recomendación muy estudiado ha sido el de los *sistemas recomendadores basados en contenido* (de Gemmis et al., 2015). A diferencia de los de filtrado colaborativo, estos sistemas trabajan con descripciones muy detalladas de los elementos a recomendar. A partir de las interacciones de los usuarios, estos sistemas generan un perfil de usuario basado en las preferencias obtenidas a través de una interfaz o consulta. Este perfil de usuario contendrá las características comunes de los elementos que los usuarios han ido eligiendo. Los sistemas basados en contenido recomiendan aquellos productos cuya descripción se parece más al perfil del usuario generado. Este tipo de sistemas se han beneficiado de que la cantidad de información que existe sobre los productos es cada vez más detallada. La investigación en estos y otros paradigmas adicionales sirvió para crear *sistemas recomendadores híbridos*. Estos sistemas híbridos solucionan algunos errores que tienen cada uno de los paradigmas individualmente a partir de la combinación de ambas técnicas (Burke, 2002).

Con el énfasis de los dispositivos móviles se inició otra área de investigación en los sistemas de recomendación centrada en estudiar factores adicionales que podrían condicionar la opinión de los usuarios. A estos factores adicionales se les denomina *información contextual* (Dey, 2001). La información contextual describe el estado y entorno de los usuarios y los productos

en un momento concreto. Esta información puede ser, por ejemplo, la posición del usuario, el clima o el horario de una actividad. Esta información se ha ido añadiendo a los sistemas de recomendación como una dimensión más para tener en cuenta (Adomavicius y Tuzhilin, 2015). A estos sistemas se les denomina *sistemas recomendadores contextuales* (CARS del inglés Context-Aware Recommender Systems). Un dominio donde se utiliza mucho este tipo de recomendadores es el del turismo y ocio, sobre todo cuando los usuarios usan sus dispositivos móviles y quieren recomendaciones en tiempo real (Borràs et al., 2014).

En la actualidad existe una gran variedad de técnicas de recomendación y para implementar un nuevo sistema, un desarrollador debe conocer sus características y, además, conocer cuándo puede usar cada una de ellas. Para ello, un primer paso que debe hacer un desarrollador es analizar la información que usará su sistema. Este análisis permite detectar posibles problemas asociados a los datos, como que no existan suficientes datos iniciales para generar recomendaciones (a este problema se le denomina *problema del arranque en frío*). Este análisis influye en qué técnica de recomendación se puede utilizar y cuál dará mejores resultados.

La popularidad de los sistemas recomendadores ha influido en la gran variedad de librerías y frameworks que existen hoy en día y que facilitan su construcción (Owen et al., 2011; Ekstrand et al., 2010; Recio-García et al., 2008). Sin embargo, esta gran variedad de frameworks añade una nueva dificultad debido a que existen muchas diferencias en cada uno de ellos: el lenguaje de programación, limitación de los algoritmos que implementan o la construcción final del sistema. Algunos de estos frameworks utilizan términos que pueden diferir del vocabulario que se utiliza en la fase del diseño de un sistema recomendador. Esto hace que el diseño de un sistema recomendador necesite de una persona o un equipo de personas muy especializados con conocimiento en el área de sistemas de recomendación y conocimientos técnicos para utilizar el framework seleccionado. Asimismo, los términos que se usan en la fase de diseño deben ser adaptados en la fase de desarrollo si el framework utiliza otra terminología.

Este proceso de diseño y desarrollo se puede simplificar aplicando un diseño basado en componentes para la construcción de los sistemas recomendadores ya que la mayoría de estos sistemas utilizan un esquema similar en el flujo de la información. Esto permite que se puedan definir componentes que tengan una funcionalidad concreta dentro del sistema recomendador como, por ejemplo, la entrada de datos del usuario o técnicas específicas de recomendación (filtrado colaborativo, basado en contenido, etc.). Una de las grandes ventajas del uso de componentes es su reutilización en distintas soluciones. Esto hace que se acorten los tiempos de desarrollo y mejoren los costes de la construcción del software (Szyperski et al., 2002). Por otro lado, el uso de componentes nos permite definir las relaciones e incompatibilida-

des que existen entre ellos y crear un modelo que describa cómo construir un sistema recomendador. Por ejemplo, si el módulo de entrada de datos devuelve información contextual, podemos filtrar aquellos algoritmos que no usan este tipo de información. Gracias a esta forma de construir los sistemas de recomendación, un diseñador con menos conocimientos en el área de los recomendadores puede saber qué componentes puede aplicar a su sistema. Por último, el uso de componentes permite separar la implementación de cada componente de la construcción de un sistema recomendador. Esto hace que los componentes puedan estar implementados con diferentes frameworks, y que este hecho sea transparente para el desarrollador, que solo debe seleccionar qué componentes utilizará.

Recapitulando, la construcción de un sistema recomendador es un proceso complejo en el que el desarrollador debe tener un conocimiento alto en las distintas tareas del proceso. En primer lugar, el desarrollador debe seleccionar la técnica de recomendación que mejor se adapta al dominio y, además, analizar los posibles problemas que se pueda encontrar de acuerdo con los datos disponibles. A continuación, el desarrollador debe elegir cómo implementará su sistema, ya sea a través de un framework, o bien, implementándolo completamente. En ambos casos debe tener un conocimiento técnico avanzado. Siendo esta una carencia muy importante en este campo, en la siguiente sección, explicamos los objetivos que hemos definido para facilitar el proceso de creación de sistemas recomendadores.

1.2. Objetivos

Como hemos explicado en la sección anterior, crear un sistema recomendador es una tarea compleja. Por este motivo, el objetivo general de esta tesis doctoral es construir una plataforma basada en conocimiento que unifique y ayude en el proceso de diseño y desarrollo de sistemas de recomendación. Esta plataforma usa un modelo teórico que describe cómo se deben construir los sistemas de recomendación a partir de un conjunto de componentes. Además, se describen diferentes herramientas que, a partir de ese modelo, permiten crear sistemas recomendadores a usuarios con distintos perfiles. Estas herramientas proporcionan asistencia automática orientando al usuario en la selección de los componentes que se deben utilizar. Para poder cumplir este objetivo, se plantearon una serie de objetivos más específicos.

El primer objetivo específico (**O-1**) ha consistido en comprender cómo se construyen los sistemas recomendadores y cómo plasmar esta construcción en un modelo teórico basado en componentes. Se enumeraron distintas técnicas de recomendación que existen en el estado del arte y se han definido las restricciones que tienen dependiendo de la información que utilizan. Además, se analizaron los problemas que se pueden encontrar en sistemas recomendadores. Por último, se analizaron un conjunto de frameworks académicos y

de código libre que se utilizan para construir estos sistemas con el objetivo de conocer sus características.

El segundo objetivo específico (**O-2**) persigue comprender la construcción de los sistemas recomendadores, y por ello ha consistido en crear un sistema real en el dominio del turismo y ocio. El sistema devuelve los planes turísticos para los usuarios a partir de sus preferencias y la información contextual en el momento de la recomendación. Este sistema fue un caso de estudio donde hemos podido probar las distintas funcionalidades y técnicas que se han añadido al modelo y a la plataforma basada en conocimiento. El dominio elegido de este sistema recomendador fue el turismo y ocio debido a que este dominio nos permite incluir información contextual en el sistema recomendador como, por ejemplo, las localizaciones de los usuarios, los horarios de las actividades, etc. En este sistema se analizó la influencia de la información contextual a las recomendaciones que se daban a los usuarios.

Este sistema recomendador también nos permitió definir el tercer objetivo específico (**O-3**), que consistió en estudiar soluciones a uno de los problemas más comunes que existen en los sistemas recomendadores: el *arranque en frío* (Park y Chu, 2009). Este problema surge cuando no hay suficiente información para que el sistema recomendador tenga un rendimiento aceptable. Esto pasa, por ejemplo, si no hay suficientes valoraciones en sistemas de filtrado colaborativo o no existen suficientes descripciones de productos en sistemas basados en contenido. En esta tesis proponemos aplicar *sistemas de razonamiento basado en casos* (CBR del inglés Case-Based Reasoning) como solución a este problema en algunos sistemas recomendadores. Los sistemas CBR basan su razonamiento en buscar soluciones a un problema nuevo reutilizando soluciones que se han aplicado en situaciones similares (Kolodner, 1992). En nuestro grupo de investigación GAIA (Grupo de Aplicaciones de Inteligencia Artificial) existe una larga trayectoria en aplicar CBR a diversos problemas de inteligencia artificial.

Partiendo del estudio del estado del arte y de la construcción de un sistema real, el cuarto objetivo específico (**O-4**) fue formalizar un modelo basado en componentes que permitiera definir cómo se construyen los sistemas recomendadores. Es decir, definir cada una de las funcionalidades de los sistemas recomendadores como una caja negra donde se definan las restricciones de uso y su signatura respecto a los datos de entrada y salida. La principal ventaja de este modelo es permitir la reutilización de estos componentes en distintos sistemas recomendadores. Además, en este modelo se representan las reglas de relación entre los distintos componentes. Por último, al definir los componentes del sistema recomendador, este modelo crea un vocabulario común para construir cualquier tipo de sistema recomendador, siendo independiente del framework final que se utilice. Para este objetivo se definió que el modelo se tenía que representar mediante una ontología para que se pudiese usar como base en la plataforma inteligente. Una ontología permite crear

una definición formal de un conjunto de entidades y definir las propiedades de las entidades y la relaciones que existen entre ellas (Gruber, 2009).

Todos estos objetivos específicos nos permitieron completar el objetivo final de esta tesis (**O-5**): el desarrollo de una plataforma basada en conocimiento que dé soporte para el diseño, implementación y despliegue de los sistemas recomendadores. Esta plataforma, llamada RecoLibry-studio, se basa en el modelo teórico basado en componentes para construir los sistemas recomendadores. Esta plataforma cuenta con 3 herramientas:

- Una herramienta de alto nivel para el diseño de los sistemas recomendadores. Esta herramienta usa el modelo teórico para orientar al desarrollador a la hora de seleccionar qué técnica de recomendación se adapta mejor a los componentes seleccionados previamente.
- Un framework en Java que implementa los componentes a partir de otros frameworks de sistemas recomendadores. Además, este framework permite la construcción de los sistemas recomendadores usando inyección de dependencias.
- Una herramienta web que permite el despliegue automático de los sistemas creados con esta plataforma. Además, esta herramienta despliega una API para que el sistema recomendador pueda ser utilizado en aplicaciones externas.

1.3. Resumen de las contribuciones

Como hemos explicado en la sección anterior, en esta tesis se definieron un conjunto de objetivos específicos para poder completar el objetivo final de desarrollar una plataforma para crear sistemas recomendadores. Por cada uno de estos objetivos se han obtenido uno o más resultados. A continuación, explicaremos las contribuciones resultantes de cada uno de los objetivos, el capítulo en el que se encuentran y las publicaciones asociadas a estas contribuciones:

Objetivo 1 (O-1)
Comprender la construcción de los sistemas recomendadores analizando las técnicas utilizadas, los problemas que se puedan encontrar y los frameworks disponibles.

Resultado 1 Estudio de sistemas recomendadores clásicos y sistemas recomendadores contextuales incluyendo las distintas técnicas que utilizan.

Contribuciones aportadas:

- Capítulo 2: Estudio del dominio.

Resultado 2 Clasificación y análisis de los principales problemas que se encuentran en un sistema recomendador incluyendo soluciones encontradas en la literatura.

Contribuciones aportadas:

- Capítulo 2: Estudio del dominio.

Resultado 3 Estudio de los principales frameworks para crear sistemas recomendadores clasificando sus características para su integración con la plataforma del objetivo **O-5**.

Contribuciones aportadas:

- Capítulo 2: Estudio del dominio.
- Capítulo 15: RecoLibry Suite: a set of intelligent tools for the development of recommender systems, (Jorro-Aragoneses et al., 2020).

Objetivo 2 (O-2)

Desarrollar un sistema recomendador que utilice información contextual y que permita probar los conceptos que se añadirán en el modelo teórico.

Resultado 4 Desarrollo de un sistema recomendador de planes turísticos y de ocio en Madrid que utiliza información contextual: *Madrid Live*.

Contribuciones aportadas:

- Capítulo 3: Estudio de la construcción de un sistema recomendador contextual.
- Capítulo 9: Madrid Live: a context-aware recommender system of leisure plans, (Jorro-Aragoneses et al., 2017b).

Objetivo 3 (O-3)

Desarrollar distintas soluciones al problema del cold-start en sistemas recomendadores.

Resultado 5 Propuesta de un método CBR para resolver el problema del cold-start en sistemas recomendadores.

Contribuciones aportadas:

- Capítulo 3: Estudio de la construcción de un sistema recomendador contextual.
- Capítulo 11: Addressing the cold-start problem in facial expression recognition, (Jorro-Aragoneses et al., 2015).

Resultado 6 Propuesta de una metodología que permite crear una base de casos inicial para sistemas recomendadores de planes turísticos.

Contribuciones aportadas:

- Capítulo 3: Estudio de la construcción de un sistema recomendador contextual.
- Capítulo 12: Case base elicitation for a context-aware recommender system, (Jorro-Aragoneses et al., 2018).

Objetivo 4 (O-4)

Crear un modelo teórico basado en componentes que describa la construcción de los sistemas recomendadores y plasmarlo en una ontología para utilizarlo en la plataforma del objetivo **O-5**.

Resultado 7 Propuesta de una ontología que defina la construcción de sistemas recomendadores a partir de componentes: RecOnto.

Contribuciones aportadas:

- Capítulo 4: Una plataforma inteligente para la creación de sistemas recomendadores.
- Capítulo 13: RecOnto: an ontology to model recommender systems and its components, (Jorro-Aragoneses et al., 2017a).
- Capítulo 15: RecoLibry Suite: a set of intelligent tools for the development of recommender systems, (Jorro-Aragoneses et al., 2020).

Objetivo 5 (O-5)

Desarrollar una plataforma que ayude en el diseño, implementación y despliegue de los sistemas recomendadores basándose en el modelo obtenido en el objetivo **O-4**.

Resultado 8 Creación de un framework para crear sistemas recomendadores a partir de los componentes definidos en el objetivo **O-4**: RecoLibry-core.

Contribuciones aportadas:

- Capítulo 4: Una plataforma inteligente para la creación de sistemas recomendadores.
- Capítulo 14: RecoLibry-core: A component-based framework for building recommender systems, (Jorro-Aragoneses et al., 2019).
- Capítulo 15: RecoLibry Suite: a set of intelligent tools for the development of recommender systems, (Jorro-Aragoneses et al., 2020).

Resultado 9 Desarrollo de una aplicación web que permita diseñar y crear un sistema recomendador de forma semi-automática: RecoLibry-studio.

Contribuciones aportadas:

- Capítulo 4: Una plataforma inteligente para la creación de sistemas recomendadores.
- Capítulo 15: RecoLibry Suite: a set of intelligent tools for the development of recommender systems, (Jorro-Aragoneses et al., 2020).

Resultado 10 Desarrollo de un servidor web que permita desplegar los sistemas recomendadores de forma automática a partir de un fichero de configuración: RecoServer.

Contribuciones aportadas:

- Capítulo 4: Una plataforma inteligente para la creación de sistemas recomendadores.
- Capítulo 15: RecoLibry Suite: a set of intelligent tools for the development of recommender systems, (Jorro-Aragoneses et al., 2020).

La memoria de esta tesis doctoral se ha organizado de la siguiente manera: el Capítulo 2 contiene un estudio del dominio de los sistemas recomendadores y frameworks para construir sistemas recomendadores; el Capítulo 3 describe todas las contribuciones asociadas al sistema recomendador contextual desarrollado y las soluciones propuestas al problema del *cold-start*; el Capítulo 4 contiene las contribuciones asociadas al modelo teórico y a la plataforma para generar sistemas recomendadores; en el Capítulo 5 se hace una recopilación de las conclusiones obtenidas a lo largo de la investigación de esta tesis y las líneas de trabajo futuras que quedan abiertas. Por último, la Parte III contiene todas las publicaciones que explican en detalle cada una de las contribuciones presentadas.

Capítulo 2

Estudio del dominio

El primer objetivo específico que se planteó en esta tesis doctoral es el objetivo **O-1**:

Objetivo 1 (O-1)
Comprender la construcción de los sistemas recomendadores analizando las técnicas utilizadas, los problemas que se puedan encontrar y los frameworks disponibles.

Para cumplir este objetivo, en la Sección 2.1 se presenta una revisión de las distintas técnicas de recomendación que se han propuesto en la literatura. A continuación, la Sección 2.2 enumera los problemas más comunes asociados al conjunto de datos que se utilizan en los sistemas de recomendación. Por último, en la Sección 2.3 se muestran los resultados del análisis realizado a un conjunto de frameworks para construir estos sistemas. En este análisis se han descrito las ventajas y limitaciones que tienen cada uno de los frameworks analizados.

2.1. Sistemas recomendadores

A la hora de tomar una decisión, normalmente no conocemos en detalle toda la información de cada una de las posibilidades que se nos presentan. Por esta razón, tendemos a escuchar las recomendaciones que nos hacen otras personas o que leemos a través de artículos de opinión para guiarnos en nuestra decisión. Los sistemas recomendadores intentan imitar este proceso para mostrarnos aquellas opciones que más nos pueden interesar (Resnick y Varian, 1997). Un factor importante que ha popularizado los sistemas recomendadores ha sido el crecimiento exponencial de información que tenemos en Internet. Estos sistemas son una herramienta para filtrar de forma personalizada la información, ya que muestran solo la información que puede

interesar a cada usuario según sus gustos o preferencias.

Aunque el objetivo principal de los sistemas recomendadores es facilitar el proceso de búsqueda de información a los usuarios, han surgido nuevos objetivos en los que enfocar los resultados de los sistemas recomendadores. En Ricci et al. (2015) encontramos algunos objetivos para los que se pueden aplicar los sistemas recomendadores: incrementar las ventas de un producto, mejorar la experiencia de los usuarios en un sitio web, mejorar la fidelidad de los usuarios, etc. Un ejemplo muy claro lo encontramos en los sitios web de comercio electrónico donde los productos que se muestran a cada usuario son aquellos que pueden interesarle según las compras que ha realizado anteriormente (Schafer et al., 1999). Los sistemas recomendadores se pueden aplicar a una gran variedad de dominios como libros, música, películas, viajes, etc. (Bobadilla et al., 2013).

La mayoría de los sistemas recomendadores cuentan con tres elementos básicos para generar las recomendaciones (Ricci et al., 2015). Estos tres elementos son los siguientes:

Usuarios Son los consumidores de los sistemas recomendadores. Dependiendo del objetivo que tenga el sistema de recomendación, se guardará distinto tipo de información de los usuarios. Esta información puede ser desde únicamente el identificador del usuario hasta un conjunto de características de los productos que le gustan o datos demográficos del usuario.

Ítems Son los elementos que van a ser recomendados en el sistema. Según la técnica de recomendación, el sistema almacenará diferentes datos de los ítems. Por ejemplo, algunas técnicas solo necesitan el identificador de los ítems y otras necesitan una descripción detallada de cada uno de ellos. Los sistemas de recomendación clasificarán a los ítems según la utilidad que tengan para un usuario concreto.

Transacciones Son las interacciones que existen entre los usuarios y los ítems. Estas interacciones permiten predecir qué ítems le pueden interesar a un usuario en el futuro. Se pueden obtener de dos maneras: explícita o implícitamente. De forma explícita se obtienen cuando el usuario interactúa con el sistema recomendador para indicar su opinión sobre un ítem, por ejemplo, valorando un producto de 0 a 10. Por otro lado, de forma implícita ocurre cuando esta información se infiere de una acción del usuario, por ejemplo, almacenando en una lista los productos que ha comprado un usuario.

Aunque la mayoría de los sistemas recomendadores utilizan únicamente estos tres elementos, existen otros sistemas que incluyen información adicional para mejorar los resultados de los sistemas recomendadores. Estos

sistemas incluyen información externa a los usuarios y a los ítems que puede haber influido en la opinión del usuario sobre un ítem en un momento concreto. A este tipo de información externa se le denomina *información contextual*. Por ejemplo, imaginemos un sistema recomendador turístico donde las preferencias de un usuario sea visitar parques. Sin embargo, en el momento de generar la recomendación está lloviendo. Este sistema recomendador debería recomendar otra actividad que, aunque no es la preferida por el usuario, mejore su experiencia. La lluvia en este ejemplo sería la información contextual que influye en la recomendación final al usuario. Por este motivo, en esta tesis doctoral hemos clasificado las técnicas de recomendación siguiendo la clasificación que proponen Adomavicius y Tuzhilin (2015). En esta clasificación encontramos 2 tipos de sistemas de recomendación según la información que utilizan: los *sistemas clásicos* o de *2-dimensiones* (2D) y los *sistemas contextuales*. Los sistemas de recomendación clásicos o 2D son aquellos que utilizan únicamente la información de los usuarios, los ítems y sus interacciones. Por otro lado, los sistemas de recomendación contextuales incluyen más dimensiones como atributos contextuales para calcular la sugerencia al usuario.

En las siguientes secciones se explican en detalle cada uno de los tipos de sistemas recomendadores, qué técnicas se utilizan en cada uno de ellos y algunos trabajos encontrados en la literatura que aplican cada uno de los tipos de sistemas de recomendación.

2.1.1. Sistemas recomendadores clásicos

Como hemos explicado en el Capítulo 1, los sistemas recomendadores comienzan a ser un área de investigación independiente a mediados de los años 90. Los primeros trabajos sobre sistemas de recomendación estudiaban la estructura que tenían las valoraciones de los usuarios sobre los ítems para determinar los ítems que les podían interesar. Por este motivo, la mayoría de los trabajos se centraron en crear una *técnica de recomendación basada en valoraciones*, es decir, técnicas que permitiera estimar la valoración que haría un usuario a un ítem que no había consumido aún (Adomavicius y Tuzhilin, 2005). Estos primeros sistemas de recomendación utilizaban algoritmos que tenían como objetivo entrenar una función f que devolvería las estimaciones de las valoraciones ($Ratings$) que un conjunto de usuarios ($Users$) harían sobre un conjunto de ítems ($Items$) (Zafarani et al., 2014).

$$f : Users \times Items \rightarrow Ratings \tag{2.1}$$

En concreto, la función f tenía que devolver un valor real por cada par (u, i) donde u es el identificador de un usuario del conjunto $Users$ e i es el identificador del ítem perteneciente al conjunto $Items$. El valor devuelto en

los sistemas basados en valoraciones dependerá del intervalo de valoraciones que hagan los usuarios en el sistema recomendador (Ekstrand et al., 2010). Sin embargo, el objetivo general de los sistemas recomendadores es devolver el conjunto de ítems que más puedan interesar a los usuarios. Por este motivo, la función anterior se puede generalizar como una función R que utiliza la información entre usuarios ($Users$) e ítems ($Items$) para devolver un conjunto ordenado de ítems ($Recommendations$) según lo útil que sean los ítems para un usuario (Adomavicius y Tuzhilin, 2005). La función R puede usar la función f para conocer el interés del usuario por cada item, pero incluye un sistema de selección para devolver el conjunto de ítems que se le recomendará al usuario.

$$R : Users \times Items \rightarrow Recommendations \tag{2.2}$$

Por este motivo a estos sistemas se les denomina sistemas de recomendación clásicos o de 2-dimensiones. Estos algoritmos entrenaban las funciones a partir de las valoraciones o interacciones de los usuarios con los ítems que conocían. Dentro de este grupo de sistemas recomendadores encontramos tres técnicas distintas: *técnicas de filtrado colaborativo*, *técnicas basadas en contenido* y *técnicas híbridas*. A continuación, describimos en detalle cada una de estas técnicas.

2.1.1.1. Filtrado colaborativo

Los *sistemas de recomendación basados en filtrado colaborativo* utilizan los patrones de las valoraciones que hacen los usuarios a los ítems para obtener nuevos ítems que les puedan interesar (Koren y Bell, 2015). Las técnicas basadas en filtrado colaborativo utilizan como dominio de la información una matriz donde las dimensiones sean los usuarios y los ítems del sistema recomendador, y los valores de la matriz son las valoraciones que han hecho los usuarios a los ítems. Por supuesto, existen muchos valores que no se conocen. A estos se les llama *valores desconocidos* (Ekstrand et al., 2010). El objetivo de la mayoría de los sistemas que utilizan estas técnicas es predecir la opinión de los usuarios en los valores desconocidos y, a partir de estas predicciones, recomendar ítems que puedan interesar a los usuarios. Las técnicas de filtrado colaborativo se pueden dividir en dos grupos (Breese et al., 1998): las *técnicas basadas en memoria* y las *técnicas basadas en modelos*.

Las *técnicas basadas en memoria* necesitan las valoraciones que han hecho los usuarios a los diferentes ítems para encontrar similitudes entre ellas. Dentro de las técnicas basadas en memoria, la más utilizada es la *basada en los vecinos más próximos* sobre el conjunto de usuarios o de los ítems (Schafer et al., 2007). Los *algoritmos basados en los usuarios más próximos* utilizan las valoraciones que ha hecho un usuario objetivo, al que se le recomendará

un ítem, para buscar usuarios similares. Para buscar estos usuarios parecidos se suelen utilizar funciones de similitud (Shimodaira, 2014) como, por ejemplo, las funciones de *similitud del coseno* o de *correlación de Pearson*. Los usuarios similares permiten predecir la valoración que haría el usuario objetivo a ítems que no ha consumido aún, pero que sí han valorado esos usuarios similares. Por otro lado, están los *algoritmos basados en los ítems más próximos*. En este caso, el sistema obtiene los ítems que han recibido valoraciones parecidas al ítem objetivo, el que se va a recomendar. A partir de la valoración que ha hecho el usuario objetivo a los ítems similares se puede predecir la valoración que hará al ítem objetivo.

El segundo tipo de técnicas son las basadas en modelo. Estas técnicas entrenan un modelo con el que predecir los valores desconocidos a partir de las valoraciones que han hecho los usuarios anteriormente (Sarwar et al., 2001). A continuación, los sistemas recomendadores utilizan el modelo resultante para generar las recomendaciones. Algunos de los algoritmos que se utilizan en este tipo de técnicas son *sistemas basados en reglas*, *redes Bayesianas* y *algoritmos de agrupamiento* o *clustering*. Uno de los algoritmos que mejores resultados han dado para predecir valoraciones son los *métodos basados en factores latentes*, como por ejemplo la *factorización de matrices no negativas* (NMF del inglés *Non-Negative Matrix Factorization*) (Lee y Seung, 2001). Éste fue uno de los mejores algoritmos que se presentaron en la competición del Netflix Prize en 2006 (Funk, 2006).

Los sistemas recomendadores basados en filtrado colaborativo son los más utilizados. Muchas de las grandes empresas comerciales utilizan este tipo de sistemas recomendadores para filtrar los productos que muestran a los usuarios. Por ejemplo, *Netflix* (Gomez-Uribe y Hunt, 2016) utiliza algoritmos de filtrado colaborativo para recomendar películas a los usuarios. Otro ejemplo lo encontramos en *Amazon* donde utilizan un algoritmo de filtrado colaborativo basado en ítems (Smith y Linden, 2017) para mostrar a los usuarios recomendaciones en los productos de su catálogo. Un sistema que también se popularizó en este tipo de técnicas de recomendación fue *MovieLens*. MovieLens usa un sistema de recomendación de filtrado colaborativo basado en usuarios para predecir qué películas pueden interesar a un usuario.

2.1.1.2. Sistemas basados en contenidos

Por otro lado, existen los sistemas recomendadores basados en contenido. Estos sistemas utilizan técnicas que comparan las características de los ítems con las características que gustan a los usuarios (Lops et al., 2011). Mientras que las técnicas de filtrado colaborativo solo necesitan tener acceso a las valoraciones o interacciones que hacen los usuarios con los ítems, los sistemas basados en contenido deben tener acceso a dos fuentes de información (Aggarwal, 2016): las descripciones de los ítems a recomendar y los perfiles de usuario donde se guardan las preferencias de los usuarios. Este tipo de

sistemas de recomendación cuenta normalmente con tres etapas (Lops et al., 2011):

1. **Análisis de contenido.** En muchas ocasiones la información que describe a los ítems no se encuentra en una forma estructurada. Se puede encontrar, por ejemplo, en un formato de texto libre. Por este motivo, la mayoría de estos sistemas deben considerar una primera etapa en la que se extraiga la descripción de los ítems con la estructura que utilizará el sistema recomendador. Esta etapa se puede hacer de forma automática o manual.

2. **Entrenamiento de perfiles de usuario.** Los perfiles de usuario que utilizan estos sistemas se generan a partir de las interacciones que van haciendo los usuarios con los ítems que van consumiendo. Este paso es el encargado de almacenar en los perfiles de usuario aquellas características comunes de los ítems que han preferido cada uno de los usuarios. Normalmente, en esta etapa se utilizan técnicas de aprendizaje automático que permiten extraer los atributos de los ítems más significativos para guardarlos en el perfil del usuario.

3. **Filtrado de contenido.** La última etapa es el proceso de recomendación. En este paso se comparan las descripciones de los ítems con las características almacenadas en los perfiles de usuario. Una vez hecho esto, el sistema mostrará aquellos ítems que puedan ser más relevantes para el usuario según la comparación realizada. El usuario, cuando recibe una recomendación, puede continuar valorando los ítems. En estos casos, el sistema recomendador vuelve a ejecutar la etapa de entrenamiento de perfiles para actualizar las preferencias del usuario en su perfil.

Dentro de este tipo de sistemas recomendadores, existen diversas metodologías que se han aplicado con éxito. Entre estas metodologías encontramos *clasificación Bayesiana* (Yang et al., 2013), *clasificación basada en reglas* o *redes semánticas* (Shishehchi et al., 2010) o *modelos basados en regresión* (Van den Oord et al., 2013). Sin embargo, la técnica más popular en este tipo de sistemas es la *técnica de los vecinos más próximos* (K-NN del inglés *K-Nearest Neighbors*) (Pazzani y Billsus, 2007).

A pesar de que los sistemas recomendadores de filtrado colaborativo son los más utilizados, existen algunos proyectos comerciales que utilizan técnicas basadas en contenido para sus recomendaciones. Un ejemplo es *The Music Gnome Project* (Castelluccio, 2006) donde se han generado 450 atributos musicales a cada una de las canciones de la plataforma Pandora para generar recomendaciones a partir de estos atributos. Otro ejemplo es el proyecto *Enterteinment Genome* de la empresa *Jinni*[1]. Este proyecto etiqueta todos

[1] http://www.jinni.com/

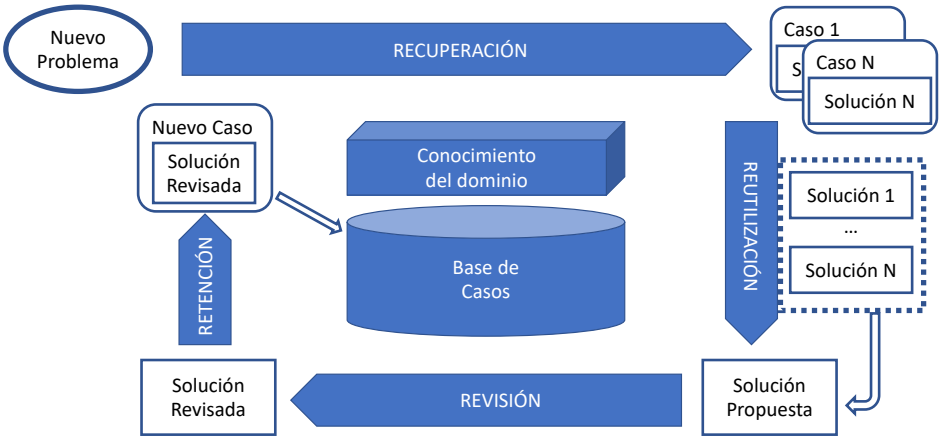

Figura 2.1: Diagrama de los pasos en sistemas CBR (Aamodt y Plaza, 1994).

los videos de entretenimiento (películas, programas de televisión, etc.) con una gran cantidad de atributos para que puedan ser utilizados en sistemas recomendadores basados en contenido. Este proyecto es utilizado por grandes empresas de entretenimiento como *Microsoft Xbox*[2], *VUDU*[3] o *Telefónica*[4] para sus plataformas de televisión o vídeos.

Dentro de los sistemas basados en contenido encontramos una técnica llamada *sistemas recomendadores basados en casos*. Esta técnica es la que se ha utilizado para crear el sistema recomendador que se explica en el Capítulo 3. Estos sistemas se basan en la metodología del *razonamiento basado en casos* (CBR del inglés *Case-Based Reasoning*). Esta metodología consiste en resolver un nuevo problema reutilizando o adaptando soluciones a otro problema similar que ha surgido anteriormente (Riesbeck y Schank, 1989). Esta metodología imita un tipo de razonamiento del ser humano al aplicar soluciones que recuerda para resolver problemas similares que se le presentan en el futuro (Kolodner, 2014). El CBR se suele aplicar en dominios donde es muy difícil crear un modelo o donde no es posible hacer una adquisición de conocimiento (Shiu y Pal, 2004).

Los sistemas CBR tienen un conjunto de soluciones que se han aplicado a diversos problemas de un mismo dominio. En los sistemas CBR se denomina con el término *caso* al elemento que une la descripción del problema y la solución aplicada. A partir de este conjunto de casos, o también llamado base de casos, se aplican cuatro pasos para generar soluciones a nuevos problemas. A estos pasos se les denominan las cuatro Rs (Aamodt y Plaza, 1994), que como se pueden ver en la Figura 2.1 son los siguientes:

[2]https://www.xbox.com/
[3]https://www.vudu.com/
[4]https://www.telefonica.com/

- **Recuperación.** A partir de una consulta de un nuevo problema, el sistema utiliza funciones de similitud para recuperar aquellos casos almacenados que son más similares a la consulta.
- **Reutilización.** A continuación, a partir de las soluciones que contienen todos los casos recuperados, se copian o integran estas soluciones para generar una posible solución al problema de la consulta.
- **Revisión.** El sistema revisa la solución recuperada para determinar si resolverá el problema correctamente. En este paso el sistema puede adaptar la solución al problema de la consulta.
- **Retención.** Esta nueva solución se almacena en la base de casos junto con la descripción del problema en el momento que se ha validado. Esto permitirá resolver problemas similares en el futuro.

Los sistemas de recomendación basados en casos se han aplicado en diversos dominios como son la recomendación de cursos masivos (Bousbahi y Chorfi, 2015), películas (Quijano-Sánchez et al., 2012) o terapias de bienestar (Pheng et al., 2013).

2.1.1.3. Sistemas híbridos

El último tipo de sistemas recomendadores clásicos son aquellos que mezclan dos o más técnicas de recomendación. A estos sistemas se les denomina sistemas de recomendación híbridos. La principal característica es que la combinación de técnicas de recomendación permite solventar los problemas que existen en cada una de las técnicas individualmente (Ricci et al., 2015). En Burke (2002) se hace una clasificación de los métodos híbridos existentes en los sistemas de recomendación. Según esta clasificación existen siete tipos de métodos híbridos:

- **Ponderado.** Se utiliza una técnica de puntuación o votación para determinar la recomendación final a partir de las recomendaciones obtenidas por un conjunto de técnicas.
- **Switching.** El sistema recomendador elige distintas técnicas de recomendación dependiendo de la situación.
- **Mezclados.** Las recomendaciones obtenidas por distintas técnicas son mostradas al usuario al mismo tiempo.
- **Combinación de propiedades.** Las características de las distintas fuentes de información de las recomendaciones son combinadas para generar un algoritmo basado en un modelo con la combinación de características.

- **Cascada.** Las recomendaciones propuestas por una técnica de recomendación son refinadas por otra técnica de recomendación.
- **Aumento de funciones.** Los resultados de una de las técnicas de recomendación son los datos de entrada para otra técnica de recomendación.
- **Meta-niveles.** El modelo entrenado para una técnica de recomendación es utilizado como entrada para otra técnica de recomendación distinta.

Existen muchos ejemplos de métodos híbridos aplicados en sistemas recomendadores. En la Sección 2.2 se exponen algunos ejemplos de estas técnicas aplicados a problemas concretos que se encuentran en sistemas recomendadores.

2.1.2. Sistemas recomendadores contextuales

Todos los sistemas de recomendación vistos en la sección anterior utilizan únicamente la información de los usuarios, los ítems o las interacciones entre ellos para predecir qué ítems les pueden interesar en el futuro. Sin embargo, existe un área de investigación en sistemas recomendadores que incluyen factores externos que pueden hacer cambiar las preferencias del usuario en un momento concreto. A estos sistemas se les denomina *sistema recomendadores contextuales* (CARS del inglés *Context-Aware Recommender Systems*).

Este tipo de sistemas incluyen el contexto en el modelo de recomendación. El contexto tiene múltiples definiciones dependiendo del dominio en el que se utilice. En los sistemas informáticos, la definición más utilizada para definir el contexto es la que se propone en Dey (2001): *El contexto es cualquier información que puede ser utilizada para describir la situación de cualquier entidad. Una entidad es una persona, lugar u objeto que es considerado relevante en la interacción entre un usuario y una aplicación, incluidos el propio usuario y la propia aplicación.* En Zimmermann et al. (2007) se explica qué tipo de información contextual existe y cómo se puede utilizar.

Utilizaremos un ejemplo para comprender mejor cómo funcionan este tipo de sistemas. Imaginemos que tenemos un sistema recomendador contextual de películas. Al usuario objetivo al que se le hará la recomendación le encantan las películas de terror, sin embargo, en un momento concreto quiere ver una película con su familia, compuesta por su pareja y un niño o niña. En este caso, el sistema recomendador debe adaptar la recomendación a la nueva situación, que es verlo en familia. Por este motivo, el sistema recomendador buscará otro tipo de película que, a pesar de que no es la categoría favorita del usuario objetivo, se adapta mejor a esta situación.

Como explica Adomavicius y Tuzhilin (2015), los sistemas recomendadores contextuales se pueden definir como una función R_{ctx} que extiende

la función definida en la Sección 2.1.1. Esta función R_{ctx} utiliza la información del dominio de los usuarios ($Users$), ítems ($Items$) y la información contextual ($Context$) para devolver un conjunto ordenado de ítems ($Recommendations$) según su utilidad para el usuario en un contexto concreto:

$$R_{ctx} : Users \times Items \times Context \rightarrow Recommendations \tag{2.3}$$

Estos sistemas recomendadores se pueden clasificar en tres paradigmas según se ha implementado el uso de la información contextual (Adomavicius y Tuzhilin, 2015):

- **Pre-Filtrado contextual.** Este tipo de sistemas recomendadores filtran del catálogo de ítems a recomendar aquellos que no se adaptan a la información contextual de la consulta. Una vez hecho esto, se puede aplicar un sistema de recomendación clásico, como los explicados anteriormente, para determinar los ítems más relevantes para el usuario objetivo.

- **Post-Filtrado contextual.** Estos sistemas de recomendación invierten los pasos vistos en el paradigma de pre-filtrado contextual. En primer lugar, se utiliza una de las técnicas de recomendación clásicas para ordenar los ítems según el interés que puedan tener para el usuario objetivo. A continuación, se filtran aquellos ítems que tienen condiciones contextuales diferentes a los de la consulta.

- **Modelo Contextual.** El último paradigma agrupa todos los algoritmos que incluyen algún tipo de información contextual dentro del modelo del sistema recomendador junto con la información de los usuarios y la de los ítems. Este tipo de sistemas recomendadores obtienen los ítems más relevantes para el usuario y el contexto de la consulta directamente del modelo.

En Panniello et al. (2009), se hizo un estudio con diversos sistemas de recomendación contextuales para determinar qué paradigma entre pre-filtrado y post-filtrado funciona mejor. En este estudio se determinó que el paradigma de pre-filtrado contextual es el más rápido, debido a que el sistema recomendador solo utiliza un subconjunto de todos los ítems, y que las recomendaciones obtenidas son más relevantes comparados con los otros los algoritmos que utilizan post-filtrado.

Los sistemas recomendadores contextuales se aplican a diversos dominios como películas (Ono et al., 2007; Colombo-Mendoza et al., 2015), música (Baltrunas et al., 2011; Hariri et al., 2012) o herramientas de aprendizaje (Wang y Wu, 2011). Sin embargo, uno de los dominios donde más sistemas

de recomendación contextual se pueden encontrar es el dominio del turismo y ocio. Esto se debe a que las actividades turísticas cuentan con mucha información contextual como horarios, distancia al usuario objetivo, distancia entre actividades, etc. Además, estos sistemas suelen tener como objetivo organizar rutas a usuarios en tiempo real (Borràs et al., 2014), y es necesario incluir la información contextual para mejorar los resultados. Uno de los primeros sistemas recomendadores contextuales que se aplicó al turismo fue COMPASS (Van Setten et al., 2004) que mostraba a los usuarios aquellos lugares que le podrían interesar cerca de su ubicación. Otro ejemplo más actual es Pythia (Drosatos et al., 2015) que recomienda distintos puntos de interés a partir de la información contextual del usuario.

2.1.3. Nuevas tendencias en sistemas recomendadores

Además de todas las técnicas vistas anteriormente, existen otras técnicas de recomendación que se han popularizado en los últimos años. La más popular de estas técnicas, debido a su gran precisión en otros campos de la inteligencia artificial, ha sido el *deep learning* o *aprendizaje profundo*. El deep learning es un método de aprendizaje que utiliza diferentes niveles interconectados para representar un problema y permiten transformar un problema desde los datos en crudo, como por ejemplo una imagen, hasta un nivel de representación más abstracta, la categoría de la imagen dentro de una clasificación (Lecun et al., 2015). En el área de los sistemas recomendadores, el uso de deep learning ha permitido capturar de forma eficiente relaciones no lineales y no triviales entre usuarios e ítems (Mu, 2018). Además, la técnica del deep learning ha permitido resolver de manera eficiente algunos de los problemas en sistemas recomendadores que se trabajan en esta tesis como el problema del cold-start (Wei et al., 2017). Sin embargo, esta técnica tiene algunas limitaciones como la obtención de modelos poco interpretables o la cantidad de datos de entrenamiento necesarios para obtener el modelo final (Zhang et al., 2019).

Otra estrategia en los sistemas recomendadores es el uso de grafos de conocimiento. Un grafo de conocimiento es una representación compuesta por entidades, representadas con nodos del grafo, y las relaciones que tienen estas entidades, representadas con las aristas del nodo (Ji et al., 2020). Dependiendo del problema a resolver, la información representada es diferente. Un ejemplo de representación híbrida lo encontramos en el sistema entity2rec (Palumbo et al., 2017) que utiliza grafos donde las entidades son los usuarios del sistema recomendador, las películas a recomendar y características de las películas (actor, director, etc.) y las relaciones son las interacciones de los usuarios con las películas y las características con las películas. A partir de esta representación, los autores generar medidas de puntuación para generar un ranking de n recomendaciones. Sin embargo, en Caro-Martinez y Jimenez-Diaz (2017) los autores proponen un grafo para representar las interacciones

de un juez en línea de ejercicios de programación. Este grafo relaciona a los usuarios y los ejercicios dependiendo de los ejercicios que los usuarios hayan completado anteriormente y, a partir de esto, recomienda nuevos problemas. Estas recomendaciones las obtienen aplicando métricas de análisis de redes sociales.

A parte de nuevas técnicas, en el área de investigación de los sistemas recomendadores se ha trabajado en sistemas que devuelvan otros resultados. Por ejemplo, un área de investigación muy activa en los últimos años es la de los sistemas recomendadores basados en secuencias. Estos sistemas de recomendación devuelven una lista de ítems para que el usuario los consuma en ese orden (Quadrana et al., 2018). Para entrenar esos sistemas de recomendación se utilizan los registros de los ítems que han consumido en el pasado y teniendo en cuenta en el orden que ha consumido cada uno de esos ítems. Estos tipos de sistemas recomendadores los podemos encontrar en muchos dominios como el comercio electrónico para recomendar el siguiente producto que comprar (He y McAuley, 2017), en el turismo para recomendar el siguiente punto de interés a visitar a partir del recorrido que ya lleva un usuario (He et al., 2016) o para crear listas de reproducción de música (Turrin et al., 2015).

Todas estas nuevas técnicas y nuevos tipos de sistemas recomendadores se deben tener en cuenta para que en un futuro el modelo propuesto en esta tesis doctoral pueda incluir estas y otras nuevas soluciones que aparezcan en el futuro.

2.2. Problemas en sistemas recomendadores

Existen una gran cantidad de problemas que un diseñador debe tener en cuenta a la hora de crear un sistema recomendador. Algunos de estos problemas son la privacidad del usuario, la escalabilidad del sistema, problemas de latencia, etc. (Khusro y Ali, 2006). Sin embargo, en la investigación de esta tesis doctoral nos centramos en tres problemas comunes asociados a los datos de entrada que aparecen en los sistemas recomendadores. Estos problemas son: *dispersión de los datos*, el *problema de la estela* y el *problema del cold-start*.

2.2.1. Dispersión de datos

Debido a la gran cantidad de usuarios e ítems que puede llegar a tener un sistema recomendador, existe un porcentaje muy pequeño de valoraciones con respecto al total de valoraciones potenciales. Debido a esto, los sistemas recomendadores pueden no proporcionar buenas soluciones para algunos usuarios (Shahabi y Chen, 2003). A este problema se le denomina *dispersión de datos* (*sparsity* en inglés). Este problema es muy común en los sistemas

Figura 2.2: Problema de la estela. Representación del número de valoraciones que tiene cada película en el dataset de MovieLens.

de recomendación de filtrado colaborativo basados en memoria.

Existen modelos de sistemas recomendadores para solventar este problema. Algunas *técnicas basadas en reducciones de dimensionalidad* como el *método de descomposición de valores singulares* (SVD) (Sarwar et al., 2000) o *técnicas basadas en factores latentes* como *factorización de matrices no-negativas* (NMF) (Luo et al., 2016) pueden llegar a resolver este problema. Otras soluciones utilizan *sistemas de recomendación híbridos* para mejorar los resultados cuando se encuentran con el problema de la dispersión de datos (Maneeroj y Takasu, 2009; Kim et al., 2017).

2.2.2. Problema de la estela

Muchos de los sistemas recomendadores se centran en técnicas que funcionan correctamente para los ítems que cuentan con un número muy alto de valoraciones. Sin embargo, existe un problema en aquellos ítems que no tienen muchas valoraciones. El *problema de la estela* o *problema de la cola* (en inglés: *long-tail problem*) se refiere a un problema en la distribución de los datos, donde existen unos pocos elementos de los que se tiene mucha información y, en cambio, una gran mayoría de ellos tienen poca información (Park y Tuzhilin, 2008). La Figura 2.2 muestra este problema con el conjunto de datos de MovieLens, un sistema recomendador de películas que contiene una gran cantidad de valoraciones de usuarios reales. En esta figura están ordenadas las películas según el número de valoraciones que han recibido. Como se puede observar, existe un gran número de películas que no tienen

muchos ratings.

En la literatura se pueden encontrar bastantes trabajos donde se han aplicado distintas soluciones al problema de la estela como el uso de *algoritmos de agrupamiento* (Park, 2012), uso de *factorización de matrices* (Jing et al., 2015) o *algoritmos basados en grafos* (Shi, 2013). Como la mayoría de los problemas de sistemas recomendadores, también encontramos soluciones basadas en *sistemas de recomendación híbridos* (Domingues et al., 2013; Alshammari et al., 2017).

2.2.3. Problema del cold-start

El problema del *cold-start* o *arranque en frio* es un problema muy común en los sistemas recomendadores. Este problema ocurre cuando no se tiene la suficiente información de un ítem o de un usuario para poder calcular una recomendación (Park y Chu, 2009). Esto puede ocurrir, por ejemplo, al añadir una nueva película en un sistema recomendador de filtrado colaborativo. Esta película no tiene valoraciones de los usuarios y, por lo tanto, no aparecerá en las recomendaciones a los usuarios. Se puede producir también por parte de los usuarios. En los sistemas recomendadores, cuando existe un nuevo usuario no se tiene información sobre sus preferencias.

Existen numerosos trabajos que intentan resolver este problema. En Zhang y Lee (2010), los autores añaden categorías sociales en las descripciones de los ítems y los usuarios. Esto permite utilizar esta información para generar las recomendaciones cuando no se tiene suficiente información de los usuarios o los ítems. Otros trabajos como Lika et al. (2014) intentan clasificar a los nuevos usuarios en grupos predefinidos para poder hacer las primeras recomendaciones. Aunque una solución muy común es el de utilizar un *sistema recomendador híbrido* para resolver este problema (Popescul et al., 2001; Schein et al., 2002).

De los tres problemas principales que hemos descrito, en esta tesis doctoral nos hemos centrado en proponer soluciones al *problema del cold-start*. Esto es debido a que es uno de los más comunes en sistemas recomendadores y a que este problema también se encuentra en sistemas CBR.

2.3. Frameworks para el desarrollo de sistemas recomendadores

Como hemos visto en las secciones anteriores, existe una gran variedad de técnicas para implementar sistemas recomendadores y esto ha llevado a la creación de un gran número de frameworks o librerías para facilitar el desarrollo de estos sistemas. En la Tabla 2.1 se muestra una lista con los frameworks de código abierto o académicos más relevantes que han aparecido desde 2011. En total se han analizado las características de 21 frameworks

Framework	Lenguaje	Año	FC	BC	AA	IG	DE	EV	DS
Lenskit	Java	2011	X				X	X	
MyMediaLite	C#	2011	X	X			X	X	
Mahout	Java, Scala	2012	X		X		X	X	
PredictionIO	Java, PHP, Python, Ruby	2013			X	X	X	X	X
HapiGer	Node.js	2015	X				X		
LightFM	Python	2015	X	X				X	
LibRec	Java	2015	X	X	X			X	
jCOLIBRI	Java	2015		X		X	X	X	
RankSys	Java	2016	X		X			X	
LIBMF	R	2016	X					X	
RecDB	SQL	2017	X						X
Surprise	Python	2017	X				X	X	
Rexy	Python	2017	X	X				X	
QMF	C++	2017	X					X	
SpotLight	Python	2017			X			X	
CaseRecommender	Pyhton	2017	X		X			X	
proNet-core	C++	2017			X				
Seldon	Java, Python, R, PySpark	2018			X		X		X
Oryx v2	Java	2018	X		X		X		X
Tensorrec	Pyhton	2018			X			X	
recommenderlab	R	2018			X		X	X	

Tabla 2.1: Comparativa de frameworks para crear sistemas recomendadores de código abierto o académicos desde 2011.

diferentes. En primer lugar, se han analizado las técnicas de recomendación que permiten implementar. Estas técnicas pueden ser filtrado colaborativo (**FC**), basadas en contenido (**BC**) o aprendizaje automático (**AA**). A continuación, hemos enumerado aquellos framework que cuentan con una interfaz gráfica (**IG**) para simplificar la tarea de desarrollo a los usuarios con menos conocimientos técnicos. Las siguientes características que hemos analizado en estos frameworks es cuáles cuentan con la posibilidad de desarrollar nuevos algoritmos (**DE**), cuáles tienen métodos de evaluación (**EV**) para los sistemas implementados y, por último, cuáles disponen de herramientas para el despliegue automático (**DS**) de los sistemas construidos.

Como se puede observar en la Tabla 2.1, las técnicas de recomendación que más implementan estos frameworks son las técnicas de filtrado colaborativo y aprendizaje automático. Esto es debido a la popularidad que han tenido ambos métodos, aunque por motivos distintos, según explicaremos a continuación.

En primer lugar, como hemos explicado en la Sección 2.1.1.1 las técnicas de filtrado colaborativo fueron las más utilizadas en sistemas de recomendación desde que Netflix implementó está técnica de recomendación en su sistema después del famoso concurso Netflix Prize (Bennett et al., 2007). Esto demostró la gran precisión de esta técnica a la hora de generar recomendaciones y propició la creación de numerosos frameworks para su desarrollo. Entre todas las librerías que implementan la técnica de filtrado colaborativo, encontramos 2 frameworks que son los más utilizados. El primero de ellos es Lenskit (Ekstrand et al., 2010). Esta librería es muy utilizada a nivel

académico para introducir a los estudiantes en los sistemas recomendadores. También, la aplicación MovieLens (Harper y Konstan, 2015) utiliza esta librería para generar un sistema recomendador de películas. Además, la aplicación MovieLens ha proporcionado numerosos conjuntos de datos que se han utilizado en el área de la investigación de sistemas recomendadores para evaluar las técnicas propuestas. El segundo framework más utilizado para crear sistemas recomendadores de filtrado colaborativo es Mahout (Schelter y Owen, 2012). La principal característica de Mahout es la facilidad que proporciona a la hora de crear un sistema recomendador debido a la gran cantidad de metodologías que tiene implementadas. Por otro lado, un framework destacable en este grupo es RecDB (Sarwat et al., 2017). Su principal diferencia con los demás es que RecDB es un módulo que se instala dentro de la distribución de bases de datos PostgreSQL y permite aplicar las técnicas de filtrado colaborativo directamente a las consultas SQL que se hacen sobre la base de datos.

La segunda técnica que más encontramos en los frameworks analizados es el uso de metodologías basadas en aprendizaje automático. Esto se debe a que la utilización de estas técnicas se ha incrementado debido al aumento de información disponible sobre usuarios e ítems y al abaratamiento de los dispositivos que permiten almacenarla (Landset et al., 2015). Los frameworks que utilizan técnicas de aprendizaje automático se pueden clasificar en 2 grupos: aquellos que utilizan una librería o servicio externo para ejecutar los algoritmos de aprendizaje automático, y aquellos que implementan técnicas de aprendizaje automático dentro del framework. Dentro del primer grupo encontramos los frameworks PredictionIO (Chan et al., 2013), Apache Mahout (Owen et al., 2011), Oryx v2 (Sean, 2018), y Seldon[5] que utilizan la librería Apache Spark (Apache, 2016) para ejecutar los algoritmos de aprendizaje automático. En este mismo grupo encontramos Tensorrec (Kirk, 2018), que utiliza el servicio Tensorflow (Abadi et al., 2016), y SpotLight (Kula, 2017), que utiliza la librería PyTorch (Paszke et al., 2017). Ambos frameworks usan estos servicios para aplicar técnicas de Deep Learning para sus sistemas recomendadores. El resto de frameworks de la Tabla 2.1 tienen incluidos sus propios algoritmos de aprendizaje automático.

La técnica menos utilizada por los frameworks de sistemas de recomendación es la basada en contenido. La mayoría de estos frameworks tiene implementadas este tipo de técnicas como complemento a las técnicas de filtrado colaborativo. Es el caso de Surprise (Hug, 2017), LightFM (Kula, 2015), LibRec (Guo et al., 2015) y MyMediaLite (Gantner et al., 2011). En este análisis solo hemos encontrado un framework orientado exclusivamente a la técnica de recomendación basada en contenido: jCOLIBRI (Recio-García et al., 2014). Aunque este framework fue diseñado para la construcción de sistemas CBR, los autores explican en Recio-García et al. (2008) como uti-

[5]https://www.seldon.io

lizar su framework para crear sistemas recomendadores. Además, como ya explicamos en la Sección 2.1.1.2, el CBR nos permite crear un tipo concreto de sistemas basados en contenido que son los recomendadores basados en casos.

Otra característica que podemos extraer de la mayoría de los frameworks analizados es que están orientados a usuarios que tienen un alto conocimiento en el desarrollo de aplicaciones con distintos lenguajes de programación. Por este motivo solo encontramos 2 frameworks que disponen de una interfaz gráfica para crear sistemas recomendadores. El primero de ellos es PredictionIO (Chan et al., 2013) que permite personalizar un conjunto de plantillas para crear sistemas recomendadores. El otro sistema es jCOLIBRI que cuenta con una aplicación llamada COLIBRI-Studio (Recio-García et al., 2014) que permite crear sistemas CBR usando una interfaz gráfica.

Una de las características más importantes desde el punto de vista de la investigación es que los frameworks permitan implementar nuevas técnicas de recomendación reutilizando los componentes que tienen. En la Tabla 2.1 podemos observar que solo la mitad de los frameworks analizados permiten la implementación de nuevas técnicas de recomendación. En alguno de estos casos, como Tensorrec, esta limitación se debe a que utilizan un servicio externo donde se ejecuta el algoritmo. Otros frameworks, como el ya citado RecDB, están orientados a crear sistemas recomendadores finales. Otra característica interesante cuando se construye un sistema recomendador es la posibilidad de que los frameworks cuenten con herramientas para evaluar los sistemas implementados. Por este motivo casi todo los frameworks cuentan con esta característica.

La última característica analizada en los frameworks de la Tabla 2.1 es determinar si cuentan con herramientas o métodos que permitan desplegar el sistema de recomendación para usarlo en aplicaciones finales. Como se puede observar en la Tabla 2.1, solo 4 frameworks cuentan con esta posibilidad. Esto se debe a la dificultad de generalizar los pasos para poder desplegar cualquier sistema como aplicación web y proporcionar un conjunto de servicios web que permitan utilizar el sistema recomendador en aplicaciones externas.

2.4. Conclusiones del capítulo

En este capítulo, incluimos un análisis que nos permitió conocer en que estado se encuentra la investigación de sistemas recomendadores. El primer paso ha sido enumerar algunas de las técnicas más comunes que se utilizan en estos sistemas. Hemos observado que existe una gran cantidad de técnicas y que cada una de ellas se utilizan en situaciones distintas. Un ejemplo muy claro es que dependiendo de la información de los ítems o los usuarios podremos usar técnicas de filtrado colaborativo o basadas en contenido. Esta revisión del estado del arte nos ha permitido definir las técnicas que podemos

incluir en el modelo propuesto en esta tesis. En esta definición se ha podido definir las condiciones en las que se pueden aplicar las diferentes técnicas a partir de la información que utilizará el sistema recomendador final.

Además de enumerar diversas técnicas de recomendación, hemos encontrado algunos problemas que pueden ocurrir por falta de información en los datos de entrada. Conocer estos problemas y ver cómo se pueden resolver nos permiten enriquecer el modelo, añadiendo dependencias entre las técnicas de recomendación y los problemas detectados en los datos de entrada.

Por último y antes de diseñar la plataforma basada en conocimiento de esta tesis, era necesario conocer las características que encontramos en los frameworks más populares para construir sistemas recomendadores. Este análisis nos ha permitido definir un conjunto de características que debería tener la plataforma propuesta. En el Capítulo 4, explicamos como las diferentes herramientas de la plataforma intentan abarcar todas estas características y, además, aplican el modelo propuesto para ayudar a los desarrolladores en la construcción de los sistemas de recomendación. A pesar de este análisis, no teníamos suficiente información de las fases de desarrollo de un sistema recomendador final. Por este motivo, en el próximo capítulo, describimos la construcción de un sistema recomendador real como caso de estudio para definir todas las fases en el desarrollo de este tipo de sistemas.

Capítulo 3

Estudio de la construcción de un sistema recomendador

Para poder completar el objetivo principal de esta tesis doctoral, ha sido necesario comprender los pasos de desarrollo de un sistema recomendador complejo. Es decir, definir los pasos que debe seguir un desarrollador a la hora de construir un sistema de recomendación. Por este motivo, se definió el objetivo **O-2**:

Objetivo 2 (O-2)
Desarrollar un sistema recomendador que utilice información contextual y que permita probar los conceptos que se añadirán en el modelo teórico.

Con este fin, se creó un sistema de recomendación contextual en el dominio del turismo y ocio llamado *Madrid Live*. Este sistema recomendador propone planes turísticos en Madrid a partir de las preferencias y la información contextual de los usuarios y las actividades en el momento de la recomendación. Por otro lado, se han propuesto distintas soluciones para cumplir con el objetivo **O-3**:

Objetivo 3 (O-3)
Desarrollar distintas soluciones al problema del *cold-start* en sistemas recomendadores.

En este capítulo, detallamos los objetivos y las contribuciones asociadas al sistema recomendador Madrid Live. En primer lugar, la Sección 3.1 describe en detalle los objetivos definidos para Madrid Live. La Sección 3.2 describe el sistema que se ha implementado para Madrid Live y un módulo para reconocer las emociones de los usuarios a partir de sus expresiones

faciales. A continuación, la Sección 3.3 expone las distintas soluciones que se han propuesto para resolver el problema del *cold-start*. Por último, en la Sección 3.4 resumimos algunas de las conclusiones más importantes que hemos obtenido trabajando con Madrid Live.

3.1. Objetivos planteados

Antes de desarrollar una plataforma que permita crear sistemas de recomendación, es necesario comprender los pasos por los que debe pasar el desarrollo de este tipo de sistemas. Para completar esta tarea, se definieron dos objetivos específicos que se cubren en este capítulo.

Para el caso de estudio en sistemas recomendadores, decidimos desarrollar un sistema de recomendación contextual. La principal razón de esta decisión fue que este tipo de sistemas, como se explica en la Sección 2.1.2, pueden implementarse a partir de un sistema recomendador clásico e incluir la información contextual al modelo del sistema recomendador clásico. Por lo tanto, implementar un sistema de recomendación contextual nos permitió conocer el desarrollo de un sistema clásico. Además, el análisis de sistemas recomendadores contextuales nos permite preparar la futura plataforma para sistemas recomendadores más complejos como, por ejemplo, los recomendadores sociales. Una vez decidido el tipo de sistema recomendador, era necesario elegir el dominio al que se iba a aplicar. En este caso se decidió aplicar el sistema recomendador al dominio del turismo. El motivo es que este dominio tiene una gran cantidad de información contextual que puede utilizarse en el sistema recomendador como, la localización del usuario o el clima en un momento concreto, incluso, obtener el catálogo de actividades para recomendar es bastante sencillo gracias a la gran cantidad de recursos abiertos que existen. Además, para este dominio se pueden encontrar un gran número de trabajos a tener como referencia como se puede ver en Borràs et al. (2014).

Dentro de la tesis doctoral se definieron dos objetivos específicos orientados al desarrollo de un sistema recomendador que agrupamos en este capítulo:

- El primer objetivo específico fue desarrollar un sistema recomendador contextual que devuelva planes turísticos a los usuarios a partir de sus preferencias y la información de su entorno. Para completar este objetivo se siguieron estas directrices:

    - En primer lugar, el desarrollo del sistema recomendador tenía que permitir evaluar la formalización del diseño y construcción de un sistema recomendador. Para ello se siguieron los pasos de desarrollo que se han extraído a partir del estudio que se ha hecho

en el Capítulo 2. Esto nos permitió evaluar las etapas por las que tenía que pasar el desarrollo de este tipo de sistemas y nos permitió definir las fases por las que ha pasado la plataforma del objetivo **O-5**.

- En segundo lugar, el desarrollo de este sistema recomendador debía utilizar el diseño basado en componentes. Esto nos permitiría probar componentes y funcionalidades que serían añadidas a la plataforma inteligente. Esto está relacionado con el estudio del Capítulo 2 ya que es necesario comparar estas funcionalidades con las existentes en otros tipos de sistemas recomendadores.

- Otro objetivo específico que se definió es el objetivo **O-3**: encontrar soluciones al problema del *cold-start*. Cómo se explicó en el Capítulo 2, el *cold-start* es el problema más común que se encuentra en los sistemas recomendadores. Para este objetivo, nos hemos centrado en soluciones basadas en la gestión de la base de casos de sistemas CBR. Esta decisión viene determinada por dos motivos. El primero porque este problema también es muy común en sistemas CBR que no tienen un gran número de casos almacenados. El otro motivo es que el sistema Madrid Live utiliza un sistema CBR para buscar planes turísticos y recomendarlos. Este objetivo se ha enfocado de dos maneras:

  - *Soluciones al problema del cold-start orientado al usuario.* Esto es, el sistema no tiene suficiente información del usuario para obtener una buena recomendación. En esta tesis doctoral las soluciones que se proponen se centran en resolver los problemas para reconocer el estado emocional de un usuario cuando no se tiene información de este usuario anteriormente.
  - *Soluciones al problema del cold-start orientado a los ítems.* El sistema no tiene un catálogo inicial de ítems suficientemente grande como para devolver recomendaciones satisfactorias para el usuario. En esta tesis se propone obtener un catálogo inicial de ítems lo suficientemente representativo para la mayoría de las preferencias y restricciones contextuales.

Por último, y aunque no era uno de los objetivos específicos propuestos en esta tesis, este sistema recomendador nos permitió evaluar nuevas funcionalidades como, por ejemplo, usar el estado emocional del usuario para saber si la recomendación propuesta por el sistema es satisfactoria.

Una vez que se conocen los objetivos propuestos para esta línea de trabajo, en la siguiente sección se explica el funcionamiento del sistema recomendador contextual Madrid Live.

Versión	Información Contextual	Tipo de Actividades
1ª	Tiempo Localización Uso de transporte público Presupuesto Clima	Museos Restaurantes Parques Paseos
2ª	Tipo de transporte Duración Coste del plan Social	Puntos de Interés Eventos Temporales Edificios emblemáticos Exposiciones
3ª	Turismo Accesible	Actividades Accesibles

Tabla 3.1: Evolución de la información contextual y el tipo de actividades que se han ido incluyendo en las diferentes versiones de Madrid Live.

3.2. Madrid Live: un sistema recomendador turístico

Madrid Live es una de las contribuciones a nivel de software de esta tesis doctoral. Como se explica al comienzo del capítulo, Madrid Live es un sistema recomendador de planes turísticos en Madrid. Este sistema de recomendación utiliza la información contextual del usuario a la vez que sus preferencias sobre el tipo de actividades que le interesan. Los planes recomendados en Madrid Live se representan como una lista de actividades con un horario. En la Tabla 3.1 se muestra la información contextual y las actividades turísticas incluidas en las distintas versiones de Madrid Live. La primera versión se encuentra en el Capítulo 9 (Jorro-Aragoneses et al., 2017b) y la segunda versión en el Capítulo 12 (Jorro-Aragoneses et al., 2018). Existe una tercera versión que se ha propuesto como una línea de investigación para trabajar en un futuro y que está explicado en el Capítulo 10.

El proceso de recomendación se basa en un algoritmo CBR. La Figura 3.1 muestra cómo funciona el proceso de recomendación en Madrid Live. El módulo CBR almacena los planes que han realizado otros usuarios como *casos* para obtener futuras recomendaciones. Cada uno de estos casos contiene dos elementos: una *descripción* y una *solución*. La descripción incluye un conjunto de atributos que etiquetan los tipos de actividades que tiene el plan y las restricciones contextuales de dicho plan. Por otro lado, la solución es el conjunto de actividades que realizó el usuario con el horario correspondiente.

Para obtener una nueva recomendación, el módulo CBR recibe una consulta que contiene los tipos de actividades preferidos por el usuario y su información contextual. El sistema compara la consulta con las descripciones de cada uno de los casos almacenados en el módulo CBR. La comparación se

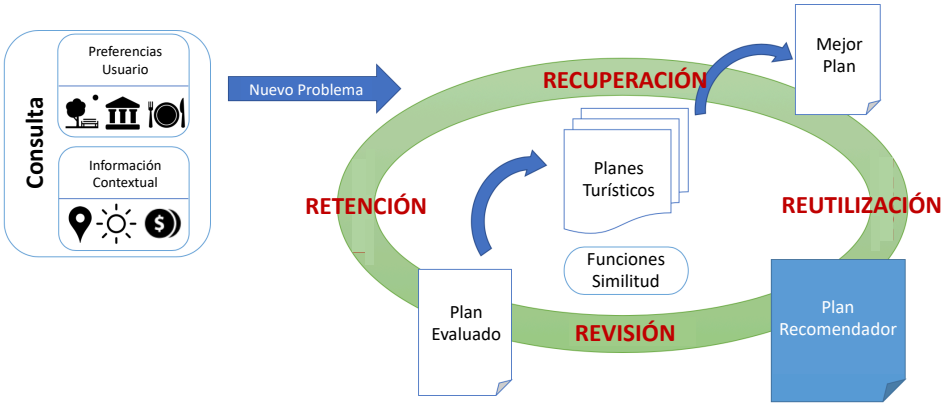

Figura 3.1: Proceso del módulo CBR en el sistema recomendador Madrid Live.

hace a través de una función de similitud que devuelve un valor entre 0 y 1. Cuando la similitud entre la consulta y el caso se aproxima a 1, significa que el caso contiene un plan que satisface las preferencias del usuario y se adapta a la información contextual. En cambio, si la similitud entre la consulta y la descripción de un caso es 0 significa que no tienen ningún atributo en común. Esta función de similitud se compone de un conjunto de funciones que compara cada atributo de la consulta con el correspondiente atributo del caso. A continuación, el módulo CBR recupera el caso con la máxima similitud y que contiene un plan que el usuario no ha realizado anteriormente. La recomendación final se obtiene del plan almacenado como solución de dicho caso. Para conocer más detalles del funcionamiento de Madrid Live y las funciones de similitud que utiliza, se puede consultar el Capítulo 9 (Jorro-Aragoneses et al., 2017b).

Además, en ese capítulo se analiza el problema al incluir la información contextual en las funciones de similitud: los planes recuperados por el módulo CBR tienen menor similitud con respecto a las preferencias del usuario. Como se explica en dicho capítulo, esto se debe a que cuanta más información contextual se añade, más restricciones se imponen al plan que se debe recomendar. Incluso, esta relación no solo depende de la cantidad de información contextual. En los experimentos se observa que existen algunos atributos contextuales, como la localización, que influyen más en las restricciones de los planes que se recuperan. Por este motivo, en ese capítulo se propone un sistema de explicaciones para hacer entender al usuario por qué un plan puede no satisfacer completamente sus preferencias.

En la siguiente sección, se describen distintas soluciones que se han propuesto para resolver un problema muy común en los sistemas CBR y que se ha encontrado en Madrid Live: el problema del *cold-start*.

3.3. Soluciones para el problema del *cold-start* en sistemas CBR

Como se explica en la Sección 3.1, durante el desarrollo de esta tesis se ha investigado el problema del *cold-start* en 2 vertientes: orientado al usuario y orientado a los ítems. En el primero de los casos se hicieron propuestas para resolver el problema de no tener suficiente información para un usuario, por ejemplo, cuando un nuevo usuario empieza a usar este tipo de sistemas. En el segundo caso se trabaja en el supuesto de que no exista un catálogo inicial para hacer las recomendaciones, por ejemplo, no existan suficientes ítems que cubran las distintas preferencias de usuarios. Cada uno de estos problemas y las soluciones propuestas se explican a continuación.

3.3.1. *Cold-Start* orientado a los usuarios

Una dificultad que se puede encontrar en un sistema recomendador es no tener suficiente información del usuario. Por ejemplo, si un usuario no ha realizado un suficiente número de valoraciones, un sistema recomendador de filtrado colaborativo no cuenta con la mínima información para calcular una recomendación.

En esta tesis hemos estudiado el problema a partir de un módulo que se añadió a Madrid Live. Este módulo obtiene las emociones de un usuario a partir del gesto de su rostro. Este módulo se llama PhotoMood y su primera versión se explica en Lopez-de Arenosa et al. (2014). PhotoMood tiene dos fases para detectar esta información: el preprocesado de la imagen y el proceso CBR. En el preprocesado de la imagen se obtiene la posición de ocho gestos de la cara a partir de la imagen. El uso de ocho gestos fue una primera mejora que se incluyó en la publicación de Jorro-Aragoneses et al. (2014). Cada uno de estos gestos se representa con un vector que contiene la posición de los puntos más representativos del gesto. A continuación, esta representación del rostro se envía como consulta al proceso CBR del sistema.

En la Figura 3.2 se puede observar el funcionamiento del proceso CBR. El sistema CBR contiene una base de casos con la representación de imágenes con los gestos del usuario y con una etiqueta de la emoción correspondiente. Este sistema intenta detectar dos emociones básicas: *Me gusta* y *No me gusta*. Usando una función de similitud se recuperaban los casos más similares. Finalmente, se calculaba la emoción del usuario a través de un sistema de votación con la emoción más representada entre los casos recuperados. Este módulo se utilizó en Madrid Live para saber si al usuario le gustan los planes que se le recomendaban.

El objetivo principal que se completó en Jorro-Aragoneses et al. (2014) era minimizar el error a la hora de detectar la emoción del usuario. Para cumplir el objetivo se trabajó en dos aspectos del sistema de reconocimiento

Figura 3.2: Funcionamiento del proceso CBR para detectar las emociones de un usuario a partir de una imagen de su rostro.

de emociones. El primero de ellos trataba de asignar a cada gesto un peso. Este peso se añadió a la función de similitud y cada peso indicaba cómo de importante es ese gesto a la hora de detectar la emoción de un usuario. Además, para cada usuario se realizó un entrenamiento de los pesos para obtener la configuración que minimice el error en la detección de emociones al usuario correspondiente. El segundo aspecto fue mejorar la base de casos que se utilizaba en el sistema. Se trabajó con dos bases de casos: una base de casos personal y una base de casos general. La base de casos personal contenía únicamente representaciones de los gestos del propio usuario obtenidas a partir de imágenes anteriores. La base de casos general contenía representaciones de usuarios anónimos obtenidas de bancos de imágenes públicos. Los resultados de esta publicación mostraron que el sistema detectaba mejor las emociones usando la configuración de pesos personalizada a cada usuario y con su respectiva base de casos. Sin embargo, los resultados obtenidos con la base de casos general y una configuración de pesos global son los suficientemente buenos como para usar esta opción como solución al problema del *cold-start*.

Posteriormente, se hizo una publicación donde se describía totalmente el sistema PhotoMood y, además, se hizo una evaluación comparando PhotoMood con respecto otros sistemas de detección de emociones, como se describe en el Capítulo 11 (Jorro-Aragoneses et al., 2015). En esta evaluación se comparó el error del sistema PhotoMood con respecto otros sistemas usando un dataset específico para la evaluación de este tipo de sistemas. Además, de un dataset específico, fue necesario reconocer más emociones. Los primeros resultados mostraron que el error de reconocimiento de emociones en Pho-

toMood es mayor que el de los sistemas comparados. Este incremento en el error se debía a la confusión de emociones que tenia PhotoMood como, por ejemplo, enfado y disgusto. Sin embargo, el sistema PhotoMood se utiliza en un sistema recomendador y no es necesario reconocer tantas emociones. Por este motivo, se agruparon todas las emociones para que el sistema reconociera tres: *Sorpresa*, *Me gusta* y *No me gusta*. Por ello, se propuso separar las imágenes del dataset en distintas bases de casos. Cada una de estas se clasifica a partir de un conjunto de características. A PhotoMood se le añade un paso previo donde, dependiendo de las características del usuario, se selecciona la correspondiente base de casos. Esta configuración es la que devuelve mejores resultados.

Por lo tanto, a partir de estos resultados se observa que el uso de una base de casos con representaciones anónimas de usuarios permite obtener resultados buenos cuando nos encontramos con el problema del *cold-start*. Incluso, se pueden generar diversas bases de casos a partir de la clasificación de las características de un usuario. En este caso, estas características se podían obtener de las imágenes almacenadas. Sin embargo, esta solución se podría extrapolar a otros tipos de datos como, por ejemplo, a partir de los datos que se obtienen en un test de preferencias. Esta información podría permitirnos clasificar a un usuario y usar una base de casos con usuarios del mismo tipo para generar recomendaciones.

3.3.2. *Cold-Start* orientado a los items

Otro problema en el que se trabajó durante el desarrollo de esta tesis fue el del *cold-start* aplicado al catálogo de items. Este problema aparece cuando no se tienen suficientes ítems para recomendar. También se investigó en el sistema Madrid Live. El objetivo era poder crear un conjunto de planes que sirva como base de casos inicial para el sistema recomendador Madrid Live. Una solución es la que se presenta en el Capítulo 12 (Jorro-Aragoneses et al., 2018) donde se propone una metodología que permite crear esta base de casos inicial. Esta metodología se expone a continuación.

En el caso concreto de Madrid Live, existen dos características importantes que se deben respetar en esta base de casos inicial. La primera de las características es que estén representadas un gran número de restricciones contextuales y cubran un gran número de situaciones. La segunda característica es que estos planes deben tener una lógica en el orden de las actividades. Una de las ventajas de usar planes reales que han hecho otros usuarios es que son validados por esos usuarios. Al usar estos planes se incorpora un conocimiento implícito que no se tiene que modelar, ya que está incluido en esos planes.

Dadas estas restricciones, se propuso una metodología de generación de planes pseudo-aleatorios que, a continuación, son validados por un conjunto de usuarios. En el primer paso se generaron 300 planes turísticos, a los que se

les asignaba unas restricciones contextuales que debían cumplir. Estos planes se generaban añadiendo actividades de forma aleatoria de un catálogo obtenido del Ayuntamiento de Madrid y respetando las restricciones contextuales. Con este paso se tienen suficientes planes para múltiples restricciones contextuales.

El siguiente paso es validar esos planes para que sean lógicos. Para ello, se organizó una evaluación con usuarios en el que se valoraban qué planes eran mejores. Para esta evaluación se utilizó un test Elo (Elo, 1987). En este test se les mostraban a los usuarios 2 planes diferentes, pero con las mismas restricciones contextuales. Entonces el usuario selecciona el plan que más le guste. Con esos votos, se genera el ranking Elo con los 300 planes turísticos. Los planes con las mejores posiciones en el ranking serían planes aceptables para añadir en la base de casos inicial.

A partir de los resultados, se determinó que alrededor del 50 % de los casos generados se podían utilizar en la base de casos inicial. La primera razón de esta conclusión era que esos planes obtuvieron una puntuación positiva en el ranking Elo. Esto significa que estos planes eran más atractivos para los usuarios que el resto de los planes propuestos. Por otro lado, se midió la cobertura y la densidad de la base de casos. Ambas medidas permitieron conocer la probabilidad que existe de encontrar una solución en esa base de casos a nuevos problemas. En ambas medidas se observó que sustituyendo algunas actividades de los planes se podían resolver una gran cantidad de recomendaciones. Este sigue siendo un buen resultado teniendo en cuenta que los planes cuentan con actividades temporales que deben ser sustituidas para nuevas recomendaciones como, por ejemplo, conciertos que no se repiten en la nueva recomendación.

Esta forma de adquirir conocimiento permitió obtener un conjunto de ítems que representen gran parte de las futuras recomendaciones. Además, la utilización del test Elo permitió filtrar aquellos ítems que no deben incluirse en el catálogo por no tener lógica o no ser atractivos.

Una de las conclusiones que obtuvimos de los anteriores resultados es la necesidad de añadir una fase de adaptación en las recomendaciones de Madrid Live. Esta fase de adaptación debía sustituir aquellas actividades que no deben aparecer en las recomendaciones. Esta línea se propuso como trabajo futuro de Madrid Live y la describimos en las Sección 5.2.

3.4. Conclusiones del capítulo

En este capítulo hemos presentado Madrid Live, un sistema recomendador de planes turísticos en Madrid. La principal característica de Madrid Live es que utiliza la información contextual para recomendar planes turísticos a los usuarios en tiempo real. Como hemos explicado en la Sección 3.1, este sistema nos ha permitido comprender las distintas fases de desarrollo de

un sistema recomendador, desde definir los datos de entrada hasta aplicar diferentes soluciones al problema del cold-start.

Gracias a este sistema, hemos podido proponer una metodología de desarrollo, como se explicará en la Sección 4.3.1, para la construcción de sistemas recomendadores y, a partir de este caso de estudio, hemos podido probar diferentes técnicas de recomendación y otros módulos que se han incluido en la versión final del modelo para nuestra plataforma. En el siguiente capítulo, describiremos el resultado final de nuestra tesis a partir del estudio teórico (Capítulo 2) y práctico (Capítulo 3) de los sistemas recomendadores.

Capítulo 4

Plataforma para construir sistemas recomendadores

En los capítulos anteriores se ha descrito los objetivos específicos previos que nos permitían cumplir el objetivo principal de esta tesis doctoral: desarrollar una plataforma basada en conocimientos que guíe a los usuarios en el desarrollo de sistemas recomendadores. En este capítulo nos centraremos en los objetivos específicos enfocados al diseño y desarrollo de esta plataforma. El primero de estos objetivos es el **O-4**:

Objetivo 4 (O-4)
Crear un modelo teórico basado en componentes que describa la construcción de los sistemas recomendadores y plasmarlo en una ontología para utilizarlo en la plataforma del objetivo **O-5**.

La principal aportación de este objetivo fue RecOnto, una ontología que define los componentes que existen en los sistemas recomendadores y como se relacionan entre sí. Este modelo nos permitió completar el objetivo específico **O-5**:

Objetivo 5 (O-5)
Desarrollar una plataforma que ayude en el diseño, implementación y despliegue de los sistemas recomendadores basándose en el modelo obtenido en el objetivo **O-4**.

Como resultado se ha creado RecoLibry Suite, una plataforma que incluye una metodología de desarrollo y un conjunto de herramientas para crear sistemas recomendadores y que está orientado a múltiples perfiles de usuarios.

Este capítulo describe en detalle los objetivos y las contribuciones enfo-

cadas a la construcción de todas las herramientas de la plataforma RecoLibry Suite. En la Sección 4.1 se detallan los objetivos que se plantearon para la construcción de la plataforma. A continuación, en la Sección 4.2, se expone la ontología que describe la construcción de un sistema recomendador y de los componentes que pueden utilizarse. En la Sección 4.3 se describe la metodología de construcción propuesta en RecoLibry Suite y, además, cada una de las herramientas que componen la plataforma RecoLibry Suite: RecoLibry-core, RecoLibry-studio y RecoServer.

4.1. Objetivos planteados

Una vez que hemos estudiado el dominio de los sistemas recomendadores -Capítulo 2- y se ha desarrollado un sistema recomendador complejo como caso de estudio -Capítulo 3- ya se tenía suficiente información en el proceso de diseño y desarrollo de este tipo de sistemas. El siguiente paso en esta tesis doctoral fue plantear los objetivos específicos orientados a la construcción de una plataforma basada en conocimiento para ayudar a los usuarios en la creación de un sistema recomendador. Para ello, se plantearon 2 objetivos específicos. El primero de ellos orientado al proceso de diseño de un sistema recomendador. El segundo orientado al desarrollo de la plataforma de desarrollo que permita instanciar dicho proceso. A continuación, se describen en detalle cada uno de estos objetivos:

- El objetivo específico **O-4** era crear un modelo teórico que describiese la construcción de sistemas recomendadores. Este modelo tenía que describir las reglas que seguiría la plataforma inteligente del objetivo **O-5**. Por esta razón, el modelo teórico resultante debía cumplir las siguientes características:
    - La primera característica del modelo era que tenía que estar basado en componentes. Es decir, el modelo debía dividir el comportamiento total de los sistemas recomendadores en funcionalidades más específicas. Estas funcionalidades se encapsularían en componentes. Cada uno de estos componentes tendrían definidos el tipo de entrada que esperan y la salida que devolverían.
    - La segunda característica era que debería definirse un conjunto de dependencias que regulen cómo se deberían unir los componentes definidos anteriormente. Este conjunto de reglas debería indicar qué componentes son necesarios para construir un sistema recomendador. Por otro lado, el conjunto de reglas también tenía que definir las incompatibilidades que existan entre componentes.
    - Por último, la representación del modelo tenía que poder ser instanciado en la plataforma de desarrollo. Era necesario que se buscase una representación donde la plataforma pudiese consultar

el conjunto de reglas y, así, guiar a los usuarios en el diseño y desarrollo de sistema recomendadores.

- El segundo objetivo específico era el objetivo **O-5**, desarrollar una plataforma inteligente que guie a los usuarios en la construcción de sistemas recomendadores. Esta plataforma debía basarse en la construcción de los sistemas de recomendación en el modelo teórico creado en el objetivo **O-4**. Para esta plataforma también se definieron unas características que se tenían que cumplir:

  Completa La plataforma debía contener la mayoría de los algoritmos que se utilizan actualmente en sistemas recomendadores.

  Extensible La plataforma debía permitir que los usuarios puedan incluir nuevos algoritmos y que estos algoritmos fuesen reutilizables en el modelo.

  Flexible Cada uno de los componentes debía incluir un conjunto de parámetros que permitan ajustar su comportamiento en los distintos algoritmos.

  Adaptado al usuario La plataforma debía ajustarse a los distintos conocimientos técnicos que tendrían los usuarios. Por ejemplo, era necesario hacer herramientas visuales para usuarios con pocos conocimientos de desarrollo.

  Autocontenido Para la construcción de sistemas recomendadores solo se debía utilizar los recursos de la propia plataforma. Los sistemas creados con esta plataforma no tenían que necesitar otros frameworks externos.

En esta tesis doctoral se expone una contribución por cada uno de los resultados obtenidos para cada objetivo:

- A partir del objetivo **O-4** se creó RecOnto: una ontología que definía la construcción de los sistemas recomendadores usando componentes. Esta contribución se explica en la Sección 4.2.
- La plataforma RecoLibry Suite es la contribución al objetivo **O-5**. Esta plataforma está compuesta por 3 herramientas que facilitaban el diseño y desarrollo de sistemas recomendadores. La Sección 4.3 describe la plataforma RecoLibry Suite y las herramientas que lo componen.

4.2. Modelo teórico para construir sistemas recomendadores

Como se explica en Szyperski (2000), el diseño de software basado en componentes reduce el coste de desarrollo de sistemas complejos. Esto se

debe principalmente a dos razones (Heineman y Councill, 2001): primero porque se consigue que los componentes se reutilicen en múltiples sistemas; y segundo porque cada componente se centra en una funcionalidad concreta del sistema y, como consecuencia, facilita su desarrollo. Después del estudio que se ha hecho en la literatura sobre sistemas recomendadores, ver Capítulo 2, se pudo observar que muchos de estos sistemas se podían descomponer en funcionalidades más específicas y podían ser reutilizadas entre distintos sistemas de recomendación. Por ejemplo, la funcionalidad de gestionar los datos del usuario o generar una consulta para el sistema recomendador. Por este motivo, una de las contribuciones en esta tesis doctoral fue crear un modelo teórico que describiese el funcionamiento de los sistemas recomendadores usando componentes y definiendo la relación entre ellos.

El resultado fue RecOnto, una ontología que describía semánticamente los distintos componentes que integran un sistema recomendador y el conjunto de reglas lógicas que definían la relación entre los componentes. RecOnto organiza la terminología encontrada en la literatura sobre sistemas de recomendación y, como consecuencia, nos permitía crear un lenguaje común donde estaban definidos todos los elementos existentes en las diversas técnicas de recomendación. Además, esta terminología era independiente a los múltiples dominios donde se podían aplicar estos sistemas. RecOnto está representado usando el *Lenguaje Web de Ontologías* (OWL-DL del inglés *Ontology Web Language*). La representación en OWL-DL permite que otras aplicaciones externas puedan hacer consultas sobre el modelo representado, cumpliendo la tercera característica que se proponían en el objetivo **O-4**, ver Sección 4.1. El desarrollo de este modelo se ha completado en 2 etapas:

1. En la primera etapa se han identificado todos los componentes que se aplican en las distintas técnicas de recomendación. Para ello, se ha separado todo el funcionamiento de cada una de estas técnicas en un conjunto de funcionalidades atómicas, es decir, funcionalidades que ya no se pudieran descomponer en otras más específicas. Por cada una de estas funcionalidades se ha definido un componente.

2. En la segunda etapa se ha trabajado en describir cómo se construye un sistema recomendador basándonos en la unión de los componentes definidos en la etapa anterior. Para ello, se estudió como se enlazan cada uno de los componentes del modelo con los demás. A partir de este estudio, se definieron un conjunto de dependencias por cada componente. Estas dependencias definían cuales de estos componentes son necesarios en un sistema recomendador, y las compatibilidades o incompatibilidades a la hora de unirlos.

Durante el desarrollo de esta tesis doctoral, RecOnto ha tenido dos versiones. La primera versión se describe en detalle en el Capítulo 13 (Jorro-

Aragoneses et al., 2017a). En esta versión, el modelo representaba los sistemas de recomendación de una forma más teórica. Esto hizo que se utilizaran muchos términos que añadían demasiada complejidad a la hora de clasificar los componentes. Un ejemplo muy claro son las categorías fundamentales de los datos como se ve en dicho capítulo. Para evaluar este sistema, se clasifica un sistema de recomendación contextual en el dominio de la salud usando el modelo descrito en RecOnto. Esta versión nos permitió tener una definición muy concreta de todos los elementos que se encuentran en un sistema recomendador. Sin embargo, en la versión presentada en el Capítulo 15 (Jorro-Aragoneses et al., 2020), se hizo una revisión para simplificar el modelo y ajustarlo para su integración en las herramientas de RecoLibry Suite. Esta segunda versión se centró en el proceso de diseño del sistema recomendador. Por este motivo, se eliminaron algunos términos confusos y se modificó el conjunto de reglas para definir el orden en el que se tienen que elegir los componentes a la hora de crear un sistema recomendador.

Por último, en esta segunda versión se trabajó mucho en las reglas que describían cómo unir los diferentes componentes. Para ello se utilizo *Lógica Descriptiva* (DL del inglés *Description Logics*). Con esta modificación las herramientas de la plataforma RecoLibry Suite podrían realizar consultas al modelo para saber que técnicas de recomendación se pueden utilizar a partir de los componentes ya seleccionados anteriormente. El modelo definido en esta segunda versión fue el que se utilizó en las distintas herramientas de RecoLibry Suite. Actualmente. RecOnto se encuentra en un repositorio público de GitHub[1] para su consulta o extensión. Además, esta ontología se puede visualizar desde la herramienta *WebVOWL* de la página web del grupo GAIA[2].

4.3. RecoLibry Suite

El proceso de construcción de un sistema recomendador está compuesto por un conjunto de tareas complejas en las que se deben tomar decisiones técnicas sobre el diseño y desarrollo. En el diseño de un sistema de recomendación, es necesario que el diseñador tenga un gran conocimiento de este tipo de sistemas, y conozca qué técnicas puede aplicar según la información de la que dispone. Por otro lado, debido a la gran cantidad de frameworks y sus distintas características (ver Capítulo 2) el desarrollador debe adaptar sus conocimientos técnicos al framework seleccionado. Por estas razones, el objetivo principal de esta tesis doctoral (**O-5**) era crear una plataforma basada en conocimiento que ayude a diseñadores y desarrolladores en la construcción de los sistemas de recomendación.

El resultado de este objetivo ha sido RecoLibry Suite, una metodología

[1] https://github.com/UCM-GAIA/RecOnto
[2] http://gaia.fdi.ucm.es/ontologies/#reconto

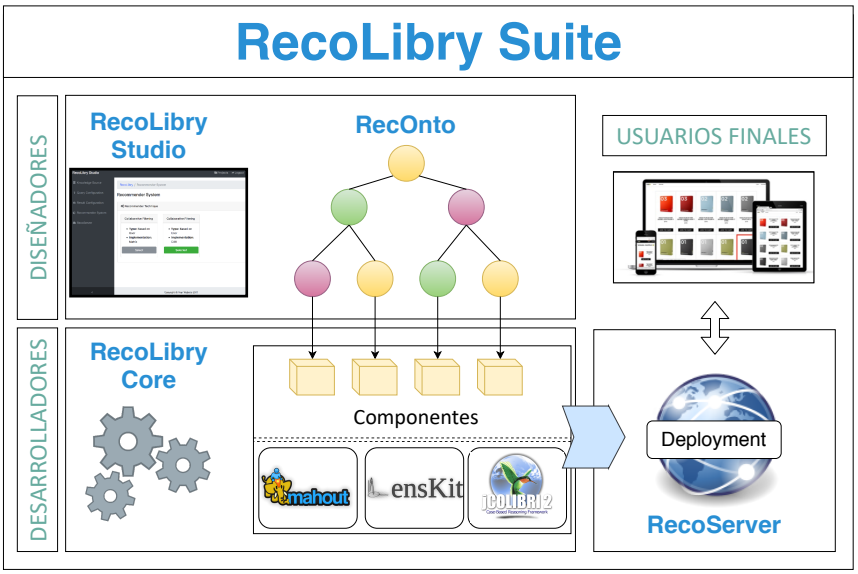

Figura 4.1: Arquitectura de RecoLibry Suite

de desarrollo y un conjunto de herramientas que facilitan el diseño, la implementación y el despliegue de sistemas recomendadores. RecoLibry Suite está compuesto por 3 herramientas como se muestra en la Figura 4.1. La primera de ellas es RecoLibry-core, un framework orientado a objetos en JAVA que implementa los componentes identificados en el modelo teórico y la arquitectura necesaria para combinarlos. La segunda herramienta es RecoLibry-studio, una aplicación web que, a partir de una interfaz visual, ayuda a los usuarios en los pasos necesarios para la creación de un sistema recomendador. La tercera herramienta es RecoServer que se encarga de desplegar en un servidor web los sistemas de recomendación creados con las herramientas anteriores.

A continuación, explicaremos en detalle la metodología de construcción de sistemas recomendadores y cada una de las herramientas que componen RecoLibry Suite.

4.3.1. Metodología para construir sistemas recomendadores

El proceso de desarrollo de sistemas recomendadores con RecoLibry Suite se compone de 4 pasos, como se muestra en la Figura 4.2. Estos pasos son: definir el conocimiento que utilizará el sistema recomendador; definir las entradas y salidas de información; seleccionar el algoritmo de recomendación y desplegarlo para que se pueda utilizar. Estos pasos están descritos en detalle en el Capítulo 15 (Jorro-Aragoneses et al., 2020):

1. **Conocimiento.** El primer paso consiste en definir que información

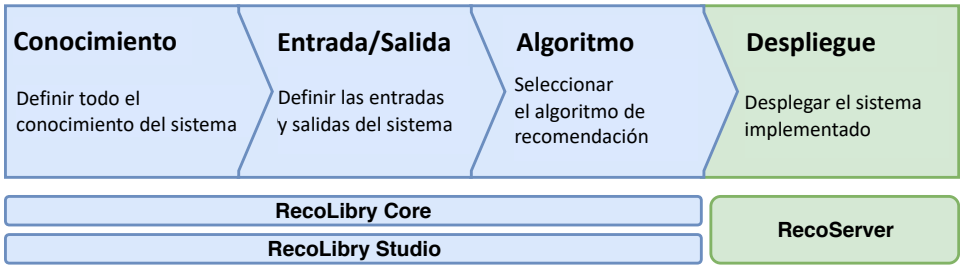

Figura 4.2: Flujo de trabajo en RecoLibry Suite para desarrollar un sistema de recomendación

va a utilizar el sistema recomendación. A este conjunto de datos se le denomina Conocimiento.

2. **Entrada/Salida.** A continuación, es necesario definir que datos de entrada tendremos para cada recomendación y cómo es la salida que espera el usuario.

3. **Algoritmo.** A partir de los pasos 1 y 2, se utiliza el modelo RecOnto para buscar aquellos algoritmos que son compatibles con la información proporcionada.

4. **Despliegue.** El último paso consiste en desplegar el sistema recomendador final para que pueda ser utilizado por los usuarios finales.

Como se puede ver, estos pasos están cubiertos por todas las herramientas que componen RecoLibry Suite. Los tres primeros pasos de la metodología están soportados tanto por RecoLibry-core si el usuario quiere utilizar los componentes programáticamente, como por RecoLibry-studio si prefiere utilizar la interfaz gráfica. Esto dependerá de los conocimientos técnicos del desarrollador. Si el desarrollador es experto en el desarrollo de este tipo de sistemas, puede utilizar RecoLibry-core combinando los componentes que ya existen a partir de frameworks de terceros como Mahout, Lenskit o jCOLIBRI o, incluso, creando nuevos componentes. Por otro lado, si el usuario no es experto en el desarrollo de software, puede utilizar RecoLibry-studio. En este caso, la herramienta le va guiando en los pasos que hemos visto antes para construir su sistema recomendador. RecoLibry-studio utiliza como fuente de conocimiento la ontología RecOnto para comprobar los pasos que va siguiendo el diseñador. Finalmente, RecoServer despliega de forma automática el sistema que se haya creado con las herramientas anteriores. RecoServer se encargará de crear una instancia del sistema de recomendación y generar un conjunto de servicios web que permitan ejecutar las distintas funciones del sistema recomendador.

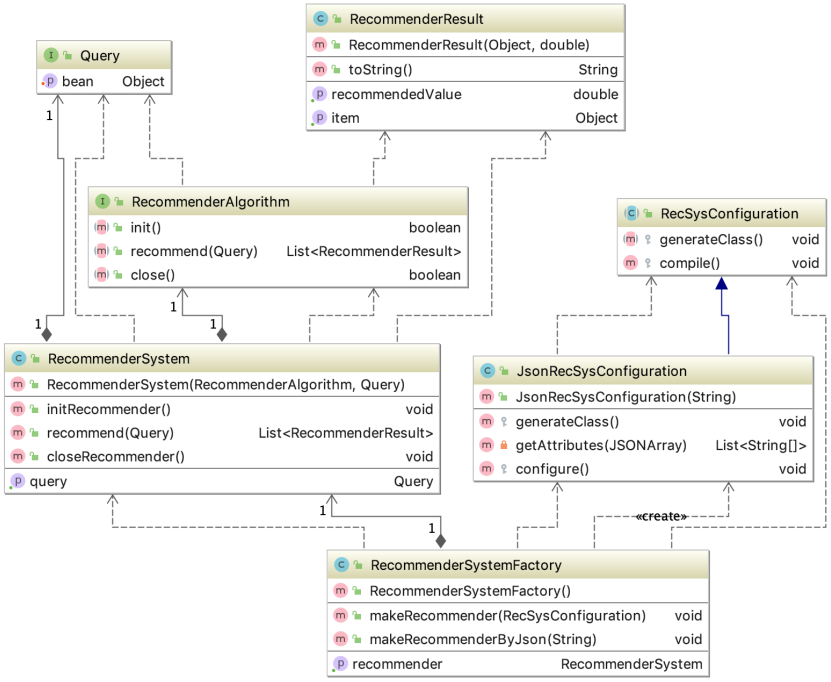

Figura 4.3: Diagrama de clases con la arquitectura principal de RecoLibry-core

4.3.2. RecoLibry-core: framework basado en componentes para la construcción de sistemas recomendadores

La herramienta de más bajo nivel es RecoLibry-core. Este es un framework en JAVA que implementa todos los componentes que se han definido en el modelo descrito en RecOnto. Además, RecoLibry-core se encarga de implementar la arquitectura para poder combinar los componentes y construir los sistemas recomendadores finales.

La arquitectura de RecoLibry-core se centra en implementar las funcionalidades básicas para el correcto funcionamiento de un sistema recomendador. Estas funcionalidades básicas son: inicializar el sistema recomendador, obtener una recomendación a partir de una consulta, cerrar el sistema de forma segura. La Figura 4.3 muestra el diagrama UML con las clases que definen el esquema principal de un sistema recomendador en RecoLibry-core. El componente principal es la clase `RecommenderSystem`. Esta clase define un sistema recomendador se compone de los siguientes elementos:

- `RecommenderAlgorithm`: Esta interfaz define los métodos que son necesarios implementar en un algoritmo de recomendación para poder

integrarlos en RecoLibry-core. Estos métodos son las funcionalidades básicas que deben tener los sistemas recomendadores.

- `Query`: Esta interfaz define la estructura que deben tener las consultas que reciba el sistema recomendador. Esta interfaz gestiona las consultas usando objetos beans que contienen los atributos de las consultas y métodos para acceder y modificar esos atributos.
- `RecommenderResult`: Esta clase almacena los resultados generados por el sistema recomendador. Esta clase contiene dos atributos: el ítem recomendador y un valor de recomendación, que dependiendo del algoritmo este valor será diferente.

A la hora de implementar los distintos componentes se tomó la decisión de encapsular las funcionalidades que existían en otros frameworks, ya que el objetivo no era implementar nuevas técnicas sino reutilizar componentes que ya existían. RecoLibry-core serviría para encapsular los elementos de otros frameworks adaptándolos al modelo que habíamos diseñado. Para esta versión de RecoLibry-core, se adaptaron los elementos de los frameworks Mahout (Schelter y Owen, 2012), con técnicas de recomendación colaborativa, y jCOLIBRI (Recio-García et al., 2008), con técnicas de recomendación basadas en contenido. De cualquier modo, RecoLibry-core permite la implementación de nuevos componentes y sobrescribir componentes ya existentes.

Otra característica que se quería incluir en RecoLibry-core es que los sistemas de recomendación se pudiesen construir a partir de un fichero de configuración. Es decir, un fichero de configuración en el que se especificara qué componentes y configuración tendría el sistema recomendador a construir. Esto permitiría crear los sistemas recomendadores usando una interfaz gráfica. Por este motivo, se aplicó el patrón de *inyección de dependencias* (Prasanna, 2009) en RecoLibry-core. El patrón de inyección de dependencias es un patrón basado en objetos que cumple dos características importantes para nuestro desarrollo. En primer lugar, se define un esquema de cómo se tienen que unir los diferentes objetos de un sistema. En nuestro caso, nos permitía definir una plantilla común para todos los sistemas recomendadores. En segundo lugar, este patrón cuenta con una clase proveedora. llamada `RecommenderSystemFactory` que se encarga de generar los objetos que necesite el sistema y ensamblar todo para ejecutar una instancia del sistema final. Para aplicar este patrón se utilizó la librería de *Google Guice* (Vanbrabant, 2008). Esta librería incluye un conjunto de anotaciones al lenguaje JAVA que permite definir cómo combinar los objetos de distintas clases. A partir de este patrón se implementaron las dos formas que existen para crear un sistema recomendador: implementando el código necesario, usando la clase `RecSysConfiguration`, o a partir de un fichero de configuración ejecutando los métodos de la clase `JsonRecSysConfiguration`.

RecoLibry-core está accesible para su uso bajo licencia GPL versión 3.0. El código es accesible a través de un repositorio público en GitHub[3] que, además, contiene un manual de usuario donde se describe cómo construir un sistema recomendador. Por otro lado, existe otro repositorio[4] que contiene un conjunto de ejemplos de sistemas recomendadores creados con RecoLibry-core. Cada uno de los ejemplos está construido usando un fichero de configuración y un proyecto en JAVA. Esta herramienta se describe con más detalle en el Capítulo 14 (Jorro-Aragoneses et al., 2019) y el Capítulo 15 (Jorro-Aragoneses et al., 2020).

4.3.3. RecoLibry-studio: herramienta para diseñar sistemas recomendadores

La segunda herramienta de RecoLibry Suite está orientada a usuarios que no tienen un nivel alto en el desarrollo de software. Esta herramienta es RecoLibry-studio, una aplicación web que permite el diseño de un sistema recomendador a través de una interfaz gráfica. RecoLibry-studio cuenta con un asistente inteligente, basado en las consultas que se realizan a la ontología, que guía al diseñador a la hora de elegir el algoritmo de recomendación. Además, esta herramienta comprueba si los componentes que ha seleccionado el diseñador pueden combinarse. Para guiar al diseñador, RecoLibry-studio realiza distintas consultas a la ontología RecOnto para comprobar que se respeta el modelo propuesto.

Como se explica en detalle en la Sección 4.3.1 y en el Capítulo 15 (Jorro-Aragoneses et al., 2020), RecoLibry-studio divide el diseño del sistema de recomendación en 4 pasos. Estos pasos se basan en la metodología de desarrollo propuesta en RecoLibry Suite que en el caso de RecoLibry-studio se implementa:

1. *Definiendo la fuente de conocimiento.* El primer paso es indicar a RecoLibry-studio la información que va a utilizar el sistema recomendador. Para ello, el diseñador puede agregar un conjunto de datos donde estén representados los ítems, los usuarios y/o las valoraciones.

2. *Diseñando la consulta.* A continuación, el diseñador seleccionará la información que se introducirá a la consulta que se haga al sistema recomendador. Esta información se basará en los atributos que se han añadido en el paso anterior.

3. *Diseñando la salida.* El siguiente paso del diseñador es indicar qué tipo de salida quiere del sistema recomendador. RecoLibry-studio

[3] https://github.com/UCM-GAIA/RecoLibry-Core
[4] https://github.com/UCM-GAIA/RecoLibry-Examples

permite obtener distintos tipos de salida como una lista de ítems recomendados o la estimación de valoración que un usuario dará a un ítem.

4. *Seleccionando el algoritmo.* A partir de la información que ha introducido el diseñador en los pasos anteriores, RecoLibry-studio lanza una consulta a la ontología para obtener una lista de los algoritmos que se pueden utilizar. Esta lista se le mostrará al diseñador para que seleccione el algoritmo final que quiere utilizar.

Una vez que el diseñador ha completado todos los pasos, RecoLibry-studio permite descargarse el diseño en un fichero de configuración. Con este fichero, el diseñador puede ejecutar localmente el sistema recomendador usando el framework RecoLibry-core. Otra opción que da RecoLibry-studio es que el sistema recomendador se despliegue automáticamente en RecoServer, como se explica a continuación.

4.3.4. RecoServer: despliegue automático de sistemas recomendadores

La última herramienta en RecoLibry Suite es RecoServer. Cómo se explica en el Capítulo 15 (Jorro-Aragoneses et al., 2020), el principal objetivo de esta herramienta era automatizar el despliegue de sistemas recomendadores creados por RecoLibry-core o RecoLibry-studio. Esta herramienta se encarga de dos tareas a la hora de generar un sistema recomendador:

- Crear una aplicación de ejemplo donde se pueda ejecutar el sistema recomendador a través de una interfaz visual sencilla. Esta aplicación permite a los usuarios probar el sistema que han diseñado y comprobar posibles errores en la selección del algoritmo.
- Crear una interfaz que permita a aplicaciones externas poder usar cada una de las funcionalidades del sistema recomendador. Para crear esta interfaz, RecoServer activa un conjunto de servicios web donde cada uno de ellos pertenece a una funcionalidad concreta. Para crear esta interfaz se utiliza el estándar *REST* (Masse, 2011) (del inglés Representational State Transfer – Transferencia de Estado Representacional) que define el protocolo web que hay que seguir para cada uno de los servicios web.

El despliegue de cada sistema recomendador se hace a través de la tecnología Docker (Merkel, 2014). Los *Docker* o *contenedores* son una versión ligera de una máquina virtual. Cada uno de estos contenedores contiene únicamente los servicios del sistema operativo que va a necesitar la aplicación desplegada. Los sistemas recomendadores que se despliegan se encuentran en su propio

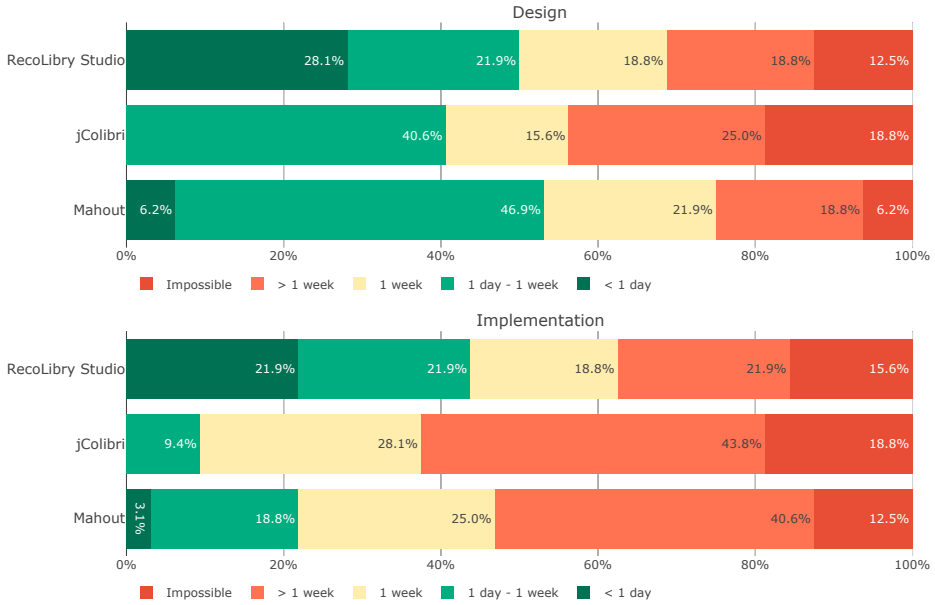

Figura 4.4: Comparación del tiempo estimado de diseño (arriba) y de implementación (abajo) de un sistema recomendador con cada framework.

contenedor. Una vez que se ejecutan, RecoServer genera una URL única para que los usuarios puedan acceder a la API y la aplicación web del sistema recomendador que han creado. Los detalles de esta herramienta se pueden ver en el Capítulo 15 (Jorro-Aragoneses et al., 2020).

4.4. Evaluación de RecoLibry-studio

Esta herramienta fue evaluada con usuarios reales para comprobar su facilidad de uso con respecto a otras herramientas. En el Capítulo 15 se explica el experimento que se realizó. En este experimento participaron 32 usuarios con un nivel básico en el desarrollo de aplicaciones. A estos usuarios se les explicó cómo desarrollar un sistema recomendador de películas con 3 frameworks diferentes: Mahout, jCOLIBRI y RecoLibry-studio. Después realizaron un cuestionario en el que debían indicar cuánto creen que tardarían en crear un sistema recomendador con cada uno de los frameworks y cuál de ellos habían comprendido mejor.

Uno de los objetivos que se planteó para RecoLibry-studio es incrementar la eficiencia a la hora de implementar y desarrollar sistemas recomendadores. Basándonos en las respuestas que se muestran en la Figura 4.4, podemos observar que según la percepción de los usuarios RecoLibry-studio podría reducir el tiempo de diseño y desarrollo de sistemas recomendadores. En el diseño de sistemas recomendadores, el 28,1 % de los usuarios

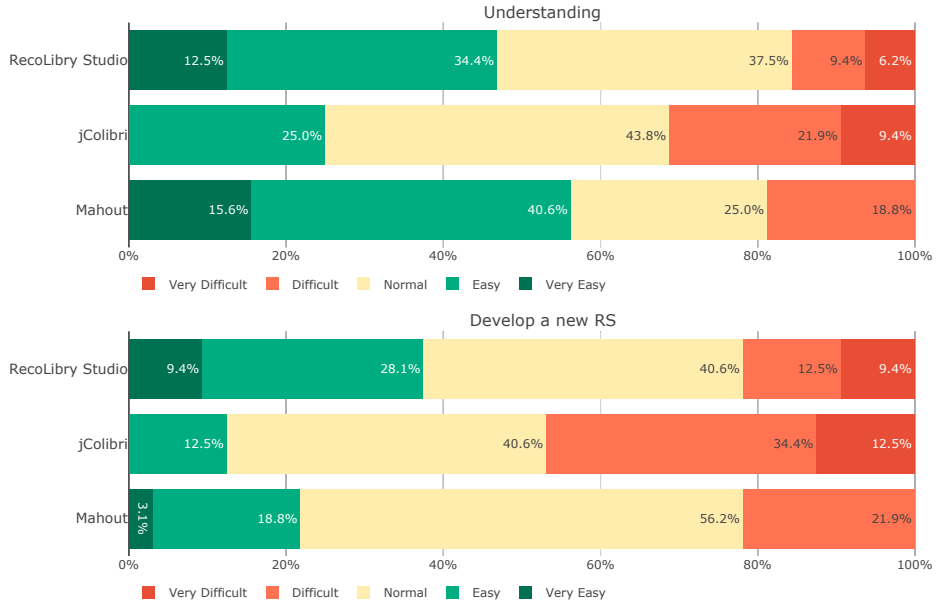

Figura 4.5: Comparación de la dificultad de comprender (arriba) y de crear (abajo) un sistema recomendador con cada framework.

consideran que tardarían menos de un día y, sin embargo, solo el 6.2 % de los usuarios consideran que tardarían menos de un día con Mahout. Estos resultados también se reflejan en la implementación de los sistemas recomendadores, donde el 21,9 % de los usuarios consideran que tardarían menos de un día, mientras que con Mahout este porcentaje baja hasta el 3,1 %. Aunque estos resultados son estimaciones de los usuarios, podemos extraer que RecoLibry-studio puede incrementar la eficiencia a la hora de crear un sistema recomendador.

Otro de los objetivos que se pretendía conseguir con RecoLibry-studio es que los usuarios comprendan mejor cómo se construye y funciona un sistema recomendador. Según los resultados que se muestran en la Figura 4.5, los usuarios consideran a Mahout el framework más sencillo para entender cómo funciona un sistema de recomendación (56,6 %), seguido de RecoLibry-studio con un 46,9 % y, en tercer lugar, jColibri con un 25 %. Este resultado puede deberse a que estamos comparando un framework (Mahout), donde los usuarios ven cómo se programa un recomendador, con una aplicación con interfaz gráfica (RecoLibry-studio) donde se encapsula la construcción del sistema y pueden no comprenderlo correctamente. Por otro lado, a los usuarios se les preguntó la dificultad de crear otro sistema recomendador con cada uno de estos frameworks. En este caso, RecoLibry-studio es el que mayor porcentaje de usuarios consideran que es fácil o muy fácil (37,5 %), seguido de Mahout (21,9 %) y de jCOLIBRI a 12,5 %. Este resultado confirma

que según la percepción de los usuarios, RecoLibry-studio es la herramienta más sencilla para construir un sistema recomendador comparada con los otros frameworks mostrados.

Todos estos resultados mostraron que RecoLibry-studio facilita el proceso de diseño de los sistemas recomendadores. Sin embargo, esta evaluación basada en la opinión subjetiva de los usuarios incluye algunos detalles que debemos analizar para mejorarla en un futuro.

4.4.1. Análisis de la evaluación

El principal objetivo a la hora de diseñar RecoLibry-Suite fue crear un conjunto de herramientas para facilitar la construcción de sistemas recomendadores. Estas herramientas usan un sistema semi-automático de construcción basado en componentes siguiendo las dependencias definidas en la ontología RecOnto. Además, estas herramientas deben facilitar a los usuarios la comprensión de la construcción de estos sistemas.

Existen muchas otras formas para evaluar nuestras herramientas y compararlas con otras opciones. Una primera evaluación puede consistir en recopilar métricas basadas en tiempo. Por ejemplo, el tiempo que tarda un usuario en desarrollar un sistema recomendador con cada uno de los frameworks. Sin embargo, el uso de estas métricas podría no ser la mejor opción debido a que una solución rápida puede llevar a una solución ineficaz en tareas como la programación (Frøkjær et al., 2000). Por otra parte, esta evaluación tiene otro inconveniente. Este tipo de evaluación requiere un conjunto de sujetos con conocimiento técnico en el desarrollo de sistemas informáticos. Además, este conjunto de sujetos debería tener un nivel de conocimiento similar para que el experimento fuese válido. Otro inconveniente en una evaluación basada en el desarrollo de sistemas recomendadores reales es que los usuarios no pueden completar los desarrollos con cada uno de los frameworks en un solo día. Esto hace que existan factores como el cansancio o la motivación que desvirtúe los resultados y la comparación entre cada framework.

En nuestro caso, para salvar estas dificultades, hemos diseñado un experimento basado en explicar a los sujetos las diferentes formas que existen a la hora de implementar un sistema recomendador con cada uno de los frameworks. Es cierto que esta evaluación tiene algunas amenazas que deben ser resueltas en un futuro experimento. Sin embargo, este estudio nos ha permitido realizarlo a un gran número de usuarios con diferentes niveles de conocimiento y que nos han dado unos resultados válidos para poder comparar nuestro sistema con los otros frameworks.

4.5. Conclusiones del capítulo

El objetivo final de esta tesis era crear una plataforma que permitiese a usuarios con distintos perfiles crear sistemas recomendadores. En este capítulo, hemos descrito la plataforma RecoLibry Suite. Esta plataforma se basa en un modelo basado en componentes llamado RecOnto y una metodología que definen cómo se deben construir los sistemas recomendadores.

A partir de este modelo y la metodología, se han creado tres herramientas para automatizar algunas tareas en el desarrollo de sistemas recomendadores. La primera de estas herramientas es RecoLibry-core, un framework en JAVA que implementa los diferentes componentes descritos en el modelo e implementa la arquitectura para combinarlos. La segunda herramienta es RecoLibry-studio, una aplicación web que permite al usuario crear los sistemas recomendadores usando una interfaz gráfica. Además, RecoLibry-studio asiste a los usuarios a la hora de elegir los componentes mostrando únicamente aquellos componentes que son compatibles con los que se han seleccionado previamente. La última herramienta es RecoServer, un servidor que despliega automáticamente todos los sistemas recomendadores creados en la plataforma RecoLibry Suite.

Para evaluar la plataforma, se realizó una sesión con usuarios para valorar la facilidad de uso de RecoLibry-studio a la hora de crear un sistema recomendador. En esta evaluación vemos que, aunque las conclusiones no son definitivas, RecoLibry-studio puede facilitar la creación de sistemas recomendadores finales. Sin embargo, esta investigación nos ha abierto nuevas líneas de investigación que podemos explorar en el futuro para mejorar la plataforma. En el próximo capítulo, enumeraremos todas las conclusiones que hemos obtenido durante nuestra investigación y describiremos nuevas líneas con las que continuar trabajando en el futuro.

Capítulo 5

Conclusiones y trabajo futuro

El objetivo general de esta tesis doctoral ha sido crear una plataforma basada en conocimiento que ayudase a usuarios con distintos perfiles a construir sistemas recomendadores complejos. Para llegar a este objetivo, se definieron unos objetivos previos que permitieron comprender cómo funcionan los sistemas de recomendación y los pasos necesarios para su construcción. Este capítulo enumera todos los resultados obtenidos por cada uno de los objetivos que se definieron al comienzo de esta tesis doctoral. Estos resultados están respaldados por los artículos presentados a la comunidad científica y que están recopilados en la Parte III.

En este capítulo, enumeramos las diferentes contribuciones que hemos conseguido y realizamos un análisis de las conclusiones obtenidas en nuestra investigación. En la Sección 5.1 se hace un resumen de los objetivos planteados y las aportaciones realizadas para cada uno de ellos. A continuación, la Sección 5.2 propone algunas líneas de investigación que podrían surgir a partir de nuestro trabajo. Por último, la Sección 5.3 expone las conclusiones finales de esta tesis doctoral.

5.1. Resumen de aportaciones

En esta sección se enumeran todos los objetivos definidos en el planteamiento del trabajo (ver Capítulo 1) que se han cumplido en esta tesis doctoral. Además, se explican brevemente las aportaciones asociadas a cada uno de ellos. La lista con los objetivos cumplidos es la que se muestra a continuación:

- **Objetivo 1 (O-1): Comprender la construcción de los sistemas recomendadores analizando las técnicas utilizadas, los problemas que se puedan encontrar y los frameworks disponibles.** En el Capítulo 2 hemos estudiado las distintas técnicas de recomendación, así como los problemas más importantes que se pueden encontrar

en estos sistemas. Además, en la Sección 2.3 y el Capítulo 15 (Jorro-Aragoneses et al., 2020) se hace un estudio detallado de los frameworks existentes para la construcción de sistemas recomendadores.

- **Aportación 1: Estudio de sistemas recomendadores clásicos y sistemas recomendadores contextuales incluyendo las distintas técnicas de recomendación que utilizan.** En el Capítulo 2, hemos analizado los sistemas recomendadores más utilizados. En este análisis hemos estudiado las diferentes técnicas de recomendación clásicas que se han utilizado hasta el momento y, además, se ha realizado una revisión de las técnicas de recomendación que utilizan información contextual.
- **Aportación 2: Clasificación y análisis de los principales problemas que se encuentran en la implementación de los sistemas recomendadores incluyendo soluciones encontradas en la literatura.** En el Capítulo 2 se hace un estudio de los problemas más comunes que se pueden encontrar en el desarrollo de los sistemas recomendadores: dispersión de datos, el problema de la estela y el problema del *cold-start*. Además, para cada uno de los problemas, se describen las soluciones que se han encontrado en la literatura.
- **Aportación 3: Estudio de los diferentes frameworks para crear sistemas recomendadores clasificando las características que tienen y definir las características para la plataforma resultante del objetivo O-5.** En el Capítulo 15 (Jorro-Aragoneses et al., 2020) se ha realizado un estudio detallado de todos los frameworks de código abierto o académicos creados desde 2011. Por cada uno de ellos se han analizado qué técnicas de recomendación implementan, si existe una interfaz gráfica para el usuario, si se pueden implementar nuevas técnicas de recomendación, si tienen elementos para la evaluación de los sistemas construidos y si permiten el despliegue automático de los sistemas recomendadores. A partir de este estudio obtendremos las características que debe cumplir la plataforma RecoLibry Suite.

- **Objetivo 2: Desarrollar un sistema recomendador que utilice información contextual y que permita probar los componentes que se añadirán al modelo teórico a implementar en O-4.** En el Capítulo 3 se describe el sistema recomendador contextual que se ha desarrollado como caso de estudio.

  - **Aportación 4: Desarrollo de un sistema recomendador de planes turísticos y de ocio en Madrid que utiliza información contextual.** En el Capítulo 3 se describe Madrid Live,

un sistema recomendador contextual de planes turísticos y ocio en Madrid. Esta aplicación tiene en cuenta las preferencias de los usuarios y la información contextual en el momento de hacer la recomendación. Esta aportación se encuentra detallada en el Capítulo 9 (Jorro-Aragoneses et al., 2017a).

- **Objetivo 3 (O-3): Desarrollar distintas soluciones para el problema del *cold-start* en sistemas recomendadores.** En el Capítulo 3 se describen soluciones propuestas para resolver el problema del *cold-start* que, como se explica en el Capítulo 2, es el problema más común en los sistemas recomendadores.

  - **Aportación 5: Propuesta de un método CBR para resolver el problema del *cold-start* orientado a los usuarios en sistemas recomendadores.** En el Capítulo 3 se propone una solución al problema del *cold-start* orientado a la falta de información de un usuario. En este caso, se propone una metodología para construir distintas bases de casos que permitan inferir la información que falta de un usuario a través de otros usuarios anónimos. En el Capítulo 11 (Jorro-Aragoneses et al., 2015) se detalla está aportación.
  - **Aportación 6: Propuesta de una metodología que permite crear una base de casos inicial para resolver el problema del *cold-start* orientado a los ítems.** En el Capítulo 3 se describe una metodología para construir una base de casos inicial para el sistema Madrid Live. Esta metodología utiliza la opinión de los usuarios para crear un conjunto de casos iniciales que resuelvan el problema del *cold-start* orientado a los ítems. El Capítulo 12 (Jorro-Aragoneses et al., 2018) describe esta metodología.

- **Objetivo 4 (O-4): Crear un modelo teórico basado en componentes que describa la construcción de los sistemas recomendadores y plasmarlo en una ontología.** Después del estudio de los sistemas recomendadores en O-1 y de desarrollar un caso de estudio en O-2, en el Capítulo 4 presentamos el modelo teórico que describe cómo se construyen los sistemas recomendadores.

  - **Aportación 7: Propuesta de una ontología que defina la construcción de sistemas recomendadores a partir de componentes**. En el Capítulo 4 presentamos RecOnto, una ontología para modelar la construcción de sistemas recomendadores a partir de componentes. Esta ontología define la funcionalidad de cada uno de los componentes y un conjunto de reglas para combinarlos. Esta aportación se presenta en los Capítulos 13 (Jorro-Aragoneses et al., 2017a) y 15 (Jorro-Aragoneses et al., 2020).

Además, esta ontología se encuentra en un repositorio público para su uso[1].

- **Objetivo 5 (O-5): Desarrollar una plataforma que ayude en el diseño, implementación y despliegue de los sistemas recomendadores basándose en el modelo teórico propuesto.** Este era el objetivo principal que se definió para esta tesis doctoral. En el Capítulo 4 se describe RecoLibry Suite, una metodología de desarrollo y un conjunto de herramientas para crear sistemas recomendadores. Esta plataforma cuenta con 3 herramientas desarrolladas para cubrir los pasos descritos en la metodología de desarrollo.

  - **Aportación 8: Creación de un framework para construir sistemas recomendadores a partir de los componentes definidos en el modelo teórico.** En el Capítulo 4 se describe la primera herramienta: RecoLibry-core. Esta herramienta es un framework en JAVA que implementa los componentes definidos en el modelo teórico y, además, implementa la arquitectura para poder combinarlos. Los Capítulos que describen en detalle este framework son 14 (Jorro-Aragoneses et al., 2019) y 15 (Jorro-Aragoneses et al., 2020). Además, este framework se encuentra disponible para su uso con un manual de usuario y un conjunto de sistemas recomendadores como ejemplo[2].
  - **Aportación 9: Desarrollo de una interfaz gráfica que permita diseñar y crear un sistema recomendador de forma semi-automática.** La segunda herramienta de RecoLibry Suite es RecoLibry-studio. Esta herramienta es una aplicación web que asiste a los usuarios en la construcción de los sistemas recomendadores. La herramienta está orientada a usuarios menos expertos en el desarrollo de software. RecoLibry-studio está descrito en el Capítulo 4. Esta aportación se describe en el Capítulo 15 (Jorro-Aragoneses et al., 2020).
  - **Aportación 10: Desarrollo de un mecanismo de despliegue mediante servicios web para ejecutar los sistemas recomendadores de forma automática a partir de un fichero de configuración.** La última herramienta es RecoServer. Esta herramienta es un servidor que despliega automáticamente los sistemas recomendadores creados con RecoLibry-core o RecoLibry-studio. Además, RecoServer crea una API para que el sistema recomendador pueda ser utilizado por aplicaciones externas. El Capítulo 4 describe el funcionamiento de esta herra-

[1] https://github.com/UCM-GAIA/RecOnto
[2] https://github.com/UCM-GAIA/RecoLibry-Core

mienta. El Capítulo 15 (Jorro-Aragoneses et al., 2020) describe el funcionamiento de RecoServer.

Aunque se han completado todos los objetivos propuestos en esta tesis doctoral, nuestra investigación deja algunas líneas para futuras investigaciones. A continuación, presentamos estas líneas.

5.2. Trabajo futuro

Después de finalizar esta tesis doctoral, proponemos diferentes líneas de investigación futuras que pueden mejorar al sistema recomendador Madrid Live y a la plataforma RecoLibry Suite.

5.2.1. Adaptación del conocimiento al turismo accesible

Usar un sistema CBR para recomendar planes turísticos permitió obtener un conocimiento implícito debido al uso de casos reales. Este conocimiento no se puede modelar de una forma sencilla, por ejemplo, en qué orden tienen que estar las actividades en un plan turístico. Por este motivo, pueden surgir complicaciones cuando se quiere añadir nueva información en los planes. Una opción sería volver a hacer una base de casos inicial, aunque esto conllevaría un alto coste y no se aprovecharía el conocimiento obtenido anteriormente. Otra opción, es añadir un proceso de adaptación para modificar el plan recomendado y que incluya la nueva información.

Una posible línea de trabajo futuro y que se plantea en el Capítulo 10 (Jorro-Aragoneses y Bautista-Blasco, 2018) es añadir una etapa de adaptación de los planes que se recomiendan en Madrid Live. Se intenta generar planes accesibles para personas con necesidades especiales sin tener que crear una base de casos inicial desde cero. El objetivo sería aprovechar el conocimiento implícito que ya se tiene en Madrid Live y modificar aquellas actividades que no se consideren accesibles.

La Figura 5.1 muestra cómo es la fase de adaptación propuesta en el Capítulo 10 (Jorro-Aragoneses y Bautista-Blasco, 2018). Esta fase comienza recibiendo el plan turístico devuelto por el módulo CBR de Madrid Live. El primer paso de esta fase de adaptación es detectar las actividades no accesibles en el plan y eliminarlas. A continuación, se descarga en tiempo real un catálogo de actividades turísticas accesibles del Ayuntamiento de Madrid. Una vez descargado el catálogo, se usa la información contextual de la actividad eliminada para filtrar aquellas actividades del catálogo que no cumplen con dichas restricciones. Las actividades restantes se clasifican en una taxonomía de tipos de actividades. Con esa taxonomía se busca una actividad accesible cuyo tipo sea lo más parecido a la actividad eliminada. Por último, se añade esta nueva actividad al plan en la posición de la actividad eliminada.

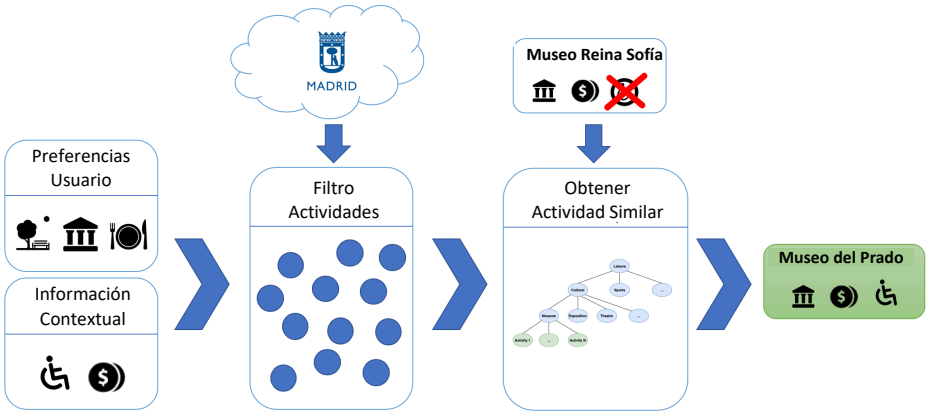

Figura 5.1: Proceso de adaptación propuesto para incluir actividades accesibles en los planes turísticos devueltos en Madrid Live.

5.2.2. Capacidad de interpretabilidad en RecoLibry-studio

Como hemos explicado en el Capítulo 4, la herramienta RecoLibry-studio está orientada a usuarios que no tienen un conocimiento muy profundo en el desarrollo de sistemas y/o en sistemas recomendadores. Por este motivo, RecoLibry-studio utiliza la ontología RecOnto para hacer un filtro de las técnicas de recomendación que puede usar el diseñador a partir de los componentes que ha seleccionado antes. Sin embargo, una posible mejora sería añadir un sistema de explicaciones que devuelva al diseñador los motivos por los que se le están mostrando un conjunto de técnicas de recomendación.

En los últimos años, la investigación en sistemas inteligentes explicables ha crecido mucho, como se puede ver en Miller (2019). El principal motivo es que al incluir explicaciones en sistemas inteligentes se aumenta la confianza del usuario en el uso de este tipo de sistemas. Además, la inclusión de explicaciones en RecoLibry-studio permitiría usarla como herramienta educativa para aprender a crear sistemas de recomendación. Una de las posibilidades que se puede explotar es analizar las reglas de RecOnto que devuelven un conjunto de técnicas y, a partir de esas reglas, generar una explicación que sea comprensible para el usuario. La investigación en sistemas de explicaciones nos permitiría estudiar metodologías para obtener los motivos de un resultado obtenido de RecOnto y, por otro lado, estudiar cómo mostrar esos motivos al usuario a través de RecoLibry-studio. Esto abriría un gran abanico de posibilidades donde orientar el trabajo para mejorar esta herramienta.

5.2.3. Añadir herramientas para el análisis de datos en RecoLibry Suite

Cómo se ha explicado en el Capítulo 2, algunas técnicas de recomendación tienen pequeñas modificaciones para solventar algún tipo de problema con los datos. Un ejemplo muy claro, como se ha visto en esta tesis, es cuando el sistema recomendador tiene el problema del *cold-start*. Una línea de trabajo muy interesante en RecoLibry Suite sería añadir una herramienta para analizar los datos que va a utilizar el sistema recomendador. Esta herramienta podría ayudar a los usuarios a detectar problemas como dispersión de datos, problema del *long-tail*, atributos no necesarios, etc. Incluso, esta herramienta podría ayudar a los usuarios en la modificación del conjunto de datos.

Una herramienta de este tipo se podría incluir como un paso previo en RecoLibry-studio para mejorar la selección de las técnicas de recomendación. En el diseño de RecOnto está contemplado clasificar los problemas de los datos en sistemas recomendadores. Esto permitiría mejorar la recuperación de técnicas de recomendación añadiendo más atributos a las descripciones de las técnicas y a las consultas al modelo.

5.2.4. Facilitar la inclusión de nuevos componentes en RecoLibry Suite

En la versión de RecoLibry Suite presentada en esta tesis doctoral, hay 2 elementos que controlan los componentes que se pueden usar. En primer lugar, está RecOnto que hace una definición teórica del componente y describe cómo se relaciona con otros componentes. Por otro lado, RecoLibry-core implementa la funcionalidad de dichos componentes para que pueda ser usado en los sistemas recomendadores. Un problema existente en este diseño es que cuando un desarrollador quiere añadir uno o más componentes nuevos debería modificar ambas herramientas. Sobre todo, si quiere que estos nuevos componentes estén disponibles en RecoLibry-studio.

Para mejorar este aspecto se podría trabajar en dos líneas de trabajo futuro. La primera de ellas sería crear un sistema que actualice la información de RecOnto a partir de los componentes nuevos que se vayan implementando en RecoLibry-core. Por otro lado, se podría trabajar en un sistema que, a partir de una nueva definición de componente en RecOnto, genere el código necesario en RecoLibry-core para incluirlo. Ambos planteamientos tendrían como objetivo facilitar a la comunidad científica o a los desarrolladores de software poder mejorar ambas herramientas, ya que el dominio de los sistemas recomendadores es muy amplio y está en continuo desarrollo.

5.3. Conclusiones finales

A partir de las aportaciones presentadas a lo largo de esta tesis doctoral y que se encuentran recogidas en los artículos publicados que se presentan en la Parte III, se concluye que:

1. Nuestro trabajo hace un estudio completo del estado del arte generando un modelo teórico que describe el dominio de los sistemas recomendadores.

2. Se han propuesto metodologías novedosas para resolver el problema del *cold-start*.

3. El modelo teórico que proponemos describe de forma detallada cómo se construyen los sistemas recomendadores. Además, este modelo crea un vocabulario único para los distintos componentes de los sistemas recomendadores.

4. Las herramientas y metodologías de desarrollo creadas en RecoLibry Suite ayudan en la construcción de los sistemas recomendadores, incluso a los usuarios con menos experiencia en este campo.

5. Tanto el modelo teórico como el framework están accesibles para su uso en repositorios públicos.

6. Todos los resultados de nuestra investigación han sido presentados a la comunidad científica.

Tras hacer un análisis de cómo se construyen los sistemas recomendadores, hemos conseguido crear un modelo y un conjunto de herramientas que facilitan el desarrollo de este tipo de sistemas. Además, se ha demostrado que estas herramientas simplifican la construcción de los sistemas de recomendación para usuarios con distintos perfiles. Sin embargo, observamos que hay muchas características que pueden mejorarse en estas herramientas y que podrían incluir funcionalidades interesantes. Esta tesis doctoral ha resuelto con éxito todos los objetivos que se propusieron al principio de la investigación y proporciona ideas para ampliar el trabajo en futuras líneas de investigación.

Parte II

Contributions

Chapter 6

Introduction

This chapter demarcates the research of this Ph. D. thesis in the recommender systems area. The main objective of our study was developing a platform to ease the creation of recommender systems. For this reason, in this chapter, we describe the difficulties ussually found by developers when they build this type of systems. Additionally, we explain our research planning to solve these difficulties. Section 6.1 illustrates the reasons why we think that this research is necessary. Next, in Section 6.2, we show a set of specific objectives defined in our study. Finally, Section 6.3 resumes the contributions reached for each of the proposed objectives in this thesis.

To present this thesis, we have chosen a format based on publications. The contributions reached in this thesis are described in detail in each of the accompanying publications, collected in Part III.

6.1. Motivation

The development process of a recommender system is not an easy task. Developers have to: do a precise analysis of all information used in recommender systems, choose the best technique for this information, and develop the final system from scratch or using an external framework. This process is becoming more and more complicated due to the continuous development of new recommender techniques and new frameworks.

Although research in recommender systems started in the early 90s (Goldberg et al., 1992), the greatest popularity of recommender systems was between 2000 and 2005. This popularity is linked to the beginning of the so-called Web 2.0 (O'Reilly, 2009). The web sites builtin the end of the 2000s and beginning of the 2010s allowed a bidirectional communication between users; indeed, users could participate actively, adding their content in the websites, for example, in blogs, wikis, etc.

As a consequence of Web 2.0, tow factors were introduced that encouraged the development of recommender systems: information overload and

the ease of obtaining the users' feedback. Firstly, the amount of information generated was growing, and that made it hard for users to find useful information. Recommender systems resolved this problem because they showed interesting content based on users'preferences. Secondly, the new Web 2.0 sites allowed to collect the users' opinions and preferences. For that reason, that facility to collect opinions fostered the development of recommender systems based on users' reviews, such as *collaborative filtering recommending systems* (Bobadilla et al., 2009) that suggest items selected by users with a similar opinion. The number of reviews pushed the research of this type of algorithms, as it happened in 2006 when Netflix organized a competition to implement a recommender system using a set of users'ratings from its platform (Bennett et al., 2007). On the other hand, the use of recommender systems enabled personalize the user experience in web sites and e-commerce sites as it is explained in Karat et al. (2004). For example, the recommendation systems of Netflix, Amazon, and Spotify show different products for each user in a personal way.

Another much-studied recommender system paradigm has been the *content-based recommender systems* (de Gemmis et al., 2015). In contrast with collaborative filtering techniques, these systems work using a detailed description of the elements to recommend. From the user reviews, these systems generate a user profile based on the preferences obtained by a graphical user interface or a query. This profile contains the common features from the elements that have been selected by a user. Next, the content-based recommender systems return a set of elements which description is similar to the generated user profile. This type of system benefits from the amount of information that describes the products, which is increasingly detailed. However, these and other additional techniques have some limitations. One way to resolve them is combining two or more methods in a *hybrid recommender system* (Burke, 2002).

The rise of mobile devices started another research area in which the recommender systems focus includes additional information that could influence the user opinion. This additional information is called contextual information (Dey, 2001). Contextual information describes the state and enviroment of users and products in a specific moment. This information could be, for example, user location, weather, or the schedule of a leisure activity. This information has been added to the recommender systems as another dimension to take into account (Adomavicius y Tuzhilin, 2015). Recommender systems that implement this approach are called *context-aware recommender systems (CARS)*. A typical domain for these recommender systems is the tourism and leisure domain, mostly when users use their mobile devices and want recommendations in real-time (Borràs et al., 2014).

Nowadays, there is a large variety of recommender techniques, and a developer needs to know these techniques and when to use each one to im-

plement a new recommendation system. For this purpose, the first step is to analyse the information that they will use in the new system. This analysis allows us to detect possible data problems; for example, there not being enough information to generate recommendations (this issue is called the *cold-start* problem). The results of this analysis can influence which recommender technique can be used or which one will return better results.

On the other hand, there is a wide variety of frameworks that help in the building process of recommender systems (Owen et al., 2011; Ekstrand et al., 2010; Recio-García et al., 2008). However, this wide variety is another challenge because there are many differences between each framework: the programming language used, limitation in the algorithms implemented, or different ways to build the final system. In addition, some frameworks use different terms than the terms used in the design process of recommender systems. For this reason, the full process to build a recommender system needs a team that contains people with a high level of experience in recommender systems and other people who know how to implement the final system using a framework. In addition, sometimes, the vocabulary uses in the design process should be adapted to the development process.

This design and development process can be simplified by applying a component-based design. This is possible because most of them use a similar scheme in the information flow. This allows us to define components that have concrete objectives inside recommender systems, for example: managing users information, defining each algorithm (like collaborative filtering, content-based, etc.). One of the significant advantages of this approach is the reuse of these components in different solutions. It allows us to reduce the time of development and reduces the cost of the software build (Szyperski et al., 2002). On another hand, the use of components allows us to define the restrictions and incompatibilities that exist between each component and make a model to describe how to build a recommender system. For example, if the system will use a component to read contextual information, then we can filter every algorithm that cannot use this type of information. Thanks to this way of building recommendation systems, a designer with a low level of knowledge in these systems can choose which components to use. Finally, the use of components allows implementing each component separately, and we can implement each component using different frameworks. This is transparent to a developer, who only needs to select which components will use in the final system.

In summary, the building of a recommender system is a complex process. It needs a developer o a team with a high level of knowledge in the different tasks. First, they should select that recommender technique that best suits in the domain and analyze the problems that could be found. Next, they have to decide how to implement the recommender systems, either using a framework or fully implementing the system themselves. In both cases, they

need to have advanced technical skills. For this reason, our research aids the development process, and, in the next section, we explain the objectives defined to complete our research.

6.2. Objectives

As we explained in the previous section, creating a recommendation system is a complex task. For this reason, the general objective of this Ph. D. thesis is to build a knowledge-based platform to unify the design and development process of recommender systems. For this purpose, we use a theoretical model to describe the building process of recommender systems based on a set of components. This platform contains a set of tools that assists users in the building process. They provide automatic assistance, guiding the user in the selection of the components based on the theoretical model. To complete this general goal, we have defined a series of more specific objectives.

The first specific objective (**O-1**) was to understand how to make recommender systems and how to define this construction into a theoretical model based on components. To complete this objective, we have performed three studies. First, we have enumerated different recommender techniques used in state of the art. For each of these method, we have described its restrictions based on the input information they use. Second, we have analyzed what type of problems occur in the recommender systems. Finally, we have examined a set of academic and open source frameworks to build these systems. We have reviewed their features to define the features of our final platform.

The second specific objective (**O-2**) associated with understanding the construction of recommender systems was to create a real system in the domain of tourism and leisure. This system creates tourism plans to users based on their preferences and contextual information at the time of the recommendation. This system was a case study to test the different functionalities and techniques included in our model. We chose the tourism domain because it contains a great variety of contextual information like user location, activities time, weather, etc. and we could include this information in our recommender process. Also, it allowed us to analyze the influence of contextual information in the final recommendations.

We have used this recommender system to work in the third specific objective (**O-3**): studying solutions to the cold-start problem (Park y Chu, 2009), which is one of the most common issues in recommender systems. This problem arises when there is not enough information in a recommender system to return useful results. It occurs, for example, when there are not enough ratings in a collaborative-filtering system, or there are not enough product descriptions in a content-based method. In this thesis, we propose to apply different configurations of the case-based reasoning method (CBR) to resolve this problem. CBR systems solve new problems by reusing solutions

used in similar situations (Kolodner, 1992). In our research group GAIA (Artificial Intelligence Applications Group), there is a long history of applying CBR to different domains, including recommender systems.

After completing objectives **O-1** and **O-2**, the fourth specific objective (**O-4**) was to formalize a theoretical model based on components to define the building process of a recommender system. This formalization comprises describing the functionality of each component as a black box, enumerating its restrictions, and describing the information returned. The main advantage of this model is to reuse most of these components in different recommender systems. Additionally, the model also determines the rules to combine these components. Finally, it yields a common vocabulary for building recommender systems. This means that this vocabulary is independent of any final framework, and it is the same vocabulary across the design and development processes. For these reasons, we have represented this model in an ontology to use it in our knowledge-based platform. An ontology creates a formal definition of entities, and it defines the properties and the relationships between these entities (Gruber, 2009).

All these specific objectives have allowed us to complete the main purpose of this thesis (**O-5**): the development of a knowledge-based platform to helpaid users in the design, implementation and deployment processes of recommender systems. This platform, called RecoLibry Suite, is based on the theoretical model of components defined in objective **O-4**. This platform consists of three tools:

- A high-level tool to design recommender systems in our platform. Based on the theoretical model, this tool helps designers select which recommender technique can apply based on the components chosen before.
- A JAVA framework that implements the components defined in the theoretical model. It uses third-party frameworks to implement the components. And it implements the process to build a recommender system using dependency injection.
- A web tool to deploy automatically all recommender systems built with RecoLibry Suite, which also implements an API to use the recommender system in external aplications.

6.3. Contributions summary

In the previous section, we have explained a set of specific objectives defined to implement a knowledge-based platform to build recommender systems. During our research, we have obtained one or more results for each particular goal, which we enumerate in the following. In the next list, we

describe the result of each objective and the chapter that explains it in detail:

Objective 1 (O-1)

Understanding the build process of recommender systems analyzing: techniques used, problems founded, and available frameworks.

Result 1 Study of classic recommender systems and context-aware recommender systems, including the different techniques used.

Contributions:

- Section 7.1.1: Recommender systems.

Result 2 A classification and analysis of the main problems founded in recommender systems. It includes some solutions proposed in the literature.

Contributions:

- Section 7.1.2: Problems in recommender systems.

Result 3 A study of different frameworks used to build recommender systems. This study includes a classification of all features detected in each framework to define which features should be included in RECOLIBRY SUITE.

Contributions:

- Section 7.1.3: Frameworks to build recommender systems.
- Chapter 15: RECOLIBRY SUITE: a set of intelligent tools for the development of recommender systems (Jorro-Aragoneses et al., 2020).

Objective 2 (O-2)

Developing a recommender system as a case study that uses contextual information and allows testing the concepts included in the theoretical model.

Result 4 Developed a context-aware recommender system of tourist and leisure plans in Madrid: *Madrid Live*.

Contributions:

- Section 7.2: Study of creation of a recommender system.
- Chapter 9: Madrid-Live: a context-aware recommender system (Jorro-Aragoneses et al., 2017b).

Objective 3 (O-3)

Developing different solutions to resolve the Cold-Start problem in recommender systems.

Result 5 A CBR method to resolve the Cold-Start problem in the user information of recommender systems.

Contributions:

- Section 7.3: Solutions proposed to resolve the cold-start problem.
- Chapter 11: Addressing the cold-start problem in facial expression recognition (Jorro-Aragoneses et al., 2015).

Result 6 A methodology for building an initial case base to resolve the cold-start problem in the items information of recommender systems.

Contributions:

- Chapter 7.3: Solutions proposed to resolve the cold-start problem.
- Chapter 12: Case base elicitation for a context-aware recommender system (Jorro-Aragoneses et al., 2018).

Objective 4 (O-4)

Designing a component-based model to describe the recommender system building process and implement it in an ontology to use it in the platform of objective **O-5**.

Result 7 An ontology to define the building process of recommender systems based on components: RecOnto.

Contributions:

- Section 7.4.1: A theoretical model to build recommender systems.
- Chapter 13: RecOnto: an ontology to model recommender systems and its components (Jorro-Aragoneses et al., 2017a).

Objective 5 (O-5)

Developing a platform to help users in designing, developing, and deploying recommender systems based on the theoretical model obtained in objective **O-4**.

Result 8 A framework to build recommender systems based on the components defined in the theoretical model in objective **O-4**: RECOLIBRY-CORE.

Contributions:

- Section 7.4.2: RECOLIBRY SUITE.
- Chapter 14: RECOLIBRY-CORE: a component-based framework for building recommender systems (Jorro-Aragoneses et al., 2019).
- Chapter 15: RECOLIBRY SUITE: a set of intelligent tools for the development of recommender systems (Jorro-Aragoneses et al., 2020).

Result 9 A graphical user interface for designing and building recommender systems in a semi-automatic way: RECOLIBRY-STUDIO.

Contributions:

- Section 7.4.2: RECOLIBRY SUITE.
- Chapter 15: RECOLIBRY SUITE: a set of intelligent tools for the development of recommender systems (Jorro-Aragoneses et al., 2020).

Result 10 A deploy mechanism using web services to execute the recommender systems automatically from a configuration file: RECOSERVER.

Contributions:

- Section 7.4.2: RECOLIBRY SUITE.
- Chapter 15: RECOLIBRY SUITE: a set of intelligent tools for the development of recommender systems (Jorro-Aragoneses et al., 2020).

The rest of this memory is organized as follows: Chapter 7 describes all the contributions of our research and which objective they met, and Chapter 8 collects all conclusions obtained in the research of this thesis and explain some research lines to do as future work. Finally, Part III contains all publications that explain in detail each contribution.

Chapter 7

Contributions

This chapter summarizes the contributions obtained during our research in this Ph. D. thesis. We have divided all contributions into different sections depending on the objective met. Section 7.1 contains a summary of our study about recommender systems, the problems associated with this type of systems, and an analysis of the frameworks to build the recommendation systems. Next, Section 7.2 describes the recommender system implemented as a case of study: Madrid Live. The solutions proposed to resolve the cold-start problem are explained in Section 7.3. Finally, Section 7.4 describes the theoretical model developed to describe the build process of the recommender systems, and we explain the platform RecoLibry Suite, the main objective of this thesis.

7.1. Domain study

To develop a platform to build recommender systems is necessary to understand the build process of these systems. For this reason, we defined the objective **O-1**:

Objective 1 (O-1)
Understanding the build process of recommender systems analyzing: techniques used, problems founded, and available frameworks.

This section contains the results obtained in this objective. First, it includes a literature revision where we have enumerated the different recommendation techniques. Next, we have listed the main problems founded in recommender systems associated with the input data. Finally, we have included an analysis of features from different frameworks to build recommendation systems.

7.1.1. Recommender systems

Often, we do not have all the information about one problem to decide on a solution. For this reason, we typically listen to the recommendations of other people or read opinion articles to guide our decision. The recommender systems try to copy this process automatically (Resnick y Varian, 1997). These systems became popular because they were necessary to filter the information contained on the Internet. Their main goal was to show the information most interesting to each user based on their interests and preferences.

Despite filtering information, the recommender systems have had new targets on which to focus their results. Ricci et al. (2015) describes other goals to apply the recommender systems: increase sales of a product, improve the user experience, improve user loyalty, etc. For example, these systems are used in e-commerce to recommend products based on other bought before (Schafer et al., 1999). In addition, there are multiple domains where it is possible to use a recommender system (Bobadilla et al., 2013): books, movies, music, travels, etc.

Recommender systems divide the knowledge used in three essential elements, to search the most interesting information to users (Ricci et al., 2015):

Users They are the consumers of recommender systems. Depending on the system goal, it saves different user information: demographic data, a list of product features preferred, etc.

Items Elements recommended by the system. Depending on the recommender technique used, the system archives different item information. The main goal of these systems is to classify the utility of each item based on the interests of a concrete user.

Transactions The interactions between users and items. Based on these transactions, a system can predict which items can interest to a user in the future. A system can obtain transactions in two way: *explicit* or *implicit*. The explicit transaction is when users interact with the system to indicate an opinion, for example, rating a book. On the other hand, the implicit transaction occurs when the system infers this information based on a users actions, for example, analyzing products bought by users.

Although most of the recommender systems only use these elements, other systems include more information to improve their results. They include external information from users and items, and this type of information is called *contextual information*. This information is in, for example, a tourist recommender system. This system needs to know the weather information and, based on this information, recommend an indoor or outdoor activity. In this case, the weather information is external data from users and items.

Base on the three essential elements and the possibility to include external information, in this thesis we classified the recommender techniques based on the Adomavicius y Tuzhilin (2015) classification. This classification proposes two groups of systems based on the information used: *classic systems*, or *2-dimensions system*, and *context-aware systems*. The first group contains all techniques that only use the essential elements, i.e., users, items, and transactions. The second group includes all techniques where is included contextual information.

In next sections, we describe the techniques included in each group and some works found in the literature where authors used these techniques.

7.1.1.1. Classic recommender systems

The first works about recommender systems started in the middle of the 90s. These works studied the ratings made by users to determine which items can interest them in the future. For this reason, most of these techniques was techniques based on ratings, i.e., techniques to estimate the rating of a user in an item that the user did not rate. These algorithms trained a function f that returned the estimating rating ($Ratings$) of a set of users ($Users$) and a set of Items ($Items$) (Zafarani et al., 2014):

$$f : Users \times Items \rightarrow Ratings \tag{7.1}$$

Function f should return a real value for each pair (u, i) where u is in $Users$, and i is in $Items$. This value depends on the rating range (Ekstrand et al., 2010). However, the main goal of a recommender system is to return a set of items as a recommendation. For this reason, we can generalize the function f and define recommender systems as a function R that uses the users ($Users$) and items ($Items$) information to return an ordered set of items ($Recommendations$) based on the utility of each item to a user (Adomavicius y Tuzhilin, 2005). The function R can use function f to know the user interest for each item. However, R includes a selection algorithm to choose a set of items to recommend.

$$R : Users \times Items \rightarrow Recommendations \tag{7.2}$$

This function explains why we can call these recommender systems 2-dimensions systems. There are three different types of techniques in this group: *collaborative filtering techniques*, *content-based techniques*, and *hybrid techniques*.

The *collaborative filtering recommender systems* use the rating pattern to find new items that can interest to a specific user (Koren y Bell, 2015). Their information domain is a matrix where the dimensions are users and items,

and the matrix values are the ratings. In this matrix, there are many values without a rating called *unknown values* (Ekstrand et al., 2010). Collaborative filtering recommender systems can be divided into two groups: *memory-based techniques* and *model-based techniques*.

Memory-based techniques use the ratings to find similarities between users or items. The algorithm most used is the *Nearest-Neighborhood algorithm* (Schafer et al., 2007). One application of this algorithm is to find a set of similar users based on the rating pattern, and then recommend items that these similar users prefer. Another possible application is to search for items that have similar rating patterns than the items liked by the user.

On the other hand, the model-based techniques train a model to predict the unknown ratings based on the other ratings (Sarwar et al., 2001). Next, the recommender systems use these models to generate their results. There are many algorithms in this group. Some of them are: *rules-based systems*, *bayesian networks*, *clustering algorithms*. However, one of the methods with the best results is the *Non-Negative Matrix Factorization* (NMF) (Lee y Seung, 2001).

Other classic systems are the *content-based recommender systems*. These recommender systems compare the detailed features of items with the features that users prefer (Lops et al., 2011). These recommendation systems have two information sources (Aggarwal, 2016): the items description and the user profiles. This recommender system should apply three steps (Lops et al., 2011):

1. **Content analysis.** This step extracts the items information to save this information with the structure used in the recommender system.

2. **Train user profiles.** Based on the interactions between users and items, the system generates user profiles. Each profile contains the items features that a user prefers. Usually, the system uses machine learning algorithms in this step.

3. **Content filter.** It is the recommendation process. It compares the items description with the user profiles. Next, it shows the items most relevant from users. Users can rate the new items, and the system executes the step of the train user profile again to update the user preferences.

There are many algorithms applied in these recommender systems with good results, like *bayesian classification* (Yang et al., 2013), *semantic networks* (Shishehchi et al., 2010), or *regression-based models* (Van den Oord et al., 2013). However, the *Nearest Neighbors algorithm* is the most used (Pazzani y Billsus, 2007).

Within the content-based recommender systems, there are a specific technique called *case-based recommender systems*. This type of recommender

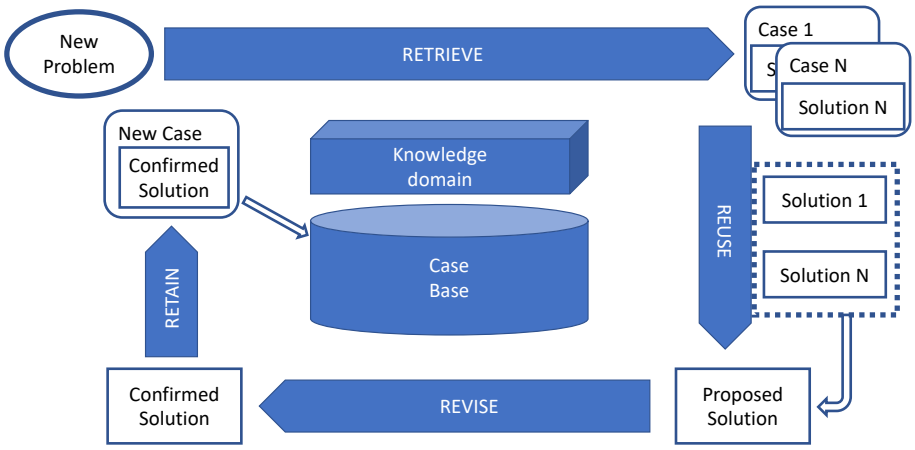

Figura 7.1: Diagram of the steps in CBR systems (Aamodt y Plaza, 1994).

systems uses the methodology *case-based reasoning* (CBR). The CBR methodology consists of resolving new problems reusing or adapting solutions applied to similar problems in the past (Riesbeck y Schank, 1989). This methodology is applied in domains where it is not possible to create a model, or it is difficult to acquire knowledge (Shiu y Pal, 2004). The CBR method has a set of solutions applied in problems of the same domain. Each solution contains the problem description, and it is called *case*. Based on these cases, the CBR method applies four steps to resolve new problems. As Figure 7.1 shows, these steps, called 4-Rs, are the next (Aamodt y Plaza, 1994):

- **Retrieval.** From a new problem, the system uses a set of similarity functions to retrieve the most similar cases.
- **Reuse.** Next, the CBR system selects which retrieved cases will use to propose a solution to the new problem.
- **Revision.** The system analyzes if the proposed solution will resolve the new problem. In this step, the system can adapt to the solution.
- **Retain.** If the solution resolves the problem, the system adds the problem description and save both in a new case to use in the future.

Case-based recommender system is applied in many domains. Some examples are: recommender massive online courses (Bousbahi y Chorfi, 2015), recommend movies (Quijano-Sánchez et al., 2012) and wellness therapies (Pheng et al., 2013).

The last type inside of the classic recommender systems group is the *hybrid systems*. These systems combine two or more recommender techniques to solve the problems that these techniques have individually (Ricci et al.,

2015). Burke (2002) contains a clasification about the combined method used. This classification includes the next seven types of hybrid methods:

- **Weighted.** The algorithm uses a score or vote method to select the final recommendation based on the solution of each technique.
- **Switching.** The algorithm selects the technique depending on the situation.
- **Mixed.** The algorithm shows the recommendation obtained in every technique.
- **Feature combination.** It uses an algorithm based on a model combining the features of each technique.
- **Cascade.** The solution proposed for a technique is improved using another technique.
- **Feature augmentation.** The output from one technique is used as an input feature to another.
- **Meta-level.** The model learned by one recommender is used as input to another.

In Section 7.1.2, we enumerate some examples that use hybrid techniques to resolve concrete problems in recommender systems.

7.1.1.2. Context-Aware recommender systems

All techniques reviewed before only use the three essential elements: users, items and transactions. However, there is a research area in recommender systems that studies how external factors can influence in the user decisions in a particular moment. These systems are called *context-aware recommender systems* (CARS).

They include *contextual information* to obtain a recommendation. The definition about contextual information is in Dey (2001): *Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.* It is a general definition, howevere, in Zimmermann et al. (2007), there is a classification of contextual information and how to use it.

We explain how the contextual information works using an example. Imagine that we have implemented a context-aware recommender system about movies. We should recommend a movie to a user who loves horror movies. However, this day, the user goes to the cinema with her family. This family has two children. In this case, the recommender system should search

for another movie type because it should adapt to the new situation. For this reason, the recommender system returns a movie that is not the favorite for the user, but which is better in this situation. In this example, the group type (family) is the contextual information.

The context-aware recommender system can be difined by an extension of the function explained in the classic recommender systems. This function R_{ctx} uses the domain of $Users$, $Items$ and the contextual information ($Context$) to return an ordered set of items ($Recommendation$) based on the utility of each item in a concrete moment:

$$R_{ctx} : Users \times Items \times Context \rightarrow Recommendations \tag{7.3}$$

These recommender systems can be classified into three paradigms based on how they use the contextual information in the systems (Adomavicius y Tuzhilin, 2015):

- **Pre-filtering paradigm.** These systems filter all items that cannot adapt to contextual information. Next, these systems apply a classic recommender technique with the rest of the items to calculate the recommendation.
- **Post-filtering paradigm.** These systems reverse the steps of the pre-filtering paradigm. First, they apply a classic recommender technique to order all items based on their utility. Next, they filter all items that not can be used with this contextual information.
- **Context model paradigm.** The last type of system makes a model that includes the contextual information inside of the model based on user and items.

Each paradigm has its advantages and its disadvantages. Panniello et al. (2009) contains a study to analyze which paradigm, between pre-filtering and post-filtering, has better results. This study concludes that the pre-filtering is faster, because it removes items before calculating the recommendation, and the results are more relevant comparison with the results obtained whit post-filtering methods.

There are many domains where applied the context-aware recommender systems is, like movies (Ono et al., 2007; Colombo-Mendoza et al., 2015), music (Baltrunas et al., 2011; Hariri et al., 2012) or learning tools (Wang y Wu, 2011). However, a popular domain in context-aware recommender systems is the tourism domain. The main reason is that this domain contains contextual information: user location, timetable of activities, weather, transport, etc. Also, these systems usually recommend tourist activities in real time and, to do that, they need to know the contextual information to improve the results (Borràs et al., 2014). The first context-aware recommender

system in the tourism domains was COMPASS (Van Setten et al., 2004), which shows interest locations near the user location. Another example is Pythia (Drosatos et al., 2015) that recommends different points of interest depending on the contextual information of users.

In conclusion, we can classify recommender techniques based on the information used. It allows us to divide the techniques into two groups: classic techniques and context-based techniques. This classification is necessary to include them in the future model. When this study is finished, the next step was to analyze the common problems found in recommender systems based on the input data.

7.1.1.3. New trends in recommender systems research

In addition to all techniques explained previously, other techniques have become popular in the last years. The technique most popular, based on its high accuracy in different artificial intelligence areas, is the *deep learning* method. Deep learning is a learning method that uses different interconnected levels to resolve a problem. It allows us to transform a problem from raw data, such as image, to an abstract level of representation, for example, the image category in a classification *Lecun2015*. In the recommender systems research area, deep learning made it possible to capture non-linear and non-trivial relationships between users and items in an efficient way (Mu, 2018). In addition, this technique can solve some of the problems in recommender systems described in this thesis, such as the cold start problem (Wei et al., 2017). However, deep learning has some limitations, such as the models obtained are not interpretable or the amount of information needed to train a model in this technique (Zhang et al., 2019).

Another new method in recommender systems is the use of *knowledge graphs*. A knowledge graph is a representation of information composed by entities, represented with nodes, and the relationships between these entities, represented by edges (Ji et al., 2020). Knowledge graphs represent the information depending on the problem to be solved. An example is the hybrid representation in entity2rec (Palumbo et al., 2017). It uses networks where entities are: the users from the recommender system, the movies to recommend, and some movies features (actors, directors, etc.). The relations are the interactions between users and movies, and the features with each corresponding movie. Authors used these knowledge graphs to generate a measures score to obtain a ranking of n recommendations. We can find other representation in (Caro-Martinez y Jimenez-Diaz, 2017), where authors proposed a knowledge network to represent the interactions from users in an online judge of programming exercises. This graph relates to users and the exercises completed by them. Next, they applied metrics from social network analysis to recommend new problems to users.

Apart from new techniques to apply in recommender systems, authors

propose new types of recommender systems that return new type of results. For example, one research area that has been active in recent years is to create recommender systems based on sequences. These systems return a list of items for a user to consume in that order (Quadrana et al., 2018). To train these systems, they use records that contains a set of items consumed by a user in the past. In the training step, these systems take into account the order of each item to understand how to recommend a new sequence of items. These recommender systems are applied in different domains like e-commerce to recommend the next product to buy (He y McAuley, 2017), in tourism to suggest the next point of interest to visit (He et al., 2016), or to create a music playlist (Turrin et al., 2015).

The design process of the model proposed in this thesis should take into account these new techniques and new types of a recommender system, and other that appear in the future, to allow other developers include them into the model.

7.1.2. Problems in recommender systems

There are multiple problems in the build process of recommender systems. Some of them are the users' privacy, the system scalability, latency issues, etc. (Khusro y Ali, 2006). However, in this thesis, we have studied three common problems based on the distribution of the input data. These problems are *sparsity*, the *long-tail problem*, and the *cold-start problem*.

7.1.2.1. Sparsity

Due to the high number of items and users, usually, there is a low number of transactions, i.e., we have not got enough ratings for all users in each item. This problem does that sometimes the recommender systems do not return valid solutions (Shahabi y Chen, 2003). This problem is called sparsity, and it is common in collaborative filtering recommender systems based on memory.

There are some models to resolve this problem. Some solutions use *techniques based on dimensional decomposition*, for example, *Singular Value Decomposition* (SVD) method (Sarwar et al., 2000) or use *techniques based on latent factors* like *Non-Negative Matrix Factorization* (NMF) (Luo et al., 2016). Other solutions use *hybrid recommender systems* to improve the results found in sparsity data (Maneeroj y Takasu, 2009; Kim et al., 2017).

7.1.2.2. Long-tail problem

This is another common problem in recommender systems. Recommender systems work very well to items that have a high number of ratings. However, many items have not got enough ratings. This problem is called the *long-tail problem*, and it is a distribution problem where a few elements have many

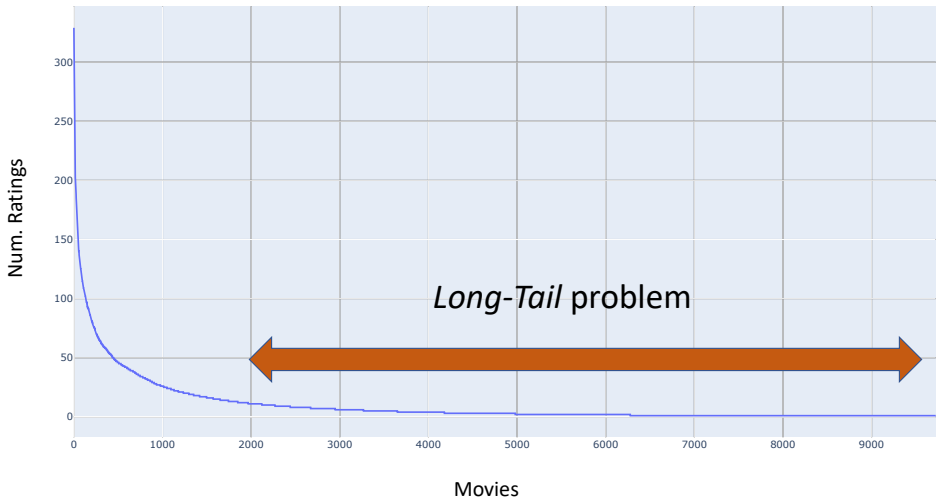

Figura 7.2: Long tail problem. Representation of the number of ratings that each film has in the MovieLens dataset.

ratings, and many items have few ratings (Park y Tuzhilin, 2008). Figure 7.2 shows an example of the long-tail problem in the MovieLens dataset.

In the literature, there are some solutions to resolve this problem. Possible solutions are *clustering algorithms* (Park, 2012), *matrix factorization* (Jing et al., 2015), or *graph-based algorithms* (Shi, 2013). In addition, we found *hybrid techniques* to resolve this problem (Domingues et al., 2013; Alshammari et al., 2017).

7.1.2.3. Cold-start problem

The last problem studied was the *cold-start problem*. It is a common problem in recommender systems when there is not enough information about items or users to make a recommendation (Park y Chu, 2009). It occurs, for example, when a new movie is introduced in a recommender system. This movie has not got enough ratings to include it in a recommendation.

This is one of the problems most studied in recommender systems. In Zhang y Lee (2010), authors propose to add social tags in users and items descriptions to use them when a user or item is in the cold-start problem. Other works, like Lika et al. (2014), try to classify new users in groups to make the first recommendations. However, like sparsity and long-tail problem, a common solution is to use hybrid recommender systems (Popescul et al., 2001; Schein et al., 2002).

Based on the literature, in this Ph. D. thesis we studied solutions for the

Framework	Language	Year	CF	CB	ML	UI	DE	EV	DP
Lenskit	Java	2011	X				X	X	
MyMediaLite	C#	2011	X	X			X	X	
Mahout	Java, Scala	2012	X		X		X	X	
PredictionIO	Java, PHP, Python, Ruby	2013			X	X	X	X	X
HapiGer	Node.js	2015	X				X		
LightFM	Python	2015	X	X				X	
LibRec	Java	2015	X	X	X			X	
jCOLIBRI	Java	2015		X		X	X	X	
RankSys	Java	2016	X		X			X	
LIBMF	R	2016	X					X	
RecDB	SQL	2017	X						X
Surprise	Python	2017	X				X	X	
Rexy	Python	2017	X	X				X	
QMF	C++	2017	X					X	
SpotLight	Python	2017			X			X	
CaseRecommender	Pyhton	2017	X		X			X	
proNet-core	C++	2017			X				
Seldon	Java, Python, R, PySpark	2018			X		X		X
Oryx v2	Java	2018	X		X		X		X
Tensorrec	Pyhton	2018			X			X	
recommenderlab	R	2018			X		X	X	

Tabla 7.1: Comparison of open source or academic frameworks to create recommending systems from 2011

cold-start problems. This is due to it is one of the most common problems in recommender systems and, also, it is common in CBR systems.

7.1.3. Frameworks to build recommender systems

The next step to understand how to build a recommender system is to analyze the available frameworks to develop these systems. The results of this analysis allow us to design the platform of objective **O-5**. Table 7.1 contains the results of this analysis in 21 different frameworks. First, we have analyzed which techniques include each framework between collaborative filtering (**CF**), content-based (**CB**), or machine learning (**ML**). Next, we have enumerated which frameworks contain a graphic user interface (**UI**) to help users in the developing process. The next features analyzed were if users can develop new algorithms in each framework (**DE**), which one has methods to evaluate the recommender systems (**EV**), and, finally, which framework has tools to deploy automatically de system developed (**DP**).

A detailed review of our analysis is in Chapter 15 (Jorro-Aragoneses et al., 2020). This section only enumerates the main conclusions obtained in this analysis. First, we have observed that the most implemented algorithms are collaborative-filtering and machine learning. The main reason for collaborative-filtering is because it is the most popular algorithm to implement a recommender system based on rating. The case of machine learning is different. At present, machine learning techniques are trendy due to the amount of information that exists. Some of these frameworks use web services

to execute the machine learning algorithm like TensorFlow or Apache Spark. On the other hand, the content-based technique is the least implemented by the analyzed frameworks.

Another conclusion obtained in this analysis was that most of these frameworks are oriented to people with a high level of programming skills. We can see that only two of them contain a graphic user interface: PredictionIO (Chan et al., 2013) and jCOLIBRI (Recio-García et al., 2008). Also, we can observe that only four of them have a tool to deploy the systems implemented. The main reason is that it is difficult to automate the developing process. However, a feature common in most of these frameworks is that they contain methods to evaluate the recommender systems implemented. It is important to know the quality of the techniques used.

7.2. Study of creation of a recommendation system

The next step to complete the main objective of our thesis was to understand the steps needed to develop a recommender system. For this reason, we defined the objective **O-2**:

Objective 2 (O-2)
Developing a recommender system as a case study that uses contextual information and allow to test the concepts included in the theoretical model.

To achieve this objective, we implemented a context-aware recommender system of leisure plans in Madrid. The name of this recommender system is *Madrid Live*. It suggests a set of leisure activity to users based on their preferences, their contextual information, and the information about the available activities. This recommender system was a case of study to evaluate the model proposed in objective **O-4** and test different components and functionalities to add in this type of systems.

Madrid Live uses the user contextual information as well as user preferences and activities description. The plans recommender in Madrid Live contains a set of leisure activities with a timetable that indicates the moment to do each leisure activity. Chapter 9 (Jorro-Aragoneses et al., 2017b) includes a detailed description of Madrid Live, and Chapter 12 (Jorro-Aragoneses et al., 2018) describes a version where we included temporal events, like concerts or sport matchs.

The recommender process of Madrid Live is implemented using a CBR module. Figure 7.3 shows the steps in the CBR process. This module stores the plans made by other users in the past. For each plan, the module generates a case with a description, where the activities information and contextual restrictions are included, and a solution that saves the set of activities and its

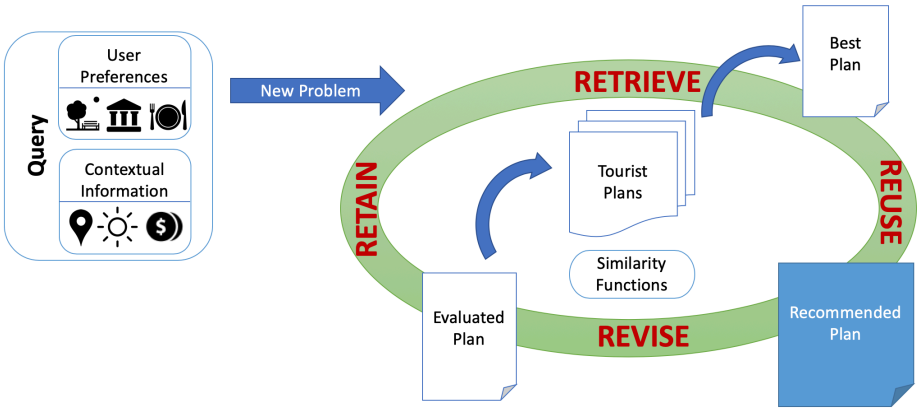

Figura 7.3: CBR module process in the Madrid Live recommendation system.

timetable. The CBR module starts receiving a query that contains the user preferences and the contextual information at this moment (location, time, weather, etc.). Next, it compares this query with the description of each case stored in its case base. To do that, it uses a set of similarity functions that compares each attribute. The CBR module retrieves the case most similar to the query and returns its solution as the final recommendation.

Apart from describing Madrid Live, Chapter 9 (Jorro-Aragoneses et al., 2017b) analyses a problem detected using the contextual information. We observed that the CBR module retrieves cases with low similarity respect to user preferences. In this chapter, we explain that if we apply too many contextual restrictions, we remove more plans, and we retrieve plans with less satisfactory according to user profile. In addition, these problems do not depend on the number of restrictions. There are some contextual restrictions, like location, that filter more cases than others and make that the recommendation has a less similarity respect the user preferences. To resolve this case, we proposed an explanation system to describe to the user the differences between the best plan, based exclusively on the user preferences, and the final recommendation.

However, this is not the unique problem founded in our recommender system. In the next section, we describe different solutions proposed to resolve a common problem in CBR systems and our recommender system: the cold-start problem.

7.3. Solutions proposed to resolve the cold-start problem

As we explained in Section 7.1.2, the cold-start is a common problem in recommender systems. This problem appears when there is not enough information to calculate a recommendation. For this reason, in our thesis, we propose the objective **O-3**:

Objective 3 (O-3)
Developing different solutions to resolve the cold-start problem in recommender systems.

To complete this objective, we have proposed some solutions based on the case base configuration in CBR systems. We take this decision because our recommender system, Madrid Live, was implemented using a CBR system, and the cold-start is a common problem in the CBR systems too. We have classified the solutions into two groups: solutions to the cold-start problem in users' information, and solutions to the cold-start problem in items solutions. Next, we explain the solution proposed in each group.

7.3.1. Cold-start problem in user information

A significant difficulty in recommender systems occurs when there is not enough information of a user to recommend items. For example, when the user does not have enough ratings to know his/her preferences.

In this thesis, we have studied this problem using a module that we included in Madrid Live. This module, called *PhotoMood*, detects users' emotions using images of their faces. Chapter 11 (Jorro-Aragoneses et al., 2015) describes how works PhotoMood to detect two emotions: *like* and *dislike*. In summary, PhotoMood has two stages. The first stage is an imagen preprocessing to detect the user gestures in the picture and save their positions in eight vectors, one vector for each gesture. The next step is a CBR module that recognizes the user's emotion.

Figure 7.4 shows how works the CBR module. It contains a case base where it stores a set of gestures with its corresponding emotion. To detect the user's emotion, it receives the vectors generated by the image pre-processing step. Next, it compares these vectors with the vectors stored in the case base, and it retrieves a set of the most similar cases. Next, it uses a voting strategy to determine the emotion of the query based on the emotions of the retrieved cases.

In this module, the cold-start problem appears when the case base does not have enough images to detect the user's emotion. To resolve this problem, we proposed two modifications in PhotoMood (Jorro-Aragoneses et

Figura 7.4: CBR process to detect a usersémotions from images of their face.

al., 2014). The first one was to set a weight to each gesture to indicate which gestures are more important for each user when the user expresses an emotion. We used a genetic algorithm to train the best weight configuration for each user and another general configuration for all users. The second modification was to create two types of case bases: a personal case base and a general case base. The first one only contains images from the same user, and the second one includes 300 pictures of anonymous people. With these modifications, we concluded that the best precision to detect emotions is achieved the user weight configuration with the personal case base. However, the results showed that the precision of the general case base with the general weight configuration is high and could be used to resolve the cold-start problem.

Later, we described the complete system and we performed another experiment to compare PhotoMood with other work. Photomood and this experiment are explained in Chapter 11 (Jorro-Aragoneses et al., 2015). The experiment consists on comparing the error detecting emotions using a specific dataset. In this experiment, PhotoMood had worse results than other solutions. The main reason was that PhotoMood could not detect many different emotions like anger or disgust. However, PhotoMood is used in a recommender system, and some of these emotions were not necessary to detect. For this reason, we grouped the images into three groups: *surprise*, *like*, and *dislike*. In addition, we improve the general case base to use in the cold-start problem. To do that, we designed different case bases where each one contained images of users with the same features, as sex or age,. The of these case bases allowed us to detect emotions with good results.

Therefore, we proposed different representations of a case base using the

information of anonymous people to resolve the cold-start problem. In these cases, this information was images, but, it can be extrapolated to other information. For example, make anonymous preference profiles. In addition, it is possible to create different case bases depending on the users' features.

7.3.2. Cold-start problem in items information

Another problem resolved in this thesis was the cold-start problem in the catalog of items. This problem appears when there are not many items to recommend. This problem was analyzed using Madrid Live again. As is explained in Chapter 12 (Jorro-Aragoneses et al., 2018), the main goal was to create a set of plans and use them as an initial case base for Madrid Live.

The cases in Madrid Live should respect two important features. The first one is that the cases should represent most of the contextual restrictions and cover a significant number of situations. The second one is that each plan should have a logical order in its activities. The main advantage of a case base with real plans is that they were validated by other users, and the activities order was logical. There is an implicit knowledge that it is not necessary to model.

Taking into account these restrictions, we proposed a methodology to build an initial case base for Madrid Live. The first step in this methodology was to create a set of pseudo-aleatory plans. Each plan should respect a contextual restriction configuration, and then we complete it with random activities. In this step, we created 300 leisure plans. The next step was to evaluate this plan. In this evaluation step, we used the Elo test (Elo, 1987) to make a ranking of best leisure plans. This test showed two plans with the same contextual restriction, and a user selects which one prefers. Based on these votes, we obtained the Elo ranking to classify the plans.

We observed that we could use 50 % of cases as in the initial case base. However, we should evaluate the coverage and density of the case base to know the probability of recommending a plan in any situation. In both measures, we observed that we cover most of the recommendations if we change any activity of a plan. This was a good result because some of the activities included in plans are temporal events that do not repeat in the future, for example, music concerts. In summary, this methodology is valid to create an initial catalog of items to recommend because it covers many situations and, using the test Elo, we found a method to obtain the best plans based on the users' opinions. This solution allows us to resolve the cold-start problem in the items domain.

7.4. Platform to build a recommender system

After the contributions of all these objectives, we were ready to complete the main objective of this Ph. D. thesis: implementing a knowledge-based platform to build recommender systems. The first step to develop this platform was to create the necessary knowledge to use in our platform. For this reason, we defined the objective **O-4**:

Objective 4 (O-4)
Designing a component-based model to describe the recommender system building process and implement it in an ontology to use it in the platform of objective **O-5**.

Next, based on this model, we could implement the platform. It was defined in objective **O-5**:

Objective 5 (O-5)
Developing a platform to helping users in designing, developing, and deploying recommender systems based on the theoretical model obtained in objective **O-4**.

In this section, we explain all contributions obtained in each specific objective. First, we explain the model that describes how to build recommender systems. Next, we describe the methodology and tools proposed to complete the last objective.

7.4.1. A theoretical model to build recommender systems

The first step to develop a knowledge-based platform to build a recommender system is to create the knowledge used in this platform. In this case, we proposed to create a model that describes how to build a recommender system based on the contributions obtained in the previous objectives. This model should apply some features to use it in the final platform. The first feature is that it must define all functionalities inside recommender systems and encapsulate each one in a component. Another feature is that it should contain a set of rules that defines how to combine these components. Finally, it has to be represented in a format where external applications can use it.

To design the model, we decided to use the model based on components design pattern because this type design reduces the costs in the implementation of complex systems (Szyperski, 2000). The two main reasons are: the components can be reused in multiple systems, and each component has concrete functionality, and it eases the development process (Heineman y Councill, 2001).

As a result, we implemented RECONTO, an ontology that describes different components used in recommender systems and a set of logic rules that define the relationship between components. RECONTO defines a common vocabulary that designers and developers can be used to build recommender systems, and is independent of each domain. RECONTO was implemented using Ontology Web Language (OWL) to use this model in external applications.

We divided the developing process of this model in two steps. The first step consists of identifying all components used in recommender techniques. To do that, we have divided up the operation of each recommendation technique in atomic operations, and for each one, we define a component. In the next step, we describe the set of rules that describe how to join these components and the incompatibilities that exist.

During this thesis, we worked with two versions of RECONTO. The first one, described in detail in Chapter 13 (Jorro-Aragoneses et al., 2017a), was a theoretical model based on the related work analysis. This version contained some terms that were especially complex to use in the classification of the components — for example, the fundamental categories of input data. We evaluated this version classifying a real recommender system and its components using RECONTO. This version allowed us to have a detailed definition of each component.

However, we simplified it in the second version to include RECONTO in the RECOLIBRY SUITE platform. In this version, defined in Chapter 15 (Jorro-Aragoneses et al., 2020), we simplified some descriptions, and we added some rules to determine the order of component selection to build recommender systems. Currently, the second version of RECONTO is in a public repository[1] for consultation or extension.

7.4.2. RECOLIBRY SUITE

The build process of a recommender system contains a set of complex tasks. In addition, designers and developers should have a high level of knowledge about recommender systems and the different ways to implement them. For this reason, we have proposed to create a knowledge-based platform to help users to build recommender systems as our main objective in this thesis (**O-5**).

The result of this objective have been RECOLIBRY SUITE, a developing methodology and a set of tools that help users in the desinging, developing, and deploying processes. RECOLIBRY SUITE contains three tools, Figure 7.5 shows them. The first one is RECOLIBRY-CORE, a JAVA framework that implements the components defined in RECONTO and the architecture to combine them. Next, RECOLIBRY-STUDIO is a graphical user interface that

[1] https://github.com/UCM-GAIA/RecOnto

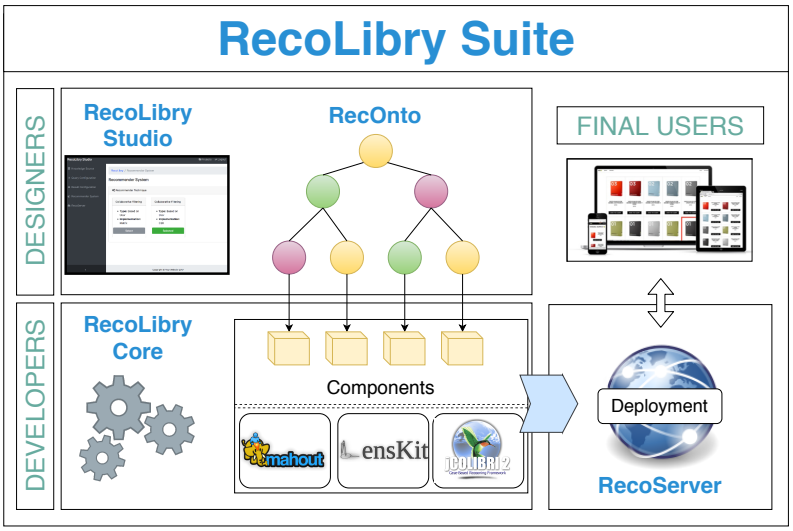

Figura 7.5: RecoLibry Suite architecture

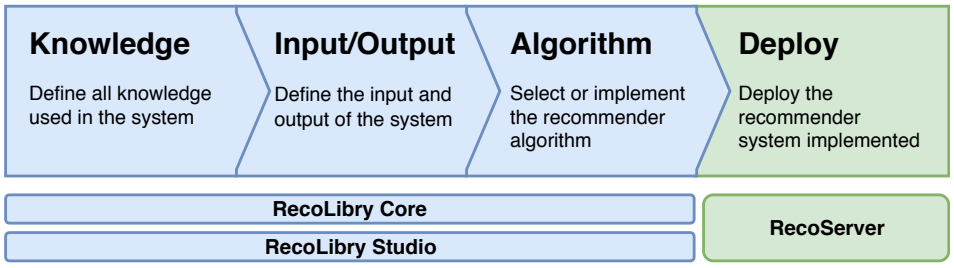

Figura 7.6: Workflow in RecoLibry Suite to develop recommender systems

assists users in the building process of recommender systems. The last tool is RecoServer that deploys automatically all recommender systems done by the above tools.

Next, we explain the methodology and the tools of RecoLibry Suite. Finally, we summarize the evaluation of this platform.

7.4.2.1. Methodology to build recommender systems

We have defined the development process of recommender systems in four stages (Figure 7.6). The first one is to determine the *knowledge* available to generate recommendations, that is, identify the information used by the recommender system. The next step is to define the *input and output* of the final system. In this stage, users should describe what information will insert in the recommender system and how they want to obtain the re-

commendations, based on the knowledge defined in the before step. When users have completed both stages, the systems can help users in the *algorithm selection* step. This step is guide by the ontology described in Section 7.4.1. Our platform suggests a set of suitable algorithms to users based on the information introduced. The final step is to *deploy recommender systems* using the components defined before. In Chapter 15 (Jorro-Aragoneses et al., 2020), we explained each stage including some examples.

The RECOLIBRY SUITE tools cover these steps in different ways. RECOLIBRY-CORE and RECOLIBRY-STUDIO cover the three first stages to different user profiles. RECOLIBRY-CORE is oriented to users with high programming skills, and, on the other hand, RECOLIBRY-STUDIO is for users who prefer to use a graphical user interface. The main task of RECOSERVER is to automate the deploying stage in the systems built-in RECOLIBRY SUITE. Also, RECOSERVER creates a set of web services to use the recommender system in external applications. In the next sections, we describe briefly the three tool included in RECOLIBRY SUITE.

7.4.2.2. RECOLIBRY-CORE

RECOLIBRY-CORE, Chapter 14 (Jorro-Aragoneses et al., 2019), is a JAVA framework that implements the functionality of all components defined in RECONTO. In addition, it contains the architecture necessary to combine these components and built the final recommender systems.

RECOLIBRY-CORE defines a schema of recommender systems using a set of interfaces and abstract classes. This schema manages the workflow to execute the recommender systems correctly. The correct order is: initialize the recommender system, compute a recommendation from a query, and close the system safely. Figure 7.7 shows the UML diagram with the classes that define the main scheme of a recommendation system in RECOLIBRY-CORE. The main component is the `RecommenderSystem` class. It defines a recommender system based on three elements:

- `RecommenderAlgorithm`: This interface defines a set of methods necessary in a recommender algorithm to include it in RECOLIBRY-CORE. These methods are the steps defined in the RecoLibry workflow.
- `Query`: It is an interface to define the queries structure to use them in recommender systems. It defines queries as bean (pojo) objects.
- `RecommenderResult`: This class stores the results generated by a recommender system. It contains the recommended item and a recommendation value that its value depends on the algorithm selected.

In Chapter 14 (Jorro-Aragoneses et al., 2019), we described in detail the full scheme of RECOLIBRY-CORE and how to build a recommender system with this scheme.

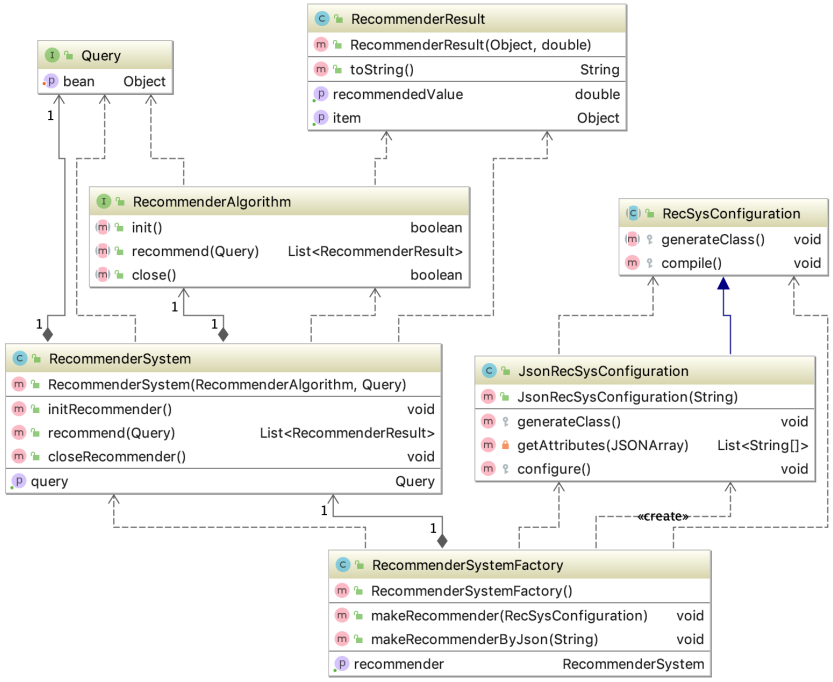

Figura 7.7: Recommender systems scheme in RecoLibry-core

Our framework reuses the implementations of recommendation techniques from third-party libraries. However, it does not use this implementation directly. RecoLibry-core acts as a wrapper the encapsulates these implementations to apply them in the components scheme. In the current version, RecoLibry-core uses components to build content-based recommender systems, based on the jCOLIBRY framewrok, and collaborative filtering methods, based on the Apache Mahout library.

To facilitate the creation of recommender systems using components, RecoLibry-core uses the dependency injection programming pattern. This pattern allowed us to define how to combine all components to make different types of systems. Another advantage of this pattern was that it is possible to create a recommender system using a few lines of code or writing a configuration file. The dependency injection pattern was implemented with the *Google Guice* library (Vanbrabant, 2008).

RecoLibry-core is available in a public repository[2]. This repository includes a development guide with additional details and a set of examples to build recommender systems with this framework.

[2]https://github.com/UCM-GAIA/RecoLibry-Core

7.4.2.3. RECOLIBRY-STUDIO

Create a recommender system using RECOLIBRY-CORE can be difficult for users with low experience developing systems. For this reason, we included RECOLIBRY-STUDIO. This tool consists on a web application that users can build recommender systems using visual elements. Also, it contains an intelligent assistant that, based on the model described in RECONTO, guides the designers to select the recommender algorithm. Another feature of RECOLIBRY-STUDIO is that it can check the correctness combination of components selected by the user.

The recommender design process in RECOLIBRY-STUDIO has the same four stages defined in Section 7.4.2.1. For each stage, the application shows a page to complete the necessary information. Finally, when designers completed all stages, they can download a configuration file to instate the recommender system with RECOLIBRY-CORE or deploy it in RECOSERVER.

In Chapter 15 (Jorro-Aragoneses et al., 2020), there is a detailed description of all designing stages in RECOLIBRY-STUDIO. It explains the process using an example to create a movie recommender system using the MovieLens dataset.

7.4.2.4. RECOSERVER

The last tool within RECOLIBRY SUITE is RECOSERVER. Its main goal is to deploy automatically recommender systems created with these tools in a web server. For each recommender system, it deploys a RESTful API and a web application. The API acts as the back-end of recommender systems to use it in external applications. The web application deployed allows designers to test the recommender system designed.

RECOSERVER uses *Docker* technology (Merkel, 2014) to create, deploy, and run recommender systems in containers. A container is a light version of a virtual machine because it only contains the operating systems services required by the application. All recommender systems deployed in RECOSERVER has the same scheme: an *Apache Tomcat* server to run the application, a *MySQL* database to store the system information, a RECOLIBRY-CORE instance to build the systems, and the configuration file that defines the components used.

When RECOSERVER deploys the container, it creates a unique URL to access both the web application and web-services. Designers have a web site where they can see how to call each service included in the recommender system API. Chapter 15 (Jorro-Aragoneses et al., 2020) explains in detail all these features.

As we have explained, RECOLIBRY-STUDIO contains a set of tools that ease the build process of recommender systems. They cover all the necessary stages in the developing process. Next, we summarize the evaluation done to

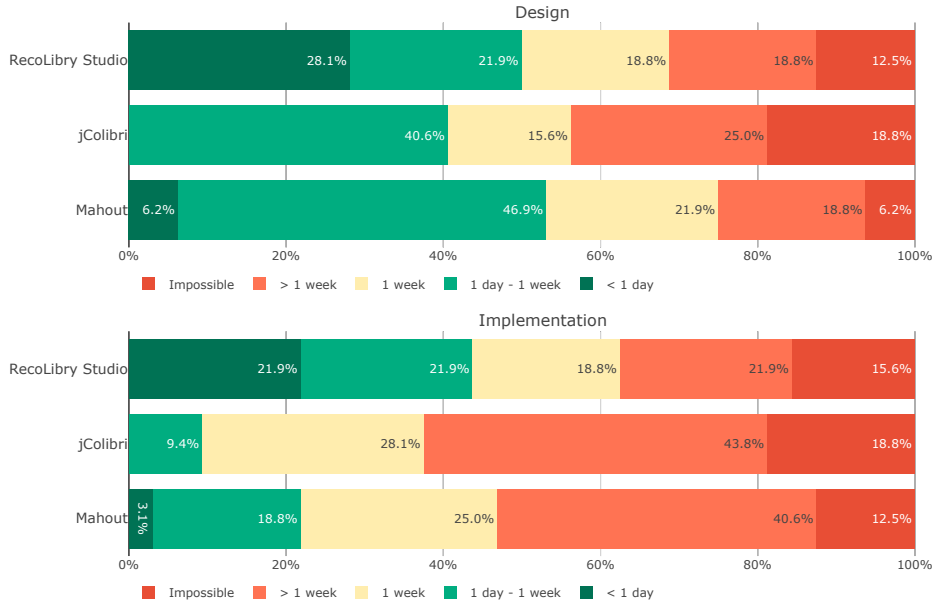

Figura 7.8: Estimated time to design (top) and implement (bottom) a recommender system using each framework.

RecoLibry-studio.

7.4.2.5. Evaluation of RecoLibry-studio

To evaluate our platform, we evaluated the RecoLibry-studio tool. In this section, we enumerate the main results, and the conclusions obtained, although the detail of this experiment and its results are described in Chapter 15 (Jorro-Aragoneses et al., 2020).

Thirty-two users participated in this experiment. All of them had a basic level in developing applications. In this experiment, we explained how to build a movie recommender system using three different frameworks: Mahout, jCOLIBRI, and RecoLibry-studio. Next, they completed a questionnaire where they marked how long they could take to develop a recommender system with each framework, and which one of them understood better.

One of the objectives that we defined in RecoLibry Suite tools was increasing the efficiency to design and implement a recommender system. Based on the users' answers that Figure 7.8 shows, we could observe that users considered RecoLibry-studio can reduce the time to build recommender systems. In the design process (Figure 7.8, top chart), 28.1 % percent of users considered that they spent less than one day to build a recommender system in RecoLibry-studio. Only 6.2 % of users considered the same with

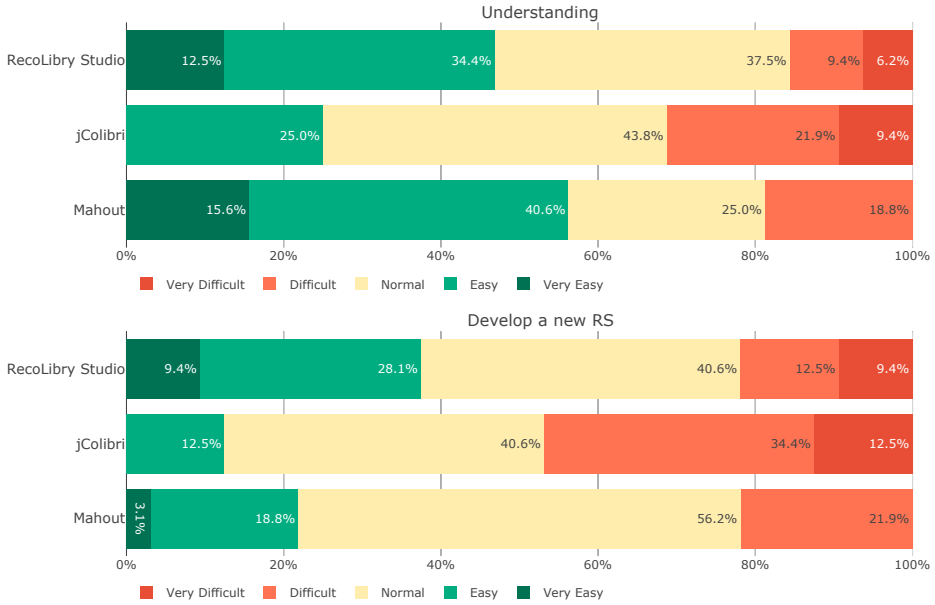

Figura 7.9: Difficulty to undesrtand (top) and to create (bottom) a new recommender system using each framework.

Mahout. In the developing process (Figure 7.8, bottom chart), we obtained similar results. 21.9 % of users considered spent less than one day to develop a recommender system in RECOLIBRY-STUDIO, and 3.1 % using Mahout. Although these answers were estimations, we could conclude that RECOLIBRY-STUDIO improves the efficiency of building a recommender system.

Another objective of RECOLIBRY-STUDIO was that users should understand better how to build and work recommender systems. The top chart in Figure 7.9 (top chart) shows that Mahout seems to be easiest to understand (56.6 %), followed by RECOLIBRY-STUDIO (46.9 %) and jCOLIBRI (25 %). This result may be caused because we are comparing two programming frameworks (Mahout and jCOLIBRI) with a graphics tool (RECOLIBRY-STUDIO) that encapsulates the behavior of our library. On the other hand, Figure 7.9 (bottom chart) shows that users considered RECOLIBRY-STUDIO easy or very easy to build recommender systems (37.5 %), followed by Mahout (21.9 %) and jCOLIBRI (12.5 %). This result confirmed our hypothesis that RECOLIBRY-STUDIO could be a simple tool to build recommender systems.

Despite these results, there are subjective opinions, and they can produce some threats. We decided to realize this experiment because it is complicated to prepare another with more objective measures, for example, measure the time of developing a recommender system with each framework. In this case, it is necessary for a group of users with the same level of programming skills, and, probably, it could spend more than one day. Both features make other

problems like the bias of users' knowledge and the fatigue of them. After an analysis of different approachs, we found our evaluation methodology a good solution to evaluate our system to a higher number of users and to make this in a one-day session.

Chapter 8

Conclusions and Future Work

The main objective of this Ph.D. thesis is to create a knowledge-based platform to help users with different programming levels to build recommender systems. To complete this objective, we define preliminary objectives to understand how recommender systems work and the necessary steps to build them. This chapter lists all results obtained for each objective defined at the beginning of this Ph.D. thesis. Part III collate all publications where we showed these results to the scientific community.

In this chapter, we make a summary of our contributions, and we analyze the conclusions obtained in our research work. In Section 8.1, we summary the objectives raised and the contribution made for each one. Next, Section 8.2 proposes some future research lines that could be raised from our work. Finally, we present some closing conclusions of this Ph. D. thesis in Section 8.3.

8.1. Contributions summary

In this section, we have enumerated all objectives defined in our research. For each objective, we have included a short description and a list of contributions. We have enumerated this information in the following list:

- **Objective 1 (O-1): Understanding the recommender system building process. It is necessary to analyze techniques used in recommender systems, identify problems in the building process, and analyze the features of common frameworks used to build these systems.** In Section Section 7.1.1, we have examined different recommender techniques and the most critical problems founded in these systems. Also, in Section 7.1.3 and Chapter 15 (Jorro-Aragoneses et al., 2020), we have presented a detailed study of common frameworks used to build recommender systems.
    - **Contribution 1: Study of classic and context-aware re-**

commender systems, including the different techniques that they use. In Section 7.1.1, we have analyzed the most common implemented recommender systems. In this study, we have enumerated different classic recommender systems used, and we have had a review of different techniques that include contextual information.

- **Contribution 2: Classify and analyze the main problems founded in the implementation process of recommender systems, including some solutions founded in the literature.** In Section 7.1.2, we have made a study of the most common problems founded in the developing process of recommender systems. We have foucused in sparsity, long-tail, and cold-start problems.
- **Contribution 3: Study of different frameworks to build recommender systems. This study classifies their features, and it allowed us to define the features of our platform developed to O-5.** In Section 7.1.3 and Chapter 15 (Jorro-Aragoneses et al., 2020), we have made a study of all open-source and academic frameworks made since 2011. For each framework, we analyzed the following features: techniques included, the inclusion of a graphical user interface, the possibility of implement new techniques, tools to evaluate the systems developed, and tools to deploy them automatically.

- **Objective 2 (O-2): Develop a recommender system as a case study that uses contextual information. This recommender system allows testing the components to include in the theoretical model to implement in O-4.** In Section 7.2, we have described a context-aware recommender system implemented as a case study.

  - **Contribution 4: Develop a context-aware recommender system of tourist and leisure plans in Madrid.** Section 7.2 describes Madrid Live, a context-aware recommender system that recommends tourist and leisure plans in Madrid. It uses the user preferences and the contextual information to return a tourist plan. Chapter 9 (Jorro-Aragoneses et al., 2017a) explains Madrid Live in detail.

- **Objective 3 (O-3): Develop different solutions to resolve the cold-start problem in recommender systems.** Section 7.3 describes a set of proposed solutions to resolve the cold-start problem. In Section 7.1.2, we explain that it is the most common problem in recommender systems.

- **Contribution 5: Propose a CBR method to resolve the cold-start problem in the user information of recommender systems.** In Section 7.3.1, we propose a solution to resolve the cold-start problem in users' information. In this case, we propose a methodology to build different case bases and infer missing information with anonymous users. We have presented this methodology in Chapter **??** (Jorro-Aragoneses et al., 2014) and Chapter 11 (Jorro-Aragoneses et al., 2015).
- **Contribution 6: Propose a methodology to build an initial case-based to resolve the cold-start problem in the items information of recommender systems.** Section 7.3.2 contains the description of a methodology to build an initial case base to the Madrid Live system. This methodology uses user reviews to make a set of cases to resolve de cold-start problem in the items catalog. Chapter 12 (Jorro-Aragoneses et al., 2018) describes this methodology.

▪ **Objective 4 (O-4): Design a component-based model to describe the recommender system building process and implement it in an ontology.** After the results of **O-1** and **O-2**, we have to build a theoretical model that describes how to build recommender systems using components. Section 7.4.1 describes this model.

- **Contribution 7: Propose an ontology to define the building process of recommender systems based on components.** In Section 7.4.1, we explain RecOnto, an ontology to model recommender systems using components. This ontology defines the functionality of each component and a set of dependencies to combine them. We have presented this ontology are the Chapter 13 (Jorro-Aragoneses et al., 2017a) and Chapter 15 (Jorro-Aragoneses et al., 2020). Also, we have published RecOnto in a public repository[1].

▪ **Objective 5: Develop a platform to help users designing, developing, and deploying recommender systems based on the theoretical model.** It was the main goal defined in our research. Section 7.4 presents RecoLibry Suite, a methodology and a set of tools to build recommender systems. This platform contains three tools developed to cover the steps described in this methodology.

- **Contribution 8: Create a framework to build recommender systems based on the components defined in the theoretical model.** In Chapter 4, we describe the first tool of our

[1] https://github.com/UCM-GAIA/RecOnto

platform: RECOLIBRY-CORE. This tool is a JAVA framework that implements the components defined in the theoretical model and implements the architecture to combine them. Chapter 14 (Jorro-Aragoneses et al., 2019) and Chapter 15 (Jorro-Aragoneses et al., 2020) describe this framework in detail. Also, this framework is public[2], and developers can use it. This repository contains a user manual and some recommender systems as examples.

- **Contribution 9: Develop a graphical user interface to design and build recommender systems in a semi-automatic way.** The second tool is RECOLIBRY-STUDIO. It is a web application that assists users in the building process of recommender systems. The tool is oriented to users with a low programming skill level. Chapter 4 and Chapter 15 (Jorro-Aragoneses et al., 2020) explains how works RECOLIBRY-STUDIO.
- **Contribution 10: Develop a deploy mechanism using web services to execute the recommender systems automatically from a configuration file.** The last tool is RECOSERVER. It is a server configuration that deploys automatically all recommender systems build by RECOLIBRY-CORE and RECOLIBRY-STUDIO. RECOSERVER builds an API to use the recommender systems in external applications. Chapter 4 describes how RECOSERVER works. Also, this contribution was presented in Chapter 15 (Jorro-Aragoneses et al., 2020).

Although we have completed all objectives proposed in this Ph. D. thesis, there are some research lines to continue our work in the future. In the next section, we present these research lines.

8.2. Future Work

After finishing this Ph. D. thesis, we propose the next lines of future work that could be interesting to improve the recommender system Madrid Live and the platform RECOLIBRY SUITE, and we think it is worth to investigate.

8.2.1. Knowledge adaptation to accesible tourism plans

One of the advantages of using a CBR system in Madrid Live is that it allows us to obtain an implicit knowledge about tourism plans, as we explained in Section 7.2. However, it is not easy to add more knowledge in the system, for example, accessible plans. There are two main methods to resolve this problem. The first one is creating a new case base where each case contains the new information. The second one is to add an adaptation

[2]https://github.com/UCM-GAIA/RecoLibry-Core

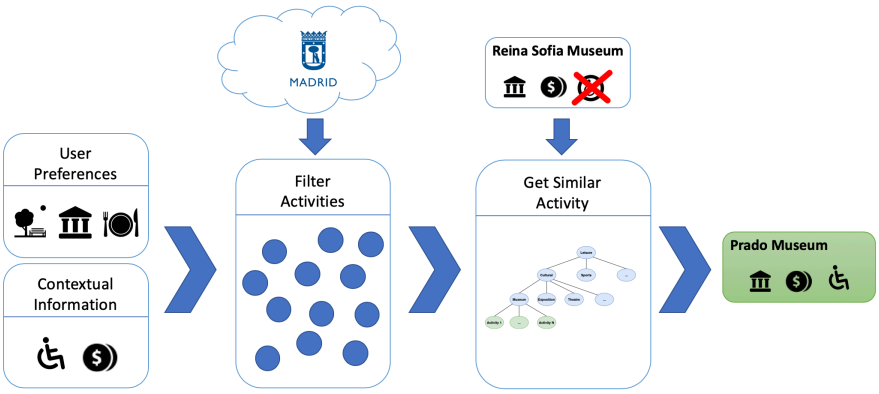

Figura 8.1: Proposed adaptation process to include accessible activities in the returned tourist plans at Madrid Live.

process in the recommender system to modify the recommended plan based on new knowledge.

In Chapter 10 (Jorro-Aragoneses y Bautista-Blasco, 2018), we analyse this second method and we proposed to include a new adaptation step in Madrid Live. This adaptation step can modify the real plans obtained by Madrid Live to make these plans accessible. The main goal is to reuse the implicit knowledge contained in Madrid Live and change the activities that are not accessible.

Figure 8.1 shows a schema of the adaptation process module proposed. This module receives a plan recommended by Madrid Live. Next, the new module search for which activities are not accessible, and it replaces each one to other accessible activities. The adaptation module uses taxonomy to replace these activities with other similar activities.

8.2.2. Explanation system in RecoLibry-studio

In Section 7.4.2.3, we have explained that RecoLibry-studio is oriented to users who do not have a high level of knowledge in recommender systems and their building process. For this reason, RecoLibry-studio uses the ontology RecOnto to filter the recommender techniques that a designer can apply based on the components defined before. However, we could include in RecoLibry-studio a system to explain why it is shown some techniques and not others.

In the last years, the explainable intelligent system research area has been growing, as it is explained in Miller (2019). The first reason for this growth is because the inclusion of explanations in intelligent systems increases the

trust of users. Also, if we included explanations in RECOLIBRY-STUDIO, we could use it as a tool to learn how to build a recommender system. One way to do that is to analyze the set of dependencies in RECONTO. Based on this analysis, we can generate an explanation understandable by the user. We could work in methodologies to obtain the RECONTO dependencies that filtered a technique, and how to show it to a user in RECOLIBRY-STUDIO. This explanation system would open up a wide range of possibilities to guide the work to improve RECOLIBRY-STUDIO.

8.2.3. Data analysis tools in RECOLIBRY SUITE

We have explained in Section 7.1.1 that some recommender techniques have modifications to resolve some data problems. For example, when a recommender system has the cold-start problem, as we have explained in this thesis. A possible improvement in RECOLIBRY SUITE could be including a tool to analyze the data used in a recommender system. This tool could help users detecting data problems like sparsity, long-tail distribution, unused attributes, etc. Also, users could modify a dataset with this tool.

A data analysis tool can be included as a step in RECOLIBRY-STUDIO to improve the filter of recommender techniques. Currently, RECONTO supports the classification of recommender techniques based on data features. It could increase the catalog of recommender techniques included in the platform RECOLIBRY SUITE.

8.2.4. Facilitate the inclusion of new components in RECOLIBRY SUITE

In the platform presented in this Ph. D. thesis, there are two tools where the components of recommender systems are defined. Firstly, RECONTO contains theoretical descriptions of each component, and it defines the relationship with the other components. On the other hand, RECOLIBRY-CORE implements the functionality of these components to build the final recommender systems. This design has a problem when a developer wants to add a new component. Currently, it should add this component into both tools. Especially if the developer wants to use the new component in RECOLIBRY-STUDIO.

To resolve this problem, we could work in two different ways. The first one is creating a system that updates the components information in RECONTO based on the new complements implemented in RECOLIBRY-CORE. Another possibility is developing a system that writes a component template in RECOLIBRY-CORE based on the information included in RECONTO. Then, the developer should implement the functionality of this component. Both ways have as objective to facilitate to software developers and scientific community improve the RECOLIBRY SUITE platform with new recommender

techniques because the area of recommender systems is large and in continuous development.

8.3. Closing conclussions

In conclusion, we implemented RecoLibry Suite, a knowledge-based platform to help users in the building process of recommender systems. Based on the contributions presented in this Ph. D. thesis, that are collected in the publications presented in Part III, we have the following conclusions:

1. Our research work makes a complete study of the related work, and it allows us to make a theoretical model that describes the recommender system domains.
2. We propose novel methodologies to resolve the cold-start problem.
3. The theoretical model explains in detail how to build recommender systems. Also, this model creates a unique vocabulary to define all components included in recommender systems.
4. The tools of RecoLibry Suite and the developing methodology help to users in the recommender system building process, including users with a low programming skill level.
5. The theoretical model and the framework are available in a public repository.
6. All results of our research have been presented to the scientific community.

After an analysis of the recommender systems developing process, we have researched to define a model and implement a set of tools to facilitate the building process of these systems. Also, we have demonstrated that our platform simplifies the developing process to users with different profiles. However, there are many features to improve, and we could add more engaging functionalities. In this Ph. D. thesis, we have successfully resolved all objectives proposed at the beginning of our research, and we have provided some ideas to expand our work in the future.

Bibliografía

Aamodt, A. y Plaza, E. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AI Communications*, vol. 7(1), páginas 39–59, 1994. ISSN 0921-7126.

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M. et al. Tensorflow: a system for large-scale machine learning. En *OSDI*, vol. 16, páginas 265–283. 2016.

Adomavicius, G. y Tuzhilin, A. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans Knowl Data Eng*, vol. 17(6), páginas 734–749, 2005. ISSN 10414347.

Adomavicius, G. y Tuzhilin, A. Context-aware recommender systems. En *Recommender Systems Handbook, Second Edition*, páginas 191–226. Springer US, 2015.

Aggarwal, C. C. Content-Based Recommender Systems. En *Recommender Systems*, páginas 139–166. Springer International Publishing, Cham, 2016.

Alshammari, G., Jorro-Aragoneses, J. L., Kapetanakis, S., Petridis, M., Recio-García, J. A. y Díaz-Agudo, B. A hybrid CBR approach for the long tail problem in recommender systems. En *Case-Based Reasoning Research and Development*, vol. 10339 LNAI, páginas 35–45. 2017. ISBN 9783319610290. ISSN 16113349.

Apache. Apache spark: Lightning-fast cluster computing. `http://spark.apache.org`, 2016.

Lopez-de Arenosa, P., Díaz-Agudo, B. y Recio-García, J. A. CBR Tagging of Emotions from Facial Expressions. En *ICCBR*, vol. 8765, páginas 245–259. 2014. ISBN 978-3-319-11208-4 978-3-319-11209-1. ISSN 16113349.

Baltrunas, L., Kaminskas, M., Ludwig, B., Moling, O., Ricci, F., Aydin, A., Lüke, K.-H. y Schwaiger, R. InCarMusic: Context-Aware

Music Recommendations in a Car. páginas 89–100. Springer, Berlin, Heidelberg, 2011.

BENNETT, J., LANNING, S. ET AL. The netflix prize. En *Proceedings of KDD cup and workshop*, vol. 2007, página 35. New York, NY, USA., 2007.

BOBADILLA, J., ORTEGA, F., HERNANDO, A. y GUTIÉRREZ, A. Recommender systems survey. *Knowledge-Based Systems*, vol. 46, páginas 109–132, 2013. ISSN 09507051.

BOBADILLA, J., SERRADILLA, F., HERNANDO, A. ET AL. Collaborative filtering adapted to recommender systems of e-learning. *Knowledge-Based Systems*, vol. 22(4), páginas 261–265, 2009.

BORRÀS, J., MORENO, A. y VALLS, A. Intelligent tourism recommender systems: A survey. *Expert Syst Appl*, vol. 41(16), páginas 7370–7389, 2014. ISSN 09574174.

BOUSBAHI, F. y CHORFI, H. MOOC-Rec: A Case Based Recommender System for MOOCs. *Procedia - Social and Behavioral Sciences*, vol. 195, páginas 1813–1822, 2015. ISSN 18770428.

BREESE, J. S., HECKERMAN, D. y KADIE, C. Empirical analysis of predictive algorithms for collaborative filtering. En *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, páginas 43–52. Morgan Kaufmann Publishers Inc., 1998.

BURKE, R. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, vol. 12(4), páginas 331–370, 2002.

CARO-MARTINEZ, M. y JIMENEZ-DIAZ, G. Similar users or similar items? Comparing similarity-based approaches for recommender systems in online judges. En *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10339 LNAI, páginas 92–107. Springer Verlag, 2017. ISBN 9783319610290. ISSN 16113349.

CASTELLUCCIO, M. The music genome project. *Strategic Finance*, páginas 57–59, 2006.

CHAN, S., STONE, T., SZETO, K. P. y CHAN, K. H. PredictionIO. En *CIKM 2013*, páginas 2493–2496. ACM Press, New York, New York, USA, 2013. ISBN 9781450322638.

COLOMBO-MENDOZA, L. O., VALENCIA-GARCÍA, R., RODRÍGUEZ-GONZÁLEZ, A., ALOR-HERNÁNDEZ, G. y SAMPER-ZAPATER, J. J. RecomMetz: A context-aware knowledge-based mobile recommender system for movie showtimes. *Expert Systems with Applications*, vol. 42(3), páginas 1202–1222, 2015.

DEY, A. K. Understanding and using context. *Personal and Ubiquitous Computing*, vol. 5(1), páginas 4–7, 2001. ISSN 16174909.

DOMINGUES, M. A., GOUYON, F., JORGE, A. M., LEAL, J. P., VINAGRE, J., LEMOS, L. y SORDO, M. Combining usage and content in an online recommendation system for music in the long tail. *International Journal of Multimedia Information Retrieval*, vol. 2(1), páginas 3–13, 2013.

DROSATOS, G., EFRAIMIDIS, P. S., ARAMPATZIS, A., STAMATELATOS, G. y ATHANASIADIS, I. N. Pythia: A Privacy-Enhanced Personalized Contextual Suggestion System for Tourism. En *2015 IEEE 39th Annual Computer Software and Applications Conference*, páginas 822–827. IEEE, 2015.

EKSTRAND, M. D., RIEDL, J. T. y KONSTAN, J. A. Collaborative Filtering Recommender Systems. *Foundations and Trends® in Human–Computer Interaction*, vol. 4(2), páginas 81–173, 2010. ISSN 1551-3955.

ELO, A. The Rating of Chess Players, Past and Present. *New York: Arco*, 1987.

FRØKJÆR, E., HERTZUM, M. y HORNBÆK, K. Measuring usability: are effectiveness, efficiency, and satisfaction really correlated? En *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, páginas 345–352. ACM, 2000.

FUNK, S. Netflix update: Try this at home. `https://sifter.org/simon/journal/20061211.html`, 2006.

GANTNER, Z., RENDLE, S., FREUDENTHALER, C. y SCHMIDT-THIEME, L. MyMediaLite. *RecSys 2011*, página 305, 2011.

DE GEMMIS, M., LOPS, P., MUSTO, C., NARDUCCI, F. y SEMERARO, G. *Semantics-Aware Content-Based Recommender Systems*, páginas 119–159. Springer US, Boston, MA, 2015.

GOLDBERG, D., NICHOLS, D., OKI, B. M. y TERRY, D. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, vol. 35(12), páginas 61–71, 1992.

GOMEZ-URIBE, C. A. y HUNT, N. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)*, vol. 6(4), página 13, 2016.

GRUBER, T. *Ontology*. Springer, 2009.

GUO, G., ZHANG, J., SUN, Z. y YORKE-SMITH, N. Librec: A java library for recommender systems. En *UMAP Workshops*, vol. 4. 2015.

HARIRI, N., MOBASHER, B. y BURKE, R. Context-aware music recommendation based on latenttopic sequential patterns. En *Proceedings of the sixth ACM conference on Recommender systems - RecSys '12*, página 131. ACM Press, New York, New York, USA, 2012.

HARPER, F. M. y KONSTAN, J. A. The MovieLens Datasets. *ACM Trans Interact Intell Syst*, vol. 5(4), páginas 1–19, 2015. ISSN 21606455.

HE, J., LI, X., LIAO, L., SONG, D. y CHEUNG, W. K. Inferring a personalized next point-of-interest recommendation model with latent behavior patterns. *30th AAAI Conference on Artificial Intelligence, AAAI 2016*, páginas 137–143, 2016.

HE, R. y MCAULEY, J. Fusing similarity models with markov chains for sparse sequential recommendation. *Proceedings - IEEE International Conference on Data Mining, ICDM*, páginas 191–200, 2017. ISSN 15504786.

HEINEMAN, G. T. y COUNCILL, W. T. Component-based software engineering. *Putting the pieces together, addison-westley*, página 5, 2001.

HUG, N. Surprise, a Python library for recommender systems. `http://surpriselib.com`, 2017.

JI, S., PAN, S., CAMBRIA, E., MARTTINEN, P. y YU, P. S. A Survey on Knowledge Graphs: Representation, Acquisition and Applications. páginas 1–25, 2020.

JING, L., WANG, P. y YANG, L. Sparse probabilistic matrix factorization by laplace distribution for collaborative filtering. En *Twenty-Fourth International Joint Conference on Artificial Intelligence*. 2015.

JORRO-ARAGONESES, J. L. y BAUTISTA-BLASCO, S. Adaptation process in context-aware recommender system of accessible tourism plan. En *Current Trends in Web Engineering: ICWE 2018*, vol. 11153 LNCS, páginas 292–295. Springer, Cham, Cáceres (Spain), 2018.

JORRO-ARAGONESES, J. L., CERON-RIOS, G. M., DIAZ-AGUDO, B., RECIO-GARCÍA, J. A. y LOPEZ-GUTIERREZ, D. M. Reconto: An ontology to model recommender systems and its components. En *IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, vol. 2017-Novem, páginas 815–821. Springer, Boston (USA), 2017a.

JORRO-ARAGONESES, J. L., DIAZ-AGUDO, B. y RECIO-GARCÍA, J. A. Optimization of a CBR system for emotional tagging of facial expressions. En *19th UK Workshop on Case-Based Reasoning*, páginas 1–14. Cambridge, United Kingdom, 2014.

JORRO-ARAGONESES, J. L., DÍAZ-AGUDO, B. y RECIO-GARCÍA, J. A. Addressing the cold-start problem in facial expression recognition. En *23rd International Conference , ICCBR 2015*, vol. 9343, páginas 197–211. Springer, 2015.

JORRO-ARAGONESES, J. L., DIAZ-AGUDO, B. y RECIO-GARCÍA, J. A. Madrid live: A context-aware recomendar system of leisure plans. En *IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, vol. 2017-Novem, páginas 796–801. Springer, Boston (USA), 2017b.

JORRO-ARAGONESES, J. L., JIMENEZ-DÍAZ, G., RECIO-GARCÍA, J. A. y DÍAZ-AGUDO, B. Case Base Elicitation for a Context-Aware Recommender System. En *26th International Conference, ICCBR 2018*, vol. 11156 LNAI, páginas 170–185. Springer, Cham, Stockholm (Sweden), 2018.

JORRO-ARAGONESES, J. L., RECIO-GARCÍA, J. A., DÍAZ-AGUDO, B. y JIMENEZ-DÍAZ, G. RecoLibry-Core: A component-based framework for building recommender systems. *Knowledge-Based Systems*, 2019. ISSN 0950-7051.

JORRO-ARAGONESES, J. L., RECIO-GARCÍA, J. A., DÍAZ-AGUDO, B. y JIMENEZ-DÍAZ, G. RecoLibry Suite: a set of intelligent tools for the development of recommender systems. *Journal of Automated Software Engineering*, 2020. ISSN 0928-8910.

KARAT, C.-M., BLOM, J. O. y KARAT, J. *Designing personalized user experiences in eCommerce*, vol. 5. Springer Science & Business Media, 2004.

KHUSRO, S. y ALI, Z. Recommender systems: issues, challenges, and research opportunities. *Springer*, 2006.

KIM, D., PARK, C., OH, J. y YU, H. Deep hybrid recommender systems via exploiting document context and statistics of items. *Information Sciences*, vol. 417, páginas 72–87, 2017.

KIRK, J. Tensorrec. `https://github.com/jfkirk/tensorrec`, 2018.

KOLODNER, J. An Introduction to Case-Based Reasoning. vol. 6, páginas 3–34, 1992. ISSN 00071447.

KOLODNER, J. *Case-based reasoning*. Morgan Kaufmann, 2014.

KOREN, Y. y BELL, R. Advances in collaborative filtering. En *Recommender Systems Handbook, Second Edition*, páginas 77–118. Springer US, Boston, MA, 2015. ISBN 9781489976376.

KULA, M. Metadata embeddings for user and item cold-start recommendations. *CEUR 2015*, vol. 1448, páginas 14–21, 2015. ISSN 16130073.

KULA, M. Spotlight. `https://github.com/maciejkula/spotlight`, 2017.

LANDSET, S., KHOSHGOFTAAR, T. M., RICHTER, A. N. y HASANIN, T. A survey of open source tools for machine learning with big data in the Hadoop ecosystem. *Journal of Big Data*, vol. 2(1), página 24, 2015. ISSN 21961115.

LECUN, Y., BENGIO, Y. y HINTON, G. Deep learning. 2015.

LEE, D. D. y SEUNG, H. S. Algorithms for non-negative matrix factorization. En *Advances in neural information processing systems*, páginas 556–562. 2001.

LIKA, B., KOLOMVATSOS, K. y HADJIEFTHYMIADES, S. Facing the cold start problem in recommender systems. *Expert Syst Appl*, vol. 41(4 PART 2), páginas 2065–2073, 2014. ISSN 09574174.

LOPS, P., DE GEMMIS, M. y SEMERARO, G. Content-based recommender systems: State of the art and trends. En *Recommender systems handbook*, páginas 73–105. Springer, 2011.

LUO, X., ZHOU, M., LI, S., YOU, Z., XIA, Y. y ZHU, Q. A Nonnegative Latent Factor Model for Large-Scale Sparse Matrices in Recommender Systems via Alternating Direction Method. *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27(3), páginas 579–592, 2016. ISSN 2162-237X.

MANEEROJ, S. y TAKASU, A. Hybrid recommender system using latent features. En *2009 International Conference on Advanced Information Networking and Applications Workshops*, páginas 661–666. IEEE, 2009.

MASSE, M. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. .ºReilly Media, Inc.", 2011.

MERKEL, D. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, vol. 2014(239), página 2, 2014.

MILLER, T. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, vol. 267, páginas 1 – 38, 2019. ISSN 0004-3702.

MU, R. A Survey of Recommender Systems Based on Deep Learning. *IEEE Access*, vol. 6, páginas 69009–69022, 2018. ISSN 21693536.

ONO, C., KUROKAWA, M., MOTOMURA, Y. y ASOH, H. A Context-Aware Movie Preference Model Using a Bayesian Network for Recommendation and Promotion. En *User Modeling 2007*, páginas 247–257. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

Van den Oord, A., Dieleman, S. y Schrauwen, B. Deep content-based music recommendation. En *Advances in neural information processing systems*, páginas 2643–2651. 2013.

O'Reilly, T. *What is web 2.0*. .º'Reilly Media, Inc.", 2009.

Owen, S., Anil, R., Dunning, T. y Friedman, E. *Mahout in Action*. 2011. ISBN 9781935182689.

Palumbo, E., Rizzo, G., Antipolis, S. y Baralis, E. Predicting your Next Stop-Over from Location-Based Social Network Data with Recurrent Neural Networks Location-Based Social Network Data. páginas 1–8, 2017.

Panniello, U., Tuzhilin, A., Gorgoglione, M., Palmisano, C. y Pedone, A. Experimental comparison of pre-vs. post-filtering approaches in context-aware recommender systems. En *Proceedings of the third ACM conference on Recommender systems*, páginas 265–268. ACM, 2009.

Park, S.-T. y Chu, W. *Pairwise Preference Regression for Cold-start Recommendation*. 2009. ISBN 9781605584355.

Park, Y.-J. The adaptive clustering method for the long tail problem of recommender systems. *IEEE Transactions on Knowledge and Data Engineering*, vol. 25(8), páginas 1904–1915, 2012.

Park, Y.-J. y Tuzhilin, A. The long tail of recommender systems and how to leverage it. En *Proceedings of the 2008 ACM conference on Recommender systems - RecSys '08*, página 11. ACM, New York, New York, USA, 2008. ISBN 9781605580937. ISSN 03029743.

Paszke, A., Gross, S., Chintala, S. y Chanan, G. Pytorch. 2017.

Pazzani, M. J. y Billsus, D. Content-Based Recommendation Systems. En *The Adaptive Web*, páginas 325–341. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-72078-2.

Pheng, T., Husain, W. y Zakaria, N. Recommender System for Personalised Wellness Therapy. *International Journal of Advanced Computer Science and Applications*, vol. 4(9), 2013. ISSN 2158107X.

Popescul, A., Pennock, D. M. y Lawrence, S. Probabilistic models for unified collaborative and content-based recommendation in sparse-data environments. En *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, UAI'01, páginas 437–444. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001. ISBN 1-55860-800-1.

Prasanna, D. R. Dependency injection. 2009.

QUADRANA, M., CREMONESI, P., MILANO, P., JANNACH, D. y KLAGENFURT, A. A. U. Sequence-Aware Recommender Systems. vol. 1(1), 2018.

QUIJANO-SÁNCHEZ, L., BRIDGE, D., DÍAZ-AGUDO, B. y RECIO-GARCÍA, J. A. A case-based solution to the cold-start problem in group recommenders. En *International Conference on Case-Based Reasoning*, páginas 342–356. Springer, 2012.

RECIO-GARCÍA, J. A., DÍAZ-AGUDO, B. y GONZÁLEZ-CALERO, P. A. Prototyping recommender systems in jColibri. En *RecSys 2008*, página 243. ACM Press, New York, New York, USA, 2008. ISBN 9781605580937.

RECIO-GARCÍA, J. A., GONZÁLEZ-CALERO, P. A. y DÍAZ-AGUDO, B. Jcolibri2: A framework for building Case-based reasoning systems. *Science of Computer Programming*, vol. 79, páginas 126–145, 2014. ISSN 01676423.

RESNICK, P. y VARIAN, H. R. Recommender systems. *Communications of the ACM*, vol. 40(3), páginas 56–58, 1997. ISSN 00010782.

RICCI, F., SHAPIRA, B. y ROKACH, L. Recommender systems: Introduction and challenges. En *Recommender Systems Handbook, Second Edition*, vol. 54, páginas 1–34. Springer US, Boston, MA, 2015. ISBN 9781489976376.

RIESBECK, C. y SCHANK, R. Inside Case-based Reasoning. *Lawrence Erlbaum Associates, Hillsdale, NJ*, 1989.

SARWAR, B., KARYPIS, G., KONSTAN, J. y RIEDL, J. Application of dimensionality reduction in recommender system-a case study. Informe técnico, Minnesota Univ Minneapolis Dept of Computer Science, 2000.

SARWAR, B. M., KARYPIS, G., KONSTAN, J. A., RIEDL, J. ET AL. Item-based collaborative filtering recommendation algorithms. *Www*, vol. 1, páginas 285–295, 2001.

SARWAT, M., MORAFFAH, R., MOKBEL, M. F. y AVERY, J. L. Database system support for personalized recommendation applications. En *ICDE 2017*, páginas 1320–1331. 2017. ISBN 9781509065431. ISSN 10844627.

SCHAFER, J., KONSTAN, J., CONFERENCE ON ..., J. R. O. T. S. A. y 1999, U. Recommender systems in e-commerce. *1st ACM conference on Electronic commerce.*, páginas 158–166, 1999.

SCHAFER, J. B., FRANKOWSKI, D., HERLOCKER, J. y SEN, S. Collaborative filtering recommender systems. En *The adaptive web*, páginas 291–324. Springer, 2007.

SCHEIN, A. I., POPESCUL, A., UNGAR, L. H. y PENNOCK, D. M. Methods and metrics for cold-start recommendations. En *Proceedings of the 25th*

annual international ACM SIGIR conference on Research and development in information retrieval, páginas 253–260. ACM, 2002.

SCHELTER, S. y OWEN, S. Collaborative filtering with apache mahout. *Proc. of ACM RecSys Challenge*, 2012.

SEAN, O. Oryx 2. `https://github.com/OryxProject/oryx`, 2018.

SHAHABI, C. y CHEN, Y.-S. Web information personalization: Challenges and approaches. En *International Workshop on Databases in Networked Information Systems*, páginas 5–15. Springer, 2003.

SHI, L. Trading-off among accuracy, similarity, diversity, and long-tail: a graph-based recommendation approach. En *Proceedings of the 7th ACM conference on Recommender systems*, páginas 57–64. ACM, 2013.

SHIMODAIRA, H. Similarity and recommender systems. *School of Informatics, The University of Eidenburgh*, vol. 21, 2014.

SHISHEHCHI, S., BANIHASHEM, S. Y. y ZIN, N. A. M. A proposed semantic recommendation system for e-learning: A Rule and ontology based e-learning recommendation system. En *Proceedings 2010 International Symposium on Information Technology - Visual Informatics, ITSim'10*, vol. 1. 2010. ISBN 9781424467181.

SHIU, S. C. y PAL, S. K. Case-based reasoning: Concepts, features and soft computing. *Applied Intelligence*, vol. 21(3), páginas 233–238, 2004. ISSN 0924669X.

SMITH, B. y LINDEN, G. Two decades of recommender systems at amazon. com. *Ieee internet computing*, vol. 21(3), páginas 12–18, 2017.

SZYPERSKI, C. Component software and the way ahead. *Foundations of component-based systems*, páginas 1–20, 2000.

SZYPERSKI, C., GRUNTZ, D. y MURER, S. *Component software: beyond object-oriented programming*. Pearson Education, 2002.

TURRIN, R., CREMONESI, P., MILANO, P. D., CONDORELLI, A. y MILANO, P. D. Large Scale Music Recommendation. *Workshop on Large-Scale Recommender Systems (LSRS 2015) at ACM RecSys*, páginas 3–6, 2015.

VAN SETTEN, M., POKRAEV, S. y KOOLWAAIJ, J. Context-aware recommendations in the mobile tourist application compass. En *International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems*, páginas 235–244. Springer, 2004.

VANBRABANT, R. *Google Guice: agile lightweight dependency injection framework*. APress, 2008.

WANG, S.-L. y WU, C.-Y. Application of context-aware and personalized recommendation to implement an adaptive ubiquitous learning system. *Expert Systems with Applications*, vol. 38(9), páginas 10831–10838, 2011. ISSN 0957-4174.

WEI, J., HE, J., CHEN, K., ZHOU, Y. y TANG, Z. Collaborative filtering and deep learning based recommendation system for cold start items. *Expert Systems with Applications*, vol. 69, páginas 1339–1351, 2017. ISSN 09574174.

YANG, X., GUO, Y. y LIU, Y. Bayesian-inference-based recommendation in online social networks. *IEEE Transactions on Parallel and Distributed Systems*, vol. 24(4), páginas 642–651, 2013.

ZAFARANI, R., ABBASI, M. A. y LIU, H. *Social media mining: an introduction*. Cambridge University Press, 2014.

ZHANG, C., BI, J., XU, S., RAMENTOL, E., FAN, G., QIAO, B. y FUJITA, H. Multi-imbalance: An open-source software for multi-class imbalance learning. *Knowledge-Based Systems*, vol. 174, páginas 137 – 143, 2019.

ZHANG, S. y LEE, M. D. Cognitive models and the wisdom of crowds: A case study using the bandit problem. En *Ratio*, vol. 32. 2010.

ZIMMERMANN, A., LORENZ, A. y OPPERMANN, R. An Operational Definition of Context. *Modeling and Using Context*, páginas 558–571, 2007. ISSN 0302-9743.

Parte III

Publicaciones Presentadas

Capítulo 9

Madrid Live: a context-aware recommender system of leisure plans

9.1. Cita completa

Jose L. Jorro-Aragoneses, Belén Díaz-Agudo, Juan A. Recio-García. *Madrid Live: a context-aware recommender system of leisure plans*. Proceedings of 29th IEEE Conference on Tools with Artificial Intelligence. Boston, United States. November 2017. P. 796-801.

9.2. Contribuciones de la publicación

En esta publicación, presentamos el sistema recomendador *Madrid Live*. Madrid Live es un sistema recomendador que utiliza información contextual para devolver a los usuarios planes turísticos en tiempo real. El desarrollo de Madrid Live nos ha servido para comprender cómo es el desarrollo de un sistema recomendador. Además, en esta publicación se incluye un análisis donde se observa como afecta la información contextual a las recomendaciones finales.

9.3. Contributions covered by this publication

In this publication, we present the recommender system *Madrid Live*. Madrid Live is a recommender system that includes contextual information to return tourism plans to users. The development of Madrid Live helps us to understand how to build real recommender systems. Also, in this publication, we include an analysis of how contextual information affects the final recommendations.

2017 International Conference on Tools with Artificial Intelligence

Madrid Live: a context-aware recommender system of leisure plans

Jose L. Jorro-Aragoneses
Universidad Complutense de Madrid
Madrid, Spain
Email: jljorro@ucm.es

Mª Belén Díaz-Agudo
Universidad Complutense de Madrid
Madrid, Spain
Email: belend@ucm.es

Juan A. Recio-García
Universidad Complutense de Madrid
Madrid, Spain
Email: jreciog@ucm.es

***Abstract*—Classical recommender systems focus on recommending the most relevant items to users. An active area of research proposes to complete the recommendation process by considering additional contextual information, such as time, location, budget, weather or social position. Researchers and practitioners in different domains have already recognized the great impact of contextual information in decision-making processes. In this paper, we focus on recommenders for tourism and leisure activities where contextual information plays a central role to modify the initial user preferences. We present Madrid Live, a context-aware recommender system (CARS) to recommend leisure activities in Madrid. In Madrid Live, users state their own restrictions and preferences to their plans. The system recommends the set of activities that satisfies these preferences together with the contextual knowledge. The main contributions of our approach are the contextual recommendation and the system explanation interface that allows the user to understand the recommendation process.**

I. Introduction

Classical recommender systems focus on either finding a match between an item's description and the user's preferences (content-based), or finding users with similar preferences (Collaborative Filtering) [1]. However, there is an active area of research on context-aware recommender systems (CARS) in which the information regarding the user environment is used in order to modify the initial user preferences. The main goal in this research area in recommender systems consists on taking advantage of additional factors that enrich the input query. These additional factors are known as contextual knowledge [2], [3].

Tourism is one of the most successful application domains for CARS [4]. In this domain there are a high number of different activity types and each type has different features. Furthermore, the majority of these activity types have restrictions such as timetables, weather conditions, price, etc. These dynamic properties make tourism one of the most popular domains in context-aware recommender systems.

In this paper, we present Madrid Live. It is a context-aware recommender system of leisure activity plans to make in Madrid. Madrid Live takes into account the user preferences and the contextual information of the user environment, just when the recommendation is computed. The plans to be recommended are representad as a timetable that contains a collection of activities to be performed by each user. At present, Madrid Live recommends 4 activity types in the plans: museums, parks, points of interest and restaurants. However, the modular architecture of Madrid Live allows to include further activity types in an simple way.

Including contextual knowledge in the recommendation process allows us to dynamically generate plans according to the current restrictions of the user. However, it has a side effect: plans have a lower similarity to the user preferences. Therefore, these recommendations will be unsatisfactory to the user. However, it can be explained to the user. The system could compute that there is a very suitable plan, with a significant higher similarity, that has not been recommended because of the context. The system could explain the possibilities to the user and let him/her decide if the context restrictions and preferences should be relaxed in order to find a more suitable plan.

The recommender process of Madrid Live is based on a Case-Based Reasoning (CBR) algorithm [5]. Madrid Live stores and reuses previous plans and it recommends these plans to users with similar preferences and contextual restrictions. The use of local similarity functions, one similarity function per feature used in Madrid Live to represent activities, enables the system to retrieve the most similar cases while taking into account the contextual restrictions and the user preferences. This approach implies that all recommendations created by Madrid Live accomplish all contextual restrictions and the user preferences in an efficient way. Finally, our system has an explanation module that can generate explanations based on the similarity functions of the CBR process. These explanations enable users to understand better the recommendations of the system.

In addition, we research how is the impact of context in the recommendation process. Our investigation demonstrates the following hypotheses:

- **H-1** The inclusion of context knowledge in the similarity function decreases the similarity of retrieved cases with respect to the user's preferences.
- **H-2** It is possible to identify which context restrictions have a higher impact in the performance of the recommendation system.

As we demonstrate in this paper, the use of context restrictions could return an unsatisfactory result for the user, although this result is the unique possible. For these reason, we implement an algorithm that generates explanations to the

2375-0197/17/31.00 ©2017 IEEE
DOI 10.1109/ICTAI.2017.00125

user. It explains why the system recommends a plan and how his/her contextual information modifies the recommendation.

To explain the proposed system and demonstrate these hypotheses, this paper is organized as follows: in Section II, we present some related works regarding context-aware recommender systems and the use of some CBR systems in leisure and tourism domains. Next, Madrid Live is described in Section III. This section explains the recommendation process and the generation of explanations. The Section IV explains the experiments made to demonstrate the hypotheses defined. Finally, Section V presents the conclusions obtained in our experiments and describes several task to be done as a future work.

II. Related Work

Commercial recommender systems, such as Tripomatic or TripAdvisor [6], recommend items based on preferences but they do not employ user-contextual information. Another tourism recommender system, Aurigo, can be integrated into review websites in order to improve the recommendations [7].

Contextual information extends the system query with information that is not included in the user preferences. The typical rating functions in recommender systems have two inputs: item description and user preferences. The main goal is to include the context in this function [8], [3]. This knowledge is very useful in mobile devices where we can obtain large amount of contextual information about the users [9]. On the other hand, thanks to the growth of social networks, the social context information can be taken into account in recommender systems [10], [11] in order to improve the results of group recommender systems [12].

Contextual information is used in many domains. It can be exploited, for instance, in domains that use contextual information in Technology Enhanced Learning (TEL) recommender system [13], or, in a mobile recommender system [14]. One of the first works into CARS for tourism was *COMPASS* [15] which included the mobile context information in tourism recommendations. At present, we can find some works in the same way such as *Pythia* [16].

III. Madrid Live

In this section, we describe the CBR process. Next, we explain how Madrid Live generates the explanations by reusing the similarity functions used in the CBR process.

A. CBR process

Recommendation in Madrid Live uses plans performed by other users in the past. To do this, the system uses a CBR module. This CBR module contains a case base where every plan performed by any user is stored. Each plan is stored as a case and it contains 2 elements: a description and a solution. The description is used to describe the plan, for example, what activity types are contained in the plan, when the plan starts, the location of the first activity, etc. The system uses this information to search the most similar case to the user preferences and the contextual restrictions. The solution stores the detailed plan that will be shown to the user after an optional adaptation step.

To make a new recommendation, the system creates a query that contains the contextual restrictions and the user preferences. The system compares this query to all the descriptions of cases contained in the case base. To do that, the system uses a similarity function that returns a value between 0 to 1, where 1 means that the case description and the query contains the same information and 0 means that the case description and the query are not similar at all. This way, the system retrieves the most similar case that the user has not previously performed in the past. The solution to this case will be the plan recommended.

Next, we will explain each element involved in this CBR process: query structure, case structure and the similarity functions.

1) Query structure: The query ($\mathcal{Q}$) contains the information used to find the most similar case. This query contains the following information: (1.)- contextual information about user obtained in the moment of the request (q^c); (2.)- the user preferences about each activity type (museum, restaurant, park and walks) (q^a); (3.)- the user preferences about museum types (q^m); and (4.)- his/her preferences about restaurant types (q^r).

$$\mathcal{Q} =< q^c, q^a, q^m, q^r > \quad (1)$$

The contextual information (q^c) in the query is composed of 6 attributes:

- *Start time (c_1^u):* time read in the user's smartphone and used to define the time to start the plan.
- *Finish time (c_2^u):* the user defines the time when he/she wants to finish the plan.
- *Location (c_3^u):* geographical location of the user obtained in the moment of the request by the GPS of his/her smartphone.
- *Weather (c_4^u):* based on the time to start the activity and the user location the system obtain the weather condition using a weather API. It is a tag that defines the weather conditions. To simplify, the weather conditions are defined either "good" or "bad".
- *Budget (c_5^u):* a tag that represents the money that the user expects to spend on this plan. The values are: cheap, regular and expensive.
- *Use of public transport (c_6^u):* a boolean value that indicates if s/he wants to use the public transport to visit each activity.

$$q^c =< c_1^u..c_6^u > \quad (2)$$

Next, the query adds the user preferences regarding his/her plans. The first preference attribute is the activity type preferences (q^a). It stores the preferred activity types of the user. It has 4 values: museums (a_1^u), restaurants (a_2^u), parks (a_3^u) and walks (a_4^u). Each one is a value between 0 and 1 and they are normalized to ensure that their total sum of them is 1.

$$q^a = < a_1^u..a_4^u > \quad (3)$$

TABLE I
EXAMPLE OF A QUERY IN MADRID LIVE

Attribute	Value
Time	11:00-16:30
Public Transport	Yes
User location	(-3.707, 40.413)
Budget	Regular
Weather	Good
Favourite Activities	Walks, Restaurants and Parks
Favourite Museums	N/A
Favourite Restaurants	American

Finally, the query contains the user preferences regarding museum categories (q^m) and food type (q^r) in restaurants. Each one has a list of values between 0 to 1 that represents the user preference of one museum category or food type. The museum category preferences contain 12 possible values and the food type preferences contain 6 choices.

$$q^m = < m_1^u..m_{12}^u > \quad (4)$$

$$q^r = < r_1^u..r_6^u > \quad (5)$$

Other activity types, parks and walks, do not need further details. Table I shows an example of a query in the CBR process.

Next, we will explain how the cases in the CBR process are represented.

2) Case Structure: As it has been explained before, a case ($\mathcal{C}$) is composed of a description ($\mathcal{D}$) and a solution ($\mathcal{S}$). The description contains the information to be compared to the query. On the other hand, the solution is the plan returned as the recommendation.

$$\mathcal{C} = < \mathcal{D}, \mathcal{S} > \quad (6)$$

The case description ($\mathcal{D}$) contains 4 attributes that summarize all the features of the plan stored in the solution. It contains: contextual restrictions (d^c), the activity types contained in the plan (d^a), the average of museum features (d^m) and restaurant features (d^r) contained in the plan.

$$\mathcal{D} = < d^c, d^a, d^m, d^r > \quad (7)$$

We explain the information represented each attribute in detail. Firstly, the contextual restriction (d^c) contains 6 attributes, one per each contextual information category used in Madrid Live:

- *Start time (c_1):* time when the first activity in the plan starts.
- *Finish time (c_2):* time when the last activity in the plan finishes .
- *Location Type (c_3):* defines the location type of activities. This information is used to determinate which cases are better depending on the weather. The values are *indoor* or *outdoor*.
- *First Location (c_4):* location of the first activity in the template.
- *Economic cost (c_5):* an estimation of the cost to realize this plan. It depends on the cost of each activity, for example, the ticket price.
- *Public transport (c_6):* it is a boolean value to indicate that all activities in the plan have a public transport station in a distance lower than the maximum walking distance (γ).

$$d^c = < c_1, c_2, c_3, c_4, c_5, c_6 > \quad (8)$$

The second information defined in the case description is the activity types contained in the plan (d^a). It is an array of 4 values. Each value represents the the percentage of each activity type contained in the plan. It has the same structure as the activity type preferences attribute in the query (q^a).

$$d^a = < a_1, a_2, a_3, a_4 > \quad (9)$$

Finally, the last 2 features in the case description are the type of museums included in the plan (d^m) and the type of restaurants (d^r). Both attributes are lists of values between 0 to 1 for each category in museum or restaurant. If there is more than one museum, the system calculates the average of all of them in this attribute. The restaurant attribute is analogues.

$$d^m = < m_1..m_{12} > \quad (10)$$

$$d^r = < r_1..r_6 > \quad (11)$$

This is an information required to describe the cases and search for the most similar plan to recommend. The solution for these cases ($\mathcal{S}$) is a list of concrete activities (a_i) with a start time and a finish time for each one.

$$\mathcal{S} = < a_1...a_n > \quad (12)$$

Next, we will explain the similarity functions used to compare the query and each case.

3) Similarity Functions: The query is compared to every case in the case base. A global similarity measure (the weighted average) aggregates the result of the corresponding local similarity values computed for every attribute [17]. The local similarity functions are divided into 3 groups: contextual similarity functions (sim_i^c), proportional activity type similarity function (sim^a) and museum/restaurant similarity function (sim^m, sim^r). The contextual similarity functions is defined by 5 functions:

- **Time similarity (sim_1^c):** It calculates the similarity between the timestamps defined in the query (t_q), and the times defined in the case (t_c). The difference between both times should be lower than the threshold (α_1), if not, the similarity is 0.

$$sim_1^c(t_q, t_c) = \begin{cases} \frac{|t_q - t_c|}{\alpha_1} & if \quad |t_q - t_c| \leq \alpha_1 \\ 0 & if \quad |t_q - t_c| > \alpha_1 \end{cases} \quad (13)$$

- **Public transport (sim_2^c):** Each user defines if s/he wants to use the public transport (c_6^u) and each case indicates if all its activities can be reached using public transport (c_6). The similarity function returns 1 if a user indicates that s/he wants to use the public transport and all activities in the plan have a public transport station. In other cases, the function returns 0:

$$sim_2^c(c_6^u, c_6) = \begin{cases} 1 & if \quad c_6^u \vee c_6 \\ 0 & if \quad i.o.c. \end{cases} \quad (14)$$

- **Location (sim_3^c):** The query defines the user location (c_3^u) using the GPS location given by his/her smartphone. On the other hand, each case description contains the location of the first activity (c_4). The similarity function used is based on the geographical distance between both locations. It must be less than the threshold (α_3). If this distance is higher, the result of the similarity is equal to the public transport similarity function.

$$sim_3^c(c_3^u, c_4) = \begin{cases} \beta & if \quad \beta \leq \alpha_3 \\ sim_2^c(c_6^u, c_6) & if \quad \beta > \alpha_3 \end{cases} \quad (15)$$

where

$$\beta \quad = \quad geo\_dist(c_3^u, c_4) \quad (16)$$

- **Budget (sim_4^c):** As we explained before, we defined 3 ordered labels to classify the budget that a user could spend and the cost estimation of each plan. The labels are: *cheap*, *regular* and *expensive*. The query contains the user prevision (c_5^u) and the real template description contains the estimation (c_5). The economic similarity function ensures that the case cost must be less or equal to the user budget.

$$sim_4^c(c_5^u, c_5) \quad = \quad ord(c_5^u) - ord(c_5) \quad (17)$$

- **Weather (sim_6^c):** It defines the restrictions about weather conditions and outdoor activities. This function compares the weather information of the query (c_4^u) and the location type of the case (c_3). It returns 0 if the weather is *bad* and the templates are tagged as *outdoor* activities. In the other case, it returns 1:

$$sim_5^c(c_4^u, c_3) = \begin{cases} 1 & if \quad \gamma \\ 0 & if \quad i.o.c. \end{cases} \quad (18)$$

where

$$\gamma \quad = \quad (c_4^u == \text{“}good\text{”}) \\ \vee \quad (c_3 == \text{“}indoor\text{”}) \quad (19)$$

The other similarity functions use a common equation. They use the cosine similarity (sim') between the vector that represents the user preferences in the query and the templates description described in each case.

$$sim'(\overline{u}, \overline{v}) \quad = \quad \frac{\overline{u} \cdot \overline{v}}{|\overline{u}| \ |\overline{v}|} \quad (20)$$

$where$

$$\overline{u} \quad \in \quad < q^a, q^m, q^r > \quad (21)$$

$$\overline{v} \quad \in \quad < d^a, d^m, d^r > \quad (22)$$

A combination of them is computed by the global similarity function. It calculates the average of the similarity of all attributes defined in cases and the query. These similarity functions are divided into contextual similarity (sim^c) and preferences similarity (sim^p).

$$sim(\mathcal{D}, \mathcal{Q}) = \frac{sim^c(d^c, q^c) + sim^p(\mathcal{D}, \mathcal{Q})}{2} \quad (23)$$

The contextual similarity function between the query and the cases is the average of the contextual information included in both. The preference similarity is the similarity of the activity types proportion and the similarity of museum and restaurant features.

$$sim^c(d^c, q^c) \quad = \quad \frac{\sum_{i=1}^{6} sim_i^c}{6} \quad (24)$$

$$sim^p(\mathcal{D}, \mathcal{Q}) \quad = \quad \frac{\sum_{i=1}^{3} sim_i'}{3} \quad (25)$$

Next, we will describe how the system generates the explanations of the recommendations.

B. Explanations

In order to implement this feature Madrid Live includes the following explanation algorithm:

- System retrieves the most similar case using context filtering C_a given a query Q.
- Next, context is ignored and the system retrieves the most similar case C_b using only the user preferences Q^- (query without context).
- If $sim(Q^-, C_a) - sim(Q^-, C_b) > \delta$ then a very suitable case C_b has been discarded because of the context and it should be explained to the user.
- Explanation runs as follows:
  - For every description feature d, its corresponding value is obtained from both cases (D_a and D_b) and their similarity to the query is computed.
  - If there is a significant difference in the feature similarity (it is higher than a threshold) $|sim(d_a, Q^-) - sim(d_b, Q^-)| > \theta$, it means that an explanation must be generated for that feature.
  - Explanations are generated as *advantages* and *disadvantages*. Due to the fact that several descriptions were used to filter cases $< d_1, \ldots, d_5 >$ and the remaining ones $< d_6, d_7, d_8 >$ used to compute the similarity to the query Q, the system explains it to the user in order to let him/her choose a more suitable plan or a more similar one regarding his/her preferences.

Let us illustrate this process by means of an example using the query Q described in Table I. The Table II shows the cases retrieved by the recommendation process. The case described on the left is taking into account the context restrictions (C_a). The case described on the right is only taking into account the user preferences (C_b).

In this example, the similarity of C_b to the query is significantly higher than C_a because $sim(Q, C_a) - sim(Q, C_b) > \delta$, and, therefore, explanations are generated. As explained before, relevant features d are identified $|sim(d_a, Q_a) - sim(d_b, Q_b)| > \theta$. Following the example, d_1, d_2 and d_4 are context restrictions that the user may relax. There are also relevant differences regarding the user preferences in d_6 and d_8. Summarizing everything, the generated explanation is: *A suitable plan has been found taking into account your current restrictions: Park “El campo de la cebada”, Restaurant “Saporte di Pizza”. However there is another relevant plan that you should consider because it better fits to your*

TABLE II
CASES OBTAINED TAKING ACCOUNT THE CONTEXTUAL INFORMATION OR THE USER PREFERENCES.

Attribute	**(C_a) Context Preference**	**(C_b) User Preferences**
Time	11:05-16:35	11:40-17:10
Public Transport	Yes	No
First activity location	(-3.709, 40.411)	(-3.723, 40.413)
Estimated cost	Regular	Expensive
Location type	Outdoor	Outdoor
Favourite Activities	Park, Restaurant	Park, Restaurant, Walk
Favourite Museums	N/A	N/A
Favourite Restaurants	Italian	American
Solution	Park "El campo de la cebada", Restaurant "Saporte di Pizza"	Park "Madrid Rio", Restaurant "Foster's Hollywood", Walk "Casa de las Siete Chimeneas".

favourite activities and restaurants. The alternative plan is Park "Madrid Rio", Restaurant "Foster's Hollywood", Walk "Casa de las Siete Chimeneas". But it starts and finishes 30 minutes later, it is more expensive and you cannot use public transport. The following section presents the experimental evaluation.

IV. EXPERIMENTAL EVALUATION

As described in Section III, by including contextual restrictions into the recommender process, the recommendations have a lower similarity with respect to the user preferences. To demonstrate the hypotheses that we described in Section I, we conducted two experiments. The first experiment studies the impact of the context restrictions on the quality of the solutions compared to the only use of user preferences. The second experiment determines the influence of the context filtering in our case base. This experiment evaluates how the context restriction influences our recommend process and which context features are more restrictive.

A. Experimental Setup

For both experiments we use a case base that contains 500 tourism plans in Madrid. Each case has a list of activities as a plan and the features and the context information of the plan as we described in Section III.

The goal is to evaluate the impact of the context features. Moreover, it was expected that some context features could have a higher impact than others in the quality of the solution being retrieved. To measure the relevance of different context features we set-up the experimental evaluation by using a cross-validation scheme where features are disabled systematically. For each evaluation we change the contextual restrictions that are available in the recommender process. We conducted 32 evaluations with each respective permutation of context restriction (from not considering any restriction to considering all of them).

In each leave-one-out step, we used the context information and the features of the case extracted to create each query. In each step we obtained 2 values: the similarity of the best cases returned by the CBR module (1-NN) and the average of the 5 best cases (5-NN). In fact, the result of each evaluation consists of the average of all 1-NN values and, on the other hand, the average of all 5-NN results.

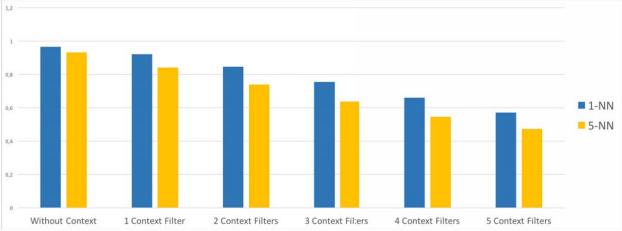

Fig. 1. Average of similarity respect the number of context restrictions available.

B. Experimental Results

To demonstrate **H-1** we want to observe the influence of the number of context restrictions in the system results. Figure 1 represents the average of the 1-NN and 5-NN similarities according to the number of restrictions that were used in the experiment. As expected, similarity decreases with the number of context restrictions. Without context restriction, we obtained our best result with an average of 0.96 in the similarity of 1-NN and 0.93 in 5-NN. But, on the other hand, when we applied all context restrictions we obtained an average of 0.57 in 1-NN and 0.47 in 5-NN.

These data enable us to observe that if we apply too many context restrictions, the plan obtained is less satisfactory according to the user profile. The reason is that in each context restriction the system retrieves cases that could be worse according to the user preferences. This results demonstrate that our hypothesis **H-1** is correct.

To demonstrate **H2** we investigate if all the context restrictions have the same effect in the recommender process. In Figure 2 each pair of columns represents the average of similarity in 1-NN and 5-NN using different configurations. Each configuration is represented by a binary number with 5 positions. Each position represents the respective context filter

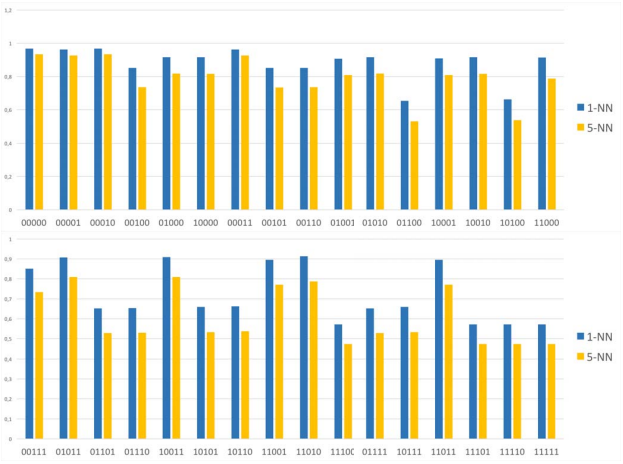

Fig. 2. Average of similarity respect different configuration of context restrictions.

feature (from left to right): start time, finish time, location, economic type and public transport available.

For example, the number 00010 means that the economic type context is available and others are disabled. Figure 2 shows that some configurations are more restrictive than others. The strongest context restriction is the location. We can see that the similarities are lower when location restriction is available, for example, columns 4, 12 or 15. And, we can observe that economic and public transport restrictions have a similar effect, for example, columns 10 and 11. These results demonstrate that depending on the context restriction our retrieved cases could be affected in a number of ways.

V. Conclusions

In this paper, we have described *Madrid Live*, a context-aware recommender system of leisure activities in Madrid. Our system exploits user context-information in order to return more satisfactory leisure activities.

We explained the problem of using contextual restrictions in recommendations. The result of these restrictions is that the retrieved cases have a lower similarity with respect to the user preferences. We conclude that this problem not only depends on the number of context restrictions, but there are context restrictions configurations that filter more cases than others. In this paper, we evaluated which context filter configurations are more restrictive. We propose the use of explanations to solve this problem. These explanations enable a user to select the best plan for him/her knowing the advantages and disadvantages of each leisure plan. All of these goals have been defined by two hypotheses in section I.

The context-aware recommender systems are an active research area. Madrid Live allows us to explore future hypotheses. An example of a future goal is an improvement to users' profiles. Several users taking to account more contextual restrictions than others. For example, there are users that prefer activities closer to their position, but there are others than can select activities far from their position. Another task is evaluate the quality of the explanations generated by the system.

References

[1] F. Ricci, L. Rokach, and B. Shapira, "Recommender Systems: Introduction and Challenges," in *Recommender Systems Handbook*. Boston, MA: Springer US, 2015, vol. 54, pp. 1–34.

[2] A. K. Dey, "Understanding and Using Context," *Personal and Ubiquitous Computing*, vol. 5, no. 1, pp. 4–7, feb 2001.

[3] G. Adomavicius and A. Tuzhilin, *Context-aware recommender systems*. Boston, MA: Springer US, 2015, ch. Context-Aw, pp. 191–226.

[4] J. Borràs, A. Moreno, and A. Valls, "Intelligent tourism recommender systems: A survey," *Expert Systems with Applications*, vol. 41, no. 16, pp. 7370 – 7389, 2014.

[5] B. Smyth, "Case-Based Recommendation," in *The Adaptive Web*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 342–376. [Online]. Available: http://link.springer.com/10.1007/978-3-540-72079-9_11

[6] P. O'Connor, *User-Generated Content and Travel: A Case Study on Tripadvisor.Com*. Vienna: Springer Vienna, 2008, pp. 47–58.

[7] A. Yahi, A. Chassang, L. Raynaud, H. Duthil, D. Horng, and P. Chau, "Aurigo : An Interactive Tour Planner for Personalized Itineraries," *IUI 2015: Proceedings of the 20th International Conference on Intelligent User Interfaces*, pp. 275–285, 2015.

[8] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 6, pp. 734–749, 2005.

[9] W. Woerndl, J. Huebner, R. Bader, and D. Gallego-Vico, "A model for proactivity in mobile, context-aware recommender systems," in *Proceedings of the fifth ACM conference on Recommender systems - RecSys '11*. New York, New York, USA: ACM Press, oct 2011, p. 273.

[10] H. Ma, T. C. Zhou, M. R. Lyu, and I. King, "Improving Recommender Systems by Incorporating Social Contextual Information," *ACM Transactions on Information Systems*, vol. 29, no. 2, pp. 1–23, apr 2011.

[11] X. Liu and K. Aberer, "SoCo," in *Proceedings of the 22nd international conference on World Wide Web - WWW '13*. New York, New York, USA: ACM Press, may 2013, pp. 781–802.

[12] L. Quijano-Sánchez, J. A. Recio-Garcıa, and B. Dıaz-Agudo, "Group recommendation methods for social network environments," in *3rd Workshop on Recommender Systems and the Social Web, RecSys*, vol. 11, 2011.

[13] K. Verbert, N. Manouselis, X. Ochoa, M. Wolpers, H. Drachsler, I. Bosnic, and E. Duval, "Context-Aware Recommender Systems for Learning: A Survey and Future Challenges," *IEEE Transactions on Learning Technologies*, vol. 5, no. 4, pp. 318–335, oct 2012.

[14] W. Woerndl, C. Schueller, and R. Wojtech, "A Hybrid Recommender System for Context-aware Recommendations of Mobile Applications," in *2007 IEEE 23rd International Conference on Data Engineering Workshop*. IEEE, apr 2007, pp. 871–878.

[15] M. Van Setten, S. Pokraev, and J. Koolwaaij, "Context-aware recommendations in the mobile tourist application COMPASS," *Adaptive hypermedia and adaptive web-based systems*, no. August 2004, pp. 515–548, 2004.

[16] P. Efraimidis, G. Drosatos, A. Arampatzis, G. Stamatelatos, and I. Athanasiadis, "A Privacy-by-Design Contextual Suggestion System for Tourism," *Journal of Sensor and Actuator Networks*, vol. 5, no. 2, p. 10, may 2016.

[17] M. M. Richter, "Foundations of Similarity and Utility," *FLAIRS'07: Proceedings of the 20th International Florida Artificial Intelligence Research Society Conference*, no. October, pp. 30–36, 2007.

Capítulo 10

Adaptation process in context-aware recommender system of accessible tourism plan

10.1. Cita completa

Jose L. Jorro-Aragoneses, Susana Bautista-Blasco. *Adaptation Process in Context-Aware Recommender System of Accessible Tourism Plan.* Proceedings of 1st International Workshop on Knowledge Graphs on Travel and Tourism. Cáceres, Spain. June 2018. P. 292-295.

10.2. Contribuciones de la publicación

En esta publicación, proponemos un proceso de adaptación para generar planes turísticos con actividades accesibles a partir de los planes devueltos por Madrid Live. Este proceso usa una taxonomía para sustituir las actividades no accesibles por otras actividades basándose en las preferencias del usuario y su información contextual. Este proceso es una de las líneas de trabajo futuro propuestas en esta tesis.

10.3. Contributions covered by this publication

In this publication, we propose an adaptation process to generate accessible plans using the retrieve plans by Madrid Live. It changes all unaccessible activities to other activities based on the user preferences and the contextual information using a taxonomy. This process is one of the lines proposed to work in the future.

Adaptation Process in Context-Aware Recommender System of Accessible Tourism Plan

Jose L. Jorro-Aragoneses[1( )] and Susana Bautista-Blasco[2( )]

[1] Universidad Complutense de Madrid, Madrid, Spain
jljorro@ucm.es
[2] Universidad Francisco de Vitoria, Madrid, Spain
susana.bautista@ufv.es

Abstract. In this paper, we propose an adaptation process to generate accessible plans based on the retrieved plans by the recommender system. In our case, if one of the activities of the retrieved plan does not apply the user preferences or contextual information, we change it for another activity to adapt the plan. We use a taxonomy based on the open data of the Madrid's council to adapt the plan and achieve an accessible tourism plan.

Keywords: Context · Accessibility · Tourism · Open data Adaptation

1 Introduction

Traditional recommender systems rely mainly on the relation between users and items. There is an active research area in context-aware recommender systems [2], where contextual and dynamic information from the environment is used to enrich the system knowledge and to personalize the recommendation to the specific situation like the location, the weather or social. Contextual information is very important in the tourism domain where any change in the context can change the activity to carry out, for example, if it is raining the user does not go to an outdoor activity. There are different previous works focusing on tourism recommenders [3] and different approaches based on contextual features [5]. Contextual information adds information to obtain a list of proper relevant activities to tourist. For example, the user location or weather state at the recommendation moment [1].

In this paper we propose an extension of MadridLive, a context-aware recommender system of tourism plan in Madrid based on a Case Based Reasoning (CBR) system [4]. Using CBR we obtain implicit information from the case base. In our case, the activities have an order in the plan, the sequence is logic and

Supported by the Spanish Committee of Economy and Competitiveness (TIN2014-55006-R); the funding provided by Banco Santander in UCM (CT17/17-CT17/18).

C. Pautasso et al. (Eds.): ICWE 2018, LNCS 11153, pp. 292–295, 2018.
https://doi.org/10.1007/978-3-030-03056-8_29

coherent based on the user preferences. This information is difficult to deduce using other approach. We include accessibility, handicaps and special needs as contextual features in the system. This accessibility context is used to adapt the system recommendation to avoid, for example, unaccessible locations and transports. This is important in order to guarantee the social inclusion for all kind of people. Given a query describing the user context and preferences, a CBR system [4] retrieves and reuse previously stored plans that fullfill the query requirements. Each case in the case base is a previously stored plan that has been performed and ranked by other users. The main contribution of this paper is an adaptation process to generate accessible plans based on a set of similar plans. Next sections describe the knowledge sources and the retrieval and adaptation processes.

2 Knowledge Sources

The system reasons with two knowledge sources: a case base of previous plans and a taxonomy of tourist activities categories based on the open data set of the Madrid's council[1]. The first one is used to recover the initial plan and, the second one is used to adapt the plan and achieve an accessible tourism plan.

We use a case base that contains 500 real tourism plans in Madrid. Each case has two main components: the case description (CD) and the case solution (CS). The CD includes tags for the categories of all the activities in the plan and the contextual knowledge associated to this plan described by the following features:

- **Time:** Time when start and finish a plan.
- **First Location:** Location of the first activity in a plan.
- **Location Type:** the place type (indoor or outdoor) of an activity.
- **Economic Cost:** An estimated cost to realize the plan.
- **Public Transport:** A boolean value to define if all activities can be visited using public transport.
- **Accessible Plan:** A boolean attribute to indicate if the plan is accessible to people with special needs.

The second knowledge source is the taxonomy. It lets classify all leisure activities that there are in Madrid in the next around 100 days. The system uses this taxonomy to search a new activity to insert in the plan. In this dataset there are around 1000 restaurants, more than 300 POIS (monuments, buildings, etc.) and nearly 1300 events. These numbers are updated daily by the council. The process of plan adaptation to achieve an accessible tourism plan uses a taxonomy of tourist activities categories. It is based on leisure categories from the activity classification in the open data of the Madrid's council. In this case, each available activity is classified using this taxonomy. Figure 1 presents a little example of our taxonomy of categories.

[1] https://datos.madrid.es/portal/site/egob/.

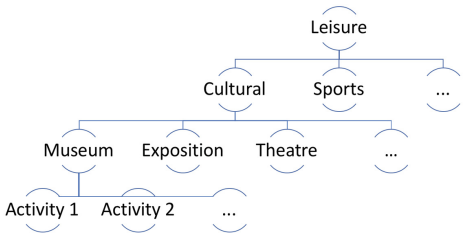

Fig. 1. An example of activity categories.

3 Retrieval

The recommendation process is based on retrieve and reuse plans previously performed and evaluated as positive experiences by other users. The CBR module retrieves the most similar case based on the query $\mathcal{Q}$. It contains the conditions that the user wants in a leisure plan. $\mathcal{Q}$ contains the *user preferences* and the *contextual information* at the moment of the recommendation: time, location, weather, budget and handicaps. To do that, it uses a similarity function that compares each query attribute with its corresponding description attribute. The solution of the case retrieved is the plan that the system offers to the user. MadridLive is explained in detail in [4]. Similarity based retrieval maximizes the plan adequacy although the selected plan could not satisfy the user preferences or the contextual information. Next section describes how the plan is adapted to improve the final solution.

4 Plan Adaptation

The selected plan ($\mathcal{P}$) is adapted by reviewing and substituting some of the activities using the Algorithm 1. The algorithm replaces any activity (a_i) that does not satisfy the contextual information ($\mathcal{C}$) or the user preferences ($\mathcal{U}_p$) defined in the query by other similar activity that respects the user preferences. We recover a similar activity using the taxonomy. For example, if the best plan retrieved has got one unaccessible activity, the system will replace this activity by other accessible activity.

The algorithm searches for an activity using the taxonomy of categories of activities (see Fig. 1). Considering the activity to replace as a query to search the similar activity, the first step is to filter all activities that not apply the contextual restrictions of the original activity. The system classifies the candidate activities in the taxonomy of categories and recovers the activities with the most similar categories. In addition, these recovered activities apply all the restrictions of the original activity. The function selects one of them and return it to the adaptation algorithm. Finally, the system returns an adapted plan based on the contextual restrictions and user preferences.

For example, the best plan obtained contains the activity to visit *the Reina Sofia museum* but it is not accessible. The next step is substitute this activity. Firstly, the system filters all activities that are unaccessible. Next, it classifies

Data: $\mathcal{P} = [a_1, .., a_n], \mathcal{U}_p, \mathcal{C}$
Result: $\mathcal{R} = [b_1, .., b_n]$
$i \leftarrow 0$;
$\mathcal{R} \leftarrow []$;
while $i < \mathcal{P}.len$ **do**
  if $\neg respectContex(a_i, \mathcal{C}) \vee \neg respectPreferences(a_i, \mathcal{U}_p)$ **then**
    $R[i] \leftarrow getSimilarActivity(a_i, \mathcal{U}_p)$;
  else
    $R[i] \leftarrow a_i$;
  end
end

Algorithm 1: Adaptation algorithm

these filtered activities using the taxonomy. Finally, the system changes the original activity by another with similar categories, in our example, the activity proposed will be *the Prado museum*.

5 Working Progress

In this approach, we explain a novel idea in order to adapt real leisure plan for people with special needs. In our case, we focus on the feature of accessibility to modify the original plan. We consider using a taxonomy of categories to obtain the best activity from the open data source. Some difficulties that we found is that there are not enough of accessible activities in the open data used in some categories or the retrieved activities do not apply the user preferences. In addition, the system could return an activity out of the logical route between the previous and next activities in the original plan. For example, an activity so far. In these cases, we need to consider to apply a new filter less restrictive in order to recover a bigger set of candidate activities. Another possible solution to replace the original activity could generate an accessible walking tour between the previous and next activities considered in the original plan.

References

1. Achmad, K.A., Nugroho, L.E., Djunaedi, A.: Tourism contextual information for recommender system. In: 7th AES 2017, pp. 1–6. IEEE (2017)
2. Adomavicius, G., Tuzhilin, A.: Context-aware recommender systems. In: Ricci, F., Rokach L., Shapira, B. (eds.) Recommender Systems Handbook, pp. 191–226. Springer, Boston (2015). https://doi.org/10.1007/978-1-4899-7637-6_6
3. Borrás, J., Moreno, A., Valls, A.: Intelligent tourism recommender systems: a survey. Expert Syst. Appl. **41**(16), 7370–7389 (2014)
4. Jorro-Aragoneses, J.L., Díaz-Agudo, B., Recio-García, J.A.: Madrid Live: a context-aware recommender system of leisure plans. In: ICTAI, pp. 796–801 (2017)
5. Vansteenwegen, P., Souffriau, W.: Trip planning functionalities: state of the art and future. Inf. Technol. Tour. **12**(4), 305–315 (2010)

Capítulo 11

Addressing the cold-start problem in facial expression recognition

11.1. Cita completa

Jose L. Jorro-Aragoneses, Belén Díaz-Agudo, Juan A. Recio-García. *Addressing the Cold-Start Problem in Facial Expression Recognition*. Proceedings of 23rd International Conference on Case-Based Reasoning. Frankfurt am Main, Germany. September 2015. P. 197-211.

11.2. Contribuciones de la publicación

Esta publicación describe otra aproximación para resolver el problema del cold-start en el sistema PhotoMood. En este caso, hemos creado diferentes bases de casos que contienen imágenes de usuarios con las mismas características (sexo, edad, etc.). Esta es otra contribución para resolver el problema del cold-start en el dominio de los usuarios.

11.3. Contributions covered by this publication

This publication describes another proposal to resolve the cold-start problem in the system PhotoMood. In this case, we made different case bases that contain images from user with similar features (age, sex, etc.) It is another contribution to how to resolve the cold-start problem in the domain of users.

Addressing the Cold-Start Problem in Facial Expression Recognition

Jose L. Jorro-Aragoneses( ), Belén Díaz-Agudo, and Juan A. Recio-García

Department of Software Engineering and Artificial Intelligence,
Universidad Complutense de Madrid, Madrid, Spain
{jljorro,belend}@ucm.es, jareciog@fdi.ucm.es

Abstract. In our previous research [5] we proposed a CBR approach to infer the emotional state of the user through the analysis of a picture taken from the front facing camera of her mobile device. We demonstrated that different people express emotions with different gestures and got the best accuracy using a personal case base with self pictures of the same user. However, in the cold start situation, where pictures of the querying user are not available, the CBR system uses a generic case base (GCB) made of pictures of anonymous people. Although the performance using the GCB was acceptable on average there were several users with a very low accuracy. In this paper we compare our GCB to other reference picture catalogues and evaluate our CBR approach with state-of-the-art Facial Expression Recognition (FER) algorithms. Results point out that our approach is only suitable for GCB including *semantically similar* users. We use an ontology to group together users with similar demographic and physiological information: sex, age and ethnic group. We evaluate our CBR approach with small and specialized case bases where pictures are semantically similar to the target population and demonstrate that it efficiently increases the accuracy in the cold start situation and minimizes the noise in the case base.

1 Introduction

Emotional tagging of facial expressions is becoming a relevant topic for e-commerce systems based on mobile devices. These devices include a wide range of sensors able to capture the emotional state of the user, which is a very valuable feedback to infer her willing to consume a concrete product being proposed by the e-commerce system. This information about the user's emotional state has many potential applications and an unmeasurable value from the commercial point of view. Although modern devices include sensors that can measure several physiological variables of the user such as heart rate, temperature or even blood pressure, the front camera is usually the best alternative to infer the emotional response.

Usage of dynamically enriched information from the user context leads the system to find better solutions that are adapted to the specific situations. In our research we have focused on the difficult problem of dynamically acquiring

E. Hüllermeier and M. Minor (Eds.): ICCBR 2015, LNAI 9343, pp. 197–211, 2015.
DOI: 10.1007/978-3-319-24586-7_14

the *emotional context* of the user during a recommendation process. In [5] we described PhotoMood, a CBR system that uses gestures to identify emotions in faces, and presented preliminary experiments with MadridLive, a mobile and context aware recommender system for leisure activities in Madrid. In the experiments, the momentary emotion of a user is dynamically detected from pictures of the facial expression taken unobtrusively with the front facing camera of the mobile device.

The emotional state of the user inferred by PhotoMood was used as the feedback -like or dislike- for our recommender system for leisure activities. This CBR system uses as queries the pictures of the user taken while she is receiving recommendations from the system. By comparing these pictures to a case base of pictures tagged as positive or negative responses, we could infer the user's opinion about each activity being proposed. The published results stated that the use of a personal case base with pictures of the user achieved a higher performance than a generic case base with pictures from several users. Although a personal case base is the best alternative, new users have no pictures in the system and, therefore, the *cold-start* problem appears. The term *cold-start* is used in recommender systems to denote those users with little information in the system. This problem is very common and requires specific techniques to circumvent it. Due to the lower performance of the generic case base compared to the personal case base, we required an alternative approach to tag those users in cold-start. This paper tries to address this limitation by proposing the use of specific case bases with a small number of cases that have been semantically annotated and that groups pictures of users that are semantically similar. we can choose the most suitable set of cases to get an efficient and precise replacement of the personal case base for those users in cold-start. This intermediate approach rises the performance of our facial expression recognition (FER) system and achieves better results than a generic case base. We have performed an experimental evaluation that not only validates our proposal but also compares our CBR strategy to other state-of-the-art algorithms for facial emotion recognition.

Section 2 describes some related work on the FER field research. Next, in Sect. 3 we explain our previous work, algorithm and results on face emotional recognition based on the CBR paradigm. Section 4 introduces the cold-start problem and a first solution using generic case bases. We introduce some of our tools for visualization and organization of cases bases and we compare our approach with other FER approaches in these generic case bases. Section 5 describes a solution based on the semantic annotation of generic case bases to obtain specialized cases that share common gestures with the current query. We use age, sex and ethnic group as the features to annotate case bases. Experiments show that results on these specialized case bases are accurate and efficient. Section 6 concludes our work.

2 Related Work

Facial expression recognition (FER) is a research field very active in the area of computer vision. We can find a general overview of FER algorithms in [18].

FER algorithms are applied to different domains, such as recognition of students engagement [21], user experience feedback [13] or different recommender systems [19,20].

FER algorithms can be classified in several ways, although a common classification is based on the method to extract facial features [18]. According to this feature, FER algorithms can divided in two groups: appearance feature extraction and geometric feature extraction.

Appearance feature extraction consists on extracting changes in the appearance of the face, for example, skin texture. We find many works whose authors use this technique. Zhen and Zilu [22] use a FER algorithm based on sparse representation and Local Phase Quantization (LQP). They based their algorithm on the LPQ method of textures of images [14]. Other example is Taheri,Patel and Chellapa [17], where authors use a dictionary-based component separation algorithm (DCS). This method separates the neutral component and from expression component of an image. In Khanum, Mufti, Javed and Shafiq [7], authors detect the facial action elements (FAEs) and uses fuzzy logic combined to CBR to detect the emotion.

The second group of FER algorithms are the geometric feature extraction ones. It consists on extracting the location of elements of the face. An example of this type of algorithms is Kotsia and Pitas [8]. It uses the location of some points to detect deformations in the face and then detect the emotion from this deformation.

Our algorithm, PhotoMood [5,11], is classified in the geometric feature extraction. PhotoMood is a CBR system to detect emotions with facial recognition. In [11], we presented a similarity function based on the external contour of the mouth. In this paper, we improve the similarity function by adding additional gestures as it is explained in the following section.

3 PhotoMood: A CBR Approach to Face Emotion Recognition

This section explains the CBR algorithm implemented in PhotoMood. PhotoMood [5,11] analyses the facial expression to detect the user emotion using two stages:

Image Pre-processing: PhotoMood obtains 46 coordinates of the user's facial gestures classified in 8 vectors:

- $\overline{v_1}$: External outline of mouth.
- $\overline{v_2}$: Internal outline of mouth.
- $\overline{v_3}$: Outline of right cheek.
- $\overline{v_4}$: Outline of left cheek.
- $\overline{v_5}$: Outline of right eye.
- $\overline{v_6}$: Outline of left eye.
- $\overline{v_7}$: Outline of right eyebrow.
- $\overline{v_8}$: Outline of left eyebrow.

CBR Process: Cases in the case base include pictures of users that have been annotated with the emotion tag t. From a query of gestures(Q) including these 8 vectors, the CBR system retrieves the most similar pictures and reuse their solution to estimate the user's emotion:

$$\mathcal{Q} = < \overline{v_1^q}, \ldots, \overline{v_8^q} > \tag{1}$$

$$\mathcal{C} = < \mathcal{D}_c, \mathcal{S}_c > \tag{2}$$

where

$$\mathcal{D}_c = < \overline{v_1^c}, \ldots, \overline{v_8^c} > \tag{3}$$

$$\mathcal{S}_c = \{t_1, \ldots, t_m\} \tag{4}$$

The system admits any number of emotions in the solution side of the cases. The system retrieves the most similar cases using the gesture vectors and reuse their emotion tags. In the recommender domain we use three tags *Like*, *Dislike* and *Surprise*, so m=3.

Retrieval. We used a K-Nearest Neighbour algorithm where the similarity value is computed by weighting the local similarity of each vector $\overline{v_i}$. Therefore each pair of vectors $< \overline{v_i^q}, \overline{v_i^q} >$ is compared and the resulting value is weighted with a value w_i that represents the relevance of the corresponding gesture in the global similarity computation:

$$Sim(\mathcal{Q}, \mathcal{D}_c) = \sum_{i=1}^{8} w_i * Sim_i(\overline{v_i^q}, \overline{v_i^c}) \tag{5}$$

where

$$\sum_{i=1}^{8} w_i = 1 \tag{6}$$

We obtain the angle between each pair of points and the horizontal axis. Following the contour of a gesture, the arctangent between a pair of points is computed to obtain their angle. Each angle of the query $\mathcal{Q}$ is compared to the corresponding angle of the case $\mathcal{D}_c$ producing $|\overline{v_i}|$ (angle-level) similarity values:

$$Sim_i(\overline{v_i^q}, \overline{v_i^c}) = \frac{1}{|\overline{v_i}|} \sum_{j=1}^{z} 1 - \sqrt{(\arctan(\widehat{p_j^q p_l^q}) - \arctan(\widehat{p_j^c, p_l^c}))^2} \tag{7}$$

$$\text{where } l = (j+1) \mod z$$

Reuse. To obtain the solution of the query, the system uses a weighted voting schema according to the similarity of the retrieves cases. The scoring function is:

$$score(t_i) = \sum sim(\mathcal{Q}, \mathcal{D}_c) \; \forall \; c \,|\, \mathcal{S}_c = t_i \tag{8}$$

The solution assigned to the query is:

$$t_i = arg\, max\{score(t_i), \; i = 1, \ldots, m\} \tag{9}$$

Revise. The revise stage is external to the PhotoMood CBR module. The user will be the responsible to modify the solution if it is wrong.
Retain. The system stores the cases that they were revised by the user.

3.1 Previous Results

In our previous experiments [5], we tested our CBR approach with a a personal case base (PCB) of self pictures for each querying user, and with a generic case base (GCB) that contains 300 images of anonymous people that we obtained with search processes in Google Images[1]. We concluded that each gesture has a different importance for each user and we used a Genetic Algorithm (GA) to calculate the optimal set of weights for each user. Finally, we concluded that the CBR system behaves better using the personal case base (PCB) and the personal set of weights (pw) for each user. However, having a personal case base for each user is not always possible. For example, when we have new users in cold start. Besides, the computation of a personal set of weights for an user is a computational expensive process. So, GCBs with very different people have been characterized and evaluated. Next section describes how different GCBs behave with different users in the cold start situation.

4 Generic Case Bases as a Solution to the Cold Start Problem

Cold start is one of the most challenging problems in recommender systems [10]. It is a potential problem in any knowledge based system based on information about users or items. It appears when the system has not gathered enough information. For example, in the domain of product recommendation for new users the system has no information about users' preferences in order to make recommendations. Although a relevant research has been conducted in this field the cold-start problem is far from being solved and many different partial solutions have been proposed (see [16] for a recent review).

In our previous research [5] we demonstrate that different people show emotions with different gestures and compute the specific set of weights that maximize the accuracy of our CBR classifier for each specific user. To do that, we used a personal case base with self pictures of each user. We have experimented with different configurations of case bases and sets of weights. As conclusion, we pointed out that the system obtains the best results using a personal case base for each user. However, in the cold start situation, where pictures of the querying user are not available, the system uses a generic case base made of pictures of anonymous people. Reasonable results were found when tagging two basic emotions –like and dislike – using 46 feature points organized in 8 gesture vectors. Experiments got an average precision in GCB of 89.34 % and an average

[1] We used Google Image Search with the queries "Happy face", "Unhappy faces" and "Surprise faces".

precision for the personal case base of 92.16 %. Our generic Case Base in these preliminary experiments consists of 300 images of anonymous people that we obtained with search processes in Google Images.

Although the performance of the GCB in our previous work was acceptable on average, there were several users with a very low accuracy. After a comparison of our GCB with other reference picture catalogues used to evaluate state-of-the-art FER algorithms, we realised that our approach is only suitable for users similar to those ones in the case base. For example, the JAFFE [12] dataset contains pictures of Japanese women. Due to specific facial features of this population our GCB was unable to correctly classify our previous pictures (from European people) when used as queries in a cross-validation experiment. Moreover, after several experimental evaluations we noticed that the Genetic Algorithm used to optimize the similarity function decreased its performance when applied to large generic datasets including a wide range of users.

This experimentation with several generic case bases led us to conclude that the cold-start problem could not be addressed with only one generic case base. Nevertheless, the use of specialized case bases including pictures semantically similar to the target population could increase the accuracy of the system. We use the term *semantically similar* to denote those features that allows us to group users with similar facial expressions.

Initially, we tried to apply automatic machine learning approaches -such as clustering techniques- to obtain these groups of users with similar expressions. After several experimental evaluations results were unsatisfactory and we concluded that these groups had to be made according to semantic features of the users. As we will present in following sections we have used the information from the profiles of the users to map them to a specialized case base through a semantic mapping function based on an ontology. This ontology captures the *semantic similarity* between users according to features like age, ethnic group an gender.

Following subsections present this experimental evaluation. First we introduce the results when applying our approach to several reference datasets in Sect. 4.1. Next, we also compare different state-of-the-art FER algorithms to our CBR system 4.2. Finally, Sect. 5 presents our semantic approach to exploit specialized case bases.

4.1 Applying PhotoMood CBR to Reference Datasets

The first solution to the cold-start situation is the use of a GCB including a large number of pictures of different type of users, in order to cover as much as possible the solution space of facial expressions. We used 2 different reference datasets as case bases to evaluate our CBR approach:

1. *CK+*[6]*:* It contains 593 images from 123 individuals. These images have been divided in scenes where each scene contains images from neutral state to any emotion. The last image of each scene is the peak of the emotion (the image most representative of the emotion tagged in this scene). We only used the last image of each user that has been tagged with an emotion. Totally we

CK+			
	Dislike	Like	Surprise
Dislike	85.55%	10.98%	3.47%
Like	30.43%	66.67%	2.9 %
Surprise	6.02%	0.0%	93.98 %

JAFFE			
	Dislike	Like	Surprise
Dislike	93.42%	5.26%	1.32%
Like	45.16%	54.84%	0.0%
Surprise	86.67%	3.33%	10.0%

Fig. 1. Confusion matrix of the CK+ (top) and JAFFE (bottom) datasets using PhotoMood CBR.

have 327 images (45 angry, 18 contempt, 59 disgust, 25 fear, 69 happiness, 28 sadness and 82 surprise) to use like queries.

2. *JAFFE*[12]*:* It contains 213 face images from 10 subjects. These images are of Japanese women. These pictures are classified in: Neutral (30), Anger (30), Disgust (29), Fear (32), Happiness (31), Sadness (31) and Surprise (30).

As we explained these datasets are classified in 7 emotions, but PhotoMood was implemented to detect 3 emotions (like, dislike and surprise). Therefore we grouped similar emotions (for example sad and fear) to reduce the number of classes to the ones used by our system. We selected and grouped the emotions equivalent to those ones in PhotoMood. For the *Like* tag we included images tagged with the happy emotion. For the *Dislike* tag we grouped images whose tag is a dislike emotion (angry, contempt, disgust, fear and sadness). Finally, we included the *Surprise* tag alone as there is not a clear classification (Like or Dislike).

After several cross-validation evaluations we obtained poor results. We tested both datasets using leave-one-out and grouping the classes of pictures into the 3 emotions used in PhotoMood (*Like*, *Dislike* and *Surprise*). Results are shown in Fig. 1 as a confusion matrix. We can observe that there is no clear classification for any emotion.

In any CBR system there are two possible explanations for this lack of performance: (1) the additional knowledge of the system -this is the similarity function- has deficiencies. And (2) the deficiencies are in the case base. Beginning with the first possible explanation we developed introspection tools to analyse the performance of our similarity function. These tools are:

– Visualization of cases grouping. Ideally a good similarity function should group same class cases together. By using a graph distance visualization tool we should be able to identify clusters of cases belonging to the same emotion. This tool is shown in Fig. 2 applied to the CK+ dataset. In this figure each emotion is represented by a colour. We can observe that there are two clear clusters where the similarity function is working properly but the center of the image is too messy, denoting a bad classification.

- Distance matrix. Although the case-base visualization tool is very useful to informally evaluate the accuracy of the similarity function it depends on the graph visualization algorithm that tries to layout the cases depending on their distance. An efficient way to circumvent this drawback is the distance matrix. This NxN matrix (being N the size of the dataset) shows the similarity of every pair of cases by using a grey scale. A dark point denotes a high similarity and a whiter point reflects low similarity. By sorting the axis of the matrix according to the class (emotion) of the cases we should, ideally, find dark rectangles in the diagonal of the matrix (corresponding to the similarity of cases from the same emotion) and white areas in the surroundings. However, when visualizing the datasets with this technique we did not find this pattern. As we can observe in Fig. 3 there is only a good classification in the third class (*like* emotion) where there is a clear dark rectangle in the diagonal of the matrix and white areas in its vertical and horizontal sides.
- Similarity function introspection. The disappointing results of the similarity function led us to consider that it was not properly designed. As we previously explained it is a weighted average of several local similarities that compares 8 gestures of the user (eyebrow, outline of mouth, etc.). Perhaps this way to compute the similarity between emotions was not suitable for this domain so we developed an introspection tool to evaluate the accuracy of the similarity function for problematic pairs of cases. The resulting tool is shown in Fig. 4. In this case, pictures being compared belong to different emotions (anger and disgust). After analysing the accuracy of the similarity function with several pairs of pictures we concluded that the similarity function was performing reasonably well.

If the similarity function was performing well the only remaining explanation for the low accuracy of the CBR system is the other source of knowledge: the case-base. As we can clearly observe in Fig. 4 pictures are quite similar but classified with different emotions. No similarity function could distinguish among these emotions because they deeply depend on the facial expression of the user[2]. This analysis supports our premise that the personal case base is the best alternative to rise the performance of the CBR system. But when the user is in cold-start and there are no personal pictures we cannot use generic case bases because the classification deeply depends on the type of user. This is the main reason to develop an approach based on specialized case bases selected by using semantic knowledge that is presented in Sect. 5. But before developing such method we had to completely confirm that the CBR approach was valid by comparing it to state-of-the-art FER algorithms as described in the following section.

4.2 Comparison of PhotoMood with Other FER Approaches

Section 2 introduced several algorithms that exploit the textures in pictures to infer the emotional state for the user, i.e. [17,22]. We have compared our CBR

[2] We leave aside possible miss-classifications of pictures in the dataset.

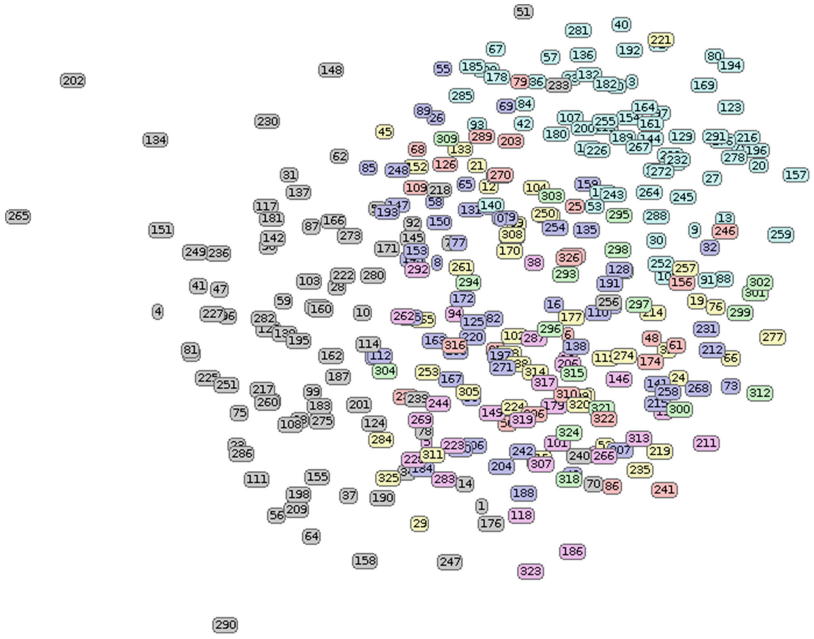

Fig. 2. Visualization of the CK+ dataset. Each colour representing a different emotion (Color figure online).

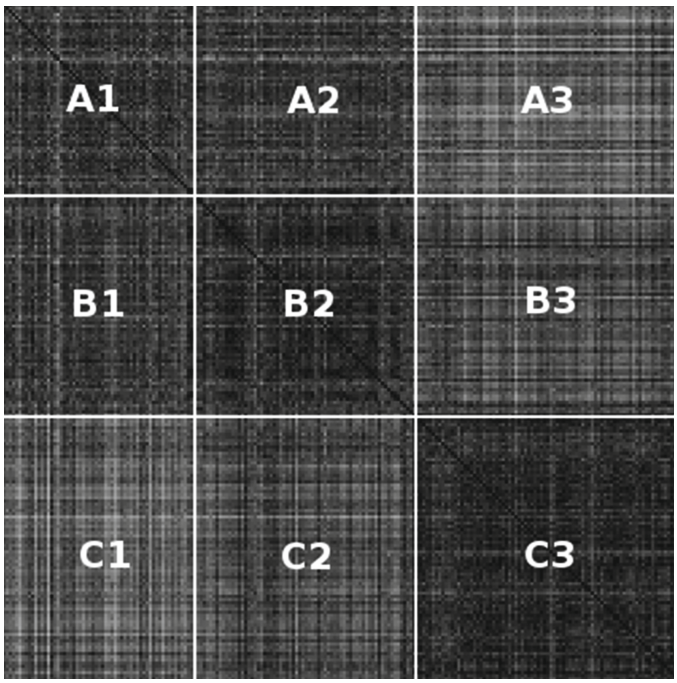

Fig. 3. Distance matrix of the CK+ dataset. A1, B2 and C3 areas representing a pictures of the same emotion.

Fig. 4. Introspection tool to analyse the similarity function. Although similar pictures denote different emotions: anger (left) and disgust (right).

approach to these algorithms in order to validate our geometric similarity function. Figure 5 shows the results of this comparison for the CK+ dataset[3].

CK+			
	Like	Dislike	Surprise
SRC + LPQ [22]	100.00%	71.156%	100.00%
DCS-S1 [17]	98.55%	72.91%	98.70 %
PhotoMood	66.67%	85.55%	93.98 %

Fig. 5. Comparison with other FER approaches.

According to these results, our approach is similar in performance to state-of-the-art algorithms. It achieves a higher accuracy for the *dislike* class, but a lower performance for other classes. These inconclusive results are rooted in the low quality of the case base regarding the classification problem being addressed.

We can also compare these algorithms from the functional point of view. Texture-based algorithms require a large storage as they must store every single pixel of the picture. In our MadridLive scenario it is not a suitable solution for devices such as mobile phones with a limited storage capacity. Our approach only requires to store the 46 points defining the geometry of the gestures. Another relevant advantage of our CBR approach is related to the alignment of the faces in the pictures. Texture-based methods usually apply a *sparse representation* technique that requires a perfect alignment of every picture. This technique uses an image overlay training process to create a model representing the emotions of the user. However, this approach is not applicable in our scenario as it is impossible to obtain aligned pictures from the front facing camera of a mobile device. Being our method based on geometrical comparison it does allow to compare pictures with different face alignments.

[3] The accuracy is taken from [9], whereas the DCS-S1 accuracy is obtained from [17].

Once the CBR approach was validated next Section presents an improvement to solve the cold-start problem.

5 Use of Semantic Case Bases

After our analysis of the performance of the studied GCBs, we have found several drawbacks when trying to address the cold-start problem. First, to achieve an acceptable accuracy, the CBR system requires a large number of pictures that share gestures with the user query. Besides, to guarantee enough coverage for different types of users we would need to provide with a very large case base. The CBR process becomes inefficient and decreases accuracy due to the irrelevant noisy cases. However, we noticed that it is possible to increase the accuracy of the CBR system if the case base includes pictures from the same physiological group than the query user. It means that by using pictures of users with similar features like age, gender and ethnic group, we could rise the performance of our FER system. In this section we describe a proposal to solve the cold-start problem using small and specialized case bases according to the demographic and physiological features of the users.

Our first approach was the use of automatic clustering algorithms to find small related groups in GCB. This way we could find groups of pictures from users with similar features (Fig. 4). However after several experimental evaluations we concluded that this approach led us to generic groups of pictures meaningless from the point of view of expression recognition.

Our proposal is to manually create these clusters as specialized case bases that capture semantic features of the user being classified. We evaluate this proposal to check if the use of semantic case bases improves the accuracy and the efficiency of the FER system in a cold start situation. It is a two stage process:

- Find the proper specialized case base for a given user.
- Retrieve and reuse the most similar cases.

This approach requires additional knowledge about the user in order to select the most suitable specialized case base. MadridLive recommender [5] obtains this information from the user profile. During registration, users must state their age, gender and country. By adding additional semantic knowledge the system can be able to find the proper case base.

If users are described by age, gender and country, and our specialized case bases are described by the age range, gender and ethnic group of the users in the pictures there is a small vocabulary gap to be solved. Previous works [15] have pointed to ontologies as an efficient way to represent the semantic knowledge required to map between a query and case base description vocabulary. In our case, ontologies can formalize the additional knowledge required to obtain the proper case base of pictures for a query described in terms of the age, gender and country. Figure 6 shows a simplified view of our ontology. It illustrates how a user from Japan is mapped to a case base for Asiatic people[4].

[4] We have used the"Geographical Races" taxonomy proposed by Garn [4].

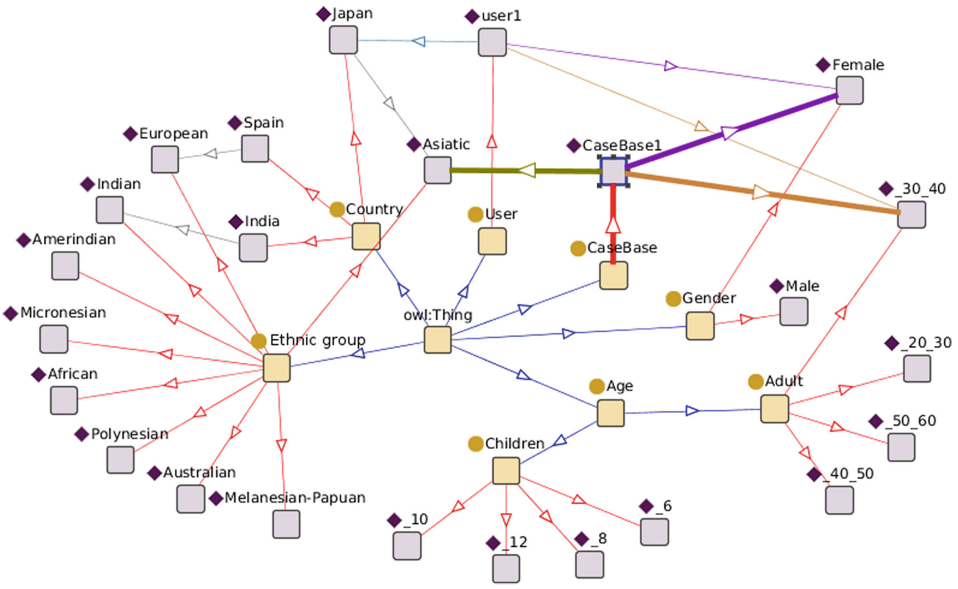

Fig. 6. Semantic annotation of specialized case bases

5.1 Experimental Results

Our approach is based on a semantic characterization of different generic case bases. We call Semantic Case Base (SCB) to these annotated GCBs. The SCB that is semantically most similar to the query is used to apply the CBR processes described in Sect. 3. The hypothesis we want to probe is that the use of a case base chosen this way repeats higher precision values. Our approach is considered as a solution to the cold start problem because the precision obtained by this specialized case base is close to the figures obtained by the user personal case base.

In the experiments to validate this hypothesis we have created 12 specialized case bases. Each case base contains images of a set of semantically similar users according to the following features:

- *Age,* classified in 2 categories, children (CH) and adults (AD).
- *Gender,* classified in 2 categories, men (M) and women (W).
- *Geographical area,* where we distinguish 3 main areas Africa (AF), Asia (AS) y Europa (EU).

By combining these categories we generated 12 case bases that were populated using Google Images:

- CHMAF: African male children. 10 Like and 9 Dislike.
- CHMAS: Asian male children. 19 Like and 20 Dislike.
- CHMEU: European male children. 20 Like and 17 Dislike.
- CHWAF: African female children. 21 Like and 16 Dislike.
- CHWAS: Asian female children. 19 Like and 19 Dislike.
- CHWEU: European female children. 20 Like and 20 Dislike.
- ADMAF: African male adults. 19 Like and 19 Dislike.

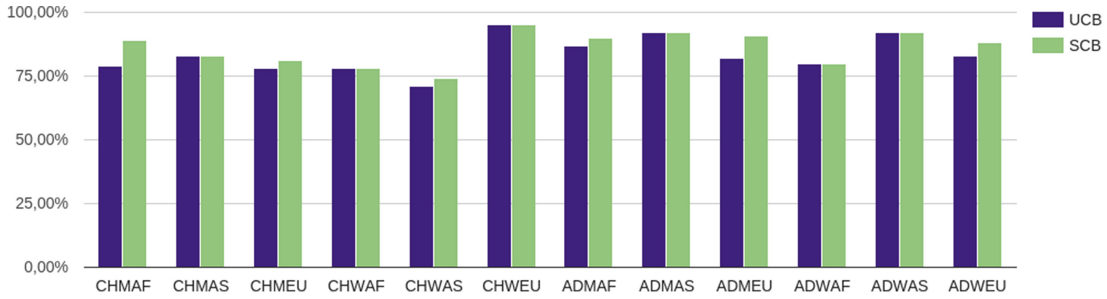

Fig. 7. Compare the use of specific case base between the union of case bases.

- ADMAS: Asian male adults. 14 Like and 12 Dislike.
- ADMEU: European male adults. 20 Like and 20 Dislike.
- ADWAF: African female adults. 10 Like and 15 Dislike.
- ADWAS: Asian female adults. 13 Like and 13 Dislike.
- ADWEU: European female adults. 20 Like and 20 Dislike.

In order to evaluate the validity of this approach we compared the performance of the system when using the most suitable semantic case base for each user to the performance when using a generic case base composed of all the semantic ones (referred as *union case base* (UCB)). Concretely, we have performed two tests:

- *Test 1:* Evaluation of every SCB using leave-one-out with cases from UCB.
- *Test 2:* Individual leave-one-out evaluation of every SCB only taking cases from itself.

Figure 7 shows the hit ratio when PhotoMood uses GCB and when it uses each semantic case base SCB. As we can observe the use of SCB always achieves at least the same performance than a generic case base, although there are many SCB improving the performance of GCB.

This experiment validates our hypothesis, that was statically confirmed by Wilcoxon test ($p-value < 0.05$), and allows us to conclude that the use of semantic case bases provides higher precision values. Besides, this approach solves the cold start problem as the precision obtained in this specialized case base is close to the values obtained in the user personal case base.

6 Conclusions

Mobility and context-awareness are two active research directions that open new potential to CBR systems [1–3]. In this paper we have addressed the cold-start problem in Facial Expression Recognition (FER). We propose a CBR system that is able to recognize emotions from pictures taken from the front facing camera of a mobile device.

In our previous work [5], we have concluded that personal case bases of user's pictures increase the performance of our CBR system. However, users in cold-start do not have enough pictures and an alternative case base must be adopted.

A plausible alternative are generic case bases composed of a large number of pictures from different users. To explore this approach we have evaluated several datasets of pictures that are used to benchmark state-of-the-art algorithms for FER. However, experimental results show that these large and generic datasets do not achieve an acceptable accuracy. In order to explain these results with the generic case bases we analysed our CBR process using a set of introspection tools, developed ad-hoc, that let us explore the best configuration of our similarity function. After a deep analysis and a concise comparison with other state-of-the-art FER algorithms, we can conclude that the proposed similarity function is correct and the only remaining explanation is the low quality of the generic case bases.

At this point, generic case bases are not an acceptable solution for those users in cold-start. Although cold-start users do not provide pictures to increase the performance of the system, they provide semantic information about their age, gender and country that can be exploited by our CBR system. Our proposal consists on the use of several specific case bases representing groups of users with similar features (those ones from the profile), and therefore, similar expressions. These specific case bases are labelled with semantic information that let the system choose the most suitable case base for each user according to the profile. However, there is a vocabulary gap between the descriptions of the profile and the labelling of the specific case bases that we solve by mean of an ontology. This ontology provides semantic information about these case bases. By using the ontology the CBR system chooses a semantic case base for each user that includes pictures from users with a similar facial expression.

Finally, this paper reports an experimental evaluation of the approach by comparing the semantic case bases to the generic ones. Results show that these case bases are a suitable solution for users in cold-start.

References

1. Adomavicius, G., Mobasher, B., Ricci, F., Tuzhilin, A.: Context-aware recommender systems. AI Mag. **32**(3), 67–80 (2011)
2. Benou, P., Bitos, V.: Context-aware query processing in ad-hoc environments of peers. JECO **6**(1), 88 (2008)
3. Braunhofer, M., Kaminskas, M., Ricci, F.: Location-aware music recommendation. IJMIR **2**(1), 31–44 (2013)
4. Garn, S.M.: Human Races, 3rd edn. Thomas, Springfield (1971)
5. Jorro-Aragoneses, J.L., Díaz-Agudo, B., Recio-García, J.A.: Optimization of a CBR system for emotional tagging of facial expressions. In: UKCBR (2014)
6. Kanade, T., Cohn, J.F., Tian, Y.: Comprehensive database for facial expression analysis. In: Proceedings of the Fourth IEEE International Conference on Automatic Face and Gesture Recognition, pp. 484–490 (2000)
7. Khanum, A., Mufti, M., Javed, M.Y., Shafiq, M.Z.: Fuzzy case-based reasoning for facial expression recognition. Fuzzy Sets Syst. **160**(2), 231–250 (2009)
8. Kotsia, I., Pitas, I.: Facial expression recognition in image sequences using geometric deformation features and support vector machines. IEEE Trans. Image Process. **16**(1), 172–187 (2007)

9. Lee, S.H., Plataniotis, K., Ro, Y.M.: Intra-class variation reduction using training expression images for sparse representation based facial expression recognition. In: IEEE Transactions on Affective Computing, p. 1 (2014)
10. Lika, B., Kolomvatsos, K., Hadjiefthymiades, S.: Facing the cold start problem in recommender systems. Expert Syst. Appl. **41**(4, Part 2), 2065–2073 (2014). http://www.sciencedirect.com/science/article/pii/S0957417413007240
11. Lopez-de-Arenosa, P., Díaz-Agudo, B., Recio-García, J.A.: CBR tagging of emotions from facial expressions. In: Lamontagne, L., Plaza, E. (eds.) ICCBR 2014. LNCS, vol. 8765, pp. 245–259. Springer, Heidelberg (2014)
12. Lyons, M., Akamatsu, S.: Coding facial expressions with gabor wavelets. In: Coding Facial Expressions with Gabor Wavelets. pp. 200–205 (1998)
13. Novak, D., Nagle, A., Riener, R.: Linking recognition accuracy and user experience in an affective feedback loop. IEEE Trans. Affect. Comput. **5**(2), 168–172 (2014)
14. Ojansivu, V., Heikkilä, J.: Blur insensitive texture classification using local phase quantization. In: Elmoataz, A., Lezoray, O., Nouboud, F., Mammass, D. (eds.) ICISP 2008. LNCS, vol. 5099, pp. 236–243. Springer, Heidelberg (2008)
15. Recio-García, J.A., Díaz-Agudo, B., González-Calero, P.A., Sánchez-Ruiz-Granados, A.: Ontology based CBR with jCOLIBRI. In: Proceedings of the 26th SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence, pp. 149–162. Springer, Cambridge (2006)
16. Son, L.H.: Dealing with the new user cold-start problem in recommender systems: A comparative review. Information Systems (0) (2014). http://www.sciencedirect.com/science/article/pii/S0306437914001525
17. Taheri, S., Patel, V., Chellappa, R.: Component-based recognition of facesand facial expressions. IEEE Trans. Affect. Comput. **4**(4), 360–371 (2013)
18. Tian, Y., Kanade, T., Cohn, J.F.: Facial expression recognition. Handbook of Face Recognition, pp. 487–519. Springer, New York (2011)
19. Tkalcic, M., de Gemmis, M., Semeraro, G.: Personality and emotions in decision making and recommender systems. In: First International Workshop on Decision Making and Recommender Systems, pp. 14–18. CEUR (2014)
20. Tkalcic, M., Kosir, A., Tasic, J.: Affective recommender systems: the role of emotions in recommender systems. In: Proceeding of the RecSys 2011 Workshop on Human Decision Making in Recommender Systems, pp. 9–13. Citeseer (2011)
21. Whitehill, J., Serpell, Z., Lin, Y.C., Foster, A., Movellan, J.: The faces of engagement: Automatic recognition of student engagementfrom facial expressions. IEEE Trans. Affect. Comput. **5**(1), 86–98 (2014)
22. Zhen, W., Zilu, Y.: Facial expression recognition based on local phase quantization and sparse representation. In: 2012 Eighth International Conference on Natural Computation (ICNC), pp. 222–225 (2012)

Capítulo 12

Case base elicitation for a context-aware recommender system

12.1. Cita completa

Jose L. Jorro-Aragoneses, Guillermo Jimenez-Díaz, Juan A. Recio-García, Belén Díaz-Agudo. *Case Base Elicitation for a Context-Aware Recommender System*. Proceedings of 26th International Conference on Case-Based Reasoning. Stockholm, Sweden. July 2017. P. 170-185.

12.2. Contribuciones de la publicación

Madrid Live es un sistema recomendador basado en casos, y para su correcto funcionamiento es necesario que tenga un conjunto de casos. En esta publicación, proponemos una metodología para generar una base de casos inicial de Madrid Live basado en las preferencias de los usuarios y las diferentes situaciones contextuales. Esta contribución resuelve el problema del cold-start en el dominio de los ítems.

12.3. Contributions covered by this publication

Madrid Live is a case-based recommender system, and it is necessary to have a set of cases to work correctly. In this paper, we propose a methodology to obtain an initial case base for Madrid Live based on user preferences and different contextual information. It is a contribution to resolve the cold-start in the items domain.

Case Base Elicitation for a Context-Aware Recommender System

Jose Luis Jorro-Aragoneses( ), Guillermo Jimenez-Díaz( ), Juan Antonio Recio-García( ), and Belén Díaz-Agudo( )

Department of Software Engineering and Artificial Intelligence, Universidad Complutense de Madrid, Madrid, Spain
{jljorro,gjimenez,jareciog,belend}@ucm.es

Abstract. Case-based reasoning can resolve new problems based on remembering and adapting the solution of similar problems. Before a CBR system can solve new problems it must be provided with an initial case base covering the problem space with a sufficient number of representative seed cases with solutions that are known to be correct. We use a CBR module to recommend leisure plans in Madrid based on user preferences and contextual information. This paper deals with the problem of how to build and evaluate an initial case base of leisure experiences in Madrid for the recommender system.

Keywords: Case-based reasoning · Cold start
Context-aware recommender system · Knowledge acquisition

1 Introduction

Case-based reasoning (CBR) addresses new problems by remembering and adapting solutions previously used to solve similar problems [18]. CBR is also a theory of skill and knowledge acquisition that overcomes some of the traditional bottlenecks of expert systems [30]. The main argument for using CBR in general is that it does not need an extensive and deep domain model and relies instead on experience-based, compiled knowledge, which humans are known to gather during and after problem solving [26]. Before a CBR system can solve new problems it must be provided with an initial case base covering the problem space with a *sufficient* number of representative *seed* cases with solutions that are known to be correct. Populating the initial case base is a hard, domain-dependent and time-demanding task. Although seed cases are typically provided by a domain expert, there are approaches of (semi)-automated acquisition [2,9,19,24,25,32]. In fact, case acquisition from raw data is one of the challenges of the CBR research [11].

Supported by the Spanish Committee of Economy and Competitiveness (TIN2014-55006-R, TIN2017-87330-R); the UCM (Group 921330) and the funding provided by Banco Santander, in cooperation with UCM, in the form of a predoctoral scholarship (CT17/17-CT17/18).

M. T. Cox et al. (Eds.): ICCBR 2018, LNAI 11156, pp. 170–185, 2018.
https://doi.org/10.1007/978-3-030-01081-2_12

The CBR system begins with this initial case base and then each experience of solving a new problem becomes the basis for a new case and enriches the experiential memory. These experiences are learnt for potential reuse in future similar problems, despite the utility problem [28]. Thus, CBR favors incremental learning from experience and acquisition of expertise rather than exhaustive extraction of domain knowledge [11].

This paper deals with the problem of how to build an initial case base of leisure experiences in Madrid for a contextual recommender system implemented as a CBR system. Contextual recommender systems have the capability to appreciate its environment and assess the situation in which the cases are to be recommended [1].

When a tourism recommender system is in cold start [15] the system could compose a plan as a sequence of activities – Point of Interests (POIs) or events– based on given composition rules. However, this kind of recommendation would require specific domain rules to capture dependencies and intrinsic congruence between the activities. Besides, it would be difficult to adjust the plan to the context restrictions. Another option to avoid the cold start situation is using a set of prototypical routes provided, for example, from a tourism expert, with the limitations described above. In this paper, we propose an approach taking advantage of the wisdom of the crowd [33] where the case base is created from a crowd of people. We use a pseudo-random method to generate the seed case base and then let the crowd vote to select and filter the best cases using an Elo test [10]. With this method we obtain a case base with diversity, congruence between the activities, and collective knowledge that is independent from a domain expert.

The rest of the paper is organized as follows. Section 2 describes the structure of the cases and an brief overview of the recommendation process. Next, Sect. 3, explains the process to generate a large set of candidate cases, and the use of the *Elo test* to obtain a ranking of the best cases. After generating and selecting the initial seed cases, in Sect. 4 we describe how this case base is evaluated using coverage measures. Section 5 details some related works about techniques to acquire knowledge in recommender systems. Finally, Sect. 6 concludes the paper.

2 The CBR Recommendation Process

The CBR module of the recommender system generates a personalized plan by retrieving and adapting the most similar plans according to the user context and preferences. In order to recommend suitable plans, the system case base stores the plan details and the contextual information associated to its activities and points of interests (POIs). This contextual knowledge is useful to avoid unavoidable plans (i.e. outdoor activities when raining). In this paper, we will describe the case structure, and the retrieval stage of the CBR system. A complete description of the system is available in [14].

The cases in the case base are plans with the typical structure description-solution, $\mathcal{C} = <\mathcal{D}, \mathcal{S}>$. In the retrieval step the case description is compared with

the query and the solution, that includes the details of the route and the sequence of activities, events and POIs in the plan, is reused. The case description is defined using two groups of attributes. The first group represents the activities, events and POIs contained in the plan (d_a). The second group of attributes represents the contextual restrictions (d_{ctx}) to perform the plan such as time, transportation availability, budget, etc. Table 1 shows an example of a case in our system. The list of attributes that defines the contextual restrictions listed in Table 2.

Table 1. The information contained in a case

	Attribute	Value
Description d_{ctx} context setup	*Time*	11:05–16:35
	Transport	Public transport
	First location	(−3.709, 40.411)
	Estimated cost	25 €
	Weather	Sunny
	Out/indoor	Outdoor
Description d_a Act. categories	Categories	Park, Restaurant, Italian Food, Museum, Art
Solution	Activities	Park "El campo de la cebada" (11:05–12:05) Restaurant "Saporte di Pizza" (12:15–13:30) Queen Sofia Museum (13:45–16:35)

These details of each plan are stored in the case solution ($\mathcal{S}$). Basically, the case solution is the list of activities contained in the plan. In addition, it contains the time when the user started the activity and when she finished it.

The retrieval process is a k-NN algorithm using a weighted average to combine the local similarities between attributes. It calculates the average of the similarity of all attributes defined in cases and the query. These similarity functions are divided in contextual similarity (sim_{ctx}) and activities similarity (sim_a).

$$similarity(\mathcal{C}, \mathcal{Q}) = \alpha \cdot sim_{ctx}(d^c_{ctx}, d^q_{ctx}) + (1 - \alpha) \cdot sim_a(d^c_a, d^q_a) \quad (1)$$

The contextual similarity function sim_{ctx} is the average of the contextual attributes described in Table 2. Analogously, the similarity of activities, sim_a, compares the type of activities in the plan (museums, restaurants, cinemas, parks, ...).

3 Case Base Acquisition

This section describes our approach to generate synthetically a case base that captures certain wisdom of crowds knowledge [33] about the congruence and

Table 2. Attributes representing contextual information in the cases.

Start-time	It contains the time when the user started the leisure plan
Duration	It is the time spent by a user to complete all activities of the plan
Location	This attribute defines the location where the user started to visit the activities
Out/indoor	It defines the type of location of every activity in the plan. The location type values can be *indoor* or *outdoor*
Weather	It represents the weather requirements to perform the plan
Cost	It is an estimation of the cost to realize all the activities
Transport	It stores the transport used by the user when he/she visited the activities. This transport type can be: *walking*, *public transport* or *car*
Social	This attribute defines the social information associated with the plan, i.e., what are the social conditions when the user did the leisure plan. Its possible values are: *alone*, *couple*, *family*, *friends* and *business*

common sense in the sequence of activities in a plan. This refers to social non algorithmic knowledge associated to a touristic route, for example, leg fatigue, crowed streets in certain seasons like Christmas, delays in big groups of people, alternate between physical and quiet activities, the best lighting conditions for visiting a POI, and others. Firstly, a large number of cases are pseudo-randomly generated complying basic restrictions about distances, timetables, activities, etc. Next, real users rank those cases according to their congruency with the context and their global quality a as a plan. It is important to note that users do not take into account their personal preferences when voting.

To perform this ranking we have used the Elo rating system [10], a well known method for calculating the relative skill levels of players in zero-sum games such as chess.

Following subsections detail the process. Firstly, we generate a large set of cases synthetically. These cases are candidates for the initial case base. Then, the next step uses the *Elo test* to obtain a ranking of the best cases.

3.1 Case Base Generation

The first step consists on generating a large set of candidate plans from a dataset of activities. These activities were obtained from the Madrid's council Open Data portal[1] that contains a large dataset of leisure activities in Madrid. Concretely, we used around 1000 restaurants, more than 300 POIs (monuments, buildings, etc.) and nearly 1300 events.

From this dataset, cases are generated randomly but according to certain restrictions, timetables, types of activities and distances. Cases are generated

[1] https://datos.madrid.es.

according to a simulated user context. A plan cannot contain two consecutive restaurants, restaurants are scheduled at certain times, plans of the same context start at the same POI, the maximum distance of the complete plan is limited depending on the public transport in the area, or in the use of private car or walking distance as indicated in the context. We generated randomly 30 different contextual setups. For each contextual setup we define concrete values for the attributes presented in Sect. 2 and described in Table 2. For example, if the simulated context defines bad weather, the generated plan does not contain outdoor activities. Although case generation was driven by the context, only some of its attributes were used to select the activities of the plan as it is not possible to model every possible restriction. This way, the generation stage returns a large set of cases, syntactically correct but, may be, semantically incorrect. We will use the Elo ranking, to filter these cases and select those ones that are good plans and congruent with the context. At the end of this step we generated 10 plans by contextual setup. In total, we generated 300 plans to evaluate as described next.

3.2 Case Base Selection

The main goal of this second step is to capture good plans according to the wisdom of crowd knowledge. To capture this knowledge we used a test based on the *Elo ranking*, that is simple and does not require technical skills, like other approaches such as Likert scales. This method allows determination of the best plans faster than other methods because scoring is very simple. Given a context, it just presents two options (plans) to the user and let her choose the best option for this situation.

In the Elo ranking, each player's performance in a game is modeled as a normally distributed random variable. The mean of that random variable reflects the player's skill and is called the player's Elo rating. If a player wins, her Elo rating goes up, otherwise it goes down. The use of the Elo rating system offers many advantages: it is a simple and fast, it has only a small number of parameters that need to be set, and it also provides comparable performance to more complex models. This way, it has been used for different purposes such as eliciting user preferences [13] or ranking posts in online forums [7]. For our concrete domain, the Elo ranking allows us to obtain cases that have a higher diversity than those generated by an expert. And these cases are not biased by the scorings of other users as they perform the Elo ranking independently contributing with their own opinion. The Elo system aggregates their scorings to obtain a global ranking that summarizes them.

A condition to apply the Elo ranking is that the scored items comply the zero-sum property, meaning that each item's gain or loss of utility is exactly balanced by the losses or gains of the utility of the other items. Therefore, we can apply the Elo ranking to score the quality of touristic plans if we assume that this ranking is also a zero-sum game, in our case, that two plans cannot have the same utility given a concrete context and user preferences. Therefore, for

our CBR system a plan is always better than the others when being compared to the query.

We applied this method to select the best cases according to their congruence with the context and its quality as a leisure plan. To do that, we created a test where every user selects the best plan between two options for the same contextual setup. For every couple of plans being shown to the user, the corresponding context used to guide the generation of both plans is also described.

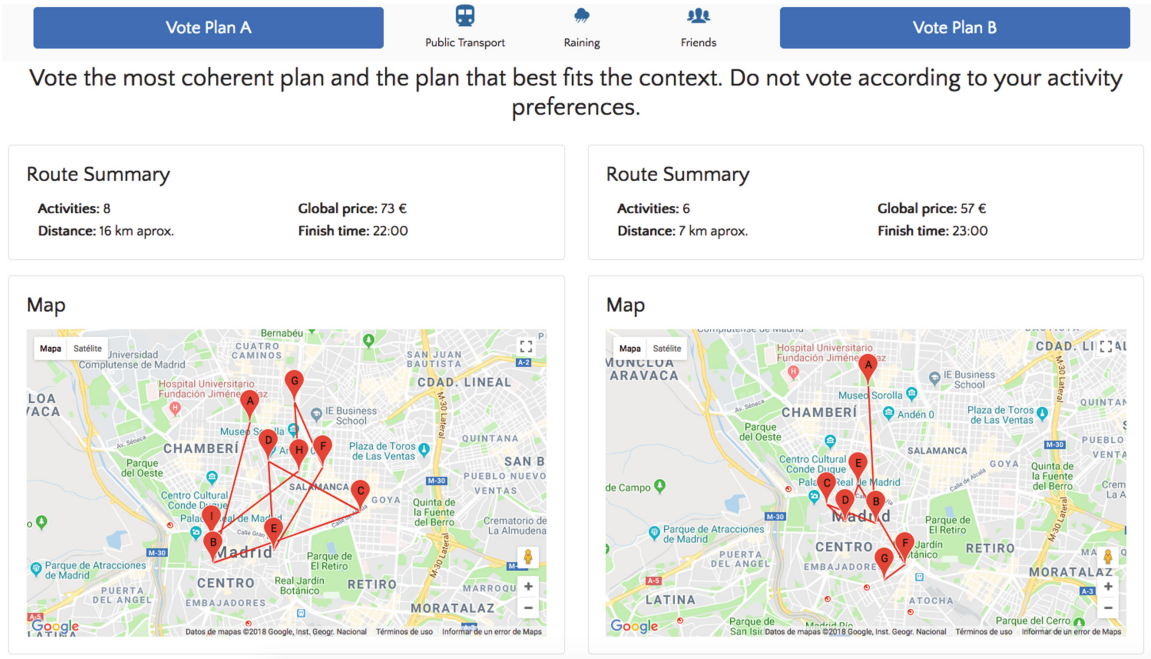

Fig. 1. Map and summary of contextual information for both plans

Figure 1 shows an example of voting using our implementation of the Elo test. First of all, the test shows 2 buttons to vote the corresponding plan; and in the middle of both it shows the contextual information to take into account when choosing the best plan (transport availability, weather conditions, traveling alone, with family, friends, ...). Next, the test shows a summary of each plan: number of activities, total cost, total distance and finishing time. Below, the system shows the route for each plan in a map. Finally, the system shows the list of activities for each plan. Figure 2 shows an example of these lists. Both are ordered according to the starting time of every activity. For each one, relevant attributes are shown to the user: title, short summary and a set of tags that describes the price, location (outdoor/indoor) and the category of the activity.

When a user reads and compares both plans, she selects one of them. Then, the application shows another couple of plans and this process is repeated in loop. Every time the user votes, our test calculates the Elo rating for both plans using the method explained next.

The Elo method calculates a rating for each case (R_x). This rating is updated when 2 cases (A and B) are compared, i.e., the user has selected which case is

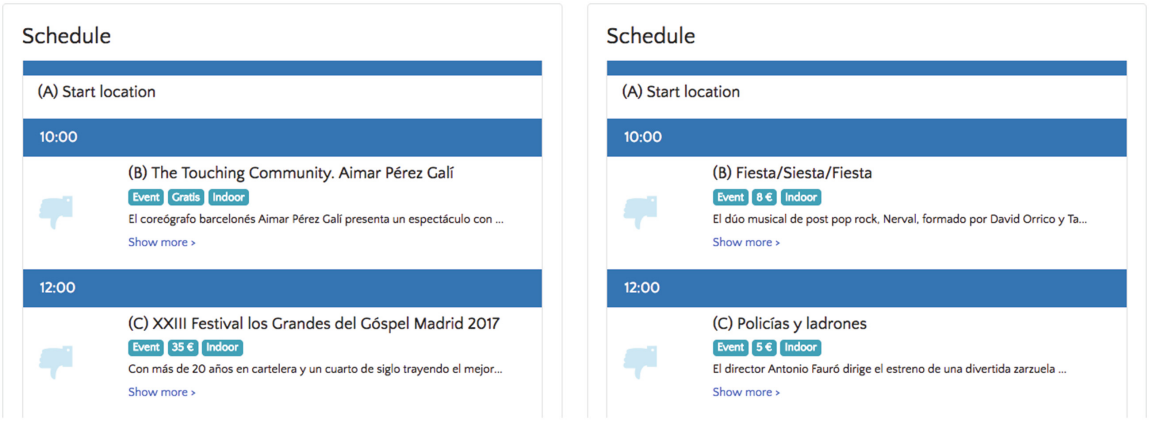

Fig. 2. Detail of activities for each plan.

better. The first step is to calculate the estimated score between them according to their current position in the ranking. The sum of both values is 1 as we assume the zero-sum property.

$$\begin{aligned} E_A &= \frac{1}{1+10^{(R_A-R_B)/400}} \\ E_B &= \frac{1}{1+10^{(R_B-R_A)/400}} \\ &where \\ E_A + E_B &= 1 \end{aligned} \tag{2}$$

The next step is to recalculate the rating of both cases after the comparison (R'_x). Elo ranking uses a constant K to adjust the lineal proportion between the estimated points and the final score. For chess players, this value changes depending on the number of matches of the player. In our experiment, we choose a value of 40 as an estimation of the average votes (both positive and negative) that every case could get. The final rating depends on the result of the vote (S_x): 1 to the winner plan or 0 to the looser plan.

$$\begin{aligned} R'_A &= R_A + K * (S_A - E_A) \\ R'_B &= R_B + K * (S_B - E_B) \\ &where \\ S_x &= \begin{cases} 1\ if\ x = \text{“}win\text{”} \\ 0\ if\ x = \text{“}lost\text{”} \end{cases} \end{aligned} \tag{3}$$

In the following section, we explain the results obtained by our experiment where we collected 1705 votes from 71 users.

4 Case Base Evaluation

After running the experiment with users we obtained an ordered list of cases according to their Elo rating. The next step consists on filtering this list using

the Elo scores that represent their congruency with the context and their quality as a plan. The main goal of this step is to evaluate the case according to several metrics to conclude if its global quality is similar to a case base that was not synthetically generated. And therefore, if the approach presented in this paper is valid to obtain the seed cases for a CBR system.

The Elo test had 1705 votes from 71 users. Firstly, we filtered invalid votes, for example, those votes that the answer time of the user is less than 5 s. After applying this filter, we obtained 814 valid votes. In addition, some candidate cases had not enough votes to be ranked properly. Therefore, we only considered cases with more than 4 votes (positive or negative). At the end of this pre-filtering, there were 222 remaining cases in our case base. This is the initial case base of our analysis, denoted as CB^0.

We will analyze the case base using different metrics. First, we will filter cases according to Elo ranking. Next, we will analyze the similarity, and coverage of the case base taking into account the contextual setups. And finally, we present a global analysis of the case base as a whole.

All analysis explained in this and next sections, with the dataset and their results, have been published in a public GitHub repository[2].

4.1 Elo-Based Similarity

The first analysis in our evaluation was to calculate the Elo ranking based on the user votes. The result of this ranking is shown in Fig. 3. As we can observe, around the 50% of the evaluated cases has a positive Elo score. We can consider these cases as good candidates to be in our initial case base. The leisure plans contained in these cases are supposed to be good plans and congruent with their context according to the opinion of the users.

The resulting case base once we have removed the cases with negative Elo score has 116 cases. We will refer to this case base as CB^2.

4.2 Similarity Analysis

The following analysis tries to evaluate the quality of the retrieval of cases. To do so, we study the performance of the similarity function described in Eq. 1 and detailed in [14].

Figure 4 shows the distribution of the pairwise similarity of every case in CB^1. This figure presents the results obtained for different values of the α parameter in Eq. 1. In this figure we can observe that similarity follows a normal distribution and most of the cases have a low similarity ($\mu \approx 0.2, \sigma \approx 0.1$). It is an indicator of the sparseness and diversity of the case base. As our goal is to obtain the seed cases for our CBR system, these values can be considered as positive because seed cases should not be very similar in order to provide a higher coverage. This

[2] https://github.com/UCM-GAIA/Case-Base-elicitation-for-a-context-aware-recommender-system.

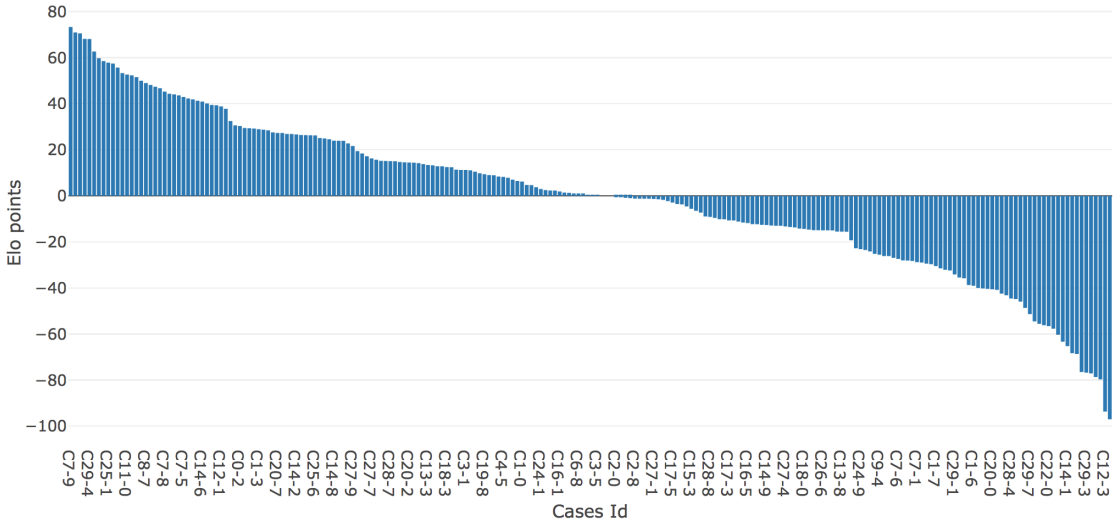

Fig. 3. Analysis of the Elo ranking of CB^0

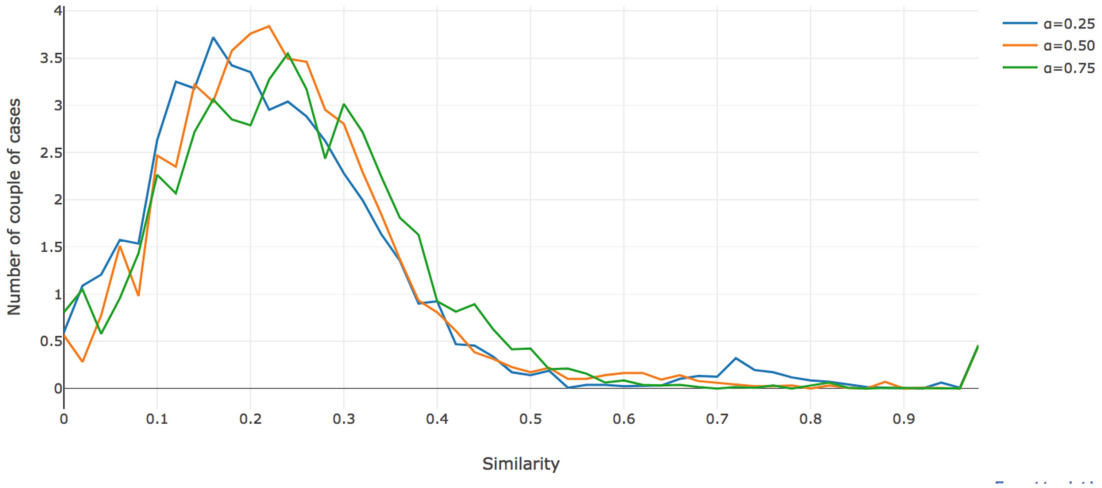

Fig. 4. Distribution of the pairwise similarity distances in CB^1

conclusion will be also corroborated by the following analysis that evaluates the coverage of the case base.

Regarding the impact of the α parameter, results do not show significant differences. This can be interpreted as a positive indicator about the quality of the similarity function in Eq. 1. Meaning that both components of the equation, the contextual similarity (sim_{ctx}) and the activities similarity (sim_a), compensate to each other. This way, there is not a predominant component when comparing cases: the context is as important as the plan itself.

We have also analyzed the pairwise similarities between cases according to the 30 different contextual setups that were used to generate them. Figure 5 shows a heatmap with the results. Every point represents the similarity of every possible couple of cases in the case base from white (no similarity) to dark blue (full similarity). Cases are organized according to the contextual setup, so we can observe similar groups of points in the form of small rectangles. The blue line

in the diagonal of the figure corresponds to the comparison of every case with itself. And the (mostly) blue rectangles around the diagonal line represent the similarities of cases from the same contextual setup. As expected, cases sharing the same contextual setup have a high similarity.

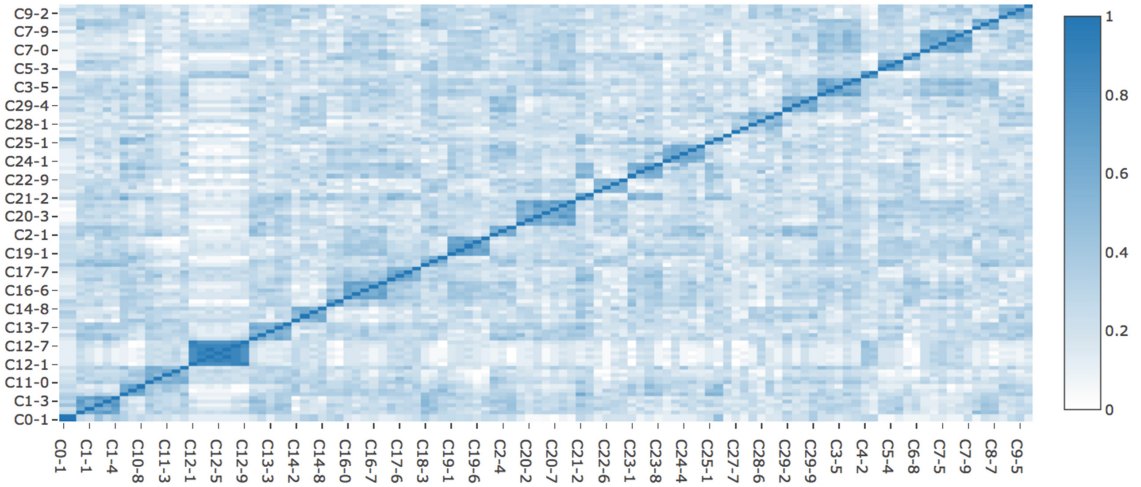

Fig. 5. Distribution of the pairwise similarity distances in CB^1 segmented by contextual setups, from white (no similarity) to dark blue (full similarity). (Color figure online)

Once we have analyzed the case base from the point of view of the similarity metric, we can extend this analysis by using the coverage metrics as explained next.

4.3 Coverage Analysis

The coverage metrics analyze the case base to find out their capability to solve new problems. These metrics have been extensively studied in the literature [17,27]. We have chosen the method described in Smyth et al. [29]. This metric is based on finding groups of cases and estimating their *density*, that averages the intra-group similarity. Having a high density it is more likely to find a proper case to be reused. In our case, it is straightforward to adapt this group-based metric to our case base, where cases are organized according to their contextual restrictions.

The *density* of a group is the average of the case densities of each group, where a group G represents a set of cases with the same contextual restrictions.

$$coverage(G) = \frac{1}{|G|} \sum_{c \in G} density(c, G) \tag{4}$$

$$where$$

$$density(c, G) = \frac{1}{|G| - 1} \sum_{c' \in G - \{c\}} similarity(c, c')$$

The blue line in Fig. 6 shows the coverage of each group defined in our case base CB^1 when using density to estimate coverage. As we can observe the coverage of some groups is really low. This result has two possible explanations: (1) there are not enough cases per group, or (2) we are not taking into account the adaptation stage of the CBR cycle.

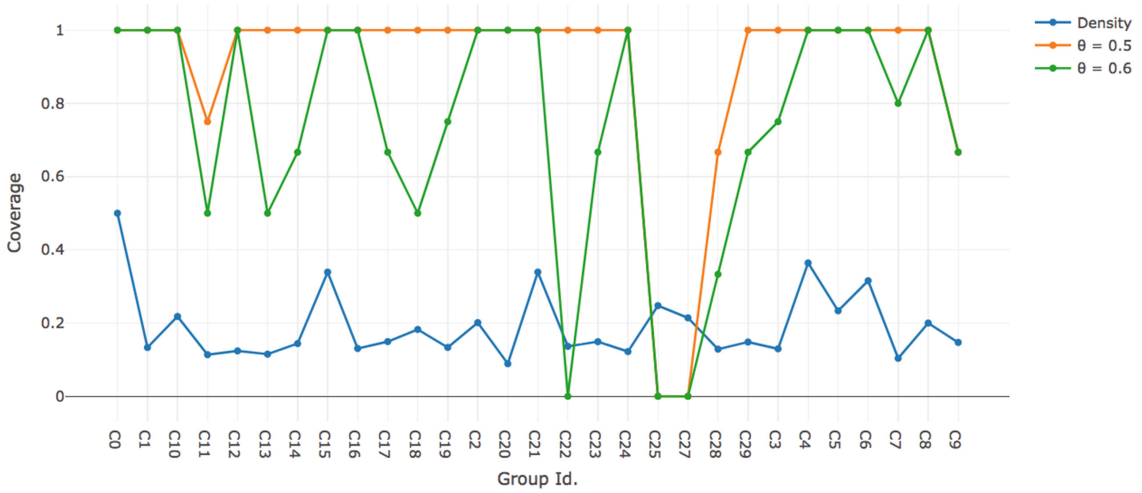

Fig. 6. Group coverage of the case base CB^1.

The probability that the same leisure plan could be repeated by another user is really low. For example, some plans contain temporal activities like a music concert or a football match. To reuse this plan, the system will change this activity because, most probably, this event is not available at the moment of the recommendation. Therefore, the adaptation stage is very important for our CBR system and needs to be incorporated into the coverage metric as explained next.

4.4 Enhanced Coverage Analysis

In our recommender system, the solution obtained by the CBR module needs to be adapted. Díaz et al. [8] describe the two main methods to transform a case solution: *transformational adaptation* and *constructive adaptation*. The first method consists of modifying the solution of the most similar case. On the other hand, the constructive adaptation creates a new solution by combining solutions from the most similar cases. Here, the key issue is the similarity of the cases with respect to the query. If cases are similar enough, the adaptation method would find a proper solution to the query. It is the basic assumption of CBR systems: similar problems have similar solutions.

Actually, coverage metrics in the literature also include the adaptation when evaluating the coverage. Therefore we redefine the coverage of a case base as the probability of being able to solve a new problem. And this probability can be estimated using the similarity of the most similar cases. Thus, we can assume

that a new problem would be solved if there are cases with a minimum similarity threshold. It can be formulated as follows:

$$coverage'(G) = \frac{1}{|G|}\sum_{c \in G} resolvability(c, G) \quad (5)$$

$$where$$

$$resolvability(c, G) = \begin{cases} 1 \; if \; highestSim(c, G) >= \theta \\ 0 \; if \; highestSim(c, G) < \theta \end{cases}$$

The $highestSim(c, G)$ function returns the similarity of the most similar case to c in G. As Fig. 6 shows, when including the adaptation in the coverage metric the performance of the case base increases significantly. Assuming that a solution can be found when the most similar case has a similarity over 0.6 the coverage improves up to 65% on average. Moreover, if we decrease this limit to 0.5 the coverage is complete for almost every context.

Although we have analyzed the case base segmented by contextual setups, we can also analyze it a as whole. Next, we will provide some insights about its features when leaving aside the contextual restrictions.

4.5 Global Analysis

We can also analyze the case base as whole leaving aside the contextual setups. This way, we can apply the $coverage'$ metric (Eq. 5) to CB^1 and study the impact of the θ parameter. We have applied this coverage metric for different θ thresholds, from 0.5 to 1.0. The Fig. 7 shows the obtained results. As expected, and according to the results shown in Fig. 6, the best coverage is achieved with a minimum similarity threshold of $\theta = 0.5$ (70% of coverage). It corroborates our hypothesis about the relevance of the adaptation step. Consequently, the coverage decreases when the similarity threshold grows.

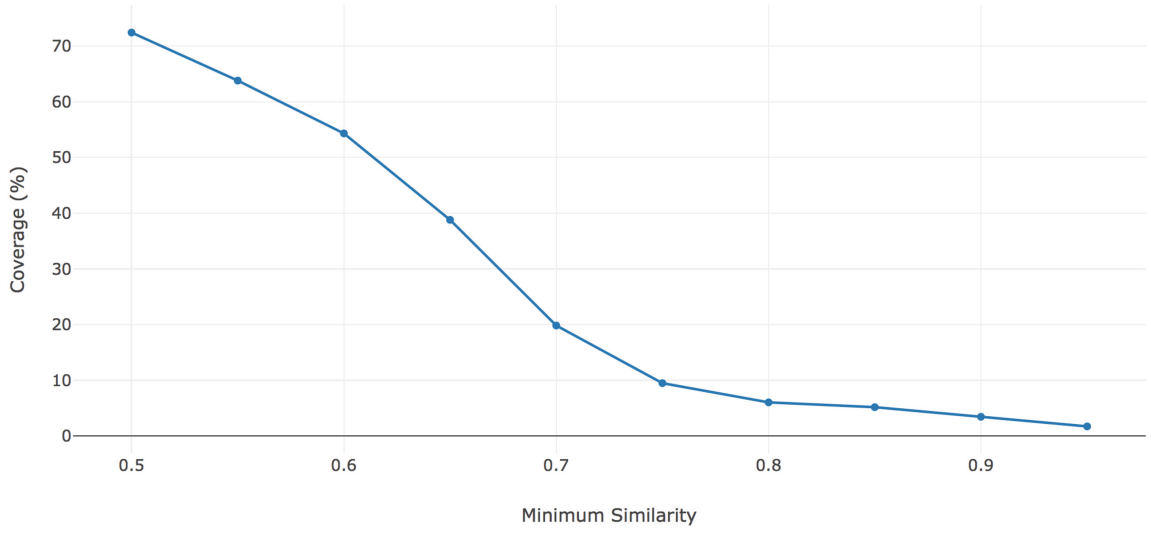

Fig. 7. Case base coverage respect different minimum similarities.

The last analysis tries to figure out the impact of the filtering according to the Elo scores of the original case base CB^0. Hypothetically, we could improve coverage by adding those cases that were deleted because of their low Elo score. We recalculated the coverage again but now we considered different sizes for the case base. We start to measure the coverage with the original CB^0 and repeated it but removing one-by-one the cases with worst Elo rating. The results of this experiment are shown summarized Fig. 8.

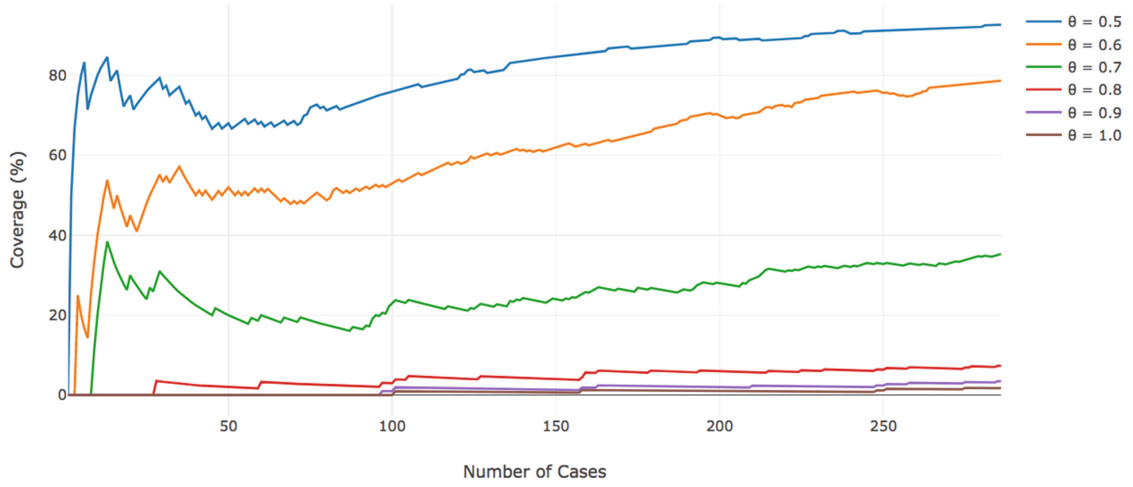

Fig. 8. Case base coverage respect different minimum similarities and the size of the case base.

These results show us that by using more cases we could improve the coverage up to approximately 20% as the coverage with $\theta = 0.6$ similarity threshold is closed to 80%, and the previous results were 65%. However, it is important to note that when including cases with low Elo scores, we are adding cases that do not make sense to the users and do not represent congruent plans. Therefore, we can conclude that a hypothetical maximum improvement of 20% does not worth the loss of semantic quality in the case base. This way, we can conclude that the filtering of cases in CB^0 according to the Elo ranking that obtains CB^1 is justified.

5 Related Work

One of the main challenges in recommender systems is acquiring the knowledge that is required to accurately recommend items [22]. Most of the research work in this area proposes capturing this information automatically based on different resources.

The growth of online resources and social networks permits acquire knowledge because they have a large amount of information. Aizenberg et al. [3] used the information from online radio stations to create a collaborative filtering recommender of music. In addition, Gottschlich et al. [12] proposed a decision support

system for investment decision using the votes from online communities. Social networks are used to acquire the knowledge used in recommender systems for groups [6,21,23]. Other works use online resources to create the knowledge for a recommender system like [4]. They created an ontology based on multiple taxonomies for profiling scholar's background knowledge of recommender systems.

In the last years, there have been many approaches to acquire the knowledge in context-aware recommender systems for tourism. Wang et al. [31] proposed a demographic recommender system of tourist attractions. They trained different machine learning methods to obtain the predicted rating and classify the demographic features to provide recommendations. To do that, they used the Tripadvisor reviews. Another similar work is proposed by Palumbo et al. [20] who created a neural network based on the FourSquare data to create a recommender system for point of interests. Other works have created some questionaries to acquire the knowledge of recommender system for tourism. For example, in [16] authors studied the influence of each context element in the recommendation of tourist items. A similar work is presented in [5]. They determined which contextual attribute is the most influential at the time of scoring a tourist activity.

All the works enumerated here need to train their systems to create the recommendations. By contrast, the use of a CBR system based on experience and learning avoids the training stage. Besides, our method deals with knowledge acquisition for contextual systems.

6 Conclusions

Contextual recommender systems have the capability to appreciate its environment and assess the situation in which the cases are to be recommended. They provide accurate recommendations that are better adjusted to a given situation or context. We have designed a contextual recommender system for leisure activities in Madrid using a CBR approach, where plans are stored and reused for similar situations. In this paper, we have discussed the knowledge acquisition difficulties of the system and how to solve the cold start situation when we have not plans to retrieve and reuse in a certain context. We have proposed a method where we first generate the candidate cases and then let the crowd vote to select and filter the best cases using an Elo test. With this method we obtain a case base with diversity, congruence between the activities, and collective knowledge that is independent from a domain expert. We have evaluated the case base using different metrics. We have analyzed the similarity, and coverage of the case base taking into account the contextual setups with and without adaptation, and we have presented a global analysis of the case base as a whole. The proposed method is generic, reusable and captures the wisdom of crowd method.

References

1. Adomavicius, G., Tuzhilin, A.: Context-aware recommender systems. In: Ricci, F., Rokach, L., Shapira, B. (eds.) Recommender Systems Handbook, pp. 191–226. Springer, Boston (2015). https://doi.org/10.1007/978-1-4899-7637-6_6
2. Aha, D.W.: The omnipresence of case-based reasoning in science and application. Knowl. Based Syst. **11**(5–6), 261–273 (1998)
3. Aizenberg, N., Koren, Y., Somekh, O.: Build your own music recommender by modeling internet radio streams. In: Proceedings of the 21st International Conference on World Wide Web 2012, p. 1. ACM Press, New York
4. Amini, B., Ibrahim, R., Shahizan, M., Ali, M.: Expert systems with applications a reference ontology for profiling scholar's background knowledge in recommender systems. Expert. Syst. Appl. **42**(2), 913–928 (2015)
5. Braunhofer, M., Ricci, F.: Selective contextual information acquisition in travel recommender systems. Inf. Technol. Tour. **17**(1), 5–29 (2017)
6. Carrer-Neto, W., Hernández-Alcaraz, M.L., Valencia-García, R., García-S'anchez, F.: Social knowledge-based recommender system. Application to the movies domain. Expert. Syst. Appl. **39**(12), 10990–11000 (2012)
7. Das Sarma, A., Das Sarma, A., Gollapudi, S., Panigrahy, R.: Ranking mechanisms in twitter-like forums. In: Proceedings of the third ACM International Conference on Web Search and Data Mining, pp. 21–30. ACM (2010)
8. Díaz-Agudo, B., Plaza, E., Recio-García, J.A., Arcos, J.-L.: Noticeably new: case reuse in originality-driven tasks. In: Althoff, K.-D., Bergmann, R., Minor, M., Hanft, A. (eds.) ECCBR 2008. LNCS, vol. 5239, pp. 165–179. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85502-6_11
9. Dufour-Lussier, V., Le Ber, F., Lieber, J., Nauer, E.: Automatic case acquisition from texts for process-oriented case-based reasoning. Inf. Syst. **40**, 153–167 (2014)
10. Elo, A.E.: The Rating of Chess Players, Past and Present. Arco, New York (1978)
11. Goel, A.K., Díaz-Agudo, B.: What's hot in case-based reasoning. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, California, USA, 4–9 February 2017, pp. 5067–5069 (2017)
12. Gottschlich, J., Hinz, O.: A decision support system for stock investment recommendations using collective wisdom. Decis. Support. Syst. **59**(1), 52–62 (2014)
13. Hacker, S., Von Ahn, L.: Matchin: eliciting user preferences with an online game. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 1207–1216. ACM (2009)
14. Jorro-Aragoneses, J.L., Díaz-Agudo, B., Recio-García, J.A.: Madrid live: a context-aware recommender system of leisure plans. In: 2017 International Conference on Tools with Artificial Intelligence, pp. 796–801 (2017)
15. Lam, X.N., Vu, T., Le, T.D., Duong, A.D.: Addressing cold-start problem in recommendation systems. In: Proceedings of the 2nd International Conference on Ubiquitous Information Management and Communication, ICUIMC 2008, pp. 208–211. ACM, New York (2008)
16. Laß, C., Herzog, D., Wörndl, W.: Context-aware tourist trip recommendations. In: CEUR Workshop Proceedings, vol. 1906, pp. 18–25 (2017)
17. Leake, D., Wilson, M.: How many cases do you need? Assessing and predicting case-base coverage. In: Ram, A., Wiratunga, N. (eds.) ICCBR 2011. LNCS (LNAI), vol. 6880, pp. 92–106. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23291-6_9

18. Leake, D.B.: CBR in context: the present and future. In: Case-Based Reasoning, Experiences, Lessons & Future Directions, pp. 1–30 (1996)
19. Manzoor, J., Asif, S., Masud, M., Khan, M.J.: Automatic case generation for case-based reasoning systems using genetic algorithms. In: 2012 Third Global Congress on Intelligent Systems (GCIS), pp. 311–314. IEEE (2012)
20. Palumbo, E., Rizzo, G., Baralis, E.: Predicting your next stop-over from location-based social network data with recurrent neural networks. In: 2nd ACM International Workshop on Recommenders in Tourism (RecTour 2017) RECSYS 2017, CEUR Proceedings, Como, Italy, 27–31 August 2017, vol. 1906, pp. 1–8 (2017)
21. Quijano-Sanchez, L., Recio-Garcia, J.A., Diaz-Agudo, B.: An architecture and functional description to integrate social behaviour knowledge into group recommender systems. Appl. Intell. **40**(4), 732–748 (2014)
22. Ricci, F., Rokach, L., Shapira, B.: Recommender systems: introduction and challenges. In: Ricci, F., Rokach, L., Shapira, B. (eds.) Recommender Systems Handbook, pp. 1–34. Springer, Boston, MA (2015). https://doi.org/10.1007/978-1-4899-7637-6_1
23. Shang, S., Hui, P., Kulkarni, S.R., Cuff, P.W.: Wisdom of the crowd: incorporating social influence in recommendation models. In: Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS, pp. 835–840. IEEE, December 2011
24. Shokouhi, S.V., Skalle, P., Aamodt, A.: An overview of case-based reasoning applications in drilling engineering. Artif. Intell. Rev. **41**(3), 317–329 (2014)
25. Shokouhi, S., Aamodt, A., Skalle, P.: A semi-automatic method for case acquisition in CBR a study in oil well drilling, pp. 263–270, January 2010
26. Sizov, G., Öztürk, P., Štyrák, J.: Acquisition and reuse of reasoning knowledge from textual cases for automated analysis. In: Lamontagne, L., Plaza, E. (eds.) ICCBR 2014. LNCS, vol. 8765, pp. 465–479. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11209-1_33
27. Smiti, A., Elouedi, Z.: Modeling competence for case based reasoning systems using clustering (2013)
28. Smyth, B., Keane, M.: Remembering to forget: a competence-perserving deletion policy for CBR systems. In: Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, Montreal, Canada (1994)
29. Smyth, B., McKenna, E.: Building compact competent case-bases. In: Althoff, K.-D., Bergmann, R., Branting, L.K. (eds.) ICCBR 1999. LNCS, vol. 1650, pp. 329–342. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48508-2_24
30. Sussman, G.J.: A Computer Model of Skill Acquisition. Elsevier Science Inc., New York (1975)
31. Wang, F.H.: On extracting recommendation knowledge for personalized web-based learning based on ant colony optimization with segmented-goal and meta-control strategies. Expert. Syst. Appl. **39**(7), 6446–6453 (2012)
32. Yang, C., Farley, B., Orchard, B.: Automated case creation and management for diagnostic CBR systems. Appl. Intell. **28**(1), 17–28 (2008)
33. Zhang, S., Lee, M.: Cognitive models and the wisdom of crowds: a case study using the bandit problem. In: Proceedings of the Annual Meeting of the Cognitive Science Society, vol. 32 (2010)

Capítulo 13

RecOnto: an ontology to model recommender systems and its components

13.1. Cita completa

Jose L. Jorro-Aragoneses, Gineth M. Ceron-Rios, Belén Díaz-Agudo, Juan A. Recio-García, Diego M. López-Gutierrez. *RecOnto: An ontology to model recommender systems and its components.* Proceedings of 29th IEEE Conference on Tools with Artificial Intelligence. Boston, United States. November 2017. P. 815-821.

13.2. Contribuciones de la publicación

El objetivo principal de esta tesis es el desarrollo de una plataforma basada en conocimiento para la construcción de sistemas recomendadores. Este conocimiento lo proporciona la ontología RecOnto que modela cómo son los sistemas recomendadores a partir de un conjunto de componentes. Esta publicación describe la primera versión de RecOnto.

13.3. Contributions covered by this publication

The main objective of this thesis is to develop a knowledge-based platform to build recommender systems. This knowledge is provided by the ontology RecOnto that models how recommender systems are based on a set of components. This publication describes the first version of RecOnto.

2017 International Conference on Tools with Artificial Intelligence

RecOnto: An ontology to model recommender systems and its components

Jose L. Jorro-Aragoneses[1], Gineth M. Ceron-Rios[2], Mª Belén Díaz-Agudo[1],
Juan A. Recio-García[1] and Diego M. López-Gutierrez[2]
[1]Universidad Complutense de Madrid, Spain
Email: jljorro, belend, jreciog@ucm.es
[2]Universidad del Cauca, Colombia
Email: gceron, dmlopez@unicauca.edu.co

***Abstract*—Nowadays, recommender systems are useful tools to filter items and information for users. There is a huge diversity of approaches to create customized recommendations. Because of this, a developer needs to know the features of these approaches to select which one is the best approach in a specific domain. In this paper, we explain the first step in the design of our intelligent framework to create recommender systems. This first step is called RecOnto: an ontology to model recommender system as a collection of components related between them. This ontology defines and classifies all components that compound a recommender system. Moreover, depending on the information used by the recommender system, it can filter the components used by the system. In addition, this ontology can be extended to add more components or apply this model in other domains. Finally, we explain an example about how to apply RecOnto to model CoCARE, a real context-aware recommender system in the health domain.**

I. Introduction

Recommender systems (RSs) have been an important research area since the mid-90's, defined as [1], tools for filtering and sorting items and information. RSs use different techniques and knowledge to identify content and items of interest from a potentially overwhelming set of choices. In the last few years the number of recommender systems in different domains (Netflix, Amazon, Last.fm,...) has increased due to the information overload that users can access [2], [3], [4]. Recommender systems help users to find interesting information using a variety of filtering and reasoning methods. There is a huge diversity of algorithms and approaches that help to create personalized recommendations.

This high number of techniques means that a developer needs to know what the best technique is each case. Our current work is to create an intelligence framework to ease the creation of recommender systems. In this paper we explain the first step. It is to design a tool that permits a future system to knows the meaning of a recommender system and its components. Our idea is divided recommender systems into modules, define each component and define the relationship with the rest of the modules. These modules and their restrictions help the system decide what are the best modules to design a recommender system and return a model based on the data used.

There are different approaches to model the information of recommender systems [5]. For our purpose we chose an ontology-based model. This model accomplishes the goals that we have in this model: a) create a meaning for each module; b) define restrictions to apply to each module and c) define a recommender system as a relationship between different modules. The use of an ontology to model recommender systems is not new [6], [7], [8], but usually is applied to a unique domain. This model ties to defining a general model that can apply to different domains.

In this paper, we present RecOnto. This is the first step in our framework. RecOnto is an ontology to model recommender systems. To apply it to most recommender system types we defined 2 groups of recommender systems based on [1]. This classification divides the recommender systems into 2 groups, depending on the number of inputs: 1) classic algorithms, like collaborative filtering, content-based filtering and hybrids, that only use the basic information of users and items, and 2) context-aware recommender systems (CARS). RecOnto defines a recommender system as a collection of modules that are connected between themselves. Then, it classifies each module based on the information used in them. The main contributions of our ontological model are: (1) define and classify all elements that take part in a recommender system and reuse these modules in a recommender system with different domains; (2) based on the information that the recommender system will use, filter the modules that cannot be applied in this system; (3) define a model that can be extended easily to apply in other domains or adding new recommender modules.

This paper runs as follows: section II reviews the related work and enumerates some classifications of recommender systems, recommender systems based on ontologies and frameworks to create a new recommender system. Next, section III explains the methodology used to design the RecOnto ontology. In section IV, we describe the recommender systems modelling ontology proposed in this paper and it explains how to include a new domain classifying below the

This work was financed by the University Complutense of Madrid, and Spanish Committee of Economy and Competitiveness (TIN2014-55006-R) and supported by the national department of sciences from Colombia called *Colciencias*.

2375-0197/17/31.00 ©2017 IEEE
DOI 10.1109/ICTAI.2017.00128

Capítulo 14

RecoLibry-core: A component-based framwework for building recommender systems

14.1. Cita completa

Jose L. Jorro-Aragoneses, Juan A. Recio-García, Belén Díaz-Agudo, Guillermo Jimenez-Díaz *RecoLibry-Core: A component-based framwework for building recommender system.* Knowledge Based-Systems. Vol. 182. 104854

14.2. Contribuciones de la publicación

En esta publicación, describimos RecoLibry-core, un framework para desarrollar sistemas recomendadores basado en la reutilización de componentes de otros frameworks. Esta es una de las herramientas incluida en RecoLibry Suite, e implementa los componentes definidos en el modelo RecOnto.

14.3. Contributions covered by this publication

In this paper, we describe RecoLibry-core, a framework to develop recommender systems based on the reuse of components provided by third-party frameworks. It is one of the tools contained in RecoLibry Suite, and it implements the components defined in the model RecOnto.

Knowledge-Based Systems 182 (2019) 104854

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Original software publication

RecoLibry-core: A component-based framework for building recommender systems

Jose L. Jorro-Aragoneses *, Juan A. Recio-García, Belén Díaz-Agudo, Guillermo Jimenez-Díaz

Department of Software Engineering and Artificial Intelligence, Universidad Complutense de Madrid, Madrid, Spain

ARTICLE INFO

Article history:
Received 13 March 2019
Received in revised form 5 June 2019
Accepted 16 July 2019
Available online 25 July 2019

Keywords:
Recommender systems
Component-based development
Java

ABSTRACT

Recommendation systems are a key part of almost every modern consumer website. These systems include techniques to filter, explore and rank a huge amount of information based on users' preferences or similar items. Designing and implementing a recommender system from scratch require skills of programming and recommending technologies. In this paper we describe RecoLibry-core, a framework to develop recommender systems based on the reuse of components provided by third-party frameworks.

1. Introduction

Recommender systems (RS) represent a very successful family of systems that explore and filter knowledge about items and users to predict the preference that a certain user would give to an item. Recommendation is based on a number of techniques that have been proposed in the literature (see [1] for a comprehensive review) and have been applied to many different domains of applications like movies, books or music [2].

The design process of a RS from scratch is a complex task where many decisions are taken. The system designer should be able to choose the most appropriate recommendation algorithm and configuration parameters. An expert designer would require an in-depth analysis on the available data and the behaviour of the algorithms. Besides, the choice depends on many different factors, such as the type of knowledge about the items and the target users, the data structure, the existence of social or contextual knowledge, the performance and the size of the knowledge base, and others.

In recent years, numerous frameworks have been created to make RS [3–6]. However, in most of these frameworks there are two major problems. First, many frameworks are oriented to a single type of recommendation methods such as Lenskit [3] (focused on collaborative filtering algorithms) or Tensorrec [6] (focused on machine learning). The second problem of these frameworks is that they are oriented to users with previous knowledge in the development of such type of systems. They do not offer non-expert users the guidance required to design or deploy recommender systems. In addition, an active research area is made tools to integrate different algorithms, for example [7], or tools to explain users algorithms or data used in a system [8].

In this paper we present a tool to solve these problems. RecoLibry-core is a Java framework to create RS by reusing components. It is a tool included in RecoLibry Suite, a set of intelligent tools to build RS. RecoLibry-core acts as a wrapper of components provided by third-party frameworks and it uses the dependency injection pattern to implement recommender systems based on the components selected by the developer.

2. Background

We have created a set of tools that facilitate the process of developing RS called RecoLibry Suite, which architecture is shown in Fig. 1. Firstly, we formalise semantically the representation of components that are typically used in RS, defining their behaviour and restrictions regarding their composition when developing a fully functional RS. This formalisation is carried out through an ontology called RecOnto. The second tool is RecoLibry-studio. It is a web application that guides the design process of an RS using RecOnto. The components described semantically by the ontology have their corresponding implementation in RecoLibry-core. It provides the components defined in RecOnto by wrapping external frameworks such as Mahout [5], Lenskit [3] or jCOLIBRI [4]. In addition, RecoLibry-core allows to use dependency injection to easily compose recommender systems from the components provided. Finally, RecoServer tool automatically deploys the RS

* Corresponding author.
E-mail addresses: jljorro@ucm.es (J.L. Jorro-Aragoneses), jarecio@ucm.es (J.A. Recio-García), belend@ucm.es (B. Díaz-Agudo), gjimenez@ucm.es (G. Jimenez-Díaz).

https://doi.org/10.1016/j.knosys.2019.07.025
0950-7051/

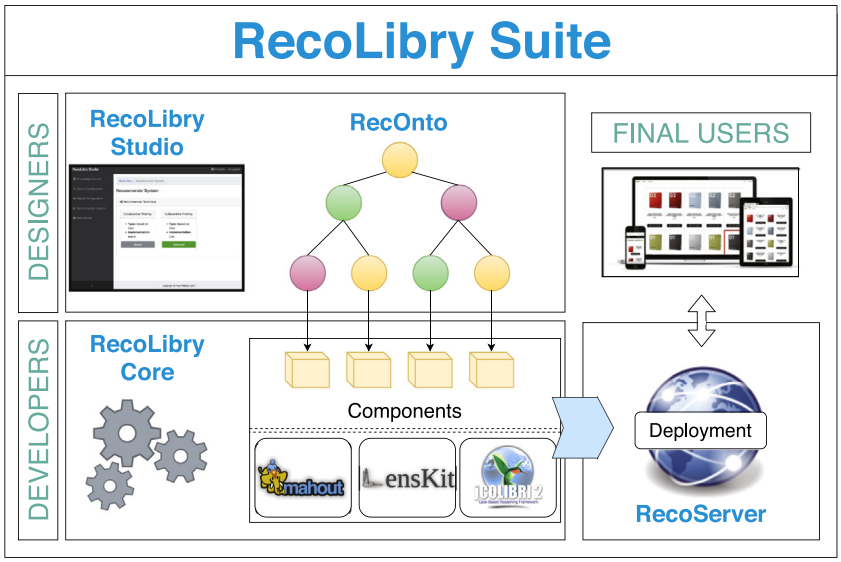

Fig. 1. RecoLibry Suite software architecture.

Table 1
Software metadata.

Nr.	(executable) Software metadata description	Please fill in this column
S1	Current software version	0.0.4
S2	Permanent link to executables of this version	https://github.com/UCM-GAIA/RecoLibry-Core
S3	Legal Software License	GNU v3.0
S4	Computing platform/Operating system	Microsoft Windows, Mac OSx, Linux
S5	Installation requirements & dependencies	Java JDK 8, Apache Maven
S6	If available, link to user manual – if formally published include a reference to the publication in the reference list	https://github.com/UCM-GAIA/RecoLibry-Core/wiki/Home-English
S7	Support email for questions	jljorro@ucm.es

and creates a RestFull-API with all the RS functionalities. It allows to use this API in an external application.

In the following section, we describe how the components are implemented in RecoLibry-core and how it uses dependency injection to build the final deployed RS.

3. Software framework

RecoLibry-core uses a scheme based on the definition of the minimum set of components necessary to build a new recommender system. The most important component in this scheme is `RecommenderSystem`. A `RecommenderSystem` component specifies the functionality of a RS. It needs two elements. The first one is a component that implements the `RecommenderAlgorithm` interface. It defines the methods that an algorithm must implement to be integrated in RecoLibry-core. The second one is an object that implements the `Query` interface. The goal of this interface is to define the structure of the queries that can be used in the implemented RS. Finally, `RecommenderAlgorithm` returns a list of `RecommenderResult` objects.

Regarding the concrete implementations provided by the framework, current version includes components to develop content-based RS, implemented through the jCOLIBRI framework, and collaborative-filtering RS, implemented by the Mahout library.

In addition, RecoLibry-core proposes a development process based on the injection dependency pattern. Concretely, it uses Google's Guice library [9] to easily compose the components included in the framework. In a nutshell, it adds a set of Java annotations that define how the components have to be combined. For example, Listing 1 shows the annotation of the `RecommenderSystem` component through the `@Inject` tag. This way, we define that a `RecommenderAlgorithm` and a `Query` objects are required to build a `RecommenderSystem`.

Following the dependency injection pattern, developers can easily build a RS by extending the `RecSysConfiguration` class or by defining a configuration file similar to the following one:

4. Conclusions

In this paper we present RecoLibry-core, a Java framework to create recommendation systems using components. The large number of existing frameworks to create recommendation systems together with the large number of recommendation techniques make the design of these systems very complex. RecoLibry-core alleviates this problem by integrating these frameworks into a homogeneous set of components. This way, RecoLibry-core provides the required components to build RS and defines a composition process through the dependency injection design pattern that eases the development of this type of systems.

Currently, RecoLibry-core provides components to build classic RS such as collaborative filtering and content-based. In future versions of RecoLibry-core we will add additional features such

Listing 1: Definition of RecommenderSystem constructor.

```
public class RecommenderSystem {
    @Inject
    public RecommenderSystem(RecommenderAlgorithm  algorithm,
                             Query query) {...}
}
```

Listing 2: Composition of recommender system with a JSON file.

```
{"injections": [{
 "type": "Class",
 "bind": "es.ucm.fdi.gaia.recolibry.api.RecommenderAlgorithm",
 "to": "es.ucm.fdi.gaia.recolibry.impl.MatrixFactorization"
},{
 "type": "Class",
 "bind": "es.ucm.fdi.gaia.recolibry.api.Query",
 "to": "es.ucm.fdi.gaia.recolibry.impl.MFQuery"
}]}
```

Table 2
Code metadata.

Nr.	Code metadata description	Please fill in this column
C1	Current code version	0.0.4
C2	Permanent link to code/repository used of this code version	https://github.com/UCM-GAIA/RecoLibry-Core
C3	Legal Code License	GNU v3.0
C4	Code versioning system used	Git
C5	Software code languages, tools, and services used	Java 8
C6	Compilation requirements, operating environments & dependencies	Apache Maven
C7	If available link to developer documentation/manual	https://github.com/UCM-GAIA/RecoLibry-Core/wiki/Home-English
C8	Support email for questions	jljorro@ucm.es

as explanations or group-based and context-aware recommendations.

Acknowledgements

Supported by the UCM (Research Group 921330), the Ministry of Economy and Competitiveness (TIN2017-87330-R) and the funding provided by Banco Santander in UCM (CT17/17-CT17/18).

Appendix. Required metadata

Current executable software version

See Table 1.

Current code version

See Table 2.

References

[1] F. Ricci, L. Rokach, B. Shapira, Recommender systems: introduction and challenges, in: Recommender Systems Handbook, Springer, 2015, pp. 1–34.
[2] D. Paraschakis, B.J. Nilsson, J. Holländer, Comparative evaluation of top-n recommenders in e-commerce: An industrial perspective, in: 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA), 2015, pp. 1024–1031.
[3] M.D. Ekstrand, M. Ludwig, J.A. Konstan, J.T. Riedl, Rethinking the recommender research ecosystem: Reproducibility, openness, and Lenskit, in: Proceedings of the Fifth ACM Conference on Recommender Systems - RecSys '11, ACM Press, New York, New York, USA, 2011, p. 133.
[4] J.A. Recio-García, B. Díaz-Agudo, P.A. González-Calero, Prototyping recommender systems in jcolibri, in: Proceedings of the 2008 ACM Conference on Recommender Systems - RecSys '08, ACM Press, New York, New York, USA, 2008, p. 243.
[5] S. Owen, R. Anil, T. Dunning, E. Friedman, Mahout in Action, Manning Publications Co., 2011, p. 375, Online, doi:citeulike-article-id:7544201, URL http://www.manning.com/owen/.
[6] J. Kirk, Tensorrec, 2018, https://github.com/jfkirk/tensorrec.
[7] C. Zhang, J. Bi, S. Xu, E. Ramentol, G. Fan, B. Qiao, H. Fujita, Multi-imbalance: An open-source software for multi-class imbalance learning, Knowl.-Based Syst. 174 (2019) 137–143.
[8] J.M. Moyano, E.L. Gibaja, S. Ventura, MLDA: A tool for analyzing multi-label datasets, Knowl.-Based Syst. 121 (2017) 1–3.
[9] R. Vanbrabant, Google Guice: Agile Lightweight Dependency Injection Framework, APress, 2008.

Capítulo 15

RecoLibry Suite: a set of intelligent tools for the development of recommender systems

15.1. Cita completa

Jose L. Jorro-Aragoneses, Belén Díaz-Agudo, Juan A. Recio-García, Guillermo Jimenez-Díaz *RecoLibry Suite: a set of intelligent tools for the development of recommender systems*. Automated Software Engineering. *To be published.*

15.2. Contribuciones de la publicación

Esta publicación describe la principal contribución de esta tesis, la plataforma RecoLibry Suite. En este artículo, describimos un modelo, una metodología y un conjunto de herramientas que facilitan el proceso de desarrollo de los sistemas recomendadores. Además, esta publicación compara la plataforma con otros frameworks.

15.3. Contributions covered by this publication

This publication describes the main contribution of this thesis, the platform RecoLibry Suite. In this paper, we describe a model, a methodology, and a set of tools that ease the developing process of recommender systems. Also, this paper compares our platform with other frameworks.

Automated Software Engineering
https://doi.org/10.1007/s10515-020-00269-4

RecoLibry Suite: a set of intelligent tools for the development of recommender systems

Jose Luis Jorro-Aragoneses[1] · Belén Díaz-Agudo[1] · Juan A. Recio-García[1] · Guillermo Jimenez-Díaz[1]

Received: 16 March 2019 / Accepted: 20 March 2020

Abstract

Recommendation systems are a key part of almost every modern consumer website. Recommender systems include techniques to filter, explore and rank a huge amount of information and items according to the user's current interests, and the similarity among users and items. Designing and implementing a recommender system usually requires high programming and machine learning skills. To alleviate these processes we present RecoLibry Suite: a set of intelligent tools to assist different types of users on the development of recommender systems. RecoLibry Suite supports not only the design and development of recommender systems but also its deployment as software as a service. We have evaluated the usability of the proposed tools with real users.

Keywords Recommender systems · Dependency injection · Framework suite · Components architecture · Ontology

Supported by the UCM (Research Group 921330), the Spanish Committee of Economy and Competitiveness (TIN2017-87330-R) and the funding provided by Banco Santander in UCM (CT17/17-CT17/18).

✉ Jose Luis Jorro-Aragoneses
jljorro@ucm.es

Belén Díaz-Agudo
belend@ucm.es

Juan A. Recio-García
jareciog@ucm.es

Guillermo Jimenez-Díaz
gjimenez@ucm.es

[1] Department of Software Engineering and Artificial Intelligence, Universidad Complutense de Madrid, Madrid, Spain

1 Introduction

Recommender systems (RS) represent a very successful family of systems that explore and filter knowledge about items and users to predict the rating that a certain user would give to an item. The extended use of recommender systems is best known for their use in e-commerce websites (Paraschakis et al. 2015), where they use input about the users interests to generate a list of recommended items. Recommender systems are useful to discover new items and personalize the list of recommendations according to the user preferences and contextual properties. For example, recommender systems typically suggest similar items to users based on past items they have reviewed or purchased. The system can also identify which items and users are similar to each other according to their properties and interaction story. Recommendation is based on a number of techniques that have been proposed in the literature (see Ricci et al. 2015 for a comprehensive review) and have been applied to a numerous domain of applications: movies, books, music, etc. (Paraschakis et al. 2015).

The development of a recommender system from scratch is a complex task where many decisions are made regarding its design and implementation. To alleviate this process, in the last few years there has been a proliferation of frameworks and code libraries that assist users in this task (Ekstrand et al. 2011; Schelter and Owen 2012; Sarwat et al. 2017; Chan et al. 2013; Recio-García et al. 2014a). However, the development process of a recommender system must be adapted to the technical skills of the users involved in its definition and development. Designers require high-level tools to import the required data, select a proper recommendation technique and deploy the system. On the other hand, developers with technical skills may prefer a low-level framework or library that can be customised or extended through a programming interface.

Despite the users' technical skills, development of a recommender system implies many processes and decisions. Users must be able to choose the most appropriate recommendation algorithm or hybrid approach together with its configuration parameters. They would require an in-depth analysis on the available data and the understanding of the behaviour of the algorithms. Besides, the choice depends on many different factors, like the type of knowledge about the items and the target users, the data structure, the existence of contextual knowledge, the performance and the size of the knowledge base, among others.

Therefore, the creation of a recommender system does not only require a low-level framework or library, but also the definition of a whole architecture and the design of a process to guide the development according to the user's requirements. Following this assumption, we present RecoLibry Suite, a development methodology and set of tools that allow users to easily design, develop and deploy recommender systems. The first tool, RecoLibry-core, is an object-oriented framework in Java that offers a catalogue of components for the implementation of recommender systems. The second tool, RecoLibry-studio, is a web application with a visual interface, where non-expert users can design a system by specifying its required functionality in terms of the inputs and the expected

outputs. Both tools use the RecOnto ontology to guide the development, connecting the components and verifying the system restrictions. Finally, the RecoServer tool deploys the generated system to a web server, creating a set of web services in order to use the recommender system.

Our approach is mainly based on reusing software components. Software reuse is a goal that the Software Engineering community has pursued from its very beginning (McIlroy 1969; Frakes and Nejmeh 1987). When designing a new recommender system the use of libraries of software components requires specific programming skills, a detailed model or software specification, and expert knowledge about the recommendation domain. Our research group[1] has a long trajectory on the development of tools and frameworks (Gómez-Albarrán et al. 1998; Bello-Tomás et al. 2004; Recio-García et al. 2014a) based on reusing components.

A contribution of this paper is an exhaustive analysis of the existing frameworks for the development of recommender systems, which is reported in Sect. 2. Next, Sect. 3 describes the design principles of RecoLibry Suite. It defines the tools workflow, i.e., the steps needed to design and develop a recommender system, followed by a description of the 3 tools contained in RecoLibry Suite and described Sect. 4: RecoLibry-core, RecoLibry-studio and RecoServer. This section also deepens on how the component-based development in applied in the design process.

Later, we evaluate the usability of RecoLibry Suite with real users. Our hypothesis is that RecoLibry Suite tools simplify the process of designing and implementing a new recommender system. To do that, we have compared the development process of recommender systems using different frameworks. The results detailed in Sect. 5 are satisfactory based on users' answers. Finally, this paper is concluded in Sect. 6, where lines of future work are presented.

2 Review of related work

There are different families of recommendation algorithms. Collaborative filtering algorithms employ a set of user ratings over items to recommend items with similar ratings, or items that have been useful to users with similar ratings (Su and Khoshgoftaar 2009). On the other hand, content-based algorithms compare user preferences, usually stored in a user profile, with the item description to know which items are relevant to users (Lops et al. 2011). Contextual recommendation systems have become popular in recent years (Adomavicius and Tuzhilin 2015). These systems use dynamical context information, like location, time, weather... that can influence the user to change their preferences (Chen et al. 2000; Van Setten et al. 2004; Jorro-Aragoneses et al. 2017b; Ayala-Gómez et al. 2018).

With the large number of algorithms and possible configurations the process of developing a recommender system is not trivial. In recent years, numerous frameworks have been developed to create recommender systems. Table 1 shows a list of frameworks created since 2011. A total of 22 different frameworks have been

[1] http://gaia.fdi.ucm.es.

Table 1 List of frameworks to create recommender systems

Framework	Language	Year	Collaborative filtering	Content-based	Machine learning	GUI	Extensibility	Evaluation	Deploy
PredictionIO	Java, PHP, Python, Ruby	2013			X	X	X	X	X
HapiGer	Node.js	2015	X				X		
Mahout	Java, Scala	2012	X		X		X	X	
Seldon	Java, Python, R, PySpark	2018			X		X		X
Lenskit	Java, Python	2011	X				X	X	
Oryx v2	Java	2018	X		X		X		X
RecDB	SQL	2017	X						X
Surprise	Python	2017	X				X	X	
LightFM	Python	2015	X	X				X	
Rexy	Python	2017	X	X				X	
QMF	C++	2017	X					X	
TensorRec	Python	2018			X			X	
SpotLight	Python	2017			X			X	
recommenderlab	R	2018			X		X	X	
CaseRecommender	Python	2017	X		X			X	
LibRec	Java	2015	X	X	X			X	
RankSys	Java	2016	X		X			X	
LIBMF	R	2016	X					X	
proNet-core	C++	2017			X				
jCOLIBRI	Java	2015		X			X	X	
MyMediaLite	C#	2011	X	X			X	X	

analysed. All of them are open source or academic frameworks. In the table we have pointed out the following features for these frameworks:

- The type of algorithms implemented in each library (collaborative filtering, content-based and machine learning algorithms).
- The existence of a graphical interface that helps users to create the recommender system.
- The flexibility to extend the framework creating new algorithms using the framework resources.
- The integration of tools or methods to evaluate the recommender systems created.
- The capabilities to help users to deploy the created recommender system.

If we analyse the types of algorithms implemented, we can see that most of them use collaborative filtering algorithms and/or machine learning. This is due to the popularity that both methods have obtained in the last years. First of all, collaborative filtering algorithms have come a long way since Netflix proposed a contest to enhance the recommendation system in their platform (Koren and Bell 2015). Thanks to this fact, most libraries supports the use of collaborative filtering algorithms. Two libraries are the most popular among the analised frameworks. Lenskit (Ekstrand et al. 2011) has a prominent popularity due to its widespread use at the academic level and in the recommender system called MovieLens (Harper and Konstan 2015), which offers different datasets employed to evaluate recommend systems. Another popular library is Mahout (Schelter and Owen 2012), which is widely used for its ease of creating recommendation systems based on collaborative filtering. Another framework that uses collaborative filtering algorithms is RecDB (Sarwat et al. 2017). Its main feature is that it is not a programming framework but a module that is installed in PostgreSQL to retrieve recommendations directly from the database created by the user.

On the other hand, the increase in information and the lower cost of devices for storing it have popularised the use of machine learning techniques (Landset et al. 2015). Libraries that use machine learning can be classified into 2 groups: those that use an external library for machine learning algorithms or, on the other hand, those that implement the algorithms within the library itself. In the first group we find libraries such as PredictionIO (Chan et al. 2013), Apache Mahout, Oryx v2 (Sean 2018) and Seldon,[2] which delegates on the Apache Spark library (Apache 2016) for their recommendation systems. In the same group we find TensorRec (Kirk 2018), which uses the Tensorflow service (Abadi et al. 2016). In addition to all these frameworks, SpotLight (Kula 2017) uses the PyTorch library (Paszke et al. 2017) to apply Deep Learning algorithms in recommendation systems. The rest of frameworks that use machine learning methods implement their own algorithms.

[2] https://www.seldon.io/.

Table 1 also shows that the least common type of algorithm implemented in the analysed frameworks is content-based. Among the frameworks that provides this type of algorithm we find LightFM (Kula 2015), Rexy (Vând 2017), LibRec (Guo et al. 2015) and MyMediaLite (Gantner et al. 2011). In addition to these frameworks, we want to highlight the jCOLIBRI framework (Recio-García et al. 2014a) that supports the creation of case-based reasoning (CBR) systems (Kolodner 2014). In (Recio-García et al. 2008) the authors explain how to use jCOLIBRI to develop recommender systems.

Another feature commonly found in the analysed frameworks is that most of them are oriented to users with high skills in application development and, more specifically, in developing recommender systems. Only two of them have a graphical interface to support the creation of recommendation systems. PredictionIO allows the customisation of a set of templates to create recommendation systems, whereas jCOLIBRI provides the COLIBRI-Studio application (Recio-García et al. 2014b) that supports the creation of CBR systems using a graphical interface.

At the experimental level, one of the most important characteristics that a framework must have is extensibility in order to ease the implementation of new algorithms reusing the elements provided by the library. Table 1 shows that only half of the frameworks allow the implementation of new algorithms. In some cases this is due to the use of an external API, as in the case of TensorRec, or because the framework is designed to create final recommender systems, as is the case of RecDB. Another important aspect at experimental level is the support to evaluate the systems created by the users. For this reason, almost all frameworks have tools and methods to evaluate the developed recommender systems.

The last aspect we have analysed is whether the framework supports the deployment of the recommender system in production. This feature is only included in a few libraries. This is due to the difficulty of generalising the steps to deploy a system automatically.

3 Design of the RecoLibry Suite tools

According to the weaknesses found in the frameworks analysed in previous section, we have defined the following underlying design goals for the RecoLibry Suite:

- Complete: the framework must contain up-to-date algorithms used in recommender systems.
- Extensible: it is easy to integrate new algorithms into the framework and make them available for future reuse.
- Flexible: each reusable component includes a set of parameters to configure the algorithm.
- User-oriented: our tools must be adapted to users with different technical skills: the framework provides different interfaces and GUI tools to create recommender systems for user without programming experience.
- Self-contained: the framework generates standalone systems without external dependencies.

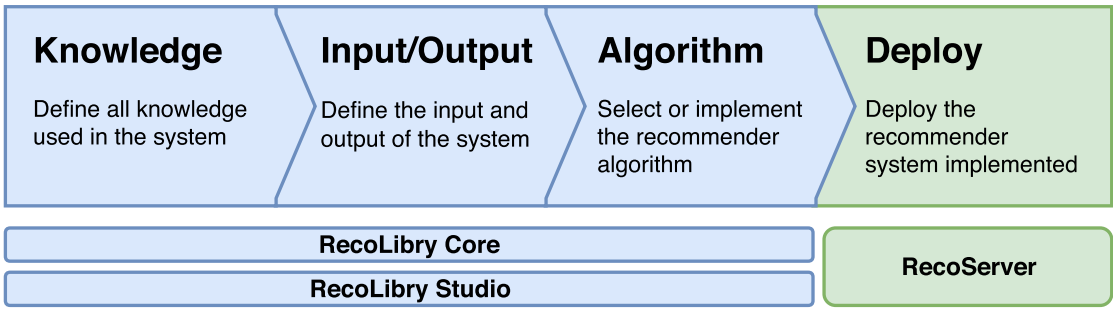

Fig. 1 RecoLibry Suite workflow to develop a recommender system

Following these design goals next section gives an overview of the development process for creating a new recommender system with RecoLibry Suite. First, we introduce the steps to design and develop a recommender system and we explain how we implemented these steps in the RecoLibry Suite workflow. Next, we present the knowledge-level description of the software components through RecOnto and how this ontology guides the user in the development process.

3.1 Development process in RecoLibry Suite

The development of a recommender system in RecoLibry Suite has been defined as a four steps process: Knowledge, Input/Output, Algorithm and Deploy. This development process is illustrated in Fig. 1 and it runs as follows:

1. Firstly, it is necessary to determinate the *knowledge* available for generating the recommendations. For example, a recommender system of movies must be implemented using a collaborative filtering approach only if movie ratings are available, or a content-based approach if the designer provides movie descriptions. This way, once the available knowledge is defined, our tools will be able to select the most suitable algorithm.
2. The next step in the design process is the definition of the *input and output* of the recommender system. The system input will establish the kind of query submitted by the final users. For example, the query could be just a user identifier and the system will return a recommendation for this user (individual recommender), or the query could be a set of identifiers that represents a group of users who want to watch a film together (group recommender). The output of the recommender system has also a direct impact in the recommendation algorithm, as it can be a rating prediction, a list of items recommended to a user, or a diversified set of items, among others.
3. When users have defined the previous steps, the system has enough information to guide them in the selection of the most suitable recommendation *algorithm*. This critical step is guided by the knowledge contained in the ontology described in Sect. 3.2. Most of the algorithms will be filtered according to the restrictions defined in the previous steps although there are more features to be taken into account. For example, sometimes the data used in a recommender system has some problems or specific features, like the long-tail problem (a few items have

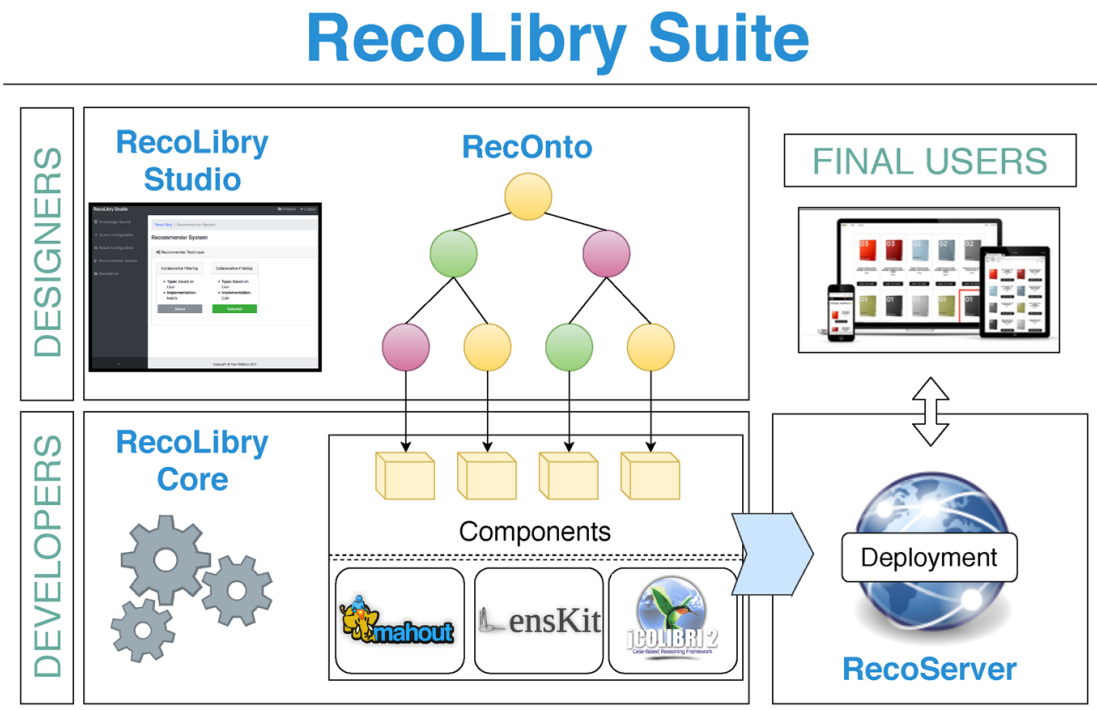

Fig. 2 RecoLibry Suite software architecture

many ratings and, on the other hand, many items have a few ratings). In these cases, there are algorithms specifically suited for this kind of datasets that should be proposed by the RecoLibry Suite tools.

4. The final step in our development process is the *deployment*. Our tools are able to provide a fully-functional and ready-to-use recommender system that can be integrated in a commercial system or, alternatively, could be evaluated in order to check its performance. The most flexible alternative to achieve these goals is the deployment of the recommender system *as a service*. However, it is technically difficult as the developer needs to configure a server, install the required packages and launch all the services.

The tools in RecoLibry Suite (Fig. 2) support the development process described above. The first three steps can be performed using RecoLibry-core or RecoLibry-studio, depending on the programming skills of the user who defines the recommender. Additionally, we provide the RecoServer tool that supports the automated deployment of the recommender system designed with RecoLibry Suite. The tools will be described in depth in Sect. 4.

Development process and tool dependencies are also orchestrated by the knowledge level descriptions in the RecOnto ontology. Next section explains in detail how this knowledge is represented in RecOnto.

3.2 Knowledge level descriptions using RecOnto

Component-based software design has long been popular in the development of complex systems (Szyperski 1999; Heineman and Councill 2001). The use of reusable components reduces the cost of system development by focusing on specific system functionalities. This type of design is equally useful in recommendation systems (Szyperski 2000). Most of these systems can be decomposed into components with a single functionality, e.g. managing user ratings, generating the query used by the recommender system, etc. The behaviour, requirements and signature of a component can be described semantically in order to use this knowledge to guide the user during the composition of a recommender system. This is the goal of RecOnto (Jorro-Aragoneses et al. 2017a), an ontology that describes semantically the components integrated into RecoLibry-core to validate the correctness of the systems implemented by developers. Moreover, this ontology is used to guide the high-level development process followed by the RecoLibry-studio tool. This way, RecOnto is a basic building block in the software architecture of RecoLibry Suite, as illustrated in Fig. 2.

A useful way for describing components is in terms of the tasks to be solved, the goals to be achieved, the methods that will accomplish those tasks, and the domain knowledge that those methods need. A description along these lines is referred to as a knowledge level description Newell (1982).

RecoLibry Suite uses the same approach employed in jcolibri (Recio-García et al. 2014a) and relies on RecOnto for organising the terminology found in recommender systems to provide a framework-independent basis for new systems. RecOnto compiles a common language to define the elements that compose a recommender system and to support the development of generic recommender system methods reusable through different domains. RecOnto[3] is represented using the Ontology Web Language (OWL) standard.

First of all, we have defined the elements that compose a recommendation system following a component-oriented schema. Each component represents a functionality, for example, a component that stores the information of an item. Once we have created a framework-independent representation of the required component, it is linked to a concrete instantation of such behaviour. This way, each semantic description of a component in RecOnto will be linked to a concrete implementation (a class or interface) in RecoLibry-core. This semantic description will also include the set of rules that specify how components can be composed with each other.

The main scheme of RecOnto, as long as the most important components, is shown in Fig. 3. A recommender system (class `Recommender System`) is associated with an algorithm (`Recommender Algorithm`). Each algorithm is connected to various components that define the information they will use and return for each recommendation. A recommendation algorithm must have at least 2 defined components: a `Query` and a `Result Returned`.

[3] An interactive visualisation of the ontology is available at http://gaia.fdi.ucm.es/ontologies/reconto.

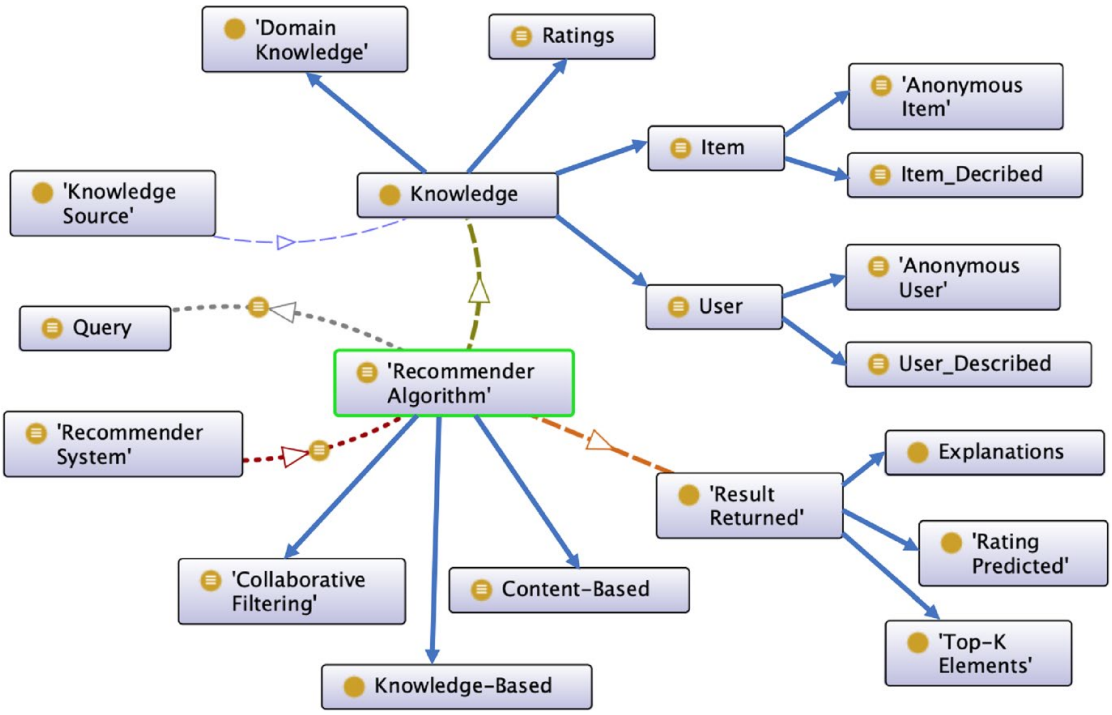

Fig. 3 Most important components in RecOnto.

A `Query` represents the data sent to the recommender system to obtain a result. For example, a `Query` stores the identifier of the user for whom you want to obtain a set of recommended items. On the other hand, `Result Returned` represents the response that we expect from the recommender system. We define three different types of recommendation results. The first one is `Top-K Elements`, which represents a list of K elements sorted by relevance. The second is called `Rating Predicted` and returns the prediction of the recommender system for an item. The last one represents the `Explanations` generated by the recommender system about the calculated recommendation. A recommendation system can return one or more types of result, e.g. a rating prediction and an explanation.

In addition to these components, a recommendation algorithm may be associated to one or more components of the `Knowledge` category. These components will provide information that a recommender system will use in its recommendations, either because they concern the catalogue of items to recommend, or because they are information that the system will use to infer the results. RecOnto defines 4 types of knowledge. `Item Data` is a representation of the information that the system has about an item. The second type of knowledge is `User Data`. It contains all the information that allows to describe a user or his preferences. Another category of knowledge is `Rating Data`. It groups together all the elements that define a rating: the user who rates, the rated item and the rating itself. This makes `Rating Data` dependent on `Item Data` and `User Data`. The last category of knowledge is `Domain Data`. Within this category is classified any data set that provides information from the recommendation domain, i.e., from the scope of the items being recommended.

All elements belonging to `Knowledge` category are connected to a `Knowledge Source`, a data structure that stores all the data associated with that knowledge. In addition, it provides a set of functions for adding, removing or modifying attributes of that knowledge.

When a recommendation system uses a `Knowledge` component, there may be some features that can characterise the information source that will be important for the recommendation process. For this reason, a `Knowledge` may be characterised by one or more `Knowledge Property` elements. For example, a set of ratings may suffer from a sparsity problem, i.e. there are many empty values in the matrix of user ratings and items.

All the components previously described are associated with different types of recommendation algorithms. RecOnto defines three large groups of recommendation algorithms: `Collaborative Filtering`, `Content Based` and `Knowledge Based`. Within these classes, we find all recommendation algorithms implemented in the RecoLibry Suite tools.

Every algorithm that exists within RecOnto defines a set of rules that fixes with which components can relate to. These restrictions are represented using the Description Logic capabilities of the OWL language. Description Logics (DLs) are a family of class-based knowledge representation formalisms that model concepts, roles and individuals, and their relationships. In addition, DLs allow to ask questions about the concepts and instances described.

Having all this knowledge available implies that our tools are able to find the algorithms that best fit the set of components defined by a user. This way, the RecOnto ontology supports the intelligent composition of recommender systems using the RecoLibry Suite tools, that we will describe next.

4 RecoLibry Suite tools

RecoLibry Suite comprises 3 tools for the design and deployment of recommender systems: RecoLibry-core, RecoLibry-studio and RecoServer.

Depending on the programming skills of the user who defines the recommender system, the design can be made using RecoLibry-core or RecoLibry-studio. If users are experts on the development of recommender systems, they can use RecoLibry-core. Using this tool, users can create a recommender system by combining the provided components programmatically. On the other hand, if users do not have enough development skills they can use RecoLibry-studio. It is a web application that guides the development of a recommender system using visual tools, which suggest the most suitable algorithm according to the available knowledge and the input/output restrictions defined by the user.

Finally, RecoServer will be responsible for the deployment of the recommender system developed either with RecoLibry-core or RecoLibry-studio, instantiating the recommender system and generating its corresponding RESTful API to be used by external applications.

In the following subsections we will explain in detail each of these tools.

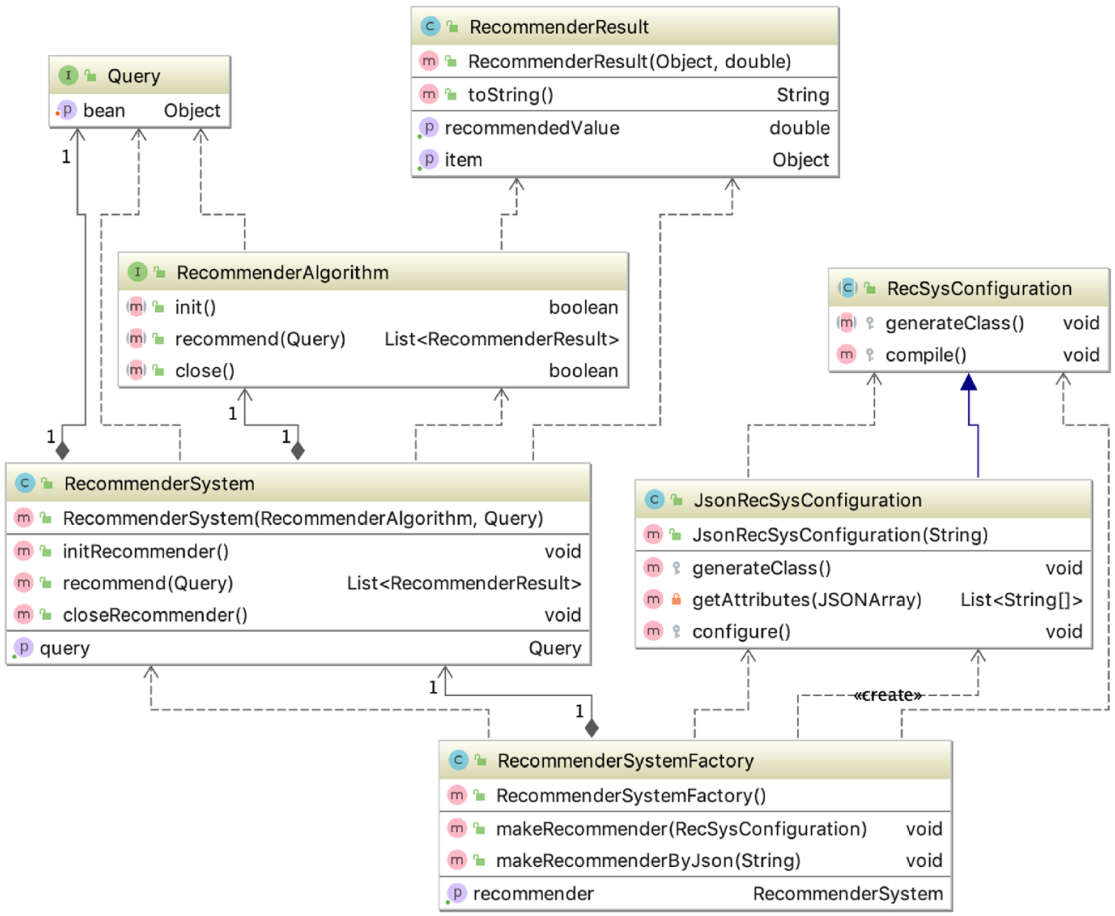

Fig. 4 Recommender systems scheme in RecoLibry-core

4.1 RecoLibry-core

RecoLibry-core is a Java framework to build recommender systems (Jorro-Aragoneses et al. 2019). It follows a component-based architecture that eases the process to create new components, reuse them and build new recommendation systems by coupling these components together. RecoLibry-core defines the schema of a recommender system using a set of interfaces and abstract classes.

This schema manages the workflow among components to correctly execute the basic functions of a recommender system. These basic functions are: initializing the recommender system, computing a recommendation from a query and closing the recommender safely. Figure 4 shows the UML diagram with the classes that define the main scheme of a recommendation system in RecoLibry-core. The most important component is the `RecommenderSystem` class. This class defines that a recommender system is composed by:

- `RecommenderAlgorithm`: it is an interface that defines the methods that a recommender algorithm must implement to be integrated into RecoLibry-core. It specifies methods to initialise and release the component and to provide the recommendation itself.

- `Query`: it is an interface that defines the structure of the queries that can be made to the recommender system. This interface manages queries as bean (pojo) objects.
- `RecommenderResult`: it is a class that stores the results generated by the recommender system. This class contains the recommended item and the recommendation value, obtained with the recommendation algorithm.

RecoLibry-core reuses the implementations for recommender system algorithms taken from several third-party libraries (some of them reviewed in Sect. 2). Obviously, we cannot use most of the components directly as they will be very heterogeneous. Therefore, the RecoLibry-core framework acts as a wrapper that encapsulates them. In the current version, RecoLibry-core has component instantiations to develop either content-based recommender systems—using the jcolibri library—or collaborative-filtering recommender systems—implemented with the *Mahout* library.

To facilitate the reuse of recommended system components, RecoLibry-core has been designed using the dependency injection programming pattern. This design allows you to create different components and define how they must be coupled to make a new recommendation system. The main advantage of using dependency injection is that it isolates the usage from the creation of the object, simplifying the creation of a recommender system with a few lines of code or even from a configuration file. Dependency injection in RecoLibry-core is supported by Google Guice library.[4] This library adds a set of Java annotations that define how the components have to be coupled and how to build the recommender system automatically.

There are tow ways to specify how to compose the components in order to create a recommender system: by implementing a class in Java or by using a configuration file. The first option consists on implementing a class that inherits the abstract class `RecSysConfiguration`. This class defines the methods that the user has to implement to build a recommender system. The second way to define the recommender system is through a JSON file. This configuration file can be defined manually or generated automatically by the RecoLibry-studio tool. Moreover, this configuration file contains all the information required to deploy the defined recommender system into RecoServer. Both processes will be detailed in the following subsections. Concretely, it is `RecommenderSystemFactory` the class in charge to generate an instance of the recommender system from that configuration file.

RecoLibry-core is publicly available in a Github repository[5] and it includes a development guide with additional details about the creation of a recommender system using this framework. Jorro-Aragoneses et al. (2019) also provides further documentation.

[4] https://github.com/google/guice.

[5] https://github.com/UCM-GAIA/RecoLibry-Core.

4.2 RecoLibry-studio

The process of creating a new recommender system using RecoLibry-core can overwhelm users with low experience in recommender systems or low programming skills. RecoLibry-studio is a more suitable tool for non-developer users, oriented to design recommender systems. It consists on a web application where designers can define recommendation system using visual elements and an intelligent assistant that guides the designers in the selection of which algorithms they should apply to their recommender systems.

Another feature of RecoLibry-studio is its capability to check the correctness of the combination of components selected by the user thanks to (Sect. 3.2). While designers are choosing the components for their recommender system, RecoLibry-studio checks whether these components are correct and which algorithms could be applied by checking them against the ontology.This feature allows RecoLibry-studio to be used also as an academic tool.

RecoLibry-studio follows the four-step design process explained in Sect. 3.1: define the knowledge, select the input and output information (query and results), select the recommender technique and deploy the system designed. RecoLibry-Studio uses the ontology to show which algorithms can be used based on the input and output selected. In addition, once the designer has completed the recommender system configuration, she will download a configuration file to instantiate the recommender system with RecoLibry-core or to deploy the recommender system in RecoServer.

Each step will be explained in detail below. We explain the process using an example to create a movie recommendation system. This system will use the Movielens dataset.[6] For each step we explain the information a final user should input and the corresponding intelligent assistance provided by RecoLibry-studio.

4.2.1 Knowledge source

As we defined before, the first step in the system design in RecoLibry-studio is to determine the knowledge sources of the recommender system. To define the data source where the system will retrieve these information, RecoLibry-studio uses `connectors`.

A connector is a component that reads the information employed by the recommender system. In addition, this connector can write back new information or modify any saved data. Designers should specify which type of format their datasets have. RecoLibry-studio allows connecting the system to a SQL database or to a file with the information (JSON, CSV or XML). Depending on the connector type selected, designers should insert the information to connect with this information source. When designers select to use their own connectors, RecoLibry-studio analyses the provided dataset, the attributes that define items and designers, and how ratings are specified.

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

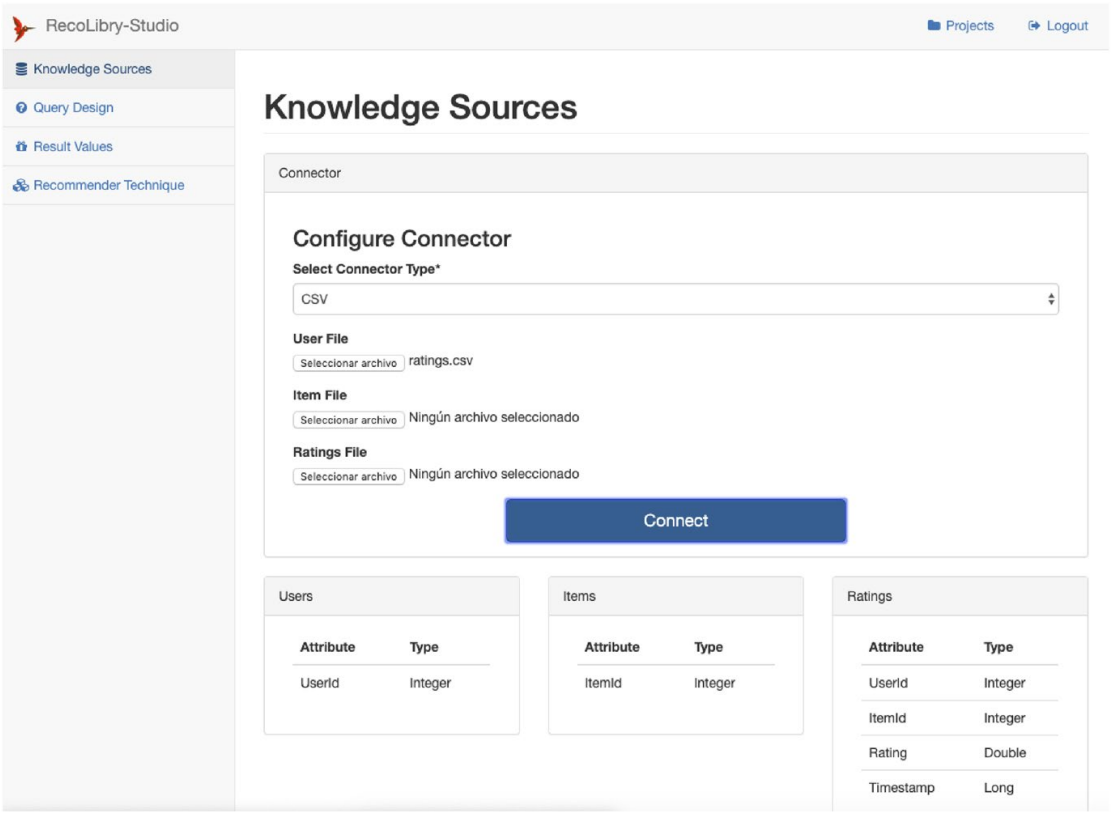

Fig. 5 Window to select the knowledge Source in RecoLibry-studio

Figure 5 shows how to configure the *Knowledge Sources* step for our example. In this case we use as knowledge source a *CSV* file that contains all the movie ratings. For that reason, in this step we select our own connector. In addition, we select the type of connector—in this example the type is *CSV*—and which files it will use—in this example it will only use a rating file. After clicking on the *Connect* button, RecoLibry-studio will read the *CSV* file header and will show which attributes represent users, items or ratings. In this example, the first 2 have only one identification attribute. In the case of ratings, we have: the id of the user, the id of the item, the rating that the user made to the item and when he made that valuation.

Once designers have defined the knowledge source used by the systems, they can complete the next step in the recommender design: the input and the output of the recommender system.

4.2.2 Query design

In the *Query Configuration* form, designers describe which information their systems will receive from final users when they want to obtain a recommendation. In this step, the system shows all the previous information added in the knowledge source step. This way, designers can select which attributes will be in the query.

Following our example, the recommender system only needs to receive the identifier of the user for whom the recommendation is to be made. In the configuration of the query we select the user identifier as the unique value of the query, as shown in Fig. 6.

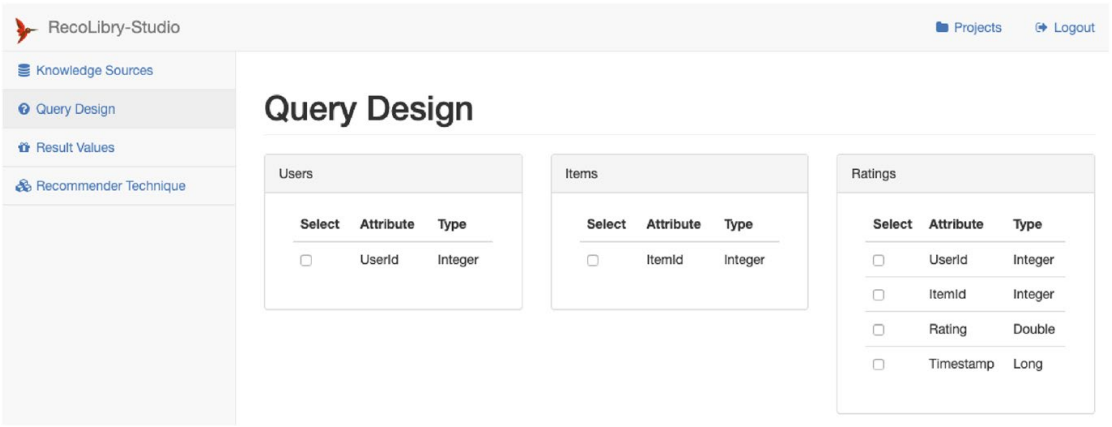

Fig. 6 Window to define the data contained in a query

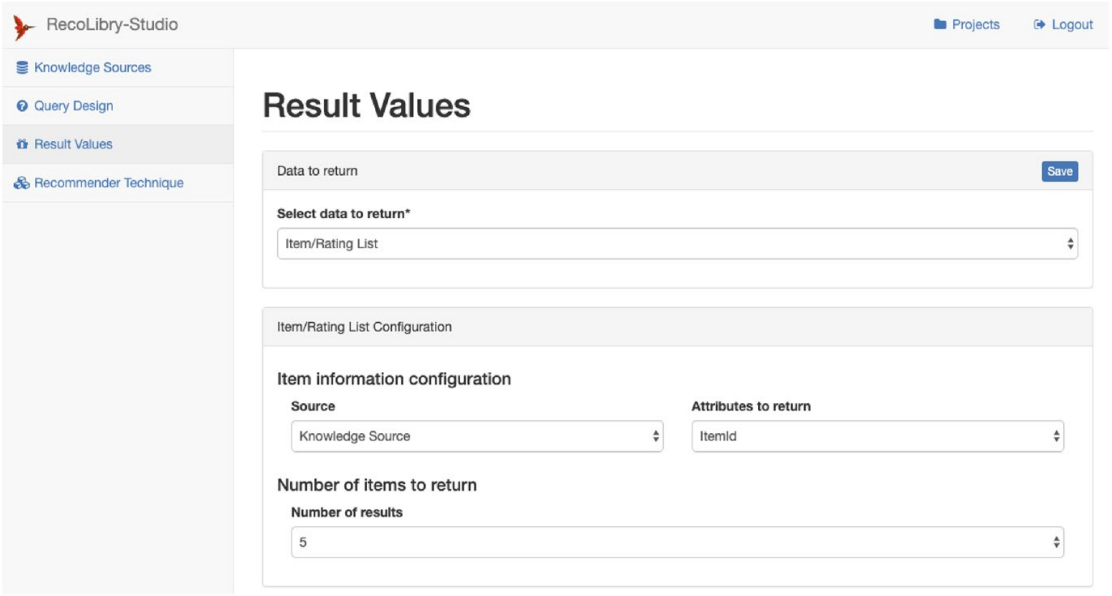

Fig. 7 Window to define the data contained in a query

4.2.3 Output data

Next, designers should define the information that they expect as the result of the recommender system. This configuration is done in their *Result Values* form. It shows a list where designers select how the solution is displayed by the system. For example, a recommender system can return a user rating prediction for an item. Depending on the result configuration selected, RecoLibry-studio shows a set of features to configure this type of solution. In our example, we want to receive a list with the top-5 recommended films and their recommendation value. Figure 7 shows how this step has been configured for our example.

When designers finish this step, RecoLibry-studio has enough information to assist the designer to select the most suitable recommendation algorithm according to the defined knowledge source and input/output descriptions. We will explain this step in the next section.

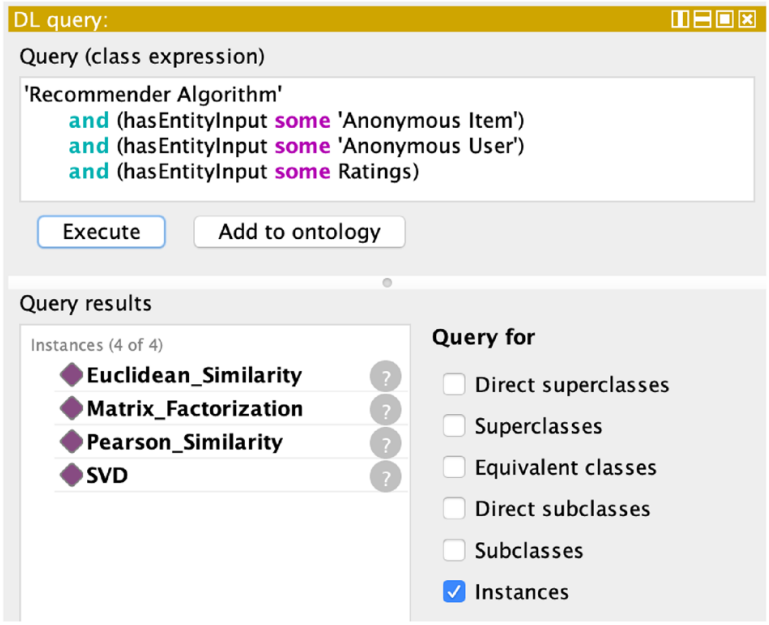

Fig. 8 Query to get the most suitable algorithms to use in the recommender system. Visualized using the Protégé ontology editor

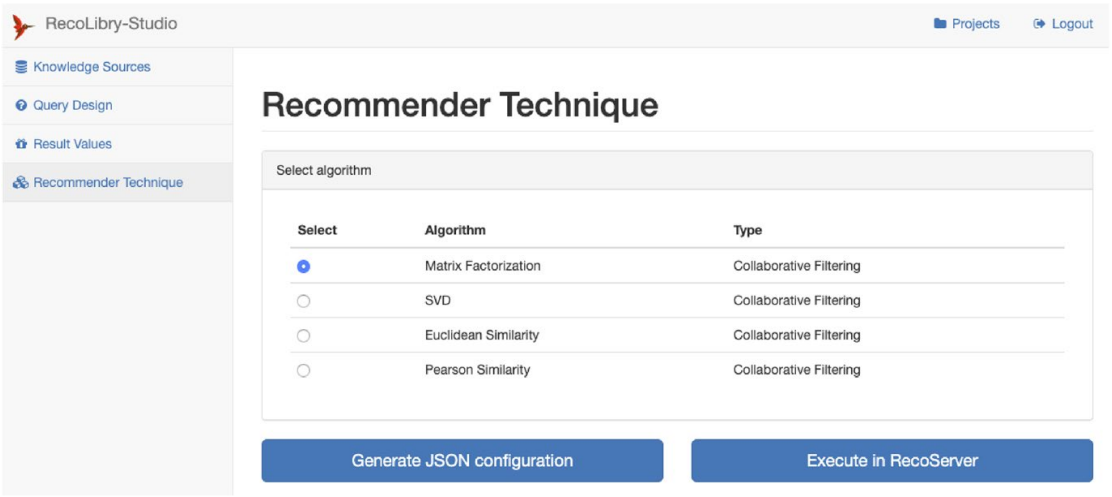

Fig. 9 Window to select the algorithm for the recommender system

4.2.4 Algorithm selection

This is the most important step in our design process. Using the restrictions defined by the designers in previous steps, RecoLibry-studio shows all the recommendation algorithms compatible with these restrictions. This way, designers select the algorithm to use in their systems.

In this step, RecoLibry-studio collects the information defined in the previous steps. Then, RecoLibry-studio sends these restrictions to RecOnto in order to filter all the algorithms compatible with these specifications. When RecoLibry-studio receives these algorithms, it shows them to the designer, together with their

features (name of the algorithm, algorithm type, implementation, etc.). This way, the designer can make an informed decision about the most suitable algorithm to use in the recommender system.

For our example, RecoLibry-studio prepares the information to send to RecOnto. Figure 8 shows the aspect of this query. In this case, the query asks for any algorithm that uses *Anonymous Items*, i.e. they are defined only by an identifier; *Anonymous Users*, i.e. they are defined only by an identifier; and *Ratings*. Next, RecOnto returns a set of individuals that match these properties. In this case is a set of collaborative filtering algorithms. RecoLibry-studio shows the result as shown in Fig. 9. According to the restrictions that we have included previously, these are the algorithms that best adapt to our recommender system. RecoLibry-studio recommends four algorithms: two of them based on kNN using different similarity functions (Euclidean or Pearson), a matrix factoring algorithm and the SVD algorithm. In our case we select the matrix factorization.

When designers finish this step, they have their system completely configured. Then, RecoLibry-studio is able to generate the configuration file to either configure a local recommender system using RecoLibry-core or to deploy that recommender into RecoServer as software as a service. The first method that configures the recommender system locally using RecoLibry-core and the configuration file is detailed in Sect. 4.1, whereas the second option where RecoLibry-studio sends the configuration to RecoServer for the deployment of the recommender is explained next.

4.3 RecoServer

The third tool within RecoLibry Suite is *R* eco *S* erver . The main goal of this tool is the automatic deployment to a web server of the recommender systems created with the tools explained above. For each recommendation system, a web application and a RESTful API will be deployed with all the web services required to let external applications use the recommender. This RESTful API acts as the back-end of the RS whereas the web application is a front-end by default.

To do so, RecoServer uses Docker technology (Merkel 2014) which facilitates to create, deploy, and run applications by using containers. A container is a light version of a virtual machine. Each container only contains the operating system services required by the encapsulated application. This allows the recommendation systems to be deployed in a simple way and to be lighter than a full web server.

All the recommender systems deployed in RecoServer have the same scheme. The Docker container has an Apache Tomcat server that runs the application and a MySQL database to manage the required dataset. Besides, each system has a an instance of the RecoLibry-core framework. As we explained in Sect. 4.1, this framework is in charge of the instantiation of the recommender system. Finally, the configuration file that we have seen in Sect. 4.2, which defines the structure of the recommender system to be deployed, is included in the contanier. Once the container is deployed, RecoServer creates a unique URL to access both the web application (front-end) and the web services (back-end).

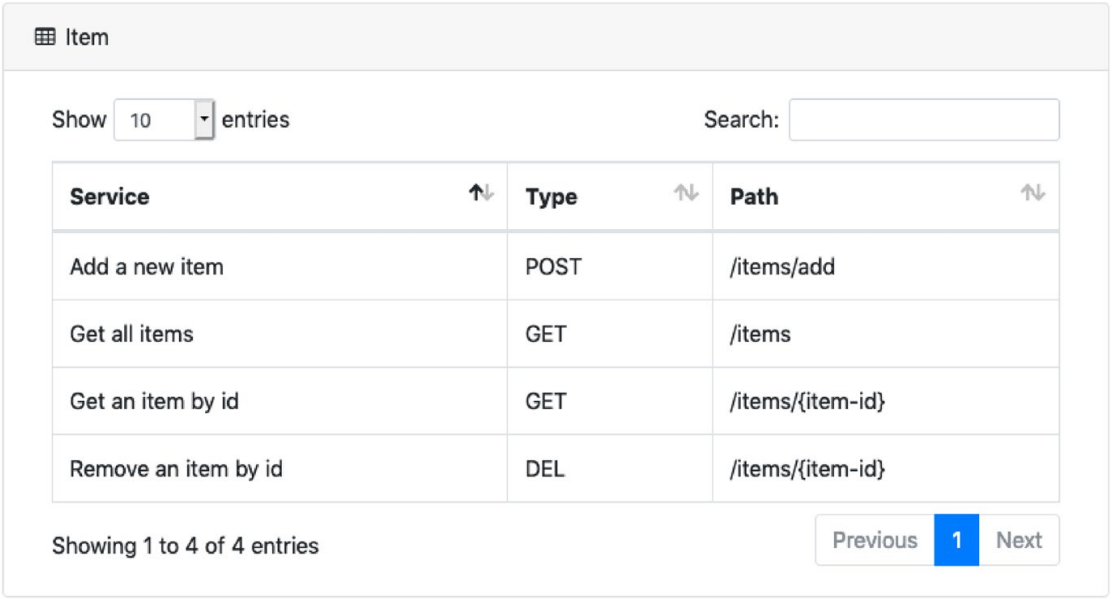

Fig. 10 Example of web services to manage the Recommender System's items

Table 2 Basic web services included in all system deployed in RecoServer

URL	Method	Description
/query	GET	This web service allows the developer to know all the attributes to complete in the system query
/recommender/init	PUT	This service executes the init function of the recommender system
/recommender	GET	Returns the result of the recommender system from a query object

This URL allows designers to access all the services associated with their recommender system. When accessing that URL, the designer will see a web page like the one shown in Fig. 10. This interface shows all the web services deployed and their associated parameters. The number, nature and signature of the web services depends on the recommender system created by the designer. For example, if the recommender system has some source of associated knowledge, it will have different services to manage that information (add or delete data, etc.). However, all recommender systems have some basic web services to obtain a recommendation, and to obtain query. Table 2 shows the common web services include in all recommender systems deployed in RecoServer.

In addition to previous web services, RecoServer also deploys a basic web application that acts as a front-end of those web services. This application allows to test the recommendation systems created with a visual interface through a web browser. Figure 11 shows an example of such a front-end created by RecoServer.

As we have presented, the applications contained in *R* eco *L* ibry *S* uite ease the generation of a recommendation system. These tools cover all the necessary steps in the development of this type of systems: design, implementation and deployment. Next, we describe how we have evaluated these tools.

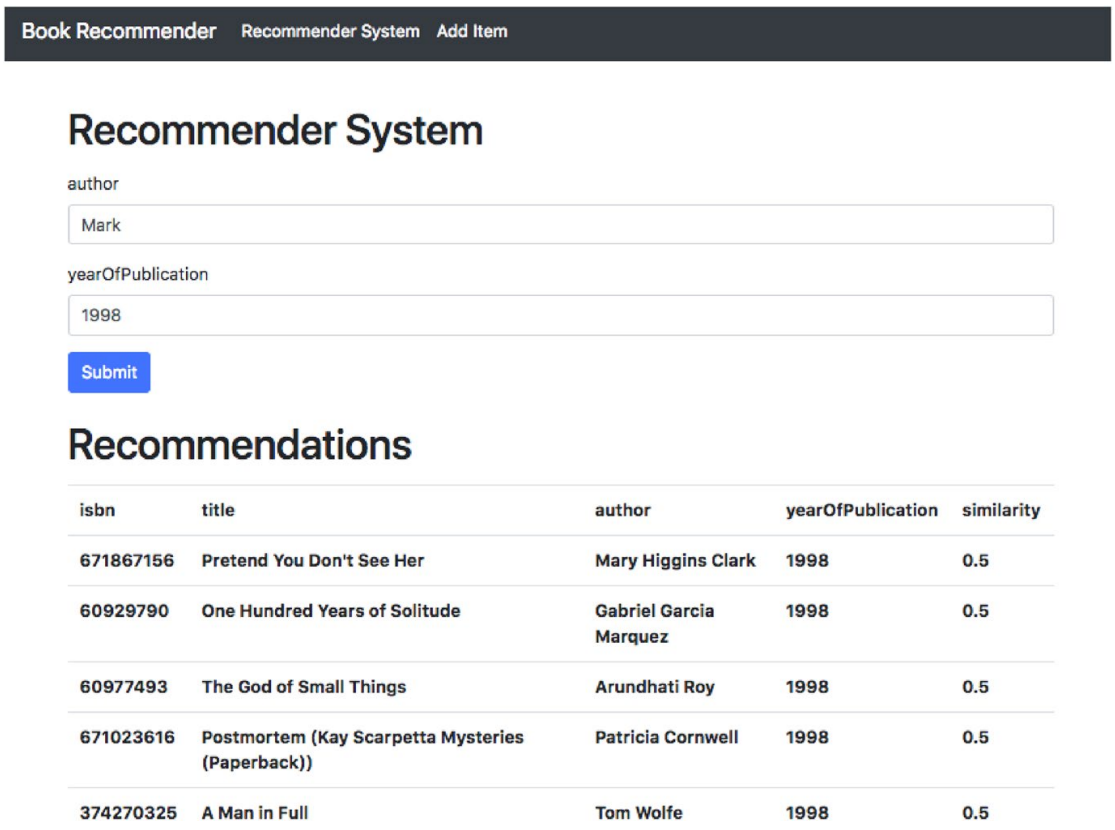

Fig. 11 A recommender system web application

5 Evaluation of the approach

The main objective of RecoLibry Suite tools is to facilitate the process of design and development of recommender systems for all types of users. For this reason, the RecoLibry Suite uses a component-based design process and a common vocabulary, described by the RecOnto ontology, to define and reuse these components.

In order to evaluate the proposed design process and the associated support tools, we have conducted an experimental evaluation that compare the development process using RecoLibry Suite to two widely used frameworks: Mahout and jcolibri. The choice of these frameworks is motivated by its specialisation in collaborative filtering and content-based recommendation, respectively. As each framework has its own focus, we tried to select a representative example of the two most common approaches, in order to compare them to RecoLibry-studio, which may be considered a wider system that supports the design of different types of recommender systems.

Our hypothesis before conducting the evaluation are the following:

1. RecoLibry-studio increases the efficiency to design and implement a recommender system.
2. RecoLibry-studio is understandable and easy to use.
3. RecoLibry-studio helps to understand how a recommender system works.

Therefore, we develop an evaluation with users where the participants would design a movie recommender system using the MovieLens dataset and evaluate several aspects of the development processes associated to the three aforementioned frameworks. The recommender system chosen for the comparison was a standard and well-known recommender: a user-based collaborative filtering system. Specifically, the recommendation system uses a K-Nearest Neighborhood (KNN) algorithm to estimate the most similar users to the target user in order to recommend her a new movie. In order to calculate the similarity between users, the metric to apply was the Euclidean distance. As every framework may be specialised in a concrete recommendation approach (content-based, collaborative filtering, etc.) we chose this concrete recommender system because it was common to the three frameworks being compared.

Thirty two users participated in the experiment. These users were users with a basic level of computer skills, i.e. users who know office applications and have worked with web applications or users who can develop basic software applications and have not worked with artificial intelligence algorithms.

1. During the introductory session, we explain the participants the theoretical foundations of recommender systems. Concretely, we detail how a recommendation system based on collaborative filtering works. We introduce the KNN algorithm and the Euclidean distance.
2. Next, we show users how to implement the recommender system using the Mahout framework. At this point, we explain which Mahout components are required to implement this functionality, and how they are composed to obtain the desired recommender system.
3. Next, the implementation using jCOLIBRI is presented. We explain how this framework, oriented to content-based recommendations, can be used to implement recommender systems. Again, we explain the components required and how they are composed in order to obtain the target recommender system.
4. Finally, we show users how to make the recommender system using the RecoLibry-studio application. At this point, we explain each of the steps they have to follow to configure the recommender system. At the end of the process, we also show the participants how the deployment of the recommender system in RecoServer works.

At the end of steps 2, 3, and 4, users have to complete a questionnaire that analysed the difficulty of designing and implementing the recommendation system using the corresponding framework. Concretely, we asked the users for evaluating (using a 1–5 Likert scale) the following metrics related with the framework employed:

- Q1: Estimated time to *design* a recommender system.
- Q2: Estimated time to *implement* a recommender system.
- Q3: Difficulty to understand how the framework works.
- Q4: Difficulty to *develop by themselves* a new recommender system using the framework.

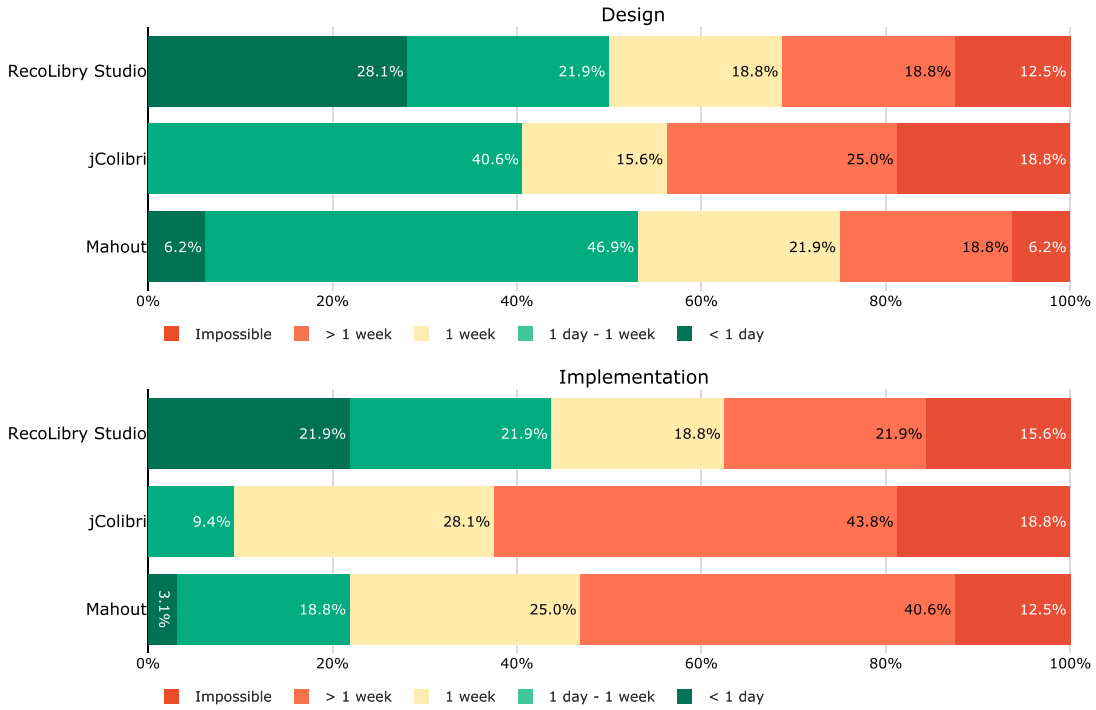

Fig. 12 Estimated time to design (top) and implement (bottom) a recommender system using the selected frameworks, corresponding to metrics Q1 and Q2

- Q5: Difficulty to understand the components and how a recommender system works using the framework.

In the following section, we analyse the results obtained by this experimental evaluation.

5.1 Results

Our first hypothesis supposes that RecoLibry-studio increases the efficiency to design and implement a recommender system. The results in Fig. 12 shows the distribution of the answers for Q1 and Q2, which support our expectations. These results clearly show that RecoLibry-studio is considered the fastest design tool as 28.1% of users believe that it would take less than a day to design a recommender system with it. In Mahout's case this percentage is only 6.25%.

Regarding the implementation (Fig. 12, bottom chart) corresponding to metric Q2, RecoLibry-studio is again the framework that users consider would take less time. The users who consider that they will take less than a day using RecoLibry-studio are 21.9%, while in Mahout they are 3.1%. If we also add the users who estimate that it would take less than a week to implement the example, RecoLibry-studio has the highest percentage (43.8%), followed by Mahout (21.9%) and jcolibri (9.4%). These results reveal that the user perception is that RecoLibry-studio improves the efficiency to design and develop these systems.

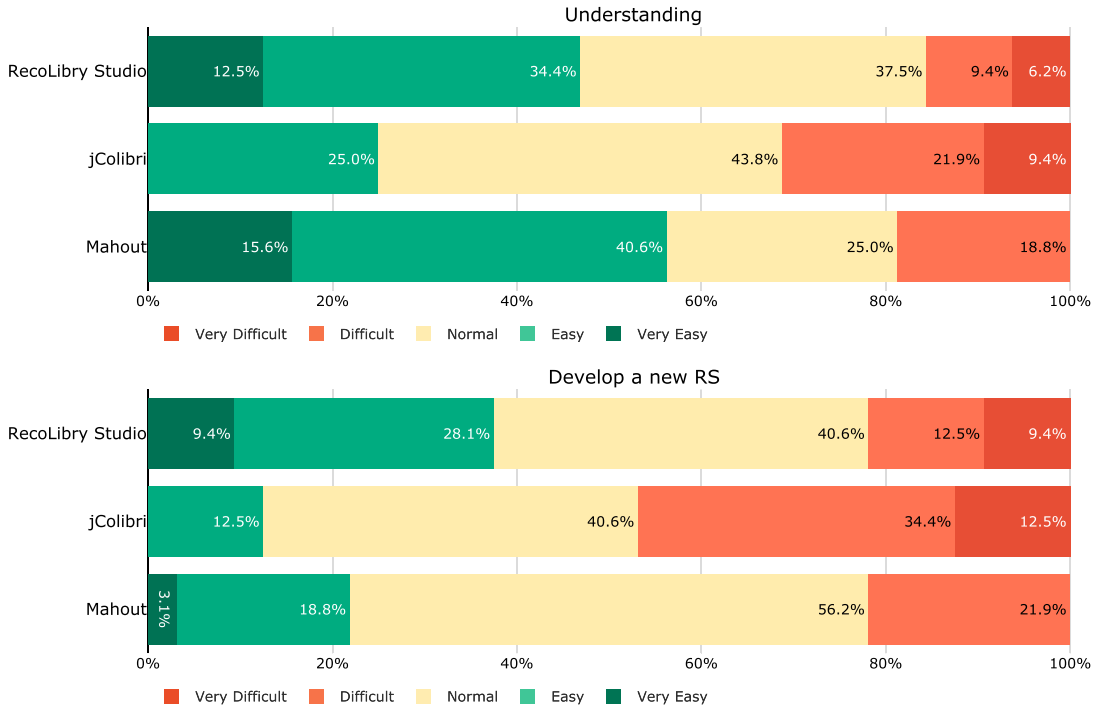

Fig. 13 Difficulty to understand (top) and to create (bottom) a new recommender system using the selected frameworks. Corresponding to metrics Q3 and Q4

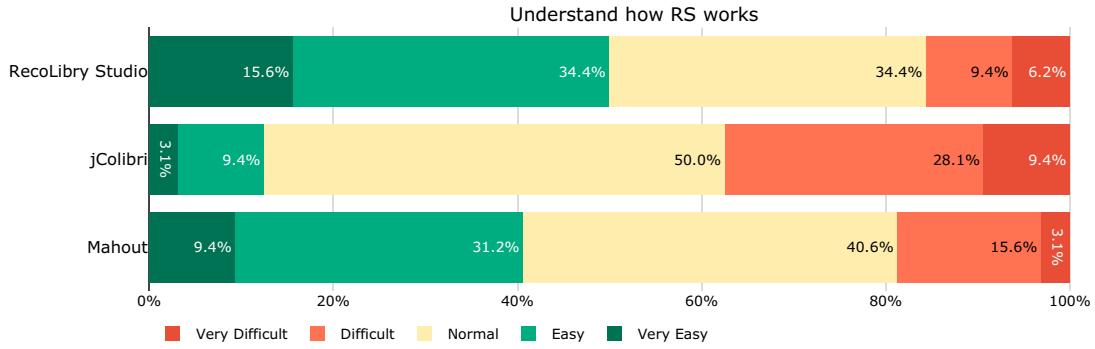

Fig. 14 Difficulty to understand how a recommender system works using the selected frameworks

Another of the hypothesis that we validate with the proposed experiment is that RecoLibry-studio is easy to understand and to use in comparison with the other two approaches. First, we evaluated how users perceived the global difficulty to understand each approach. The top chart in Fig. 13 (corresponding to metric Q3) shows that Mahout seems to be the easiest to understand, since total number of users who consider it easy or very easy is 56.6%. The second place is for RecoLibry-studio, with 46.9% of users, and, in third place, jcolibri with 25%. This result, where Mahout beats RecoLibry-studio, may be caused because we are comparing 2 libraries (Mahout and jcolibri) with a tool,RecoLibry-studio, that encapsulates, and somehow hides, the behaviour of the library. This result let us conclude that users do not understand how is the implementation process using RecoLibry-studio because it process is not visible to them.

Additionally, we asked the users about the difficulty of developing another recommender system using each one of the frameworks. The results of metric Q4 are shown in Fig. 13 (bottom chart). In this case, RecoLibry-studio has the highest percentage of users who consider it easy or very easy with 37.5%, followed by Mahout with 21.9% and, thirdly, jcolibriwith 12.5%. This result let us confirm that users perceive that the development of recommender systems with RecoLibry-studio is easier than using the other approaches.

Finally, we asked users how difficult it is to understand the structure and the components of a recommender system after designing it using each framework. According to Fig. 14, 15.6% of users consider that understanding a recommender system using RecoLibry-studio is very easy. It is the highest percentage followed by Mahout (9.4%) and jcolibri(3.1%). RecoLibry-studio is also the one with the highest percentage if we add the users who consider it easy or very easy to understand a recommendation system with 50% of users, followed by Mahout with 40.6% and jcolibriwith 12.5%. These results validate our last hypothesis, which supposes that our library helps to understand how a recommender system works.

5.2 Evaluation analysis

The main objective proposed when we designed RecoLibry Suite was to make a set of intelligent tools to facilitate the building of recommender systems. They are semiautomatic component composition tools based on semantic rules defined by an ontology. These tools should ease the building of recommender systems and help users to understand the underlying process.

There are several alternatives to evaluate our tools and compare them with other options. The first approach consists of collecting time-based metrics. For example, the time required to develop a recommender system with each framework. However, the use of such metrics to compare frameworks could not be the right choice because sometimes a faster solution could be an inefficient solution in tasks like programming (Frøkjær et al. 2000). Moreover, this approach has another major drawback. It shall require a set of subjects with technical skills, mainly programming. Therefore, the variability on the programming skills will have an impact on the development times unless we select users with the same development experience.

Besides, the evaluation through the complete development of the same recommender system using different tools may introduce another additional bias as users will not be able to complete all the developments in just one session. If we force users to perform several implementations, it will require several seasons where users may finish annoyed or fatigued, paying more attention to the initial implementations. Furthermore, the comparison of the tools after several sessions is less objective than a comparison performed after using all the alternatives in a one-day session.

In order to alleviate these bias produced by the evaluation based on the complete development of a recommender system using several tools, we designed the experimental set-up based on the explanation of the alternative development processes to the users and the estimation of several quality metrics through a questionnaire. Indeed, this evaluation also has several threats that we should resolve in a future

experiment. First of all, the order could influence users' evaluations. RecoLibry-Studio was the last framework presented, and users could rate it better because they have a deeper understanding of the construction of recommender systems after studying the other frameworks. Secondly, users are not developing a recommender system actually. It means that their estimations about the design and development process are less accurate. However, as we described before, the "full-development" alternative also has significant drawbacks. After considering both evaluation approaches, we found that our evaluation methodology was preferable as it enabled us to collect a higher number of subjects with heterogeneous skills, and perform the evaluation in a one-day session to prioritise the objectivity in the comparison of frameworks.

6 Conclusions and future work

Recommender systems have become a very useful tool to give personalized information and items for individual and groups. However, the process of developing these systems is not trivial. The large number of algorithms and possible configurations require of expert developers.

In this paper we present RecoLibry Suite, a set of tools to design and develop recommender systems using reusable components. First, we present an analysis of 22 academic and open source frameworks comparing their characteristics with respect to the properties of RecoLibry Suite. Next, we describe a vocabulary to define the components and how they relate to each other using RecOnto. In addition, we present and describe the 3 tools contained in our suite: RecoLibry-core, RecoLibry-studio and Reco Server.

Thanks to the component-based design pattern, RecoLibry Suite tools comply with many of the objectives we defined in the design. According to the completeness, RecoLibry-core allows users to implement the components using third party frameworks, so their recommendation algorithms can be included in RecoLibry-core. In addition, our suite is extensible. Users can create new components and include them in the architecture of our tools taking advantage of other components already implemented. In addition, the use of components makes our framework flexible to easily configure recommender systems.

Finally, we conducted an experiment with real users to compare the usability of RecoLibry-studio with other frameworks. The results highlight that RecoLibry-studio facilitates the process of design and implementation of this type of systems. Additionally, most of users consider it is easier to develop a recommender system with RecoLibry-studio. The use of a graphical interface, in addition to the use of components that encapsulate services, eases the understanding of how a recommendation system works and what services they need. Finally, RecoLibry-studio in general is easy to understand and the users think that it is not difficult to create new recommender systems using it. This is because RecoLibry-studio, as a web application, partially hides the process of implementation of the recommender systems and non-developer users do not have to deal with how the internal RecoLibry-core library is building the final system. However, we have detected some problems about this evaluation. The order to explain each framework and the user perception about

each framework can affect the results. For this reason, we will implement another evaluation where user have to implement a recommender system with one of the frameworks presented. Then, we will be able to compare different metrics, for example, the time spend using each framework.

RecoLibry Suite provides a unifying framework that eases the development process by reusing software components from other well known libraries. As the future work we will improve extendibility throught the RecOnto ontology. In addition, we will add more features in the data to select the best recommender algorithm. For example, select algorithms that work better with data problems, like sparsity or noise. To do that, we will add a visual tool to show the data and the developer can add tags to describe problems in his/her data. This tags can be added to RecOnto and use it to filter algorithms.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: a system for large-scale machine learning. OSDI **16**, 265–283 (2016)

Adomavicius, G., Tuzhilin, A.: Context-Aware Recommender Systems, pp. 191–226. Springer, Boston (2015)

Apache: Apache spark: lightning-fast cluster computing. http://spark.apache.org (2016)

Ayala-Gómez, F., Keniş, B., Karagöz, P., Benczúr, A.: Top-k context-aware tour recommendations for groups. In: MICAI, pp. 176–193. Springer (2018)

Bello-Tomás, J.J., González-Calero, P.A., Díz-Agudo, B.: JColibri: an object-oriented framework for building CBR systems. In: ECCBR 2004 (2004)

Chan, S., Stone, T., Szeto, K.P., Chan, K.H.: PredictionIO. In: CIKM 2013, pp. 2493–2496. ACM Press, New York (2013)

Chen, G., Kotz, D., et al.: A survey of context-aware mobile computing research. Tech. rep., Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College (2000)

Ekstrand, M.D., Ludwig, M., Konstan, J.A., Riedl, J.T.: Rethinking the recommender research ecosystem. In: RecSys 2011, p. 133. ACM Press, New York (2011)

Frakes, W.B., Nejmeh, B.A.: Software reuse through information retrieval. SIGIR Forum **21**(1–2), 30–36 (1987)

Frøkjær, E., Hertzum, M., Hornbæk, K.: Measuring usability: are effectiveness, efficiency, and satisfaction really correlated? In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 345–352. ACM (2000)

Gantner, Z., Rendle, S., Freudenthaler, C., Schmidt-Thieme, L.: MyMediaLite. In: RecSys 2011, p. 305 (2011)

Gómez-Albarrán, M., González-Calero, P.A., Díaz-Agudo, B.: Software design as framework reuse: a knowledge-based approach. In: ECAI, pp. 98–99 (1998)

Guo, G., Zhang, J., Sun, Z., Yorke-Smith, N.: Librec: a java library for recommender systems. In: UMAP Workshops, vol. 4 (2015)

Harper, F.M., Konstan, J.A.: The movielens datasets: history and context. ACM Trans. Interact. Intell. Syst. **5**(4), 19:1–19:19 (2015)

Heineman, G.T., Councill, W.T.: Component-based software engineering. Putting the pieces together, p. 5. Addison-Westley, Boston (2001)

Jorro-Aragoneses, J.L., Ceron-Rios, G.M., Díaz-Agudo, B., Recio-García, J.A., López-Gutierrez, D.M.: RecOnto: an ontology to model recommender systems and its components. In: ICTAI 2017 (2017a)

Jorro-Aragoneses, J.L., Díaz-Agudo, B., Recio-García, J.A.: Madrid live: a context-aware recommender system of leisure plans. In: ICTAI 2017, pp. 796–801. IEEE (2017b)

Jorro-Aragoneses, J.L., Recio-García, J.A., Díaz-Agudo, B., Jimenez-Díaz, G.: Recolibry-core: a component-based framework for building recommender systems. Knowl. Based Syst. **182**, 104854 (2019)

Kirk, J.: Tensorrec. https://github.com/jfkirk/tensorrec (2018)

Kolodner, J.: Case-Based Reasoning. Morgan Kaufmann, Burlington (2014)
Koren, Y., Bell, R.: Advances in collaborative filtering. In: Recommender Systems Handbook, 2nd edn, pp. 77–118. Springer, Boston (2015). https://doi.org/10.1007/978-1-4899-7637-6_3
Kula, M.: Metadata embeddings for user and item cold-start recommendations. In: Bogers, T., Koolen, M., (eds.) CBRecSys 2015, vol. 1448, pp. 14–21 (2015)
Kula, M.: Spotlight. https://github.com/maciejkula/spotlight (2017)
Landset, S., Khoshgoftaar, T.M., Richter, A.N., Hasanin, T.: A survey of open source tools for machine learning with big data in the Hadoop ecosystem. J. Big Data **2**(1), 24 (2015)
Lops, P., De Gemmis, M., Semeraro, G.: Content-based recommender systems: state of the art and trends. In: Recommender Systems Handbook, pp. 73–105. Springer (2011). https://doi.org/10.1007/978-0-387-85820-3_3
McIlroy, M.D.: Mass produced software components. In: Software Engineering: Report on a Conference Sponsored by the NATO Science Committee (1969)
Merkel, D.: Docker: lightweight linux containers for consistent development and deployment. Linux J. **2014**(239), 2 (2014)
Newell, A., et al.: The knowledge level. Artif. Intell. **18**(1), 87–127 (1982)
Paraschakis, D., Nilsson, B.J., Holländer, J.: Comparative evaluation of top-n recommenders in e-commerce: an industrial perspective. ICMLA **2015**, 1024–1031 (2015)
Paszke, A., Gross, S., Chintala, S., Chanan, G.: Pytorch (2017)
Recio-García, J.A., Díaz-Agudo, B., González-Calero, P.A.: Prototyping recommender systems in jColibri. In: RecSys 2008, p. 243. ACM Press, New York (2008)
Recio-García, J.A., González-Calero, P.A., Díaz-Agudo, B.: jColibri2: a framework for building Case-based reasoning systems. Sci. Comput. Program. **79**, 126–145 (2014)
Recio-García, J.A., González-Calero, P.A., Díaz-Agudo, B.: Template-based design in COLIBRI studio. Inf. Syst. **40**, 168–178 (2014)
Ricci, F., Rokach, L., Shapira, B.: Recommender systems: introduction and challenges. In: Recommender Systems Handbook, pp. 1–34. Springer (2015)
Sarwat, M., Moraffah, R., Mokbel, M.F., Avery, J.L.: Database system support for personalized recommendation applications. ICDE **2017**, 1320–1331 (2017)
Schelter, S., Owen, S.: Collaborative filtering with apache mahout. In: Proceedings of ACM RecSys Challenge (2012)
Sean, O.: Oryx 2. https://github.com/OryxProject/oryx (2018)
Su, X., Khoshgoftaar, T.M.: A survey of collaborative filtering techniques. Adv. Artif. Intell. **2009**, 421425 (2009). https://doi.org/10.1155/2009/421425
Szyperski, C.: Component software and the way ahead. In: Foundations of Component-Based Systems, pp. 1–20 (2000)
Szyperski, C., Bosch, J., Weck, W.: Component-oriented programming. In: ECOOP 2019, pp. 184–192. Springer (1999)
Van Setten, M., Pokraev, S., Koolwaaij, J.: Context-aware recommendations in the mobile tourist application compass. In: AH 2004. Springer (2004)
Vnd, K.: Tensorrec. https://github.com/kasramvd/Rexy (2017)