

Servicio de recomendación de documentos basado en ontología

Luis Felipe Borjas Reyes
10611066

9 de junio de 2010



Universidad Tecnológica Centroamericana
(UNITEC)

Facultad de ingeniería en sistemas computacionales
En opción al título de Ingeniero en Sistemas Computacionales

Resumen

La *world wide web* ha evolucionado de tal manera que las personas han pasado de ser consumidoras de información a productoras: desde los *blogs*, las revistas en línea y las redes sociales hasta las aplicaciones que permiten editar documentos académicos en internet, las personas están dejando paulatinamente atrás los antiguos medios de consumo y producción de datos. Pero este incremento ha traído consigo un efecto secundario: la *sobrecarga de información*; esto no afecta tanto a quien escribe un blog personal o se expresa en una red social, porque nunca, o casi nunca, tiene que desarrollar o justificar su argumento. Mas aquellos cuya opinión está sujeta al escrutinio de otros deben dedicar parte del tiempo que utilizan en la producción de información a tratar de encontrar fuentes, ideas o fundamentos, muchas veces sin éxito, al no poder expresar la necesidad de información; sin embargo, esta práctica está en contra del proceso natural -e ideal- de escribir, pues las ideas fluyen y evolucionan, y pocas veces termina uno escribiendo exactamente lo que se propuso en un principio. Para ello, en el pasado se han propuesto sistemas que asistan a quien escribe, supervisando en tiempo real el proceso y recomendándole referencias en tanto. Pero la mayor parte de estas soluciones o consisten en un sistema exclusivo -*otra* aplicación que el usuario debe instalar y aprender a usar; no tratan de personalizar las recomendaciones, perdiendo en utilidad, o no son lo suficientemente flexibles como para que los desarrolladores de software las adopten en sus propios productos.

En el espíritu de la nueva generación de servicios web y aplicaciones híbridas, se propone aquí un enfoque distinto: un *servicio* de recomendación que, a través de una *interfaz de programación de aplicaciones*, pueda ser implementado en cualquier producto de software que desee mejorar la experiencia de sus usuarios al crear información; asimismo, con el fin de personalizar las referencias a recomendar, el servicio habrá de aprender *automáticamente* las preferencias de los usuarios mediante técnicas estadísticas y conocimiento representado en una ontología.

Agradecimientos

A mis padres y hermanos, por su apoyo durante estos últimos años y por soportar a alguien rondando por la casa a altas horas de la noche.

A mis maestros en la universidad por ayudar, con el conocimiento, a establecer los fundamentos de mi porvenir.

A mis asesoras, Elena Calidonio y Yanira Rivera, por su consejo y guía.

A Jorge García y Fernando Escher, porque durante el desarrollo de escolarea he adquirido experiencia y conocimiento útiles para marcar el derrotero de mi vida profesional.

A mis amigos en la facultad, Gerardo Romero y Daniel Martínez, por su apoyo durante el desarrollo de este proyecto.

A mis conocidos, colegas y amistades, por escuchar y asentir con cortesía cuando balbuceaba mis ideas.

Índice general

| | |
|---|-----------|
| Agradecimientos | 3 |
| Introducción | 8 |
| 1. Generalidades | 10 |
| 1.1. Propósito | 10 |
| 1.2. Objetivos | 10 |
| 1.3. Contexto | 10 |
| 1.4. Estado del arte | 11 |
| 1.5. Identificación del problema | 13 |
| 1.6. Descripción de la solución | 13 |
| 1.7. Definición de la metodología usada para la solución | 16 |
| 2. Marco teórico | 17 |
| 2.1. Recuperación de información. | 17 |
| 2.1.1. Definiciones y conceptos preliminares | 17 |
| 2.1.2. Componentes de los sistemas de recuperación de información | 19 |
| 2.1.2.1. La estrategia de búsqueda | 20 |
| 2.1.2.2. Presentación de los resultados | 22 |
| 2.1.2.3. Obtención de la colección documental | 23 |
| 2.1.3. Modelos de recuperación de información | 24 |
| 2.1.3.1. Modelo Booleano | 26 |
| 2.1.3.2. Modelo booleano extendido mediante lógica difusa | 27 |
| 2.1.3.3. Modelo de espacio vectorial | 27 |
| 2.1.3.4. Mejoras del modelo de espacio vectorial | 30 |
| 2.1.3.5. Indización semántica latente | 31 |
| 2.1.3.6. Análisis formal de conceptos | 31 |
| 2.1.3.7. Similitud basada en hash | 32 |
| 2.1.3.8. Modelos probabilísticos | 33 |
| 2.1.3.9. Modelos lingüísticos y orientados al conocimiento | 34 |
| 2.1.3.10. Comparación de los modelos presentados | 35 |
| 2.1.4. Evaluación de un sistema de recuperación de información. | 35 |
| 2.1.4.1. Medidas de evaluación | 35 |
| 2.1.4.2. El proceso de evaluación | 39 |
| 2.2. Sistemas de Recomendación | 40 |
| 2.2.1. Recomendación con filtrado basado en el contenido | 41 |
| 2.2.1.1. Componentes | 42 |

| | | |
|-----------|--|-----------|
| 2.2.1.2. | Limitaciones | 43 |
| 2.2.2. | Recomendación con filtrado colaborativo. | 44 |
| 2.2.2.1. | Métodos de filtrado colaborativo | 45 |
| 2.2.2.2. | Limitaciones | 47 |
| 2.2.3. | Recomendación orientada al conocimiento | 48 |
| 2.2.4. | Sistemas de recomendación híbridos | 49 |
| 2.2.4.1. | Sistemas de recomendación colaborativos y orientados al contenido | 49 |
| 2.2.4.2. | Sistemas de recomendación colaborativos y orientados al conocimiento | 50 |
| 2.2.5. | Evaluación de un sistema de recomendación | 50 |
| 2.2.5.1. | Medidas de evaluación <i>fuera de línea</i> | 51 |
| 2.2.5.2. | Medidas de evaluación <i>en línea</i> | 53 |
| 2.3. | Sistemas de recuperación oportuna de la información | 55 |
| 2.3.1. | Determinación del contexto para sistemas basados en texto | 56 |
| 2.4. | Construcción de perfiles de usuario | 57 |
| 2.4.1. | Definiciones preliminares | 58 |
| 2.4.2. | La definición del perfil | 59 |
| 2.4.3. | La utilización del perfil | 60 |
| 2.5. | Servicios e interfaces de programación de aplicaciones web (API) | 60 |
| 2.5.1. | La técnica de arquitectura REST | 61 |
| 2.5.2. | JSON y AJAX | 62 |
| 2.5.3. | Estado del arte de los servicios web | 63 |
| 2.6. | Diseño de interfaces | 64 |
| 2.6.1. | Aspectos psicológicos del diseño de interfaces | 64 |
| 2.6.1.1. | La ley de Fitt | 64 |
| 2.6.1.2. | El principio del menor esfuerzo | 65 |
| 2.6.2. | Interfaces para sistemas de asistencia al usuario | 65 |
| 3. | Planificación | 67 |
| 3.1. | Definición del ámbito e identificación de los recursos | 67 |
| 3.2. | Calendarización | 69 |
| 3.3. | Estimación | 70 |
| 4. | Análisis y Diseño. | 71 |
| 4.1. | Análisis | 71 |
| 4.1.1. | Sumario de requisitos | 72 |
| 4.1.1.1. | Requisitos funcionales | 72 |
| 4.1.1.2. | Requisitos no funcionales | 73 |
| 4.1.2. | Casos de uso | 73 |
| 4.1.3. | Análisis de contenido | 75 |
| 4.1.3.1. | Objetos del contenido | 75 |
| 4.1.3.2. | Clases del análisis | 75 |
| 4.1.4. | Análisis de interacción y funcional | 75 |

| | | |
|-----------|--|------------|
| 4.1.5. | Análisis de configuración | 76 |
| 4.2. | Diseño | 79 |
| 4.2.1. | Diseño de la interfaz, estético y de contenido | 81 |
| 4.2.2. | Diseño de arquitectura | 82 |
| 4.2.2.1. | Arquitectura en capas | 84 |
| 4.2.2.2. | Despliegue | 84 |
| 5. | Implementación | 87 |
| 5.1. | Obtención de la colección documental y creación de la base de conocimien- to ontológica | 87 |
| 5.1.1. | Elección de una ontología | 88 |
| 5.1.2. | Obtención de la ontología | 89 |
| 5.1.3. | Almacenamiento de la ontología | 90 |
| 5.1.4. | Obtención de la colección documental | 91 |
| 5.2. | Implementación del componente de búsqueda | 91 |
| 5.2.1. | Elección de sustitutos documentales | 91 |
| 5.2.2. | Elección de una implementación de recuperación de información | 92 |
| 5.3. | Implementación del componente de perfilado | 93 |
| 5.3.1. | Entrenamiento del clasificador de categorías | 93 |
| 5.3.2. | Ponderación de las categorías | 94 |
| 5.3.3. | Evolución del perfil | 95 |
| 5.4. | Implementación del servicio | 97 |
| 5.4.1. | Interfaz de programación de aplicaciones | 98 |
| 5.4.2. | Implementación de las interfaces opcionales | 100 |
| 5.4.3. | Detección de cambios en el contexto local | 100 |
| 5.4.4. | Detección de términos clave | 100 |
| 5.4.5. | Re-ordenamiento de los resultados | 102 |
| 6. | Conclusión | 103 |
| 6.1. | Visión General | 103 |
| 6.2. | Observaciones y evaluación | 104 |
| 6.3. | Trabajo futuro | 107 |
| 6.3.1. | Desarrollo | 107 |
| 6.3.2. | Análisis de implementaciones | 108 |
| 6.3.3. | Comercialización | 109 |
| 6.4. | Recomendaciones | 109 |
| | Bibliografía | 111 |
| A. | Correos electrónicos de comunicación con la facultad | 111 |
| A.1. | Requisición de recursos de laboratorio | 111 |
| A.1.1. | Copia del correo electrónico de solicitud | 111 |
| A.1.2. | Copia de la respuesta de la facultad | 112 |
| A.2. | Solicitud de acceso al servidor virtual desde fuera del laboratorio | 112 |

| | |
|--|------------|
| A.2.1. Copia del correo electrónico de solicitud | 112 |
| A.2.2. Respuesta de la facultad | 113 |
| B. Plan de proyecto | 114 |
| C. Manual del implementador | 119 |

Introducción

En este proyecto se explora el concepto de un servicio de recomendación: proveer una interfaz para que aplicaciones orientadas a la producción de información (sistemas de gestión de contenidos y procesadores de texto, por ejemplo) y desarrolladas para cualquier plataforma puedan ofrecer a sus usuarios una mejora a su experiencia mediante la recomendación oportuna de referencias.

La motivación inicial para el desarrollo de este proyecto fue poder experimentar con tecnologías de procesamiento del lenguaje natural, con la idea de desarrollar un sistema que adquiriese el conocimiento suficiente como para recomendar a docentes referencias y lecturas adicionales en sus actividades de documentación y planificación. Sin embargo, pronto se encontró que era imposible ignorar la tendencia actual a desarrollar aplicaciones híbridas (mejor conocidas, en el mundo angloparlante, como *mashups*) que, en lugar de recurrir a distintas aplicaciones para resolver problemas relacionados o crear soluciones que sacrifican la calidad y la experiencia del usuario por suplir todas las posibles necesidades del usuario, se está optando por implementar en los sistemas, mediante interfaces web de programación de aplicaciones o librerías re-utilizables, alternativas *ya existentes* que se dedican a resolver con eficiencia problemas comunes. De esta manera, la orientación cambió del desarrollo de un sistema cerrado a la implementación de un *servicio* al cual los implementadores -tanto de sistemas híbridos como aplicaciones tradicionales- pudieran acceder mediante el protocolo HTTP, siendo éste ideal para no exigir de quien lo use nada más que conexión a internet, sin restricciones de sistemas operativos o lenguajes de programación.

Se observó que una gran parte de las personas están utilizando computadoras para crear datos textuales y tienen acceso a internet; además, -dado que existen incontables fuentes alternativas de información- en la actualidad se requiere que, si esta información está dirigida a un auditorio considerable, se necesite conocer otros documentos que justifiquen, expandan, contextualicen o eliciten lo que se está escribiendo. Para este propósito, los escritores promedio contemporáneos (autores de blogs, estudiantes, periodistas informales) han dejado en segundo plano -u olvidado del todo- la consulta a bibliotecas y personas con autoridad en sus respectivos campos y han adoptado como costumbre el utilizar motores de búsqueda para obtener estos documentos. Por lo general, esta búsqueda es previa a la escritura, de modo que las nuevas necesidades de información que surgen *durante* el proceso de escritura involucran o que el usuario conjeture o que interrumpa su tarea para tratar de encontrar nuevas referencias (muchas veces sin éxito o perdiendo tiempo valioso, pues la necesidad de información apenas se viene formando y seguramente cambiará con la evolución del texto); ambas alternativas tienen como consecuencia que se cree trabajo de baja calidad o se invierta más tiempo del necesario en producirlo. Además, el ser humano trata al trabajo con la *ley del mínimo esfuerzo* de modo que

siempre está dispuesto a elegir alternativas que reduzcan el tiempo total que él estima tomará la realización de una tarea, de modo que esto también pone en riesgo la calidad del trabajo si la investigación por referencias tomará un tiempo considerable y las meras conjeturas son más expeditas.

Ya se han propuesto sistemas de asistencia a la escritura con base en estas dos realidades. El proyecto que se describe aquí supone un avance al proveer una base común para la implementación de esta funcionalidad en software nuevo y ya existente, proveyendo un componente útil para desarrolladores que deseen mejorar la productividad de los usuarios de sus productos a la hora de producir información.

Para representar el conocimiento que el sistema utilizará para hacer recomendaciones se eligió un enfoque ontológico¹: en lugar de aprender lentamente los perfiles, el sistema subyacente al servicio ya *sabr*á distinguir los conceptos de los documentos, de modo que pueda identificar las preferencias de los usuarios más rápido en tiempo de uso (una sesión, frente a varias sesiones en enfoques tradicionales, que requieren más información para identificar lo que tienen en común las sesiones entre ellas y con otras). En este proyecto, la ontología utilizada es la del *Open Directory Project*².

El resto de este documento se estructura como sigue:

Capítulo 1: Comprende las generalidades del proyecto: se plantean el propósito y los objetivos, se establece el contexto, se exploran proyectos relacionados, se detallan el problema y la solución -incluyendo un diagrama que describe a grandes rasgos el funcionamiento del servicio- y se establece la metodología utilizada para la implementación.

Capítulo 2: En éste se exploran los conceptos relacionados al proyecto, así como los fundamentos teóricos relacionados con la implementación de un sistema de esta naturaleza.

Capítulo 3: Aquí se presentan los aspectos de gestión: la metodología de planificación, la estimación y calendarización.

Capítulo 4: Este capítulo consiste en la memoria descriptiva de la implementación de la solución, aquí se detallan las decisiones de ingeniería tomadas a lo largo de la etapa de desarrollo.

Capítulo 5: Comprende las conclusiones y recomendaciones, así como el trabajo futuro, los estándares de uso esperados de los implementadores y las posibles alternativas de comercialización del producto desarrollado.

¹Es decir, se representará mediante ontologías: grafos donde los nodos son conceptos y las aristas, relaciones entre ellos

²<http://www.dmoz.org/>

1 Generalidades

Este capítulo ofrece una visión a grandes rasgos del proyecto, estableciendo un contexto para el mismo y sentando las bases teóricas y técnicas de la comprensión del desarrollo del mismo.

1.1. Propósito

Desarrollar un servicio web que recomiende al usuario documentos relevantes a un contexto local, entendiéndose por éste cualquier tarea que involucre el ingreso de texto libre en la interfaz del sistema que utilice el servicio. La recomendación consistirá en la búsqueda de documentos relacionados conceptualmente a la tarea del usuario en un índice de documentos mantenido por el sistema y la consiguiente recomendación de los que se consideren útiles, evaluando la utilidad a partir de un perfil construido y actualizado automáticamente en base al uso histórico del servicio, basando el conocimiento de las preferencias de usuarios en una ontología jerárquica.

1.2. Objetivos

1. Elegir la naturaleza de la colección documental base para la implementación y pruebas. Ya sea en la forma de documentos obtenidos de internet o mediante la interfaz de búsqueda de una librería digital o motor de búsqueda existente. Acto seguido, obtener una muestra significativa de la colección para efectos de desarrollo y muestra preliminar.
2. Elegir e implementar la manera en la que se expresarán, construirán y actualizarán los perfiles de los usuarios.
3. Implementar un componente de indización e igualación de documentos y consultas; y desarrollar un componente que permita el filtrado de los resultados del primero en base a los perfiles y contextos.

1.3. Contexto

Los sistemas de recomendación, como el utilizado en www.amazon.com analizan el contexto local de un usuario en base a un perfil de éste o a su similitud con otros usuarios, donde tanto el perfil como la vecindad -en cuanto a preferencias- se decide por los artículos

de la colección propia del sistema que se han elegido explícitamente, en este caso, entonces, se trata de una tarea complementaria a lo que el usuario esté haciendo, orientada a llevar a éste a explorar otras opciones dentro de las ofrecidas por el servicio.

Por otro lado, los sistemas de recuperación de información pretenden suplir necesidades de información : el sistema busca qué documentos en su colección guardan alguna relación de similitud con las consultas que los usuarios expresan en un intento de concretar su necesidad. Un sistema de recuperación de información, por excelencia, es *el* componente principal y la tarea primordial de un usuario en cualquier contexto, por lo que pueden darse la licencia de dejar que el usuario examine un conjunto relativamente grande de resultados y que refine o reformule las consultas.

Pero en esta era de la información activa, los usuarios se ven sobrecogidos por la masiva cantidad de resultados que los motores de búsqueda ofrecen o las infinitas elecciones de palabras, a veces inasequibles, que pueden resultar en una búsqueda exitosa o en un completo fracaso. Asimismo, pocos sistemas de recomendación tradicionales están orientados a productores de información: en la mayor parte de los casos, es todo lo contrario, están orientados a hacer consumir más a los consumidores.

El servicio que se pretende construir en este proyecto puede ser visto como un híbrido de recomendador y recuperador de información: satisfacer necesidades de información en contextos productivos en segundo plano, sin obligar al usuario a distraerse pero ofreciendo recomendaciones útiles. Este enfoque no es nuevo, y la investigación en sistemas de este tipo en especial -iniciada por ?- se ha valido de servicios ya existentes de recuperación de información, lo cual podría probarse contraproducente: los sistemas de recuperación de información tradicionales asumen que la relevancia de un documento es equivalente a su similitud a la consulta del usuario, y están en lo correcto, para aplicaciones orientadas a la búsqueda activa. Pero, en un sistema que pretenda apoyar una tarea realizada por el usuario ¿sería verdaderamente *útil* proporcionarle documentos que probablemente no están relacionados con la tarea actual más allá del nivel léxico del contexto inmediato? En realidad, el sistema debería traer a la atención del usuario artefactos que habría pasado por alto por no saber cómo buscar o ni siquiera estar al tanto de su existencia o relación con su contexto local. Para ello, la medida por la cual se debería regir un sistema como el propuesto debería ser la *utilidad*: qué artefactos pueden traerle una ganancia al usuario mediante la reducción de un esfuerzo (al ayudarlo a hacer una búsqueda que no sabía cómo realizar) o la serendipia (al mostrarle elementos que le ayuden a ampliar su tarea (??) y esta utilidad sólo se puede lograr llegando a conocer al usuario, como se hace en los sistemas de recomendación.

1.4. Estado del arte

Aunque el enfoque elegido es relativamente nuevo, como ya se mencionó, durante los últimos años la necesidad de personalización y adquisición oportuna de información útil se ha tratado de resolver mediante sistemas de personalización y recomendación, por lo usual como parte de un producto de software específico o como aplicaciones independientes que monitorean el contexto del usuario. Entre las soluciones más conspicuas, se pueden

encontrar las siguientes:

1. FIXIT: aplicado al dominio de la reparación de fotocopiadoras y bienes relacionados, permite a los asesores de servicio técnico obtener información de los manuales de usuario en base a la consulta que estén atendiendo. Construye las consultas mediante los síntomas y problemas identificados, se vale de una red de inferencia bayesiana para determinar la probabilidad de relevancia de un documento a una consulta. Definido en ?.
2. Watson: un sistema que supervisa el entorno del usuario (su uso de procesadores de texto, navegadores de internet y clientes de correo electrónico) para sugerir documentos útiles, utiliza algoritmos heurísticos que encuentran unidades conceptuales en los documentos, a partir de las cuales realiza búsquedas concurrentes en distintas bases de datos y luego elimina duplicados en los resultados mediante algoritmos de agrupamiento. Definido en ?.
3. Remembrance Agent: Un módulo que se utiliza en el editor de textos *emacs*¹, y que sirve para realizar búsquedas de información relacionada a lo que se escribe en distintos repositorios, desde el correo electrónico del usuario hasta librerías digitales, mostrando los resultados en un apartado de la ventana de edición. Disponible para descarga en <http://www.remem.org/> y descrito en detalle en ?. Utiliza búsqueda mediante palabras clave y permite buscar explícitamente otros documentos relacionados a las sugerencias.
4. Margin Notes: Un agente que revisa páginas en html y agrega, en un margen derecho, vínculos a documentos locales que podrían ser de interés a las distintas secciones del documento; extrae palabras clave de la página y se basa en la *co-ocurrencia* de éstas en los documentos locales para sugerir. Presentado en ?.
5. Jimminy: desarrollado para una computadora que se porta en el hombro con una terminal en un casco, recaba datos del contexto físico y social de la persona mediante sensores (GPS, infrarrojos, etc) y permite tomar notas mediante un teclado que se porta en el brazo. A partir del contexto local, sugiere notas que podrían ser de utilidad. También introducido por ?.
6. Implicit Queries y Stuff I've Seen: sistemas de administración de la información personal que permiten la búsqueda más allá de palabras clave, basándose en tipos de documento, personas y fechas; el sistema Implicit Queries infiere las necesidades de información mientras el usuario escribe un correo electrónico y le sugiere documentos que podrían servirle de apoyo, como correos anteriores, entradas de agendas, etc. Una descripción se puede encontrar en ??.
7. PIRA: un sistema basado en internet que, a medida que se escribe una investigación, sugiere documentos relacionados y términos de búsqueda, a la vez que permite citarlos automáticamente al elegirlos de los resultados, agregándolos también a una

¹ <http://www.gnu.org/software/emacs/>

bibliografía. Se basa en servicios web de terceros para encontrar los términos clave, ampliar la búsqueda con sinónimos y ofrecer las facilidades de cita automática. Disponible para pruebas en www.writencite.com y propuesto en ?.

8. À propos: trata de determinar en qué etapa de la composición de un documento está el usuario para sugerirle lecturas y recursos para ampliar su composición, también se basa en búsquedas basadas en palabras clave en distintas bases de datos. Presentado en ?.
9. AMIDA: supervisa lo que se discute en una reunión de trabajo y recupera textos de reuniones anteriores que puedan servir para fundamentar puntos de la presente. El concepto y la descripción del sistema se pueden encontrar en ?.
10. Scienstein: un sistema de recomendación que se basa en lo que se ha escrito y citado para sugerir documentos que puedan ampliar una investigación. Descrito en ?.
11. Zemanta: un agente de enriquecimiento del contenido: supervisa lo que un usuario -en concreto, un autor de *blogs*- escribe y le recomienda artículos, imágenes y demás medios para enriquecer su tarea. Se puede agregar a un sitio mediante un widget de javascript o a una aplicación mediante una interfaz de programación de aplicaciones (API, por sus siglas en inglés). Activo desde 2009, más información se encuentra disponible en <http://www.zemanta.com/>

1.5. Identificación del problema

Aunque, como se puede observar en la sección anterior, existen iniciativas que pudieran llevar los avances de la contextualización y la personalización de los sistemas de recomendación tradicionales al ámbito de la producción de la información, es necesaria una implementación lo suficientemente flexible como para permitir la expansión a varios lenguajes naturales y ser utilizada por los desarrolladores en cualquier ámbito mediante un protocolo popular, como lo es HTTP, sin dejar de lado las ventajas del aprendizaje de máquina y la personalización, relegando estas tareas a un servidor que implemente un sistema de recomendación, dejando solamente a los implementadores la responsabilidad de iniciar la comunicación e interpretar las respuestas en las soluciones que creen para sus propias aplicaciones, en lugar de tener que crear ellos mismos otro sistema de recomendación oportuna de información.

1.6. Descripción de la solución

El proyecto propuesto aquí busca ofrecer a los implementadores una interfaz de programación de aplicaciones que permita que, mediante solicitudes HTTP, se puedan brindar recomendaciones de referencia útiles para tareas de producción de información a los usuarios finales modelando el conocimiento de las preferencias de los usuarios mediante una ontología jerárquica y fundamentándose en los principios de la técnica de arquitectura REST (a su vez basada en las capacidades del protocolo HTTP). Mientras que los

implementadores sólo se habrán de preocupar por hacer solicitudes a las URLs definidas en la API e interpretar los datos que el servidor regrese en respuesta, el lado del servidor utilizará técnicas de recuperación probabilística de la información y aprendizaje heurístico de perfiles de usuario para descubrir conceptos en sesiones de uso y mejorar las respuestas -en forma de recomendaciones de hipermedia- conforme los usuarios finales utilicen progresivamente el sistema que los implementadores de la API ofrezcan. El uso de principios de arquitectura REST, por otro lado, permitirá que la implementación del lado del servidor pueda escalar y refinarse sin afectar ni a los implementadores ni los usuarios finales. Asimismo, el servidor proveerá almacenamiento que hará posible la persistencia de las preferencias de los usuarios de las aplicaciones, así como la posibilidad de adiciones de fuentes de conocimiento translingües (al diseñar el almacenamiento con la multiplicidad lingüística de contenido en mente).

Las ventajas de una solución basada en una API apegada a ciertos principios REST es la flexibilidad: los implementadores podrán hacer uso de ella desde aplicaciones de escritorio, aplicaciones web tradicionales o aplicaciones web que implementen la técnica AJAX (ofreciendo soporte a la técnica JSONP).

El aporte del modelado del conocimiento mediante una ontología permite que las preferencias del usuario se aprendan más rápidamente que en otros enfoques de sistemas de recomendación, evitando así el clásico problema de *arranque en frío*.

En la actualidad la única solución popular muy similar a la propuesta es zemanta.com, pero ésta no soporta la evolución automática de perfiles de usuario (aunque provee opciones de personalización explícita) y solo se ofrece en el idioma inglés. La solución aquí propuesta, en cambio, habrá de soportar, inicialmente, inglés y español, con un diseño abierto a la inclusión de nuevos idiomas, asimismo, la evolución y uso meramente automáticos de perfiles de usuario son fundamentales para este proyecto. En la figura 1.1 se puede apreciar un diagrama de actividad que explica cómo un implementador podría utilizar el servicio desarrollado y en la figura 1.2 se puede ver cómo podría utilizar el usuario final una aplicación con el servicio implementado.

Sin embargo, se debe aclarar que el servicio -en esta fase de implementación como proyecto de graduación- solamente ofrecerá recomendaciones de hipermedia en forma de texto (HTML y PDF, por ejemplo) y contará con una base documental relativamente pequeña y la funcionalidad de incrementarla ya sea de manera automática o explícita, aunque facilitada y permitida por el diseño, está fuera del alcance del mismo (pues esta funcionalidad incluiría el diseño de arañas web y módulos de aprendizaje de máquina para clasificación de documentos *nuevos*, así como de componentes de actualización de la base de conocimiento). Nótese, sin embargo, que muchos de estos componentes (módulos de clasificación, interpretación de bases de conocimiento y arañas web) se habrán de construir, en menor escala, para la obtención de la colección inicial, sentando las bases para trabajo futuro que implemente dicha funcionalidad. De ahí que cabe notar que la alternativa (zemanta.com) es mucho más madura en este aspecto, pues ofrece no solo recomendaciones de hipermedia en forma de texto, sino también imágenes; además de contar con una base documental muy extensa y en constante crecimiento.

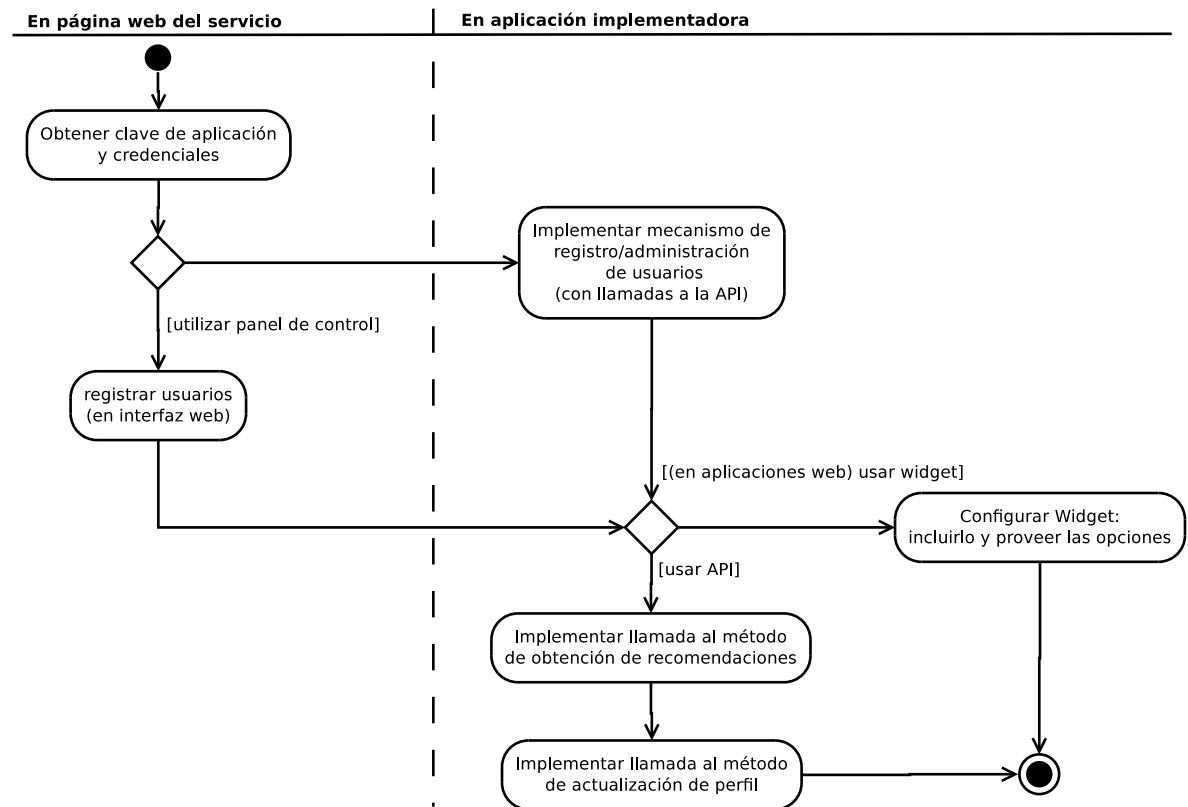


Figura 1.1: Implementación del servicio

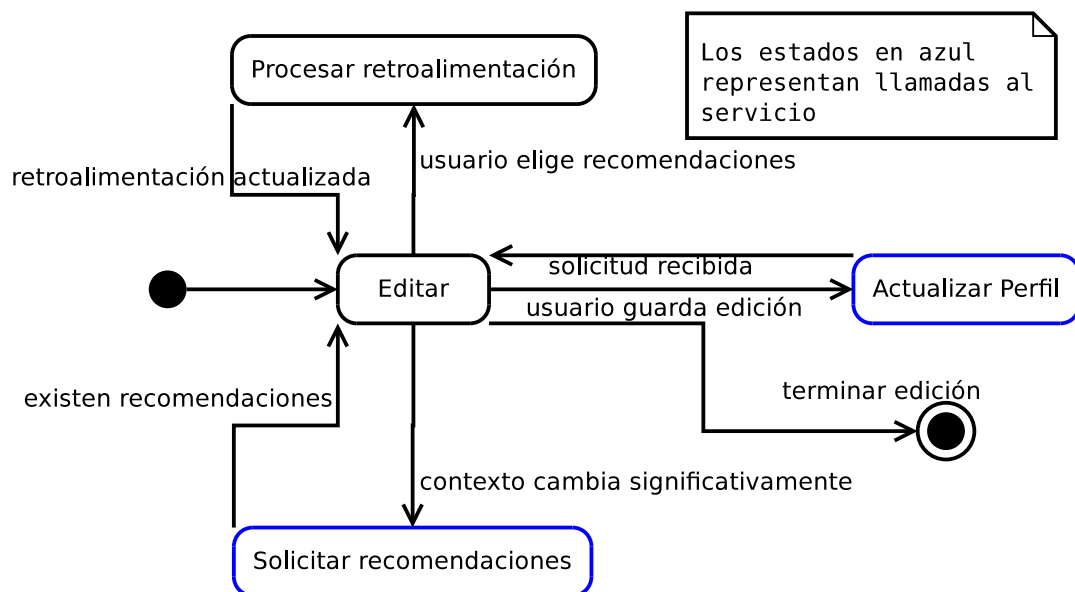


Figura 1.2: Uso de una aplicación que implemente el servicio

1.7. Definición de la metodología usada para la solución

En primera instancia, el autor debe confesar que no se utilizó ninguna metodología estricta en el desarrollo, esto debido a que se consideró lo siguiente: el poco tiempo disponible -menos de tres meses, y en éstos, sólo unas cuantas horas al día- podría convertir el rigor de una metodología formal en un obstáculo al desarrollo (en esto, quizá, se tomó una decisión propia del proceso ágil de desarrollo) y el hecho que, siendo un solo desarrollador, un proceso más laxo -mas no totalmente caótico- sería suficiente para mantener el derrotero de la implementación.

Aclarada la situación, se habrá de indicar que la metodología seguida -casi empíricamente- puede encontrar similitudes en lo prescrito por el modelo de *desarrollo rápido de aplicaciones* (DRA, definido en ?): en éste, luego de una etapa de planificación, análisis y diseño, se procede a desarrollar el software dando preferencia a implementaciones ya existentes de componentes no fundamentales; sin embargo, el DRA es un modelo no-evolutivo del desarrollo del software, mientras que, dado que la implementación era la primera incursión del autor en el desarrollo de un servicio como este, se requería que el software pudiera evolucionar y construirse incrementalmente, un componente a la vez, probándolos conforme se completaran, en iteraciones rápidas: esto se puede encontrar en metodologías ágiles del desarrollo, en concreto, el *Desarrollo conducido por características* (DCC), también definido en ?.

De esta manera, se puede concluir que se utilizó un híbrido de metodologías: del DRA se puede tomar la tendencia a acortar el ciclo de desarrollo al utilizar tecnologías ya existentes y del DCC la preferencia de iteraciones cortas y componentes independientes que puedan ser probados antes de proseguir a utilizarlos para incrementar el producto final. Las iteraciones correspondieron a los objetivos planteados: obtener la colección documental, implementar el componente de búsqueda, el de perfilado, y la consolidación del servicio.

2 Marco teórico

Se establecen los fundamentos teóricos detrás de la implementación de la solución, con el fin de ubicarla en el contexto científico actual.

2.1. Recuperación de información.

Se comienza con una puesta al día en el área de la recuperación de información (RI), ya que, a pesar de ser los sistemas de recomendación ya toda un área de investigación por mérito propio, es necesario clarificar que lo que subyace a un sistema como el que se presenta en este proyecto es, en mayor medida, una tarea de recuperación de documentos. Se presentan los componentes de un sistema computacional de RI y sus métodos de implementación, además del proceso genérico que sigue un usuario a la hora de utilizar un sistema de esta índole. Para terminar, se determina cómo se podría evaluar un sistema de recuperación de información.

2.1.1. Definiciones y conceptos preliminares

Aunque últimamente, con el extendido uso de los motores de búsqueda, se pueda creer que el área de la recuperación de información es relativamente nueva y su máxima expresión son los susodichos sistemas, la verdad es que la recuperación de información como tal nace mucho antes, quizá desde que comenzaron a existir las primeras grandes colecciones de documentos, pues la necesidad de información, y el consecuente desarrollo de técnicas para buscarla entre lo que otros ya han documentado, es inherente al desarrollo de una cultura escrita.

En un primer intento por definir la recuperación de información, se puede decir que:

La recuperación de la información es la obtención de material que satisfaga una necesidad de información.

Entendiendo por material cualquier artefacto (video, sonido, texto, etc.) que sirva para resolver una carencia de información en quien emprende la actividad de recuperación. Así, una consulta al directorio de nuestro teléfono celular para recordar el número de un amigo consistiría en una actividad de recuperación de información.

Pero si de definiciones se trata, se puede citar la definición dada en el primer capítulo de (?) como una muy buena para comenzar la discusión de la que tratará esta introducción y esta primera parte en general:

“La recuperación de la información es encontrar material (usualmente documentos) de una naturaleza no estructurada (por lo general, texto) que sat-

isface una necesidad de información en colecciones grandes (que pueden estar guardadas en computadoras).”

De lo anterior se pueden resaltar algunos conceptos que será de gran utilidad elaborar, nótese que la referencia a sistema no implica uno computacional:

Material de naturaleza no estructurada: Se pueden considerar como la *salida* del sistema. Los textos con los que nos encontraremos en grandes colecciones (e incluso, en librerías digitales) no son, necesariamente, de naturaleza homogénea: unos pueden ser formularios con campos rígidamente definidos, otros, documentos con cierta estructura, como artículos científicos con un título, resumen, palabras clave, etc; o bien en formatos electrónicos estándar, como xhtml o xml; pero la mayor parte de los documentos no tienen una estructura reconocible (lo cual no excluye que puedan tener alguna de naturaleza arbitraria) y esta situación se agrava aún más cuando la colección en uso no es ni más ni menos que la *world wide web*.

Necesidad de información: Se puede considerar como la *entrada* del sistema. Un usuario del sistema (una persona u otro sistema) expresa de alguna manera una carencia de información y se trata de procesar ésta de manera que se le puedan proveer recursos relevantes. Es importante notar que la necesidad de información puede ser explícita -en forma de consulta- o implícita -que se puede inferir del contexto o de las preferencias del usuario.

La definición dada es, a todas luces, una descripción precisa de los sistemas modernos de recuperación de información, pues especifica la noción a los sistemas de recuperación documental contemporáneos.

De las definiciones dadas, se puede apuntar el equívoco nombre que se ha dado a este concepto, o al menos a su implementación práctica: recuperación de información parecería sugerir que, de una gran colección de documentos, se puede extraer *información útil*, pero, como se puede observar, los sistemas de RI no aspiran a tanto: sencillamente dan al usuario documentos que puedan suplir su necesidad, pero es la responsabilidad última del usuario determinar que en efecto es así. Como nota marginal, sin embargo, recuérdese que otra área de las ciencias computacionales, la *extracción de la información*, sí aspira a llenar necesidades estructuradas de información.

(?) Menciona también otros conceptos que vale la pena clarificar:

Documento Por el cual se entiende aquello que consideraremos una unidad en el sistema: pueden ser las páginas de un libro, los mismos libros, los artículos de un periódico o los periódicos de un día.

Relevancia El valor que el usuario le da a un documento en base a su necesidad de información. Es un concepto que es, en realidad, bastante subjetivo, porque para algún usuario documentos que pudieran coincidir exactamente con el tópico que parecía ser el que deseaba conocer serían considerados menos relevantes -por considerarlos quizá demasiado obvios- que otros que, aunque menos relacionados, considere que le aportan más.

2.1.2. Componentes de los sistemas de recuperación de información

Un sistema de recuperación de la información de documentos en grandes colecciones, como los entendemos actualmente, desde bibliotecas hasta motores de búsqueda, a pesar de las grandes diferencias de implementación y arquitectura, tienen algo en común: todos tienen ciertos componentes que, si bien pueden variar en importancia o discernibilidad de otros, definen a un sistema de RI. Antes de discutirlos más a fondo, véase en la figura 2.1 el *ciclo de la recuperación de la información* para tener una idea de los roles que juegan los componentes de estos sistemas.

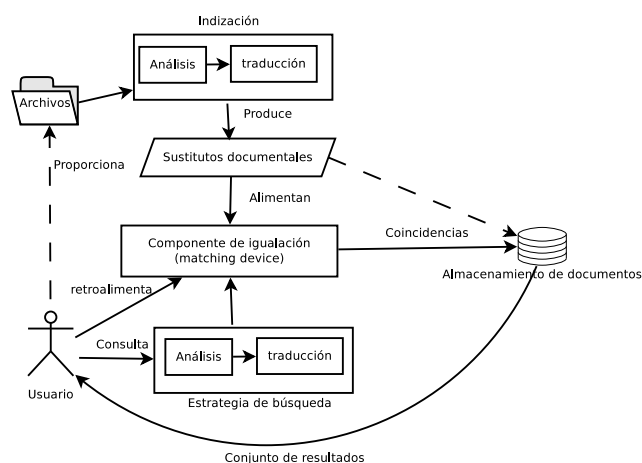


Figura 2.1: El ciclo de la recuperación de la información. Adaptado de (?)

Los componentes y actividades más notables de un sistema de recuperación de información pueden considerarse, según han determinado (??) , los siguientes:

Una colección de documentos: Es el conjunto de todos aquellos documentos que el sistema conoce y puede brindar como resultados a sus usuarios. Constituye el componente central de un sistema de RI, pero sería de poco valor¹ si no fuera por el concepto de los *sustitutos documentales* (*en inglés, document surrogates*), que son representaciones de los documentos que el sistema conoce y que permiten una obtención más rápida de los mismos. Estos sustitutos documentales se elaboran mediante un proceso de gran importancia: *la indización*; la cual, aunque es una actividad, es también, en muchos de los casos, un componente bien definido del sistema. El propósito de la indización es crear los sustitutos documentales que se almacenarán en la colección y servirán de entrada al componente de igualación. Cada nuevo documento que es agregado al almacenamiento del sistema precisa ser indexado; y esta actividad puede ser tan compleja como se quiera: puede ir desde la simple actualización del índice inverso con el documento, la representación como vector de palabras, o la clasificación del mismo en una taxonomía.

¹Si se prescinde de la indización, la otra alternativa es una búsqueda lineal, nada factible para grandes colecciones documentales.

El componente de igualación: Es la parte que sirve de intermediario entre el almacenamiento documental y las consultas del usuario. Es importante notar que la entrada de este componente no es necesariamente la consulta original del usuario, sino que puede haber sido analizada y traducida al *lenguaje interno* del sistema, que es, en última instancia, una *representación del conocimiento*. Una vez en poder de la consulta, este componente trata de asimilar ésta con los sustitutos documentales y luego, si hay coincidencias, solicita al almacenamiento los documentos.

La consulta del usuario: La consulta es la manera del usuario de expresar su necesidad de información. La cual puede ir desde una pregunta en lenguaje natural hasta unos cuantos términos claves de un vocabulario controlado, pasando por consultas escritas con lógica booleana. La consulta se forma mediante la elección de parte del usuario de una *estrategia de búsqueda*.

El conjunto de resultados: Los documentos que el sistema ha considerado relevantes a la búsqueda del usuario, presentados de la manera que se considere idónea, ya sea gráficamente, como los sistemas que se centran en mostrar al usuario no sólo los documentos relevantes sino sus relaciones o como una lista en orden de relevancia según criterios propios del sistema.

2.1.2.1. La estrategia de búsqueda

La necesidad de un *índice* de la colección documental no es algo nuevo ni propio de sistemas computacionales de RI; más bien, como hace notar (?), la recuperación de información nace de la necesidad de determinar si mediante un documento cierta carencia de información se puede suplir sin tener que recurrir a una lectura completa, o al menos significativa, del recurso. Y es de esta necesidad, tanto en documentos de extensión considerable como en colecciones grandes, que se crean los índices en los libros y los sistemas de clasificación bibliotecarios. Así, todo sistema de RI que se precie de ser útil deberá proveer alguna manera de acceder a la información contenida en los documentos; de cualquier manera, los documentos han de ser representados de forma que la técnica elegida pueda valerse de ellos, y en esto radica la importancia de la elaboración de *sustitutos documentales*.

En cualquier sistema de RI, ya sea mediante la ayuda de un profesional en una biblioteca o en un motor de búsqueda, el usuario debe ser capaz de expresar su necesidad de información, es decir, como bien señala (?), que la RI es, en el fondo, un asunto de *comunicación*.

Pero el tema de la elaboración de una consulta no es nada sencillo. En primera instancia, se debe discernir qué conceptos podrían satisfacer la necesidad y expresarlos ya sea en un lenguaje natural o en alguna otra representación conceptual de comunicación. Mas precisamente aquí se encuentra el primer obstáculo de la recuperación de la información: la naturaleza subjetiva de los conceptos, y, por tanto, las múltiples maneras de expresar uno. A esto muchos sistemas de recuperación de información responden con el uso de palabras clave o vocabularios controlados -aquellos donde se da preferencia a una expresión de un concepto sobre otro, por ejemplo, al uso de la palabra automóvil para

referirse a un carro. El usuario podría tener acceso a éstos o podría *aprenderlos* mediante la experiencia adquirida en búsquedas anteriores. Esta limitación, si se desea ver así, en los sistemas de recuperación de información es expresada de una manera idónea por (?):

Queremos buscar conceptos, pero estamos forzados a buscar palabras.

El otro lado de la comunicación es igualmente importante. Una vez que el usuario ha elegido una consulta, el sistema debería ser capaz de interpretarla y procesarla de manera que el componente de igualación tenga éxito en encontrar coincidencias en los documentos. Este proceso puede ser tan sencillo como simplemente procesarla tal cual se introdujo o tan complicado como se quiera; de hecho, muchos sistemas de recuperación de información pre-procesan las consultas de usuario expresadas en algún lenguaje natural con técnicas que han probado hasta cierto punto ser exitosas:

1. Normalización de mayúsculas y minúsculas: convertir todo en los documentos y consultas a sólo mayúsculas o minúsculas, evitando así que documentos importantes se pierdan porque los términos no coinciden en este aspecto con lo contenido en documentos. Es evidente que esta técnica tiene sus desventajas, pues puede dar lugar a falsos resultados (como documentos que contienen el término en inglés *aids*, refiriéndose quizá a apoyos económicos en el tercer mundo cuando se ha buscado por el término *AIDS*, el síndrome de inmunodeficiencia adquirida).
2. Reducir las palabras a sus raíces: -en inglés *stemming*- que consiste en eliminar terminaciones de plural/singular, femenino/masculino o los sufijos de conjugaciones en verbos regulares. Aunque algoritmos de buena utilidad existen para ello, en muchos de los casos se tiene menor ganancia de la que se podría esperar -sólo de un 2 % en inglés según ?; aunque se ha determinado que es más útil en otros lenguajes como el alemán (?).
3. Usar sinónimos para ampliar o normalizar las consultas.
4. Corregir la ortografía. Lo cual podría dar lugar a ganancias, pero, dada la variedad documental en internet y la frecuencia del surgimiento de neologismos, podría probarse hasta cierto punto contraproducente. (?) reporta el uso exitoso de un algoritmo de corrección ortográfica basado en el sonido.
5. Utilizar la estructura y la *meta-data* de los documentos. Los usuarios pensarán en conceptos que probablemente se encuentren en los títulos, resúmenes o encabezados de los recursos. Documentos internamente estructurados como artículos científicos y de periódicos o contruidos en formatos organizados como xml pueden ser candidatos a este tipo de refinamiento. Asimismo, el hecho de que documentos hayan sido citados o enlazados por otros mediante términos que no aparecen necesariamente en el documento permite ampliar las posibilidades de encontrarlo.

Como se ve, el sistema debe tratar de entender la consulta del usuario antes de buscar coincidencias. En el capítulo siguiente técnicas relacionadas al procesamiento de la consulta se presentarán más a detalle; cabe notar que, vista como un problema de comunicación, la

RI se podría valer de los avances en lingüística computacional, pero, como bien apuntan ?, debido a la naturaleza de corta extensión de muchas consultas en sistemas basados en internet, el uso del procesamiento de lenguaje natural en un sistema de RI debería usar herramientas especialmente confeccionadas para el mismo y no adaptadas de otras áreas de investigación.

2.1.2.2. Presentación de los resultados

Una vez que el sistema ha determinado los documentos que, según el modelo que utiliza, son relevantes a la consulta del usuario, debe presentarlos. Cualquier usuario de motores de búsqueda diría, según su experiencia, que este es un problema trivial: sencillamente se deberían presentar los documentos en orden descendente de relevancia. Para muchos sistemas, esto es suficiente porque los usuarios o sólo desean alguna información somera o no se disponen a investigar más a fondo. Pero, ¿qué sucede con aquellos que desean indagar a conciencia en un tema? Los motores de búsqueda de propósito general más utilizados se quedan cortos en este respecto, pero no significa que no haya otras maneras de representar los resultados de una manera más útil, ni mucho menos que no se hayan implementado.

Una manera es mostrar gráficamente las relaciones entre documentos, entendiéndose éstas, por ejemplo, como que uno cite a otro o conceptos aparecen de maneras similares en los documentos. Un enfoque así se puede encontrar en (?), donde el sistema muestra de manera gráfica los documentos y sus relaciones (y los documentos más relevantes se muestran de mayor tamaño).

Pero quizá el usuario esté más interesado en ver los documentos agrupados de alguna manera. Para esto se pueden utilizar dos técnicas estrechamente relacionadas con el campo del aprendizaje de máquina:

Clasificación: Dada una categorización preexistente de conceptos, los resultados se presentan agrupados bajo éstos. Así, por ejemplo, en un sistema de recuperación de información en la base de documentos de alguna enciclopedia, los resultados de una búsqueda respecto a “Eco” mostraría alguno resultados agrupados bajo la categoría “Escritores” con documentos refiriéndose a Umberto Eco, otros bajo la categoría “Física” con documentos relacionados al fenómenos físico del eco, etc. Un sistema computacional de RI que desee utilizar clasificación debería utilizar algún tipo de aprendizaje de máquina supervisado: en primera, la categorías se deben decidir de alguna manera (ya sea a criterio de quien compila la colección o mediante otras técnicas de inteligencia artificial), a seguir, se debe elegir un conjunto de documentos y etiquetarlos con la categoría a la que pertenecen; este conjunto se conoce como el *conjunto de entrenamiento*. El algoritmo de aprendizaje debe formular una *hipótesis* que explique por qué a los documentos se les dieron esas etiquetas de categorías. Luego, otro conjunto de documentos etiquetados se alimenta al sistema, esta vez, sin las etiquetas, para ver si lo puede clasificar con éxito -comparando al final si las etiquetas previamente asignadas coinciden con las dadas por el sistema. Los resultados se evalúan y la hipótesis se puede refinar (con una *matriz de confusión*,

por ejemplo). Nótese que la clasificación cuenta con una *taxonomía de tópicos pre-existente*, por lo que sólo es factible para colecciones de documentos manejables con categorías claramente definidas. Esta técnica es utilizada por (?) en su sistema clasificador.

Agrupación: (En inglés, *clustering*). A diferencia de la clasificación, los documentos se agrupan en categorías determinadas *a la hora de presentar los resultados* -y no pre-existent. La agrupación es un problema de aprendizaje *no supervisado* de máquina (mientras que la clasificación es supervisada) y se vale de técnicas estadísticas para agrupar los documentos según ciertas medidas de distancia (la cual es una medida estadística de frecuencia y reincidencia de palabras en los textos): documentos con poca distancia entre sí se agruparán juntos; luego, se pueden determinar las etiquetas a poner a los grupos, lo cual se puede basar en los títulos de los documentos, según alguna medida estadística, por ejemplo, los más cercanos al *centro* del grupo; o sencillamente etiquetarlos con las palabras más frecuentes. Dada su naturaleza no supervisada, esta técnica, aunque computacionalmente más costosa, es preferida por los investigadores. El sistema presentado en (?) utiliza técnicas de agrupación para presentar los resultados.

Además, el sistema podría valerse de la *retroalimentación* del usuario, ya sea implícita, mediante el acceso a documentos, el tiempo de estancia en ellos -si éstos consisten en hipervínculos- o la impresión; o explícita, mediante evaluación de relevancia. Así, con la retroalimentación, el sistema podría construir un nuevo conjunto de resultados a partir de éste, buscando aquellos documentos que se aproximen a los preferidos por el usuario -es importante resaltar que esta es una característica básica de los sistemas de recomendación.

2.1.2.3. Obtención de la colección documental

Aunque la existencia de la colección se da por sentado al hablar de la implementación de sistemas de recuperación de información, la obtención de ésta es indiscutiblemente fundamental para el sistema. Aunque existen colecciones en internet que se construyen exclusivamente mediante la colaboración de los usuarios, como wikipedia² o el *open directory project*³, y aún cuando esta tendencia es cada vez más común, dado el auge de la *web 2.0* y la *web semántica*, un sistema de RI a gran escala deberá contar con un conjunto de documentos suficientemente grande como para satisfacer la mayor parte de las necesidades de información de sus usuarios, y, para ello, deberá disponer de algún método de obtención automática de nuevos documentos.

Las *arañas web* son tales mecanismos: son programas que, partiendo de un conjunto inicial de páginas web, encuentran en éstas nuevos vínculos a otros documentos, los cuales agregan a su conjunto y exploran para encontrar nuevos vínculos y así sucesivamente. Este enfoque simple de las arañas web se puede encontrar implementado en el lenguaje de programación python por ?. Las dificultades del uso de una araña web en un ambiente

²www.wikipedia.org

³<http://www.dmoz.org/>

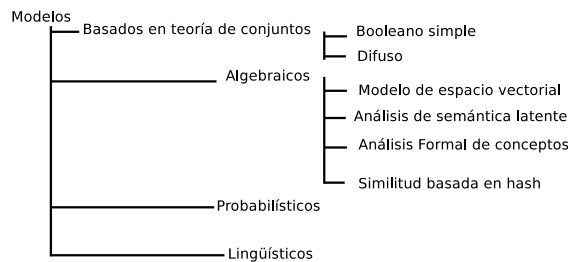


Figura 2.2: Taxonomía de modelos de recuperación de información. Adaptada de ??

comercial, como la privacidad y el volumen de datos procesados, se pueden encontrar someramente relatadas en las experiencias de ? al desarrollar google⁴. Por su parte, ? menciona las características más importantes que una araña web debería tener: *robustez*, para no caer en trampas de navegación que lo hagan quedarse demasiado tiempo en un solo dominio; *cortesía*, para respetar las políticas de sitios que tienen un límite de visitas automáticas o no desean ser agregados a un índice -téngase presente que existe un estándar⁵ para determinar, en un archivo de texto llamado usualmente robots.txt, las direcciones de un sitio que las arañas web no tienen permitido agregar a los índices de sus sistemas anfitriones; *escalabilidad, desempeño, eficiencia y posibilidad de distribución*, para usar óptimamente los recursos de los que dispone; *frescura*, obtener siempre copias nuevas de las páginas; *un estándar de calidad*, para explorar sólo aquellas páginas que podrían verdaderamente ser relevantes, ya sea por su riqueza de contenido, su *fama* entre otras páginas o su apego a convenciones estándar y *extensibilidad*, una araña se debería poder adaptar a nuevos formatos, protocolos y estándares.

Un caso interesante de implementación de arañas web es cuando no se desea obtener cualquier página para agregarla a la colección, sino que sólo aquellas que se refieran a cierto tópico; en este caso, las arañas deben tomar decisiones sobre la relevancia a un tópico y hasta aprender de recolecciones anteriores ??.

2.1.3. Modelos de recuperación de información

El componente más importante de un sistema de RI, como se vio anteriormente, es el tratamiento de la consulta del usuario en el componente de igualación; para que éste devuelva objetos verdaderamente relevantes o útiles, la indización debe haber construido una representación óptima de los mismos, y es en esta etapa en la que entran en juego los distintos modelos presentados en este apartado. En la figura 2.2 se presenta una taxonomía de algunos de los modelos de recuperación de información utilizados comercialmente y en investigación. De éstos, se explorarán someramente unos pocos en este apartado.

⁴www.google.com

⁵definido en <http://www.robotstxt.org/wc/exclusion.html>

El índice invertido Antes de explorar los modelos de recuperación de información, es menester definir una de las técnicas de indización presente en la mayor parte de las implementaciones: el *índice invertido*. Éste consiste en un archivo que contiene todas las palabras de la colección documental y en qué documentos ocurren éstas. Es esta estructura de datos la que permite que los sistemas de recuperación de información computacionales sean factibles, ya que la búsqueda por una palabra no se hace en los documentos en sí, sino en este archivo. Puesto que la construcción de esta estructura ocurre en la etapa de indización, es aquí donde se hacen decisiones fundamentales sobre el tratamiento de los documentos como qué hacer con la puntuación, utilizar las palabras completas o sólo las raíces, si distinguir entre mayúsculas y minúsculas, etc -nótese que estas decisiones deben reflejarse en el tratamiento dado a las consultas, como se define en el apartado 2.1.2.1.

El proceso general de construcción de un índice invertido se puede encontrar explicado en ??, y se resume a continuación:

1. Juntar los documentos a ser indizados, en alguna estructura de datos.
2. Separar en unidades (*tokens*) cada documento; las unidades son, por lo general, palabras, pero no necesariamente.
3. Hacer pre-procesamiento lingüístico (esto es, aislar raíces, normalizar mayúsculas o minúsculas, ignorar puntuación o palabras vacías, hacer etiquetado gramático, desambiguar palabras mediante análisis semántico, etc.) El pre-procesamiento puede parecer trivial para lenguajes cuya sintaxis y alfabeto lo permiten (como los que utilizan los alfabeto latino, griego y cirílico, por ejemplo; sin embargo, la tarea es más difícil en otros). Para una discusión más a fondo del tema del procesamiento lingüístico de documentos, consúltense ?. Además ? presenta un sumario de las técnicas más utilizadas en sistemas comerciales en esta etapa y sus resultados.
4. Por cada palabra, construir el índice agregando los documentos en los cuales ocurre ésta y, si se decide considerar la estructura, agregar también en qué parte del documento es la ocurrencia. Los documentos, por lo general, se representan mediante una clave numérica única y las partes mediante una clave o una abreviación.
5. Se ordena el índice alfabéticamente.

Nótese que el índice contiene entradas únicas para las palabras, así que es responsabilidad de quien lo implementa incluir un campo donde se incluya también la cantidad de veces que una palabra ocurre en un documento (o en una parte del documento, si es más granular); esta decisión, como las demás, depende del modelo en uso.

Búsqueda orientada a conceptos vs. la búsqueda orientada a palabras Como ya se apuntó, los sistemas tradicionales parten de la premisa de que las palabras que están en la consulta son suficientes para la búsqueda, sin embargo, como se sabe, la necesidad de información del usuario encierra *conceptos* que trascienden una mera instancia léxica, y por ello se presentarán también técnicas que parten de la premisa de que la

consulta sólo es una base semántica para la búsqueda y que ésta se puede aumentar (ya sea mediante sinónimos aportados por tesauros, búsquedas en ontologías o colecciones indexadas por conceptos) o refinar (mediante la desambiguación de términos, por ejemplo). Ambos enfoques se pueden llamar también *búsquedas léxicas* y *búsquedas semánticas*, respectivamente.

La motivación que subyace a la transición de la recuperación de información léxicamente orientada a una semánticamente orientada es la observación de la incertidumbre inherente al lenguaje natural, observable en dos aspectos del mismo: la *sinonimia* y la *polisemia*. Las consultas pueden ser expresadas en palabras que no aparezcan como tales en documentos verdaderamente relevantes sino que en su lugar se utilicen sinónimos, por lo que la exhaustividad del sistema se ve afectada, al no traer todos los artículos relevantes. Asimismo, documentos irrelevantes a una consulta pueden ser obtenidos debido a la polisemia -la variabilidad contextual del significado de una misma palabra- por lo que la precisión se ve afectada (?).

2.1.3.1. Modelo Booleano

Es el modelo más simple de recuperación de información. Podemos definirlo como una función que, dados un documento y una consulta escrita como una expresión booleana de palabras, el documento es relevante a la consulta si y sólo si la expresión evalúa a verdadera en el documento, donde las cláusulas de la expresión son las palabras mismas y la presencia de una palabra en el documento implica que la cláusula es verdadera.

Si se utiliza un índice invertido, basta con hacer búsquedas para las palabras de la consulta y devolver los documentos en los que éstas se encuentran, aplicando operaciones simples de conjuntos a los resultados según la expresión booleana. La consulta se puede hacer en un tiempo lineal respecto a la cantidad de documentos y palabras en la consulta; asimismo, optimizaciones heurísticas se pueden aplicar a las consultas.

Este modelo es tan fácil de implementar y tan eficiente computacionalmente, que se ha usado en sistemas comerciales de búsquedas en bases de datos desde hace casi medio siglo. Pero los sistemas implementados han ido más allá de los simples operadores booleanos y han introducido *operadores de proximidad* y comodines de búsqueda, con el propósito de mejorar los resultados. Un ejemplo de un sistema comercial con una implementación altamente especializada y optimizada del modelo booleano es el sistema *DIALOG*, explorado en detalle en ?.

Las desventajas del modelo booleano radican en que no hay manera de ordenar los resultados, puesto que los términos o están o no están, sin manera de saber si la palabra está realmente relacionada al tema central del documento o sólo aparece allí una o pocas veces en contextos distintos. Asimismo, los usuarios podrían no estar familiarizados con la lógica booleana, y aún siendo así, la diferencia de resultados entre una conjunción y una disyunción de términos puede ser demasiado abismal como para dar al usuario una idea de cómo refinar su consulta.

2.1.3.2. Modelo booleano extendido mediante lógica difusa

Trata de sobreponerse a las limitaciones del modelo booleano tradicional implementado la lógica difusa para las operaciones, así, los resultados de las expresiones no serán sólo verdadero o falso (0 ó 1), sino cualquier valor entre ellos, usualmente representado en el continuo $[0, 1]$.

Los documentos se tratan como conjuntos difusos y la membresía de cada término se puede calcular en base a la frecuencia de su ocurrencia, por lo que los resultados de las operaciones difusas extendidas se pueden utilizar para dar una medida de relevancia a los documentos en el conjunto de resultados. Aunque este es el enfoque más intuitivo, la lógica difusa se puede utilizar para extensiones más drásticas del modelo de conjuntos de recuperación de información (?). Nótese también que este tipo de lógica se asemeja más al lenguaje natural, pues toma en cuenta la incertidumbre y la ambigüedad. La presentación de un modelo de recuperación de información generalizado -que trasciende al mero uso de palabras clave para la búsqueda y se extiende al uso de *conceptos*- se puede ver discutida en ? y los fundamentos de la lógica difusa para la manipulación y representación del conocimiento se encuentran esbozados en ?.

2.1.3.3. Modelo de espacio vectorial

Es el modelo más utilizado en la actualidad, y ha servido de base para otros modelos que han reportado éxito en la investigación. Fue propuesto para la tarea de recuperación de información por ? y trata a las consultas y los documentos como vectores multidimensionales, donde cada término es una dimensión. A la hora de la búsqueda, se computa la similitud entre el vector de consulta y los de los documentos, retornando los resultados ordenados según su relevancia (siendo ésta proporcional a la similitud). Por lo general, las dimensiones se ponderan en base a la frecuencia de los términos en los documentos y así se pueden ordenar los resultados por relevancia.

En primer lugar, se debe tener en cuenta la representación de la colección documental. Como ya se dijo, el modelo de espacio vectorial parte de la premisa que los documentos se representan como vectores multidimensionales, tomando todas las palabras de la colección documental como las dimensiones. Nótese que estos vectores se han de construir con los términos que hayan quedado luego del pre-procesamiento lingüístico, en especial de la remoción de palabras vacías, para eliminar dimensiones posiblemente innecesarias.

El vector para un documento D se construye con los *pesos* asociados a cada término del espacio -el cálculo de éstos se discutirá enseguida. Visto de esta manera, entonces, para un vector con t dimensiones, un documento se representa así:

$$D_i = \langle w_{i1}, w_{i2}, \dots, w_{it} \rangle \quad (2.1)$$

A lo largo de este apartado, se utilizarán los documentos presentados en la tabla 2.1 para ilustrar los conceptos clave.

El peso w de un término se puede calcular respecto a ciertas medidas estadísticas y algunas de las más utilizadas en la implementación de este modelo son:

- Peso booleano: asigna 0 ó 1, dependiendo de la presencia o ausencia del término.

| id | Original | Pre-procesado | Vector |
|----|--|-----------------------------|------------|
| 1 | “Dijo alguien que decía lo que dices: Ser o no ser el ser que se es” | “decir[3]” | <3,0,0> |
| 2 | “Ser alguien en la vida, es poco decir, mi vida” | “decir[1] vida[2] poco[1] ” | <1,3.17,1> |
| 3 | “Dije que a poco es vida” | “decir[1] vida[1] poco[1] ” | <1,1.58,1> |

Cuadro 2.1: Documentos de ejemplo para el modelo de espacio vectorial. El procesamiento lingüístico elimina palabras vacías, puntuación y mayúsculas y reduce a raíces. Por claridad, las frecuencias se muestran contiguas a las palabras, aunque esto es responsabilidad del índice inverso. Las ponderaciones se han calculado con $tf*idf$

- Frecuencia local: el peso es proporcional a la cantidad de veces que la palabra ocurre en el documento. Esta ponderación es fácil de calcular y refleja de manera transparente una premisa común en mayor o menor medida a los modelos basados en palabras: que la posición de la palabra no es importante, suposición heredada de los modelos estadísticos basados en inferencia bayesiana ?. La frecuencia del k -ésimo término en el i -ésimo documento de una colección se puede expresar como f_i^k

- Frecuencia local multiplicada por la frecuencia global inversa: conocida en la literatura como $tf*idf$, se basa en la observación que dice que los términos verdaderamente importantes en una colección documental son aquellos que aparecen con la suficiente frecuencia en un documento para definirlo y a la vez son poco frecuentes globalmente. Esta ponderación será proporcional a $f_i^k \cdot IDF_k$ donde, para una frecuencia documental d_k que representa el número de documentos en los que aparece el k -ésimo término, para n documentos, la frecuencia documental inversa se define como:

$$IDF_k = \log \frac{n}{1 + d_k} = \log_2 n - \log_2 d_k + 1 \quad (2.2)$$

- Valores discriminantes: aunque la medida $tf*idf$ es ampliamente utilizada, el mismo ?, reconoció que aún así hay términos que son más decisivos en definir el carácter relevante de los documentos, en especial dentro de colecciones relativamente homogéneas; por lo cual introduce el concepto de *valores discriminantes* en función de la densidad del espacio vectorial: un término es discriminante en la medida en que su ausencia o presencia en la colección afecte qué tan cerca están, en términos de similitud, los documentos; un buen término, entonces, sería aquel que, al ser eliminado, hace que los documentos queden más cerca entre sí, afectando peyorativamente la precisión de las búsquedas. Puesto que no es factible computacionalmente encontrar el valor discriminante de todos los términos de la colección -ya que habría que calcular la densidad t veces, lo cual sería $O(n^3)$ - el mismo autor propone examinar el comportamiento de los términos respecto a las frecuencias,

y reporta que buenos términos discriminantes son aquellos que no son demasiado raros ni demasiado frecuentes, por lo que los términos poco frecuentes se deberían agrupar bajo un sinónimo (una clase de tesoro) y los más frecuentes, en *frases*, dado que éstas son, por lo general, menos propensas a tener la misma ocurrencia global que sus componentes individuales.

- Frecuencias de grupo: en el mismo trabajo, ?, propone un multiplicador similar al $tf*idf$, pero en vez de hacerlo relativo a toda la colección, calcularlo en base a las frecuencias en grupos, luego, claro, de haber agrupado los documentos relacionados mediante algún algoritmo de *clustering* (una explicación más exhaustiva de algoritmos de agrupamiento en la tarea de recuperación de información se puede encontrar en ? y una implementación simple del agrupamiento jerárquico, en ?).

En la enumeración anterior, nótese el uso indistinto de “término” y “palabra” como equivalentes, esto es debido al extendido uso de *unigramas* como los componentes fundamentales de los sistemas de recuperación de información -lo cual no es necesariamente el mejor ni único enfoque, pero es el más simple (?).

Una vez obtenidos los vectores, se pueden representar como en la figura 2.3a; pero la longitud de los vectores no es importante, sino la *similitud* que hay entre ellos, la cual se puede calcular con un producto interno o como una función inversa al ángulo que hay entre ellos (?). Por lo general, se utiliza la medida de *similitud de coseno* (definida en el apartado 2.2.2.1) porque ésta es independiente de la magnitud del vector, a diferencia de la mera diferencia de magnitudes, que puede ser ambigua, porque dos documentos muy similares pueden tener magnitudes proporcionales y, así, una diferencia significativa (?) que no refleja la verdadera similitud. El modelo se puede simplificar tomando la intersección de cada vector con la esfera unitaria y luego estirando ésta sobre dos dimensiones, de manera que los vectores ahora se aprecian como puntos en un plano y estos puntos estarán cerca unos de otros si los vectores originales estaban angularmente próximos. La figura 2.3b ilustra esta simplificación para el ejemplo de esta sección.

El quid de este modelo estriba en la siguiente hipótesis (?):

“La densidad de un espacio documental y el rendimiento de un sistema de recuperación de información están inversamente relacionados”

Lo cual, llevado al espacio simplificado, implica que los documentos más similares a una consulta estén lo más cerca posible a ésta y unos a otros; y, al mismo tiempo, que los menos relevantes estén lo más alejado posible a la consulta, esto debido a que la precisión de la búsqueda se puede ver afectada por la densidad del espacio: entre más cerca están los documentos, más posible es que, para una consulta, se obtengan resultados no-relevantes; por lo que es deseable que la densidad se minimice, distinguiendo lo más posible, mediante indización y ponderaciones óptimas, entre los documentos. Técnicas de agrupamiento basadas en la ponderación dada a los términos se pueden aplicar para hacer posibles estos requerimientos, acercando los grupos de documentos similares y alejando los disímiles.

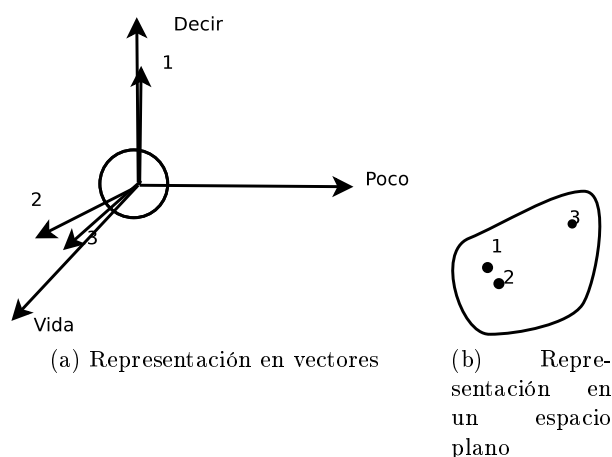


Figura 2.3: Representaciones en el espacio vectorial para el ejemplo de la tabla 2.1

2.1.3.4. Mejoras del modelo de espacio vectorial

Aunque el modelo de espacio vectorial tiene en cuenta algunos aspectos del lenguaje natural, los términos individuales siguen consistiendo en dimensiones, por lo que documentos semánticamente similares a una consulta, pero léxicamente diferentes, seguirán apareciendo con un bajo puntaje de relevancia -este fenómeno se da aún cuando se usa el valor discriminante bajo clases de sinónimos, puesto que los términos son o equivalentes o no equivalentes a una clase.

Para resolver este problema, el *modelo de espacio vectorial orientado a tópicos* asume que los términos no sólo pueden ser equivalentes o no equivalentes, si no que pueden pertenecer en distintos grados a estas dos clases -tomado de la lógica difusa; este modelo se define y compara con otros basados en el de espacio vectorial clásico en ?. Por su parte, el *modelo generalizado de espacio vectorial*, asume que los términos individuales de un vector también pueden ser similares entre sí, por lo que se representan también con vectores en un espacio combinatorio. Una implementación de este modelo se puede encontrar explicada en (?). Ambos modelos parten de la idea de aprovechar las relaciones entre términos, y difieren en que el primero utiliza información basada en tesauros y ontologías y el segundo infiere las relaciones entre términos a partir de la colección documental.

Un tercer modelo que extiende al de espacio vectorial clásico es el basado en *Semántica Distribuida* (DSIR, por sus siglas en inglés); éste utiliza la distribución contextual de las palabras, basada en una matriz de co-ocurrencia, para ponderar los términos y así sobreponerse a los problemas de sinonimia y polisemia, dejando alejados en el espacio vectorial aquellos documentos donde los términos sólo sean polisémicamente equivalentes y cercanos aquellos donde haya similitud sinónímica. Este modelo ha sido implementado y probado con una colección documental relativamente grande por ?.

2.1.3.5. Indización semántica latente

Este modelo parte de la premisa de que existe una “estructura semántica latente que subyace a la elección de palabras específicas de una consulta”?. Es decir, las palabras que el usuario elige para realizar una consulta son instancias de un *concepto* expresable de otras maneras. Este modelo trata de encontrar esta estructura latente mediante una aproximación estadística: parte de una matriz donde se representan todos los términos contra los documentos en los que aparecen (cada entrada de ésta corresponde a la frecuencia del i -ésimo término en el j -ésimo documento) y luego esta matriz se descompone en tres matrices menores mediante la técnica de *descomposición en valores singulares*, reduciendo así la cantidad de dimensiones del espacio documental y dejando que términos sinonímicamente relacionados se representen en una sola dimensión. Esta reducción dimensional basada en la contextualización estadística de los términos es la que permite que documentos semánticamente relacionados a una consulta queden en la vecindad de ésta en el espacio vectorial, mejorando el rendimiento del sistema ???. Por su parte, ?, resume el proceso de representación documental como la transformación de la matriz original de términos/documentos a un espacio con k dimensiones (donde k es una constante elegida usualmente con un valor entre los pocos cientos) donde éstas son representadas por los k principales *eigenvectores* (*vectores propios*, los que mejor representan un espacio) del espacio original, haciendo así que sólo aquellas dimensiones que son verdaderamente representativas de los documentos subsistan y que otras que en realidad representaban valores cuasi-sinonímicos desaparezcan y se vean reducidas a las dimensiones propias del espacio. Una explicación más detallada y matemáticamente fundamentada de este modelo se puede encontrar en ?. Y una comparación entre este modelo y otro basado en la expansión de consultas de usuarios mediante el uso de ontologías se puede encontrar en ?.

Este modelo no carece de problemas: en primer lugar, el costo computacional relacionado a la transformación matricial -el cual se debe hacer cada vez que la colección documental cambie *drásticamente* ? -énfasis en qué se considere como drástico, pues la adición de pocos documentos nuevos se puede hacer al espacio ya existente, ignorando momentáneamente las nuevas dimensiones que éstos pudieran introducir y sólo redefiniendo el espacio cuando se alcance algún umbral fijo o relativo al rendimiento (?) ; y, a medida que las colecciones crecen, la polisemia de los términos puede afectar negativamente a la transformación (al introducir vectores propios espurios o equívocos). Una solución a ambos problemas puede ser una partición de la colección documental en segmentos más manejables, como se propone en ?.

2.1.3.6. Análisis formal de conceptos

Modelo propuesto para recuperación de información por ?, se basa en la construcción de un *retículo* (en inglés, *lattice*) a partir de los conceptos y los documentos, de manera que cada nodo de esta estructura consista en dos conjuntos A (*la intensidad*) y B (*la extensión*) tales que A tenga todos los atributos comunes a los documentos del nodo y B tenga todos los documentos que comparten los atributos de A . De esta manera, en un

nodo se agruparían todos los documentos que tengan un concepto en común, definido éste por la intensión del nodo -el conjunto de atributos elegido para representarlo. Dada la construcción de esta estructura, todos los sub-nodos de un nodo consistirán en sub-conceptos del concepto que éste represente.

La tarea de recuperación sencillamente consistirá, entonces, en representar la consulta como un pseudo-nodo del retículo y luego encontrar el conjunto de nodos que tenga la intensión de la la consulta, siendo los documentos más relevantes la extensión combinada de éste. Se pueden sugerir refinamientos de la consulta explorando los sub-nodos del pseudo-nodo de consulta, garantizando que el conjunto de resultados será siempre más preciso (puesto que, dada la naturaleza de la estructura, los sub-nodos son siempre más específicos que sus antecesores). El ordenamiento por relevancia de los resultados se puede hacer en función de la distancia entre los nodos de los conceptos y la magnitud de las extensiones de cada nodo.

Las ventajas de este modelo radican en la facilidad que da al usuario de ampliar o refinar sus consultas: sólo se le sugieren súper-nodos y sub-nodos de la estructura, según sea el caso; además, la magnitud del conjunto de resultados de una consulta refinada o ampliada se puede conocer antes de realizar la recuperación de los documentos, por lo que el usuario puede conocer el tamaño de este conjunto antes de llevar a cabo la consulta en sí, ahorrándose tiempo y costo computacional.

Como se puede observar, el aporte de este enfoque radica más en el mejor rendimiento a la hora de refinar o ampliar las consultas, siendo la primera búsqueda quizá menos eficiente, computacionalmente, que la de otros modelos, puesto que implica búsquedas en todo el espacio reticular.

Por otro lado, se puede utilizar el análisis formal de conceptos para construir jerarquías conceptuales a partir de la colección documental (?) y expandir la etapa de indización (?): la idea básica es mejorar la ponderación de los términos mediante la determinación de clases de sinónimos y reducción de la polisemia -o la identificación de dimensiones en el caso de los modelos generalizados de espacio vectorial.

2.1.3.7. Similitud basada en hash

Este es un modelo relativamente nuevo, al menos en la aplicación a la recuperación documental, cuya idea central es simple pero poderosa: se utilizan funciones *hash* aplicadas a los documentos y cuando existe una colisión (se asigna la misma firma *hash* a dos o más documentos) es porque éstos son similares y, por tanto, probablemente igual de relevantes a una consulta. Este método se ha probado computacionalmente más eficiente que la búsqueda en un espacio vectorial (?), pero tiene un par de debilidades: en primera, la elección de una función *hash* debe ser en extremo cuidadosa, funciones muy estrictas podrían no asignar las mismas firmas a documentos relacionados y lo contrario para funciones muy permisivas; otra debilidad radica en la premisa misma del modelo: es totalmente basado en palabras clave -puesto que la función hash es intrínsecamente léxica- por lo que es inaplicable para escenarios donde una orientación más semántica que léxica sería deseable.

2.1.3.8. Modelos probabilísticos

Mientras que los modelos basados en el espacio vectorial y otros similares se basan en medidas heurísticas para maximizar la relevancia verdadera de los documentos y así poder obtener buen rendimiento, los modelos basados en probabilidades cuentan con cierta información de relevancia preliminar que utilizan para estimar la probabilidad de que un documento sea relevante a una consulta. Esta información previa puede ser obtenida al momento de construir el índice mediante análisis estadístico similar al de otros modelos: frecuencias, co-ocurrencias o distribuciones. Estadísticamente hablando, que un documento D sea relevante (r) a una consulta Q se puede expresar con la probabilidad condicional expresada en la siguiente probabilidad, la cual se desea maximizar en un sistema de recuperación de información:

$$P(r|D, Q) \quad (2.3)$$

Si se aplica la regla de Bayes, que dice que una probabilidad depende de su inversa (en este caso, la probabilidad de que una consulta sea relevante a un documento) y se aplica la regla de cadena -que dice que se puede estimar una probabilidad conjunta una vez conocido el antecedente de una probabilidad condicional de alguno de los elementos, en el caso de la recuperación de información, que la probabilidad inversa a (2.3) se puede expresar como la probabilidad de que una consulta se dé dado un documento relevante si se sabe que el documento es verdaderamente relevante- y, finalmente, asumiendo que la probabilidad de relevancia es constante (la constante α), ya que ésta sólo puede ser 0 ó 1; se obtiene la siguiente transformación:

$$P(r|D, Q) = \frac{P(D, Q|r)P(r)}{P(D, Q)} = \frac{P(Q|D, r)P(D|r)P(r)}{P(D, Q)} = \frac{\alpha P(Q|D, r)P(r|D)}{P(D, Q)} \quad (2.4)$$

Si además decimos que en vez de maximizar la probabilidad de relevancia de un documento a una consulta queremos maximizar la razón de ésta probabilidad respecto a su negación - es decir, se maximizará la razón $P(r|D, Q)/P(\neg r|D, Q)$ y luego se asume que si un documento es irrelevante a una consulta, la probabilidad de éste es independiente de la consulta, basándonos en (2.4), tenemos la siguiente transformación:

$$\frac{P(r|D, Q)}{P(\neg r|D, Q)} = \frac{P(Q|D, r)P(r|D)}{P(Q|D, \neg r)P(\neg r|D)} = P(Q|D, r) \frac{P(r|D)}{P(\neg r|D)} \quad (2.5)$$

En donde la razón $P(r|D)/P(\neg r|D)$ representa la calidad intrínseca del documento: qué tan probable es que resulte un documento relevante a cualquier consulta, ya sea en base a su riqueza de términos discriminantes o su magnitud. Así, al maximizar la razón de relevancia de los documentos, llegamos a obtener una probabilidad proporcional a su inversa, descubriendo una relación causal entre las consultas y los documentos, lo cual permitiría construir una red de inferencia bayesiana (?) que establezca que, cuando ocurran ciertas palabras en las consultas, ciertos documentos serán relevantes.

Este modelo de inferencia asume que el orden de las palabras en un documento no es importante, sino su frecuencia, por lo que la probabilidad $P(Q|D, r)$ se puede expresar

como el producto de las probabilidades de cada palabra que conforma la consulta; como ya vimos, esto está lejos de ser verdad, pero es una condición necesaria para el desarrollo de estos modelos y usualmente da buenos resultados (?).

Las ventajas de este modelo radican en el cálculo preciso de relevancia de los documentos, así como de liberar al usuario de la preocupación de los lenguajes estructurados de consulta (dado que se puede escribir en un lenguaje natural), pero sobre-simplifican el modelo lingüístico, lo cual puede ir en detrimento en la precisión del sistema; además, requieren de estimación de las probabilidades de relevancia de los documentos, que requieren un conocimiento o procesamiento previos de la colección documental (?). Una discusión más detallada del trasfondo estadístico de estos modelos, así como la presentación de otros enfoques de naturaleza similar, se puede encontrar en ?.

2.1.3.9. Modelos lingüísticos y orientados al conocimiento

Como ya se ha observado, la recuperación de la información es, en el fondo, un problema de comunicación, y, si se toma del campo de la inteligencia artificial el concepto de *agente inteligente*, se puede aducir que, si el agente ha aprendido lo suficiente de la colección documental o el dominio del conocimiento como para poder procesar preguntas y contestarlas, un sistema de RI se reduce a la “respuesta de preguntas a partir de una base de conocimiento” (?).

Sin embargo, poder representar de manera satisfactoria y utilizable la colección documental como información para agentes inteligentes sigue siendo demasiado costoso, tanto en términos de recursos computacionales como de tiempo. De ahí que la tendencia más común en el campo de la recuperación de la información es combinar la lingüística computacional con los otros modelos: utilizar información contextual para ponderar las palabras -por ejemplo, una palabra en función de sustantivo usualmente es más relevante que la misma palabra en función de objeto indirecto; valerse de *colocaciones* (frases que son más relevantes como tales que sus componentes individuales) a la hora de indexar, desambiguación de palabras según conocimiento previo del dominio, para eliminar la polisemia, entre otras. Sin embargo, el estudio de ? respecto a este enfoque híbrido es desalentador: sistemas que lo utilizan son igual o menos eficaces que otros con métodos más ortodoxos, tal falta de ganancia se debe en gran parte a la longitud promedio de las consultas de los usuarios en motores de búsqueda comerciales: son demasiado cortas como para la inferencia de contextos y desambiguación, componentes fundamentales para el procesamiento del lenguaje natural en textos. Por otro lado, sistemas con enfoques más centrados en la lingüística computacional pura, se han implementado con cierto éxito, entre los que se cuentan el famoso sistema *SMART* del promotor del modelo de espacio vectorial, Gerard Salton; el sistema *CLARIT*, que en una etapa de análisis morfológico y sintáctico elegía palabras que fueran candidatos fuertes para ser consideradas dimensiones en el índice y el sistema *SIMPR*, que tiene un enfoque iterativo similar al de CLARIT (?).

Por su parte, un modelo estadístico basado en un modelado probabilístico del lenguaje se puede incluir aquí. Este modelo, relacionado con los modelos basados en probabilidad, computa una probabilidad distinta: trata de construir un perfil lingüístico de cada doc-

umento y luego se pregunta qué tan probable es que un documento pudiera generar una consulta, en base a este perfil; los documentos con mayor probabilidad se devolverán (?).

Un enfoque más general se encuentra en sistemas orientados al conocimiento, que utilizan reglas obtenidas del dominio del conocimiento para elegir términos discriminantes o para establecer la causalidad en el caso del uso de redes de inferencia bayesiana (como en el sistema *FIXIT*, de ?). Estos sistemas son completamente basados en el uso agentes inteligentes o sistemas expertos, por lo que se requiere del uso extenso de técnicas de ingeniería del conocimiento.

2.1.3.10. Comparación de los modelos presentados

Con el fin de ofrecer un sumario sucinto y crítico de los modelos introducidos, se ha elaborado la tabla 2.3 que también evalúa someramente cada uno en términos de la utilidad al proyecto propuesto.

2.1.4. Evaluación de un sistema de recuperación de información.

Como se hace ver en ?, los sistemas de recuperación de información se pueden medir ya sea según el sistema, es decir en base al desempeño del mismo en términos estrictamente computacionales o en base a cómo se ordenan los documentos por relevancia; o se pueden medir según el usuario: el usuario considera que un sistema de RI es útil si los resultados suplen su necesidad de información.

Es definitivamente más difícil conducir pruebas con usuarios, tanto en tiempo como en los recursos de los que se debe disponer. Ambos enfoques se discutirán en esta sección, comenzando con el del usuario, cuya valoración se mide según su satisfacción con el sistema.

Antes de entrar en consideraciones precisas, nótese que los documentos de una colección se pueden clasificar, vistos como los resultados de una búsqueda, como se ilustra en la tabla 2.4

2.1.4.1. Medidas de evaluación

Se puede evaluar un sistema de recuperación de información en términos de precisión y exhaustividad. A continuación, se presentan ambas medidas, definidas con claridad en?:

Precisión Responde a la pregunta: De los documentos encontrados ¿cuántos son relevantes a la necesidad de información?

Y se puede expresar con la siguiente ecuación, para un sistema convencional:

$$p = \frac{\text{documentos relevantes obtenidos}}{\text{documentos obtenidos}} = \frac{vp}{(vp + fp)} = P(\text{relevante}|\text{obtenido}) \quad (2.6)$$

Nótese que la precisión se puede expresar como la probabilidad de que un documento sea relevante dado que fue obtenido.

| Modelo | Orientación | Ventajas | Desventajas |
|------------------------------|---|--|--|
| Booleano | Conjuntos, coincidencia exacta, palabras clave. | Fácil implementación. Computacionalmente eficientes. | Demasiado estrictos. Requieren el aprendizaje de un lenguaje de consulta. |
| Booleano extendido | <i>ídem</i> | Concilian la fácil implementación con un enfoque más natural. | Aún requieren de un lenguaje especializado de consulta. |
| Espacio vectorial (VSM) | Coincidencia aproximada, palabras clave. | Maximiza la similitud léxica entre documentos. Automatiza la indización. Representación óptima de los documentos. | Basado en coincidencias léxicas. |
| VSM extendido | Coincidencia aproximada, conceptos. | Igual que el VSM, y además permite un tratamiento más eficiente del lenguaje natural. | Los cálculos pueden llegar a ser poco factibles computacionalmente. |
| Análisis semántico latente | Coincidencia aproximada, conceptos. | Se vale de la estructura semántica de las consultas. | La transformación de la matriz es costosa. Para colecciones muy grandes pierde el rendimiento. |
| Análisis formal de conceptos | Coincidencia aproximada, conceptos. | Permite consultas más interactivas. Aprovecha la semántica de la colección. Eficiente en términos de estimación de resultados. | La búsqueda en el espacio reticular puede llegar a ser lenta y difícil de computar. |
| Basado en Hash | Coincidencia exacta. Palabras clave. | El orden de la computación es casi lineal, muy eficiente. La búsqueda se hace en tiempo constante. | Depende de la función de hash seleccionada. Sin posibilidad de consideraciones lingüísticas. |
| Probabilístico | Coincidencia estimada. Palabras clave. | Puede integrar consideraciones expertas para las probabilidades. | Asume estricta coincidencia léxica y sobre-simplificación lingüística. |
| Lingüístico | Contexto semántico. | Reduce el problema de la recuperación de información a comunicación, el modelo ideal. | Modelos puros no son actualmente factibles. Dependiente del lenguaje. |

36
Cuadro 2.3: Cuadro comparativo de los modelos de recuperación de información

| | Relevante | No relevante |
|-------------|---------------------------|---------------------------|
| Obtenido | Positivos verdaderos (vp) | Falsos positivos (fp) |
| No obtenido | Falsos negativos (fn) | Verdaderos negativos (vn) |

Cuadro 2.4: Clasificación de los documentos en una colección, vistos como resultados de una búsqueda

Exhaustividad Mejor conocida por su nombre en inglés: *recall*. Responde a la pregunta: De los documentos *relevantes* que se encuentran en la colección ¿cuántos se han obtenido? Y también se puede expresar con una fórmula:

$$r = \frac{\text{documentos relevantes obtenidos}}{\text{documentos relevantes en la colección}} = \frac{vp}{(vp + fn)} = P(\text{obtenido}|\text{relevante}) \quad (2.7)$$

La exhaustividad también se puede expresar como una probabilidad: la probabilidad de que un documento sea obtenido dado que es relevante.

Cuando se trata de evaluar la exhaustividad, surge la pregunta ¿cómo se puede saber cuántos de los documentos en la colección son verdaderamente relevantes, dado que el concepto de relevancia es subjetivo? A esta pregunta, ? responde con el concepto de *exhaustividad normalizada*: varios investigadores hacen búsquedas en la misma colección en base a una misma necesidad de información; juzga cada quien qué documentos son relevantes y el conjunto de todos los documentos relevantes elegidos por los investigadores se considera el conjunto de recursos relevantes en la colección para futuras mediciones.

Se puede ver el problema de encontrar documentos relevantes como una tarea de clasificación: donde relevante y no-relevante son dos etiquetas y el sistema utiliza aprendizaje de máquina supervisado para la labor.

Una tercera medida de evaluación se puede introducir: la *exactitud*: de todos los documentos, ¿qué tan bien se han clasificado, de manera que la mayor parte resulten, en una consulta, o verdaderos positivos (documentos relevantes y obtenidos) o verdaderos negativos (documentos irrelevantes y no obtenidos) -en otras palabras, que la colección tenga muy pocos o ningunos falsos positivos y falsos negativos para cualquier consulta; la exactitud, entonces, se expresa de la siguiente manera:

$$e = \frac{vp + vn}{vp + vn + fp + fn} \quad (2.8)$$

Pero, como ya se vio que la relevancia es en realidad bastante subjetiva, los sistemas obtienen casi 99.9% de documentos no exactamente relevantes a una consulta (?), por lo que la exactitud, aunque buena para medir el desempeño de tareas de clasificación, es irreal para medir la recuperación de información: ¿un sistema exacto no daría ningún resultado para casi ninguna consulta!

Relación entre la precisión y la exhaustividad Se puede decir preliminarmente, entonces, que un buen sistema es preciso y exhaustivo. Pero, reflexionando un poco más, salta a la vista que la precisión y la exhaustividad no siempre van de la mano, de hecho,

en muchos de los casos hay que preferir una a la otra: esto se conoce en inglés como *the precision-recall tradeoff*. Este fenómeno será evidente con un ejemplo, adaptado de ? : imagínese a dos usuarios de un sistema de RI, uno, buscando sobre teatros famosos en el mundo para una asignación escolar: este usuario querrá obtener pocos documentos de buena calidad, para terminar su tarea con rapidez; este tipo de búsqueda se conoce como una *búsqueda de alta precisión*. El otro usuario, en cambio, es un investigador preparando su tesis de doctorado, así que querrá una gran cantidad de documentos para tener una considerable base bibliográfica que revisar; esta búsqueda será entonces una *búsqueda de alta exhaustividad*. Ambos tipos de búsqueda son extremos en un continuo: las búsquedas tienen siempre un poco de ambos tipos. Como se menciona en ?, se puede entonces ver a la exhaustividad como una función no-decreciente respecto al número de documentos obtenidos (¡se puede obtener un 100 % de exhaustividad si se devuelven todos los documentos como resultado para una consulta!) mientras que la precisión es decreciente respecto al número de documentos obtenidos (Podría tenerse una precisión del 100 % si se retornase sólo un documento muy relevante).

La relación entre ambas medidas se puede representar como un promedio, para así ver en una sola cifra qué sucede al aumentar una o la otra. Para expresar este promedio, un podría recurrir a la media aritmética (el promedio convencional) pero, como es oportunamente llamado a la atención por ?, la media aritmética sería engañosa: si obtuviésemos un 100 % de exhaustividad al retornar todos los documentos (un muy mal sistema de recuperación de información) aún así obtendríamos un 50 % de promedio -una medida que, vista con optimismo, podría sugerir que el sistema es bueno. En consecuencia, se debe recurrir a otra manera de medir el promedio que dé resultados más fidedignos, para ello, se puede pensar en el rendimiento del sistema en términos de precisión y exhaustividad como la *media armónica ponderada* de ambos, siendo la media armónica la menor de las tres medias estándar (la otra es la media geométrica). Esta media armónica ponderada se conoce como *medida F*:

$$F = \frac{1}{\alpha \frac{1}{P} + (1 - \alpha) \frac{1}{R}} = \frac{(\beta^2 + 1)PR}{\beta^2 P + R} \text{ donde } \beta^2 = \frac{1 - \alpha}{\alpha} \text{ y } \alpha \in [0, 1] \text{ (} \therefore \beta^2 \in [0, \infty] \text{)} \quad (2.9)$$

De esta manera, podemos ajustar el valor de alfa para dar prioridad a una u otra medida, por ejemplo, ajustar alfa a 0.5 (equivalente a un valor de beta al cuadrado de 1) resulta en una *medida F balanceada*: se da el mismo valor tanto a la precisión y a la exhaustividad. Por conveniencia, el valor de beta cuadrada se expresa como subíndice de la medida F en uso, así, para la medida F balanceada:

$$F_1 = \frac{2PR}{P + R} \quad (2.10)$$

Claro está que se pueden elegir otros valores en función del tipo de sistema construido: valores de beta mayores a uno corresponden a énfasis en la exhaustividad; mientras que valores de beta menores a uno, en la precisión.

Para ilustrar este intercambio entre la precisión y la exhaustividad, se suele utilizar una ayuda gráfica: la curva de *Características operativas del receptor* (ROC, por sus siglas

en inglés) : el eje de las abscisas (x) representa la tasa de falsos positivos y el de las ordenadas (y), la tasa de falsos negativos; el área bajo la curva representará la eficacia del sistema, esto según ?. Por otro lado, ? propone una curva ROC como la gráfica de la *sensitividad* (sinónimo de exhaustividad) en el eje de las ordenadas y la tasa de falsos positivos -que constituye el inverso de la *especificidad* (cuántos de los documentos no obtenidos son, en realidad, no relevantes); como paréntesis, la especificidad se define de la siguiente manera:

$$especificidad = \frac{vn}{fp + vn} \quad (2.11)$$

Así, la tasa de falsos positivos se puede ver como:

$$tasa\ falsos\ positivos = \frac{fp}{fp + tn} \quad (2.12)$$

Pero, claro, la especificidad en la mayor parte de los casos tenderá a la unidad: el número de verdaderos negativos siempre será grande; de ahí que la especificidad no sea una medida tan significativa. En términos de una curva ROC, ésta crecerá con una pendiente pronunciada en el lado izquierdo de la gráfica, es decir, que la tasa de falsos positivos será mínima y la exhaustividad crecerá tempranamente.

Así, a la hora de evaluar el sistema, se debe tener en cuenta a qué extremo del continuo de precisión-exhaustividad tenderán la mayor parte de las búsquedas. En un sistema de recomendación, se puede asumir que los usuarios no desean navegar por páginas y páginas de documentos que ellos ni siquiera buscaron explícitamente (como se verá después, una de las máximas de un sistema de recomendación es su poca interferencia con la tarea del usuario).

Otras consideraciones Las medidas que se han discutido hasta este punto son para sistemas donde los documentos resultantes para una búsqueda se presentan como un *conjunto* esto es, sin ningún orden específico. Existen otras medidas y consideraciones matemáticas que tener en mente a la hora de presentar los resultados en un orden específico, por rango de relevancia, por ejemplo, ya que otros asuntos como la efectividad del algoritmo de asignación de rango vienen a consideración. Si el lector está interesado, en profundizar en estas medidas puede consultar la sección 8.5 de ?.

(?) introduce someramente otras medidas, importantes más que todo, para motores de búsqueda: *rango recíproco*, para denotar cuáles de los resultados relevantes para el usuario se encuentran entre los primeros de la lista devuelta por el sistema y *tiempo de respuesta*, refiriéndose al tiempo total en el cual el usuario satisface su necesidad de información.

2.1.4.2. El proceso de evaluación

Para sistematizar la tarea de la evaluación, tanto desde el punto de vista del sistema como del usuario,? recomienda tener en cuenta los siguientes pasos:

1. Elegir una colección de documentos de dimensiones significativas.

2. Determinar un conjunto de necesidades de información: nótese que esto no es equivalente a un *conjunto de consultas*. La necesidad de información del usuario y cómo esta sea formulada deben considerarse distintas.
3. Obtener un conjunto de juicios de relevancia; usualmente se califica a un documento, dada una consulta, como relevante o no-relevante. Estos juicios se formarán por usuarios que evalúen manualmente los documentos.
4. A partir de los juicios de relevancia, calcular el rendimiento del sistema en términos de precisión, relevancia y cualesquiera otras medidas elegidas.
5. Según los resultados de los cálculos afinar manualmente o mediante técnicas de aprendizaje de máquina el sistema.

Algunas colecciones estándar de prueba se pueden encontrar mencionadas en ?? y los lineamientos para construir una colección para efectos de evaluación de un sistema de recuperación de información se pueden encontrar en ?. En todo caso, es necesario que las necesidades de información sean coherentes con la colección elegida.

Como es evidente, en la tarea de evaluación de un sistema desde el punto de vista del usuario se debe recurrir a consultar a verdaderos usuarios humanos, pero, ¿cómo hacer para grandes colecciones documentales? Por lo usual, se elige cierta cantidad k de documentos obtenidos con algún sistema de RI, posiblemente del que se evaluará, como subconjunto de evaluación. De este subconjunto, entonces, se obtendrán los juicios de relevancia de los usuarios.

2.2. Sistemas de Recomendación

Se introducirá el concepto general de sistemas de recomendación, las técnicas más comunes para su implementación y los métodos utilizados en su evaluación.

Un sistema de recomendación se puede ver como un caso especial de un sistema de recuperación de información de alta precisión; el quid es, sin embargo, la *recomendación*: ¿cómo un sistema puede deducir qué puede preferir el usuario de entre un conjunto de resultados?

En este punto es idónea una analogía con una situación cotidiana -la cual, dicho sea de paso, es suficientemente popular como para ser el ejemplo por antonomasia de sistemas de recomendación: imagínese el lector que desea ver una película, para ello, decide ver la cartelera del cine más cercano, y elige como opciones unas cuantas que tienen muy buena publicidad en la televisión; indeciso, llama a un amigo que usted sabe que tiene un gusto similar al suyo y le pide que le recomiende una película, su amigo, que le conoce, trata de darle un par de opciones que se adecuen más a sus preferencias; aún indeciso, cuando está en la taquilla decide preguntarle a la persona que vende los boletos por una y esta persona le recomienda la que más se ha vendido en esa tanda, la cual no está entre las que su amigo le recomendó. Al final, usted sopesa las recomendaciones y decide ver una de las que su amigo le recomendó.

Del ejemplo podemos establecer que, de una cantidad considerable de opciones que responden a su necesidad original de información, qué película ver, el sistema *filtra* aquellas que, de alguna manera, considere pueden interesarle más. El amigo y el vendedor de taquilla se pueden ver como dos enfoques distintos de recomendación: el primero filtra la información en base a lo que ya sabe de usted y sus preferencias, además de los gustos que ambos tienen en común, esto tiene su contraparte en sistemas de recomendación en la técnica conocida como *filtrado colaborativo*, donde se recomiendan objetos preferidos por usuarios similares entre sí; el segundo caso también se vale del saber colectivo, pero de una manera más general, a saber, en la *popularidad general* de un artefacto.

En suma, los sistemas de recomendación se pueden ver como *sistemas de filtrado de información* en base a cierto perfil, ya sea construido con las preferencias del usuario y el contexto en el que el usuario se desenvuelve o infiriéndolo del comportamiento de éste y otros usuarios similares.

En las siguientes secciones se presentan los dos grandes tipos de sistemas de recomendación: los *basados en contenido* y los de *filtrado colaborativo*.

2.2.1. Recomendación con filtrado basado en el contenido

Los sistemas de recomendación basados en contenido se centran en el artefacto en el cual el usuario muestre interés en el momento: ya sea un documento propio en el que está trabajando o en un artículo de una colección. Un sistema de este tipo puede construir con el tiempo un perfil del usuario en base los artículos por los cuales se ha decidido antes. O, como se define en (?):

“[Los sistemas de recomendación basados en el contenido] sugieren un artículo basado en una descripción del mismo y un perfil de los intereses del usuario. [...] Tienen en común (1) un medio para describir los elementos que se recomendarían, (2) un medio para crear un perfil que describa el tipo de artefactos que un usuario favorece y (3) un medio para comparar elementos con el perfil del usuario para determinar qué recomendar.”⁶

(?) propone una analogía basada en las bases de datos relacionales: ver a un sistema orientado al contenido como una relación entre entidades, la cual se muestra en la figura 2.4

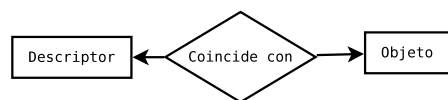


Figura 2.4: E-R representando a los sistemas orientados al contenido, adaptado de (?)

De esa relación entre entidades, se puede inferir la tabla 2.5, donde se muestran las posibles combinaciones de entradas y salidas para un sistema de filtrado de información orientado al contenido.

⁶Numeración del autor

| Entradas/Salidas | Descriptores | Objetos |
|------------------|----------------------|--|
| Objetos | Indización | Búsqueda basada en ejemplos |
| Descriptores | Tesauros asociativos | Recuperación y filtrado de información |

Cuadro 2.5: Aplicaciones de sistemas orientados al contenido, con las filas como entradas y las columnas como salidas. Adaptada de (?)

2.2.1.1. Componentes

De la definición dada por ? , los tres elementos numerados pueden encontrar su contraparte en los componentes de un sistema de recuperación de información, a saber, la *representación de los artículos* se puede asimilar al concepto de *sustitutos documentales* en la indización de la colección; el *perfil de usuario*, en papel de entrada principal al sistema, se equipararía a la *consulta* de un sistema de RI (nótese que esto refuerza lo antes mencionado como diferencia fundamental entre ambos sistemas: mientras que la consulta contiene todo lo necesario para la búsqueda, en un sistema de recomendación muchos detalles sobre la necesidad de información se deben inferir del perfil del usuario) y, por último, el componente que compara los artículos al perfil sería equivalente al *componente de igualación* de un sistema de recuperación de información.

Hecho el símil, se procede ahora a discutir los detalles y el rol de los componentes:

Representación de los artículos: Al igual que un sistema de RI, se debe elegir la representación de los artefactos que sea más idónea para la labor de recomendación; puesto que los usuarios evalúan los artículos, lo más natural sería elegir los *atributos* más conspicuos que todos los elementos tienen en común, haciendo así sencillo el almacenamiento porque se pueden ver los atributos como las columnas y cada componente como filas en una tabla. Pero para tener verdadera utilidad para el sistema, los valores de estos atributos deberían estar en un *dominio cerrado*, es decir, deberían elegirse de entre opciones previamente definidas; por ejemplo, para el atributo *género* en una película, es mucho más preferible decir “suspense” a sabiendas de que éste es uno de quizá cinco posibles valores, y no decir “película que da ganas de que no termine nunca”, porque tener demasiados atributos de texto libre vuelve inútil al sistema, o innecesariamente complicado, pues todas las ambigüedades inherentes al lenguaje natural se presentan y los análisis de similitud, fundamentales para el sistema, se vuelven difíciles de realizar. Una solución al problema de campos de texto libre es el uso de técnicas de asignación de relevancia y conversión de los términos a otras representaciones más manejables por un sistema computacional -como usar, por ejemplo, el modelo de espacio vectorial propuesto por ? y representar los campos de texto libre como vectores de términos.

Definidos los atributos a considerar y los dominios de éstos, resta el no menos importante problema de obtenerlos de los artefactos. Esta tarea es sencilla para cuando los dichos artefactos tienen una naturaleza estructurada (por ejemplo, recursos bibliográficos en formato bibtex⁷) o documentos en xml. Pero la mayor parte de documentos de texto

⁷ www.bibtex.org

tienen una estructura arbitraria o carecen de cualquier tipo de estructura; como se mencionó antes, técnicas de la indización en la recuperación de información se pueden aplicar a este tipo de entradas para indexarlas; aunque se puede también valer de técnicas de extracción de información para tratar de llenar una tabla de atributos predefinida -por ejemplo, cierto nivel de extracción de información se utilizó en el sistema de ?.

El perfil del usuario Para que un sistema orientado al contenido tenga éxito, debe *conocer* al usuario. Esto lo hace mediante el aprendizaje de las *preferencias* del usuario y el historial de las interacciones de con el sistema. La construcción del perfil se puede tratar desde el punto de vista de la clasificación: partiendo del historial de interacciones con el sistema, llevar un control de qué artículos le interesaron al usuario ya sea mediante retroalimentación explícita (asignar un juicio cualitativo de valor, por ejemplo) o implícita (comprar o utilizar un artefacto); se etiquetan los elementos -por ejemplo, como “interesantes” y “no interesantes”- y se usan estos datos para entrenar a un algoritmo clasificador. De esta manera, se construye una hipótesis de los gustos del usuario para predecir cuáles otros objetos podrían ser de interés. Métodos como Naïve-Bayes, exitosos en otras tareas de clasificación, se podrían aplicar en este tipo de sistemas de recomendación, como proponen ?, otros algoritmos de clasificación más avanzados se han propuesto para estas tareas, como RIPPER (?), árboles de decisión y k-vecinos más cercanos, clasificación en base al algoritmo de Rocchio para el modelo de espacio vectorial, el uso de clasificadores lineales -Widrow-Hoff, gradiente exponencial o máquinas de vectores de soporte- todos presentados someramente por?. Ahora bien, en vez de aprender el perfil, se puede pedir al usuario que llene formularios sobre sus preferencias (por ejemplo, www.mendeley.com, pide al usuario recién registrado que elija su área de investigación y temas de interés para recomendarle lecturas), lo cual, en la práctica, se hace poco o pobremente.

O bien, el sistema puede tener un conjunto de reglas específicas al dominio en el que funciona, las cuales, aunadas al historial del usuario, se usan para inferir las preferencias (por ejemplo, un sitio de venta de computadoras puede recomendar productos de cierta marca a usuarios que han comprado varios artículos de ella).

El componente de recomendación Dado el perfil del usuario y la representación de los elementos, se puede ver a este componente como uno que lleva a cabo una tarea de recuperación de información. Técnicas avanzadas de recuperación de información en un sistema de recomendación orientado al contenido se pueden ver en el trabajo ?; técnicas de clasificación para recomendación, utilizadas por ? y el uso del procesamiento del lenguaje natural para obtener recomendaciones donde las preferencias del usuario son escasas, mas el contenido es rico en detalles útiles, en el sistema de ?.

2.2.1.2. Limitaciones

Para ciertos tipos de artefactos, este tipo de filtrado puede mostrarse muy útil, como, por ejemplo, para entradas bien estructuradas de referencias bibliográficas, puesto que es usual que ciertos autores o citas sean reincidentes en tópicos de interés para el usuario. Sin

embargo, para datos escasos, poco estructurados o de valoraciones más subjetivas, como música o sitios web, estos sistemas se quedan atrás, al necesitar suficiente información para construir un clasificador preciso, requiriendo del usuario el uso del sistema por un tiempo considerable o llenar explícitamente un perfil detallado, cosa no factible en ciertos sistemas donde el componente de recomendación es secundario.

2.2.2. Recomendación con filtrado colaborativo.

Con el cambio de paradigma que se ha venido dando en el internet en los últimos años, las personas empiezan a esperar un ambiente más social en los sistemas que utilizan. Esto se ve reflejado en la creciente popularidad del uso de las opiniones de otros usuarios para determinar las recomendaciones en los sistemas.

Un sistema que utilice este enfoque, como lo define ?:

“Busca en un grupo grande de personas y encuentra un conjunto más pequeño cuyos gustos se asemejen a los del usuario. Mira qué otras cosas ellos favorecen y las combina en una lista de sugerencias.”

O, más sencillamente, ? expresa que un sistema de recomendación de filtrado colaborativo es:

“Es un método basado en el ambiente social, usado para proponer artículos que usuarios que piensan de manera similar prefieren.”

En consecuencia, los sistemas que utilizan filtrado colaborativo trascienden las limitaciones de los basados en contenido: se valen de la inteligencia colectiva para hacer recomendaciones y no tanto en un extenso perfil del usuario, puesto que pueden inferir en menos tiempo de uso del sistema, o con un perfil más reducido, cuáles otros usuarios opinan de manera similar al actual; las recomendaciones dependen de las evaluaciones más que del contenido de los artefactos, por lo se puede omitir la fase de análisis de contenido, la cual es computacionalmente costosa, y particularmente difícil para artefactos con multimedia y, por último, un sistema de filtrado colaborativo se orienta a la comunidad, lo cual permite que los elementos subjetivos y complejos encontrados en cualquier sociedad humana puedan tener cabida en el sistema, como asegura?.

Evidentemente, debe haber alguna manera de expresar los gustos de las personas cuantitativamente, lo cual se hace usualmente mediante valores numéricos; para sitios como www.delicious.com, se puede usar el valor de 1 para indicar que un usuario ha publicado un vínculo y 0 para el caso contrario; para sitios de películas, se podrían usar valores entre uno y cinco para indicar qué tanto disfrutó el usuario cierta película.

Incluso , pueden verse las opiniones cuantitativas de los usuarios como atributos de los elementos, permitiendo así el uso de aprendizaje de máquina para el filtrado (?).

De manera análoga a la definición dada para los sistemas orientados al contenido, en (?) también se puede encontrar una representación de los sistemas colaborativos como una relación entre entidades (figura 2.5) y las distintas combinaciones de entradas y salidas para esta relación (tabla 2.6).



Figura 2.5: E-R de los sistemas colaborativos, adaptado de ?

| Entradas/Salidas | Usuario | Objeto |
|------------------|-------------------------|---|
| Objeto | Búsqueda de expertos | <i>Filtrado colaborativo automatizado</i> |
| Usuario | Casamentero/Emparejador | <i>Filtrado colaborativo activo</i> |

Cuadro 2.6: Combinación de entradas y salidas para un sistema colaborativo, adaptada de ?

De cualquier manera, una vez en posesión de datos cuantitativos de los gustos de los usuarios, se calcula la similitud entre usuarios; las técnicas utilizadas para esto pueden ser bastante complejas: desde el uso de la distancia euclidiana (?), hasta máquinas de vectores de soporte (?). Algunos de estos algoritmos se discuten a continuación.

2.2.2.1. Métodos de filtrado colaborativo

El filtrado colaborativo se ha valido tradicionalmente de una matriz que representa las valoraciones de los usuarios respecto a los artefactos en la colección; esta matriz se conoce como *matriz de usuarios y artículos*. Por ejemplo, para un sistema de recomendación de literatura con seis libros y cuatro usuarios, se podría tener la representación mostrada en la tabla 2.7.

| Usuarios/Libros | El castillo | La caverna | Fundación | G.E.B | Hamlet | The Silmarillion |
|-----------------|-------------|------------|-----------|-------|--------|------------------|
| Luis | 1.0 | 4.5 | 4.0 | 4.9 | 5.0 | 4.5 |
| Jorge | 0.0 | 0.0 | 5.0 | 4.0 | 0.0 | 1.0 |
| Fernando | 3.0 | 3.0 | 0.0 | 4.5 | 1.0 | 2.0 |
| María | 5.0 | 3.7 | 3.0 | 0.0 | 4.5 | 4.0 |

Cuadro 2.7: Ejemplo de matriz de usuarios y artículos. Las valoraciones están en el rango [1-5], 0 equivale a no haber votado

De la matriz, si se separa en distintos vectores para las filas, se obtienen los perfiles de cada usuario (lo que juzgó de cada artículo). Tal separación es la base del primer enfoque de filtrado colaborativo: el *basado en memoria*.

Asimismo, si la matriz se separase en columnas, se obtendrían los votos de todos los usuarios para un artículo en específico, lo cual lleva al segundo enfoque: el *basado en modelos*.

Ambos enfoques pretenden *predecir* cuál sería la opinión del usuario respecto a los artículos relevantes, eligiendo así aquellos que el usuario preferiría.

Filtrado colaborativo basado en memoria

El filtrado basado en memoria trata de simular la situación cotidiana de buscar una opinión: las personas se vuelven a aquellos otros que saben que tienen gustos similares a los propios, pues encuentran de más valor esas opiniones. Para realizarlo computacionalmente, una vez existente la matriz de usuarios-artículos, se debe buscar en toda la base de datos todas aquellas filas que sean similares a la del usuario -es decir, se buscan aquellos perfiles que sean más parecidos al del usuario- y luego, en función de esto, se puede predecir la preferencia respecto a los elementos a recomendar; lo cual tiene también su ejemplo en la vida cotidiana: si sabemos que a los amigos cercanos de una persona les gusta cierto género musical, es probable que a esta persona también. Esta manera de predecir las preferencias del usuario se conoce con el nombre genérico de *cálculo del vecino* (o *k-vecinos*) *más cercano(s)*. Es de notar que el algoritmo compara el perfil del usuario con el de todos los demás, lo cual ya vaticina una labor computacionalmente exigente.

Las diferencias en implementaciones radican en la función que se use para calcular la similitud entre los distintos perfiles. A continuación se definen algunas de las funciones más comunes para calcular qué tan similares son dos vectores de usuario (las filas en la matriz):

Distancia euclidiana: Es la función más común para medir las distancias entre dos puntos. Se expresa en la siguiente fórmula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (2.13)$$

Sin embargo, la fórmula, tal como está, lo que hace es dar valores inversamente proporcionales a la similitud (esto es evidente ya que entre más similares son dos personas, más corta es la distancia). Lo que se espera, claro, es una función que dé valores *mayores* entre más grande sea la similitud, para este efecto, se puede invertir y ampliar la fórmula de esta manera:

$$\Delta = \frac{1}{1 + d} \quad (2.14)$$

Así, se puede hablar del porcentaje de similitud entre personas.

Similitud de coseno: Si se entienden los perfiles de preferencias del usuario como un vector, se puede aplicar esta técnica extensamente utilizada en el modelo de espacio vectorial de recuperación de información: si el ángulo entre los vectores se hace más corto, el ángulo de coseno se acerca a la unidad (?). Se expresa de la siguiente manera:

$$\cos \theta = \frac{a \cdot b}{\|a\| \|b\|} \quad (2.15)$$

Coeficiente de correlación de Pearson: Definida con sencillez y elegancia en (?) como

La relación entre qué tanto cambian las variables juntas e individualmente.

Lo cual se traduce en lenguaje estadístico como la razón entre la covarianza de ambas variables y la desviación estándar de cada una. Se puede escribir así, para dos usuarios i y j que han calificado un conjunto común de elementos E (adaptada de (?):

$$corr_{i,j} = \frac{\sum_{e \in E} (P_{e,i} - \bar{P}_i)(P_{e,j} - \bar{P}_j)}{\sqrt{\sum_{e \in E} (P_{e,i} - \bar{P}_i)^2} \sqrt{\sum_{e \in E} (P_{e,j} - \bar{P}_j)^2}} \quad (2.16)$$

donde $P_{e,i}$ representa la ponderación del usuario i para el elemento e y \bar{P}_i es el promedio de todas las calificaciones del usuario.

Una vez determinados los usuarios más similares, se procede a calcular cuánto le daría el usuario actual a los elementos que los otros usuarios similares han calificado. Para ello, se pueden usar dos métodos: *suma ponderada* y *regresión*.

En el método de suma ponderada, se suman los valores que los usuarios similares han dado al elemento actual, ponderándolos, usualmente, con una multiplicación, con la similitud que exista entre el usuario actual y los demás.

El método de regresión trata de ajustar una línea recta entre las ponderaciones para el elemento actual, lo cual tiene la ventaja de dar resultados más exactos, al tratar de ajustar a un mismo criterio valores que podrían estar muy distantes.

Con cualquier método, el resultado será la predicción del juicio que el usuario actual tendría sobre elementos que él no ha calificado aún; los elementos con mayor calificación consistirán en la recomendación.

Hasta el momento, el filtrado colaborativo discutido ha sido uno *orientado a los usuarios*. Sin embargo, como ya se mencionó, esto puede volverse computacionalmente difícil para grandes colecciones de elementos con un número considerable de usuarios. Una alternativa es calcular periódicamente los elementos que son similares entre sí y, a la hora de recomendar, en lugar de buscar entre todos los usuarios, sólo se busca entre todos los elementos similares y las correspondientes valoraciones. Los algoritmos involucrados son muy similares a su contraparte basada en usuarios, e implementaciones y resultados experimentales de este enfoque alternativo, así como una explicación más detallada, se pueden encontrar en ?.

Filtrado colaborativo basado en modelos

Como se discutirá en el apartado 2.2.2.2, un problema de los sistemas basados en memoria es el escaso uso del historial del usuario. Los sistemas basados en modelos tratan de superar este problema: en vez de calcular las predicciones cada vez, construyen hipótesis de *por qué* los usuarios en una vecindad de preferencias han preferido un sistema y a partir de ésta calculan las predicciones para filtrar los elementos. Técnicas de aprendizaje de máquina, como las mencionadas en el apartado 2.2.1.1 en la página 43 se pueden utilizar para ese fin. La orientación a artículos en lugar de usuarios es un componente central de este tipo de sistemas(?).

2.2.2.2. Limitaciones

La orientación a la comunidad es un arma de doble filo: si bien la mayor parte de las recomendaciones son exactas, el sistema se basa en datos que son intrínsecamente

inexactos: las opiniones de personas.

Las limitaciones más comunes, presentadas por (??), para este tipo de sistemas son:

Dispersión: Los sistemas implementados para grandes colecciones de productos probablemente tengan una gran cantidad de productos que han sido evaluados por pocos usuarios, resultando así en recomendaciones pobres para éstos.

Escalabilidad: Como se notó antes, los algoritmos convencionales para el cálculo de recomendaciones son computacionalmente costosos, lo cual puede tener un impacto significativo cuando la colección de artículos y usuarios crezca.

Aprendizaje: En los sistemas basados en memoria, nada se aprende del usuario más allá del momento en el que se le recomienda, por lo que cada vez que el usuario utilice el sistema, recibirá recomendaciones genéricas que no aprovechan en realidad su historial de consumo.

Preferencias inusuales: para usuarios que no tienen similitud con otros, las recomendaciones serían también pobres, dado que las funciones de predicción dependen de la existencia de más usuarios con preferencias similares.

Sobre-especialización: artículos de uso más común que otros tendrán más evaluaciones.

Arranque en frío: Sucede cuando el sistema aún no ha podido construir una matriz de usuarios-artículos que provea suficiente información. Esto se puede resolver con algoritmos heurísticos, como proponen ?.

2.2.3. Recomendación orientada al conocimiento

Los dos grandes enfoques discutidos antes son los más usados a nivel comercial, pero ambos tienen una desventaja fundamental: requieren que se logre obtener suficiente información de los usuarios o artículos para comenzar a dar recomendaciones útiles, lo cual es algo difícil de superar: sin suficientes usuarios, no se tienen muchas evaluaciones de artículos, y, sin recomendaciones de artículos, no se logra conseguir una base suficiente de usuarios. Para superar este problema, se ha propuesto un tercer tipo de sistemas de recomendación: los *orientados al conocimiento* -introducidos en (?). Éstos tratan de *razonar* la motivación que subyace a la elección de cierto artefacto por el usuario, determinando qué atributos, y qué relaciones entre éstos, han influenciado la decisión para así determinar otros artefactos que podrían ser de interés al usuario. Nótese la diferencia entre estos sistemas y los orientados al contenido: mientras que los de contenido infieren un patrón de preferencias del usuario, los orientados al conocimiento no sólo saben del usuario, sino que también de los artefactos, y la combinación de estas dos fuentes de conocimientos permite un razonamiento (y no sólo una conjetura, como lo derivado del mero aprendizaje de máquina) de las necesidades del usuario. Este enfoque se deriva del *razonamiento basado en casos*: dado un problema, se utiliza una base de conocimiento de problemas anteriores para asimilar el problema nuevo con algunos precedentes y derivar una solución similar a la de éstos.

Pero, a diferencia de los otros, estos sistemas requieren una interacción más estrecha con el usuario: luego de las recomendaciones preliminares, ofrecen opciones implícitas de retroalimentación al usuario, como darle más valor a un atributo o ignorar otro, para poder así construir un modelo de conocimiento a medida que se navega entre las sugerencias, convirtiéndose también en agentes de navegación asistida. Como la intromisión está orientada a recabar datos cada vez más precisos del usuario a la vez que ofrece mejores opciones a éste, se compensa con el aporte de cada vez mejores sugerencias.

Otra desventaja de estos sistemas es que requieren del uso de técnicas avanzadas de ingeniería del conocimiento, convirtiéndolos en una opción arquitecturalmente costosa, además de exigir un uso exclusivo, lo cual es indeseable para implementaciones que requieran de poca intrusión en la actividad principal del usuario.

2.2.4. Sistemas de recomendación híbridos

Luego de examinar los dos enfoques convencionales, el orientado a contenido y el colaborativo, se puede notar que ambos tienen sus fortalezas y debilidades: los orientados al contenido son exactos, pero requieren un uso extenso y no toman en cuenta el aspecto social del uso de sistemas basados en internet; por otro lado, los colaborativos pierden exactitud a cambio de la implementación del saber colectivo de los usuarios, además de su característico costo computacional. Sería ideal, entonces, poder combinar la exactitud y eficiencia de los sistemas orientados al contenido con la inteligencia colectiva conseguida mediante un enfoque colaborativo.

2.2.4.1. Sistemas de recomendación colaborativos y orientados al contenido

? resume las combinaciones de ambos enfoques en cuatro tipos:

1. Combinar sistemas de recomendación separados: consiste en tomar las listas de sistemas distintos que utilicen cualquier enfoque y luego ordenarlas mediante otras técnicas de ordenamiento por relevancia.
2. Agregar elementos orientados al contenido a sistemas colaborativos: determinando similitud entre usuarios mediante sus perfiles y no en base a artefactos en común; aunque esto requiere que se recabe una base histórica aún mayor, permite recomendaciones personalizadas.
3. Agregar características colaborativas al enfoque orientado a contenido: consiste en representar los perfiles de usuarios como vectores de términos y luego usar técnicas de agrupamiento para asimilarlos.
4. Unificar en un solo sistema: esto incluye las dos técnicas anteriores en una implementación diseñada desde un inicio para este efecto, un ejemplo de un sistema unificado se puede encontrar en el sistema *Fab* de ?.

Agregar características de orientación al contenido a un sistema colaborativo se hace tradicionalmente agregando reglas relacionadas con el contenido del dominio a la matriz

de usuarios-artículos -por ejemplo, si un artículo tiene mala ortografía, la regla puede decidir darle una puntuación desfavorable; aunque esto resuelve el problema de la dispersión y el arranque en frío -que artículos nuevos no son evaluados, ni recomendados, por carecer de evaluación- al hacer que las reglas sean los primeros evaluadores de los elementos. ⁸, sin embargo, ofrece una alternativa: en vez de *combinar* las técnicas, propone *unirlas*, en vez de agregar una a la otra⁸.

Para ilustrar esta unificación, se vale de las definiciones alternas para ambos sistemas -mencionadas en las secciones 2.2.1 y 2.2.2 para hacer una unión de la representación de entidades, que se presenta en la figura 2.6y que ofrece las aplicaciones ilustradas en la tabla 2.8; en ésta, al contar con todos los datos de ambos enfoques en un sistema unificado, se puede ir más allá del filtrado de información, pero la columna pertinente al tópico de la recomendación es la central: nótese que resulta en un filtrado colaborativo orientado a objetos y a usuarios a la vez que permite contar con recuperación de información en base a perfiles.

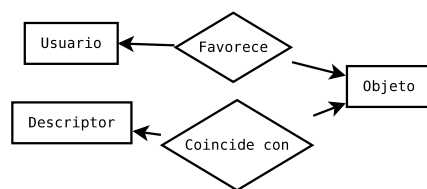


Figura 2.6: E-R Híbrido resultante de la combinación de sistemas colaborativos y orientados al contenido. Adaptado de ?

2.2.4.2. Sistemas de recomendación colaborativos y orientados al conocimiento

Aunque los sistemas orientados al conocimiento pueden encontrar recomendaciones más adecuadas, carecen del importante elemento social de los sistemas colaborativos, filtrando menos elementos que éstos, al carecer del contexto que la comunidad aporta. Por ello(?) propuso, mediante una tabla como la 2.9, un híbrido entre estos dos enfoques.

Estos híbridos, entonces, paliarían el problema de la poca atención a las tendencias entre varios usuarios, ofreciendo recomendaciones exactas mediante la base del conocimiento y filtradas mediante lo que otros usuarios han hecho con los artefactos del conjunto de resultados, dotando a los sistemas basados en el conocimiento de la alta precisión y la orientación social de la que inherentemente carecen.

2.2.5. Evaluación de un sistema de recomendación

Se puede evaluar un sistema de recomendación en base a los resultados estimados mediante alguna técnica estadística contra los resultados obtenidos, determinando así si los conjuntos de resultados obtenidos por el sistema cumplen con ciertos requerimientos

⁸ *Join*, el término en inglés para el concepto de unión en b.d. relacionales quizá clarifique el sentido de la frase

| E/S | Descriptor | Objeto | Usuario |
|------------|-------------------------|--|---|
| Usuario | Exportación de perfiles | Filtrado colaborativo activo | Emparejador |
| Objeto | Indización | Búsqueda mediante ejemplos Filtrado colaborativo automatizado | Búsqueda de expertos |
| Descriptor | Tesauro | Recuperación y filtrado de información | Búsqueda de expertos Importación de perfiles |

Cuadro 2.8: Aplicaciones de un sistema híbrido de recomendación, adaptada de ?. E/S equivale a “Entradas/Salidas”

cuantitativos y cualitativos estáticos. Pero existen otras consideraciones a medir, como el nivel de confianza del usuario en el sistema y qué tanta utilidad tienen los resultados para él. Esta distinción es hecha en ? cuando califica al primer tipo de mediciones como *fuera de línea* y a las postreras, como *en línea*.

2.2.5.1. Medidas de evaluación *fuera de línea*

Un componente importante de un sistema de recomendación es el que obtiene de la colección de artefactos aquellos que se filtrarán para luego sugerirlos; como ya se ha mencionado, este componente es, en realidad, un sistema de recuperación de información, de ahí que parte de la evaluación se pueda hacer en términos de precisión y exhaustividad del subconjunto de resultados. Además, si se tiene en mente que las recomendaciones tienen algún sistema de rango para el filtrado, medidas como la curva ROC, la precisión media, la *precisión-R* y la de *once puntos*, resumidas en ? y explicadas con mayor detalle en ? pueden ser aplicadas. Es de enfatizar que lo que se mide con la curva ROC es la relación entre precisión y exhaustividad, pero una característica interesante de los sistemas de filtrado de información es que podrían sugerir más elementos de los que un usuario encontraría verdaderamente útiles; para normalizar esta situación, se puede introducir una restricción en la presentación de resultados: sólo mostrar una cantidad constante de recomendaciones a todos los usuarios, las cuales consistirían en las k mejores sugerencias. Un sistema donde se ha introducido tal restricción se puede medir con una curva CROC (curva ROC de cliente), como sugiere ?.

Asimismo medidas de evaluación de aprendizaje de máquina en sistemas orientados

| Técnica | Ventajas | Desventajas |
|---------------------------|---|--|
| Orientada al conocimiento | A. No se necesita mucha información inicial. B. Retroalimentación detallada y cualitativa de preferencias. C. Sensible a la variación a corto plazo. | H. Ingeniería del conocimiento I. La habilidad de sugerir es estática |
| Filtrado colaborativo | D. Identifica nichos con precisión. E. No se necesita conocimiento específico al dominio. F. La calidad mejora con el tiempo. G. Las recomendaciones son personalizadas. | J. La calidad depende de los datos históricos K. Sujeta a anomalías estadísticas L. Reacciona lentamente a la variación a corto plazo. |
| <i>Híbrido Ideal</i> | A, B, C, D, F, G | H |

Cuadro 2.9: Comparación entre sistemas colaborativos y orientados al conocimiento.

al contenido se podrían aplicar, como la exactitud⁹, el afinamiento mediante matrices de confusión o la validación cruzada (más sobre estos métodos de valoración se puede encontrar en los capítulos sobre aprendizaje de máquina de ?).

Y para los sistemas de colaboración, vistos como un problema de regresión, se podrían usar medidas de correlación entre los resultados esperados y los estimados, así como otros cálculos de error, como ser la media cuadrática. En ? se examinan a fondo las métricas antes mencionadas para la medida de este tipo de sistemas, específicamente los de filtrado colaborativo.

De las anteriores consideraciones se pueden destacar tres medidas extensamente utilizadas en la investigación de la evaluación de sistemas de recomendación:

Exactitud: que mide qué tan bien clasifica un sistema (o predice) comparando los resultados obtenidos con resultados estimados o evaluados por usuarios.

Cobertura: que mide qué tanto de la colección puede ser sugerido por el sistema, puesto que muchos de los artículos podrían no ser nunca evaluados o no ser semánticamente relevantes a ningún perfil de usuario.

Tasa de aprendizaje: qué tan rápido pueden aprender los algoritmos la cantidad suficiente de información como para empezar a dar recomendaciones significativas.

⁹definida en la sección 2.1.4.1

Una de las mayores limitaciones de este tipo de análisis de desempeño es el sesgo en el conjunto de resultados a evaluar: usualmente, los usuarios sólo han calificado lo que ellos favorecen, y no dan retroalimentación explícita a aquellos artefactos que no encuentran útiles. Se puede mencionar también en contra de las métricas puramente estadísticas que ignoran la *utilidad* de los resultados: aunque ciertos conjuntos puedan significar alta precisión o exactitud, quizá no sean -por ser muy obvios- de gran aporte al usuario como observan?, esto se relaciona con los conceptos de *novedad* y *serendipia*: si el artículo sugerido es uno que no se le hubiera ocurrido, por no conocer -novedad- o ignorar su utilidad para el contexto presente -serendipia, al usuario.

2.2.5.2. Medidas de evaluación en línea

Además de las limitaciones mencionadas, y aunque el análisis puramente sistemático del desempeño de un sistema de recomendación pueda ser suficiente en ciertos casos, el papel del usuario no puede ser dejado de lado del todo, por ello, en la evaluación de su sistema ? se centran en la satisfacción del usuario utilizando dos evaluaciones:

Satisfacción del usuario respecto al sistema: se refiere a la interacción del usuario con el sistema mismo (mejor dicho, con la interfaz de usuario), se proponen cinco medidas:

1. Facilidad de uso.
2. Comprensibilidad de las funciones del sistema.
3. Fiabilidad.
4. Facilidad del aprendizaje de uso.
5. Tiempo de respuesta.

Satisfacción del usuario respecto a la información: o qué tan satisfecho está el usuario con los resultados que el sistema le presenta. Así como en el aspecto anterior, se presentan cinco medidas, respecto a las recomendaciones como tales:

1. Qué tanto de los requerimientos del usuario se ve cubierto.
2. Confiabilidad.
3. No-ambigüedad.
4. Precisión.
5. Inteligibilidad.

Asimismo, se puede argüir que una evaluación fuera de línea puede estar ignorando tanto el contexto de uso del usuario como el nivel de confianza que le tiene al sistema (si las recomendaciones le son útiles y le instan a continuar utilizando el sistema), de ahí que un marco de evaluación en línea basado en la comparación directa de rendimiento entre dos métodos de recomendación y la retroalimentación de usuarios reales sea propuesto por ?:

En primera, la evaluación se ha de llevar a cabo en una aplicación en línea utilizada por usuarios reales, en la cual los dos motores de recomendación distintos se utilizan

mediante la misma interfaz de usuario, de manera que la satisfacción del usuario respecto al sistema se mantenga constante -ya que el usuario final ignora cuál de los sistemas está utilizando- puesto que, en última instancia, se desea medir solamente el desempeño de las recomendaciones. Para ello se propone una arquitectura -ilustrada en la figura 2.7- que defina:

Los recursos disponibles: una interfaz de aplicación (API) definiendo los recursos a los cuales los sistemas tendrán acceso. Nótese que es importante normalizar esto, puesto que los sistemas orientados al contenido utilizan datos semánticos de los recursos y los de filtrado utilizan datos de evaluaciones de usuarios; esto se puede normalizar mediante el uso de un lenguaje común para representar y estructurar los datos.

Los métodos de sistemas de recomendación: otra interfaz, que defina qué métodos deberá implementar cada sistema.

Una política de presentación: definiendo cómo mostrar los resultados al usuario. Puesto que éstos pueden ser una combinación de las sugerencias de los dos métodos puestos a prueba, se sugieren tres alternativas:

1. Un conjunto combinado de los resultados de ambos.
2. Un conjunto contrastado, mostrando los dos conjuntos de resultados claramente separados -de manera que se sepa que provienen de fuentes distintas, pero sin decir cuál ha dado qué conjunto.
3. Conjuntos en cascada, mostrando, cuando un usuario pide recomendaciones, los resultados de un sistema y, cuando pida recomendaciones de nuevo, los del otro.

Lineamientos de retroalimentación: cuáles acciones del usuario, implícitas o explícitas, se considerarán un signo de preferencia

Una métrica de comparación: mediante la cual analiza la retroalimentación y determinar un ganador.

Téngase en cuenta que, las condiciones en una evaluación de dos métodos debe ser lo más justa posible: técnicas que requieran una sola interacción, como los sistemas colaborativos convencionales, deberían ser comparadas con otras que también requieran una sola interacción; mientras que otros enfoques, como ciertas implementaciones de recomendaciones basadas en el conocimiento -las cuales son más *conversacionales*- deberían ser evaluadas frente a otras de su clase. De igual manera, debe también tenerse en cuenta que existen métodos que requieren un tiempo de adaptación o aprendizaje, como la mayor parte de implementaciones orientadas al contenido o de filtrado colaborativo.

Como conclusión, parece conveniente citar lo que se menciona en ?:

“Un sistema [de recomendación] es útil en la medida en que ayude a un usuario a llevar a cabo sus tareas.”

Por ello, se deben elegir las medidas y experimentos de evaluación más acordes al propósito del sistema.

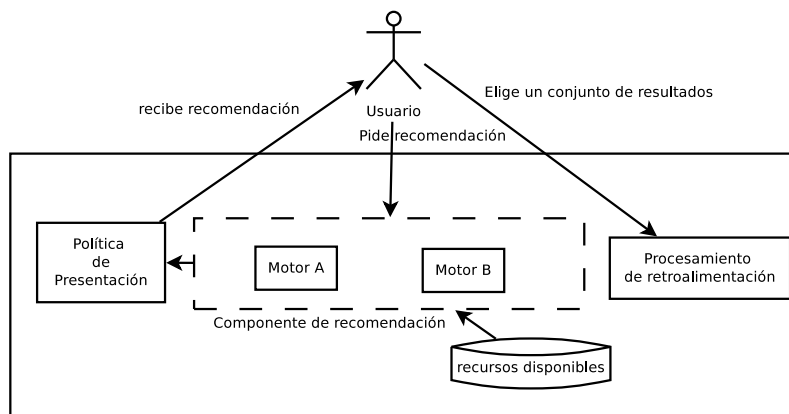


Figura 2.7: Comparación en línea de sistemas de recomendación. Adaptada de ?

2.3. Sistemas de recuperación oportuna de la información

Relativamente poca investigación se ha hecho respecto a sistemas de recomendación que asistan recuperando información útil a la tarea del usuario sin obstruirle. Pero poco no es ninguno, y, así, podemos encontrar una definición de este tipo de sistemas en la tesis que sentó las bases de este campo, la de ?:

“Un agente de recuperación oportuna de información es un programa que proactivamente encuentra y presenta información basada en el contexto local de una persona de una manera fácilmente accesible y a la vez no intrusiva.”

Los agentes descritos en la tesis antes mencionada reciben ya un nombre propio: sistemas de recuperación oportuna de información (en inglés, *JITIRS: Just-in-time Information Retrieval Systems*). Nótese que, como se había establecido antes, la principal utilidad de un sistema de esta especie es encontrar la necesidad latente de información: presentarle información al usuario cuando éste ni siquiera haya pensado en buscarla, o ni supiese que ésta existía.

Un agente de recuperación oportuna de información difiere de un sistema de recuperación de información tradicional en que la búsqueda no es la tarea principal, las consultas son potencialmente más largas -el contexto local del usuario, por el cual entiéndase cualquier contexto, desde lo que está escribiendo en un procesador de texto hasta el lugar geográfico donde está; y que la principal medida de evaluación de un sistema así es la *utilidad* que tienen los documentos -contrario a los sistemas de recuperación de información, que se miden por la relevancia, expresada en la precisión y exhaustividad, de sus resultados.

Asimismo, ?, establece tres criterios a tener en cuenta a la hora de desarrollar un *JITIR*:

¿Cómo afecta el sistema la manera en la cual una persona busca y usa información?

Tomando la teoría de la ciencia cognitiva, se establece que el sistema debería poder

permitir la serendipia: encontrar información que resulta útil y de la cual no se sabía; y poder reducir el costo: encontrar información de la cual se sabía, pero se consideraba demasiado el esfuerzo que hubiera implicado buscarla por cuenta propia.

¿Cómo puede el sistema encontrar información que resultaría útil en base al contexto?

Es en este punto donde entra en juego la recuperación de información: se ha de elegir un modelo que trascienda la mera relevancia para poder ofrecer documentos que sean de utilidad -lo cual se puede aproximar utilizando técnicas de alta precisión y trayendo a colación el filtrado: basarse en el contenido del contexto actual aunado a las preferencias del usuario o de un grupo de usuarios para eliminar documentos que se consideren, aunque relevantes, inútiles. Enfoques orientados al conocimiento del dominio específico pueden ser de utilidad, por ejemplo, en el sistema *FIXIT* ? se valieron de la existencia de una red de inferencia para el sistema principal -un sistema experto de reparación de artículos de oficina- para poder usar esas mismas reglas en la recuperación de documentación útil a la tarea.

¿Cómo debería el sistema presentar los resultados? El diseño de la interfaz debería ser tal que los resultados sean fácilmente accesibles y evaluables, pero que no interfiera con la tarea principal; así, no debería distraer completamente de ésta pero la atención debería poder dividirse entre ambas interfaces. Es práctica común en el diseño de sitios web el ubicar los elementos más importantes a la izquierda, por lo que, en principio, se podrían ubicar los resultados más a la derecha del lugar donde se realice la tarea principal, práctica adoptada por el sistema *PIRA* (?). Asimismo, se ha probado útil ofrecer un sumario del documento o una vista previa en caso de que el artefacto consista de multimedia (?).

Nótese que un sistema de recomendación proactivo no es más que un sistema de recomendación fuertemente orientado al contenido que se adapta al contexto del usuario y ofrece sugerencias sin que éste haya elegido antes otros artículos de la colección. El proyecto *À propos* (?) es un ejemplo de un sistema de este tipo: asiste a escritores construyendo consultas en base a lo que están escribiendo en el momento, las cuales alimenta a distintos motores de búsqueda; por otro lado, *Scienstein* (?), otro sistema que se puede considerar de este tipo, se vale también de las citas que el autor ha hecho para recomendar materiales relacionados, además de incorporar técnicas de filtrado de información para presentar los resultados.

2.3.1. Determinación del contexto para sistemas basados en texto

Según parece, la capacidad del sistema para determinar el contexto local del usuario es fundamental para que un agente *JITIR* sea de verdadera utilidad. Aunque para sistemas basados en contextos extraordinarios, como aquellos que se pudieran basar en información geográfica, la información va más allá de lo que se introduzca en una aplicación, para sistemas orientados a tareas puramente documentales, asumir que todo el contexto necesario se encuentra en el texto introducido por el usuario es suficiente.

Pero ¿cómo determinar, de lo que el usuario escribe, qué se puede considerar importante en la identificación del contexto y la construcción de las consultas para la búsqueda de documentos a recomendar? Esta pregunta parece encontrar su respuesta en un área que, aunque poco conocida, es vital

En primer lugar uno podría argumentar que, dado algún saber previo respecto del usuario o del dominio del conocimiento en el que se desenvuelve, se podrían extraer términos discriminantes del contexto local y con éstos realizar las consultas, ya sea con técnicas de aprendizaje de máquina -por ejemplo, ?, propone el uso de un clasificador bayesiano- o, si se dispone de una colección de documentos en el contexto local o el perfil del usuario, se pueden tomar prestadas técnicas de la recuperación de información: el modelo de vector espacial (?), ontologías construidas con el análisis formal de conceptos (?) o la aplicación del modelo de análisis de semántica latente para determinar los términos más definitivos y hasta se puede disponer de la lingüística computacional, relacionando la importancia de una palabra con su rol gramatical. Una comparación entre los enfoques estadísticos y lingüísticos para la extracción de términos clave se puede encontrar en ?.

Y cuando el conocimiento previo es nulo o escaso, un enfoque lingüístico parece cobrar más importancia, según el trabajo de ?, quienes demuestran un algoritmo que determina palabras clave con un rendimiento comparable al de la ponderación $tf*idf$ (véase 2.1.3.3).

Existen servicios basados en web para la extracción de términos clave, entre los cuales, uno de los más populares es el *Term Extraction Web Service*¹⁰, utilizado por el sistema *PIRA* (?). Asimismo, existen librerías que implementan esta funcionalidad mediante la identificación de roles gramaticales (una implementación en el lenguaje de programación python se encuentra disponible en <http://pypi.python.org/pypi/topia.termextract/>).

2.4. Construcción de perfiles de usuario

De este tema ya se habló cuando se discutieron los sistemas de recomendación (sección 2.2), pero se considera pertinente ahondar un poco más en el tema, dada la naturaleza de los sistemas que se discuten aquí. En este apartado se mostrará cómo la construcción de un perfil de usuario puede ayudar a la tarea de la recuperación de información, con miras a aportar al área de la recuperación oportuna de la información.

El área de investigación de modelado de perfiles de usuario no es nueva, de hecho, cualquier sistema se crea con un tipo de usuario en mente. ? distingue tres maneras de tratar el problema del perfil de usuario:

1. El enfoque *canónico*. En el cual el diseñador del sistema crea el mismo para un solo tipo de usuario estándar, este prototipo se puede definir arbitrariamente o mediante estudios de grupos de usuarios reales.
2. El enfoque *explícito*. En el cual el usuario tiene la responsabilidad de responder preguntas o ajustar parámetros para definir su perfil, en base a esto, el sistema se

¹⁰ <http://developer.yahoo.com/search/content/V1/termExtraction.html>

adaptará a él. Este enfoque es famoso en sistemas de recomendación comerciales o redes sociales.

3. El enfoque *automático*. En el cual el sistema se basa en las interacciones con el usuario para inferir el perfil. Una primera manera de implementar este enfoque es tener categorías de usuarios preexistentes (como novato, intermedio, y experto, por ejemplo) y luego utilizar tales interacciones para *clasificar* al usuario en la taxonomía dada. Otra manera, la que es relativamente nueva en la investigación, es la de considerar el perfil individual como una base de conocimiento que puede evolucionar y que regirá la interfaz entre usuario y sistema. Es este último enfoque, el de considerar al perfil en sí como la directriz principal para el comportamiento del sistema el que se definirá y evaluará en el resto de esta sección.

2.4.1. Definiciones preliminares

Antes de entrar en materia, es necesario definir ciertos conceptos:

Perfil: de manera general, el *perfil* de un usuario se define como las circunstancias que tienen influencia en su necesidad de información a la hora de utilizar un sistema de RI y en el consiguiente juicio de los resultados de búsquedas. En la literatura, los términos *perfil*, *contexto* y *situación* se pueden utilizar indistintamente; sin embargo, en este trabajo se entenderá el perfil como la representación de los intereses a largo y mediano plazo del usuario. El perfil puede consistir de múltiples *dimensiones*, las principales, identificadas por ?, son:

1. Información Personal: la parte estática del perfil, aquí se encontrarán datos tales como país, lenguaje de preferencia, edad, etc.
2. Dominio de interés: es la dimensión más explorada en la investigación; consiste en una representación de los intereses del usuario, por consiguiente, esta dimensión es dinámica.
3. Preferencias de presentación: los atributos que el usuario favorece en la interfaz (como los colores en los perfiles de algunas redes sociales) o los menús que se muestran, en el caso de muchas aplicaciones de escritorio.
4. Historial de interacciones: toda información recolectada a través de la retroalimentación del usuario, podría utilizarse para construir el dominio de interés.
5. Calidad esperada: alguna medida cuantitativa de la relevancia intrínseca de los resultados; un usuario profesional podría preferir sólo documentos debidamente publicados y revisados mientras que otro prefiera actualizaciones de *blogs* y redes sociales.
6. Seguridad: o las preferencias de privacidad respecto a las otras dimensiones.

Contexto: igual que el perfil, el contexto engloba las circunstancias de la “intención del usuario para búsqueda de información” (?). Pero a diferencia del perfil, el contexto solamente se refiere a la *sesión actual* de uso del sistema, por lo que representa los intereses a corto plazo.

Sesión: se refiere a cada vez que el usuario utilice el sistema para satisfacer una necesidad de información, puede incluir cualquier cantidad de reformulaciones de la consulta, siempre y cuando la tarea tenga siempre la misma finalidad.

Ontología: “es una especificación explícita de conceptos y las relaciones que puedan existir entre ellos” ?. En muchas implementaciones prácticas, esto se modela mediante una relación jerárquica en forma de un grafo dirigido acíclico o un árbol, de manera que los conceptos puedan ser sub-conceptos de otros -mediante al relación de herencia “es un”.

2.4.2. La definición del perfil

Como se ha notado, los intereses a largo plazo de un usuario son los que permitirán a un sistema de recuperación de información adaptarse a éste y retornar resultados cuya medida de relevancia sea cada vez menos genérica. De ahí que la manera en que el perfil se represente y manipule sea fundamental. ? identifican tres etapas para la consecución de esta tarea:

Representación la manera en que se representen internamente las preferencias a largo plazo del usuario tendrá un impacto directo en el sistema, puesto que es la base de cualquier otra operación. La manera más difundida de representación es la vectorial, heredada el modelo de espacio vectorial. Ahora bien, esta representación puede consistir en un solo vector con cada término clave ponderado en base a las interacciones del usuario o en varios vectores (o *clases de vectores*) no relacionadas u organizadas jerárquicamente que representen los distintos dominios de interés del usuario. La representación vectorial es utilizada por ? y las clases de vectores se implementan en el sistema de ?. Otra manera de representar un perfil es semánticamente: con el uso de ontologías, para así modelar las relaciones entre los distintos conceptos -y ya no palabras clave. Las ontologías se pueden construir a partir de la colección documental, pero los sistemas que han implementado este enfoque se valen de taxonomías de conceptos ya existentes, como el *open directory project* (??), wikipedia (?) o wordnet (?). Un último enfoque utiliza varias dimensiones del perfil, como el modelo de seis dimensiones propuesto por ? o el de dos dimensiones -los intereses y el historial de interacciones- implementado por ?.

Construcción esta etapa se ocupa de la instanciación del perfil en una sesión y de cómo se obtendrá la información que servirá para actualizarlo. En prácticamente todas los sistemas que utilizan modelos de perfil, estos datos se obtienen de la retroalimentación del usuario (en la mayoría de los casos, implícita). Esta retroalimentación se puede tratar mediante el algoritmo de Rocchio o usarse de entrada para un algoritmo de clasificación, con el fin de mejorar los resultados en la sesión actual (?).

Evolución esta etapa es la que utiliza la información obtenida de sesiones de uso para adaptar el perfil a los intereses cambiantes del usuario; y es esta misma etapa la más íntimamente ligada a la representación elegida. En los enfoques semánticos,

los conceptos se pueden ponderar mediante *puntajes de preferencia*, actualizando éstos después de cada sesión, como en el trabajo de [?] o con el uso de ponderaciones que “envejecen”, como lo proponen [?]. En el caso de representaciones vectoriales, se puede utilizar un enfoque estadístico y actualizar las probabilidades de relevancia de los documentos en relación a los términos y los intereses que éstos representan, como en el enfoque orientado a diagramas de influencia de [?].

2.4.3. La utilización del perfil

Se pueden determinar tres enfoques en la recuperación y presentación de la información utilizando los perfiles de usuario. El primero, requiere dos etapas: primero, se obtienen los resultados de un agente externo de recuperación de información con la consulta que el usuario mismo ha construido y luego, los documentos se filtran (o reordenan). Este enfoque en dos pasos ha sido implementado por [?] y [?]. La ventaja de este modelo radica en poder valerse de servicios externos o implementaciones de algoritmos de recuperación de información, concentrándose más bien en la etapa de filtrado; la desventaja estriba en el costo computacional: para contextos complejos, la determinación de la consulta podría volverse computacionalmente costosa y esto, aunado al tiempo que toma la búsqueda en sí, podría tornarse indeseable. Para resolver este obstáculo, se puede optar por sólo considerar una cantidad fija de los resultados más relevantes devueltos y de éstos sólo tomar en cuenta los resúmenes que el agente devuelve, y no todo el documento.

Por otro lado, se puede hacer la recuperación de información en una sola etapa: reescribir la consulta del usuario, como en el sistema propuesto por [?] o agregar el perfil al espacio documental, construyendo así el sistema de RI con la personalización incluida desde un principio, como propone [?] al representar los documentos como clases sinónimicas y no como términos individuales, o se puede seguir lo establecido por [?] e incluir las probabilidades entre términos, preferencias y documentos en la evaluación de relevancia. Este enfoque es en realidad, equivalente a un sistema de recomendación orientado al contenido. Con este enfoque se gana en costo computacional, pero se requiere la implementación completa del componente de búsqueda e indización.

Por último, se puede sencillamente utilizar el perfil para encontrar usuarios similares al presente y evaluar los resultados en base a los perfiles de éstos, como se encuentra documentado en el trabajo de [?]. Como es evidente, este enfoque equivale a los sistemas de recomendación colaborativos.

2.5. Servicios e interfaces de programación de aplicaciones web (API)

Una interfaz de programación de aplicaciones se puede definir como:

“Un conjunto de funciones que el programador de aplicaciones puede usar, indicándose los parámetros que hay que pasar a cada función y los valores de retorno que el programador puede esperar”[?].

En el ámbito de los sistemas web, sin embargo, el rol de funciones lo juegan URLs y los parámetros son los que se envían en solicitudes POST o GET. De esta manera, una llamada a función en el concepto tradicional de API se convierte en una solicitud HTTP POST o GET.

Un *servicio web* está definido por el consorcio de la *world wide web* (w3) como¹¹:

“Un sistema de software identificado por una URI [identificador uniforme de recursos], cuyas interfaces públicas y referencias son definidas y descritas utilizando XML [lenguaje de marcas extensible]. Su definición puede ser descubierta por otros sistemas de software. Estos sistemas pueden entonces interactuar con el servicio web de la manera prescrita por su definición utilizando mensajes en XML transmitidos por protocolos de internet ”

Esta definición implica que, dada la definición XML de un servicio, los desarrolladores de aplicaciones pueden formar solicitudes a la misma y realizar acciones sobre datos provistos por el servicio web. Una manera muy popular de implementar servicios web siguiendo este concepto es el protocolo estándar *SOAP*¹² (Simple Object Access Protocol, o Protocolo Simple de Acceso a Objetos). Los implementadores de servicio utilizan el formato *WDSL*¹³ (*Web Services Description Language*, Lenguaje de descripción de servicios web) para establecer las acciones realizables en el XML que guía la comunicación. Asimismo, se utiliza el registro *UDDI*¹⁴ (*Universal Description, Discovery and Integration*, Descripción universal, descubrimiento e integración) para poner un servicio web públicamente disponible. Pero, aunque los servicios web más robustos aún se benefician de los estándares de la industria fundamentados en SOAP, WDSL y UDDI, la posible complejidad de la interpretación de documentos XML y la dificultad de desarrollo conformaron un obstáculo para los implementadores de servicios web simples, hasta que la mayor aceptación de la arquitectura REST, el formato de intercambio de datos basado en texto JSON y la aparición del objeto `XmlHttpRequest` para el lenguaje de programación ECMAScript (con su dialecto más popular, Javascript) permitieron una mayor accesibilidad de desarrollo e implementación. Estos conceptos se definen con mayor detalle en las secciones siguientes, concluyendo con una descripción del estado del arte en la implementación de interfaces de programación de aplicaciones web.

2.5.1. La técnica de arquitectura REST

El estilo de arquitectura REST (de *Representational-State Transfer*, Transferencia de Estado Representacional), propuesta por ?, hace uso de las operaciones básicas del protocolo HTTP (PUT, POST, GET y DELETE), las URI de recursos del servicio y la meta-data estándar del protocolo (como los encabezados HTTP) para la comunicación entre un cliente y el servidor. Esta técnica prescribe seis restricciones:

¹¹<http://www.w3.org/TR/wsa-reqs/#id2604831>

¹²Definido en detalle por el w3 en <http://www.w3.org/TR/soap12-part0/>

¹³<http://www.w3.org/TR/wsdl>

¹⁴http://uddi.org/pubs/ProgrammersAPI_v2.htm

Distinción Cliente-Servidor: permite que el cliente pueda ignorar cómo se almacenan o procesan los datos en el lado del servidor, permitiendo la escalabilidad y la evolución independiente de los componentes.

Sin estado: lo cual quiere decir que cada solicitud de un cliente al servidor debe contener en sí toda la información necesaria para llevarse a cabo, sin depender de datos previos; esto permite que se incrementen la visibilidad, confiabilidad y escalabilidad, mas introduce la desventaja de tener que enviar datos que quizá sean irrelevantes o redundantes en una serie de solicitudes.

Cache: puesto que las comunicaciones entre un cliente y un servidor se basan en las operaciones y los identificadores propios de HTTP, se puede hacer uso de la capacidad del protocolo mismo de almacenar datos devueltos para una solicitud y evitar comunicaciones redundantes; esto, claro, tiene la desventaja de que el cliente pueda quedarse con datos no actualizados.

Interfaz uniforme: permite la simplificación y el bajo acoplamiento en la arquitectura. Esta, a su vez, introduce cuatro restricciones en el diseño de las interfaces:

Identificación de recursos: los recursos son identificados mediante su URI y transferidos en un formato de hipermedia estándar (JSON, XML o simple HTML).

Manipulación de recursos mediante representaciones: los datos en las solicitudes y respuestas HTTP deben ser suficientes para la modificación de los recursos.

Hipermedia como motor del estado representacional: los recursos relacionados deberían poder descubrirse exclusivamente mediante sus identificadores (URI) en las respuestas del servidor.

Mensajes auto-descriptivos: cada mensaje contiene en sí suficiente información para describir cómo procesarlo.

Sistema en capas: requiere que los sistemas se diseñen de manera tal que las funcionalidades de una capa superior sean independientes a la implementación de las inferiores, permitiendo que los componentes puedan cambiarse sin problemas. Tiene como desventaja, sin embargo, la adición del costo operativo resultante de la transmisión de datos entre capas.

Code-on-demand (código a petición): es una restricción opcional que establece que, mientras un cliente lo soporte, funcionalidad extra podría ser provista por el servidor para ejecución en la forma de *applets* de Java o *scripts* de Javascript.

2.5.2. JSON y AJAX

El lenguaje de programación Javascript se basa en una manera simple de representación de objetos: mediante mapas. Esta simplicidad llevó a que se propusiera¹⁵ un nuevo formato de intercambio de datos textuales: el *JavaScript Object Notation* (Notación de

¹⁵<http://www.ietf.org/rfc/rfc4627.txt>

objetos de javascript, JSON, por sus siglas en inglés). De esta manera, se introdujo una alternativa de representación y transferencia para datos con estructuras simples donde el beneficio de utilizar una notación más organizada como XML es menor al costo que implica su transferencia e interpretación.

Por otro lado, con la creciente necesidad de dinamicidad en los sitios web (los cuales, en principio, son simples documentos estáticos), la introducción del objeto estándar XmlHttpRequest¹⁶ soportado nativamente por los navegadores permitió que solicitudes se pudieran hacer de manera asíncrona, sin tener que re-cargar el documento, dando la impresión de dinamicidad y permitiendo que el concepto de procesos en trasfondo, popular en aplicaciones de escritorio, se popularizase entre las aplicaciones web; esta técnica se conoce como AJAX (*Asynchronous Javascript and XML*). Esta comunicación asíncrona entre clientes y servidores se vio enriquecida, a su vez, por el formato JSON.

Pero, aunque gracias a la técnica AJAX las aplicaciones pueden comunicarse con el servidor de manera más dinámica, la transferencia de datos en el formato JSON -como cualquier tipo de transferencia de datos a nivel de código de cliente- sólo es permitida por los navegadores web dentro del mismo origen (restricción conocida como *same origin policy*¹⁷, por lo que la manipulación u obtención de datos de terceros a este nivel es, en principio, imposible. Sin embargo, dadas la popularidad y la fácil accesibilidad de cada vez más interfaces de programación de aplicaciones que enriquecen el contenido o funcionalidad de un sitio, se ha propuesto que se haga uso del hecho que la transferencia de *scripts* entre distintos orígenes sí es posible y que el formato JSON se puede representar como una cadena de caracteres para introducir una técnica conocida como JSONP¹⁸ (*JSON with padding*, o JSON con relleno) que consiste en la transferencia de un *script* que, es, a su vez, la llamada a una función en el lado del cliente con los datos en formato JSON como parámetro.

2.5.3. Estado del arte de los servicios web

Con la introducción de la arquitectura REST, el formato de intercambio JSON y técnicas como AJAX y JSONP, muchas aplicaciones web actuales están introduciendo interfaces de programación de aplicaciones que permitan a los desarrolladores hacer uso de sus servicios mediante simples solicitudes sobre HTTP (y quizá en este punto aparece la sinonimia contemporánea entre servicios web y APIs web, como se explicó anteriormente). Y las APIs más populares¹⁹ implementan la arquitectura REST en el sentido que los datos se pueden encontrar y manipular exclusivamente mediante solicitudes y respuestas HTTP dejando atrás protocolos como SOAP y, en algunos casos, el manejo de archivos XML al devolver los datos en formato JSON, además del hecho que la implementación de APIs como estas permite también toda la escalabilidad inherente a la arquitectura

¹⁶<http://www.w3.org/TR/XMLHttpRequest/>

¹⁷https://developer.mozilla.org/En/Same_origin_policy_for_JavaScript

¹⁸Propuesta en <http://bob.pythonmac.org/archives/2005/12/05/remote-json-jsonp/> y también utilizada en la API para desarrolladores de yahoo (<http://developer.yahoo.com/common/json.html#callbackparam>)

¹⁹Por ejemplo, las interfaces de búsqueda de yahoo (<http://developer.yahoo.com/search/>) y google (<http://code.google.com/intl/es/apis/ajaxsearch/>)

REST. Cabe notar que estos servicios web conocidos como “RESTful” (por implementar la arquitectura REST) no son del todo fieles a la misma, pues pocos fuerzan las implicaciones semánticas del uso de las operaciones POST, GET, PUT o DELETE y, de hecho, POST y GET pueden utilizarse indistintamente (aún cuando GET sólo debería utilizarse para obtener recursos y POST para actualizarlos).

2.6. Diseño de interfaces

La investigación en el diseño de interfaces se ha convertido en todo un campo de estudio, trascendiendo lineamientos empíricos y tomando elementos de la psicología cognitiva para estudiar las experiencias de usuario. Así, aunque aspectos subjetivos como la creatividad y el buen gusto del diseñador aún juegan un papel crucial, es menester tener en cuenta el efecto psicológico del diseño de una interfaz en cómo el usuario percibe la utilidad del sistema, factor indiscutiblemente decisivo a la hora de evaluar éste.

2.6.1. Aspectos psicológicos del diseño de interfaces

Antes de ahondar de las consideraciones relacionadas a la división de atención y experiencia de usuario en la siguiente sección, se cubrirán aquí un par de aspectos generales.

2.6.1.1. La ley de Fitt

La ley de Fitt, propuesta por ²⁰, establece que:

“El tiempo para adquirir un objetivo es una función de la distancia a la que se halla y de su tamaño”.

Expresada en la siguiente fórmula:

$$t = a + b * \log_2\left(\frac{D}{S + 1}\right) \quad (2.17)$$

Donde D representa la distancia entre el punto inicial del movimiento y S es la anchura del objetivo. Establecido esto, esta ley se puede simplificar y decir que, para hacer la navegación más fácil, se habrán de poner los elementos más cerca entre sí o hacer los elementos utilizables más grandes. Y aunque esto parezca obvio, muchas veces el inverso es ignorado²⁰: elementos que no están relacionados o que no se desea que se utilicen con demasiada frecuencia (como botones de borrado, por ejemplo) deberían estar más lejos o ser más pequeños que los que se utilizan más o cumplen funciones críticas. De la misma manera, se puede hacer notar un corolario de esta ley: los elementos en los bordes se pueden considerar de tamaño infinito hacia el lado del borde²¹, por lo que se deben aprovechar los bordes naturales de la pantalla de presentación en interfaces de software.

²⁰<http://www.codinghorror.com/blog/2010/03/the-opposite-of-fitts-law.html>

²¹<http://www.codinghorror.com/blog/2006/08/fitts-law-and-infinite-width.html>

2.6.1.2. El principio del menor esfuerzo

El principio del menor esfuerzo, aplicado al comportamiento humano por G. K. Zipf en 1949, dicta que

“Una persona resolviendo sus problemas inmediatos los verá con sus problemas futuros, estimados por sí mismo, como trasfondo. De manera que tratará de manera que tratará de resolver sus problemas de forma que minimice el tiempo total utilizado en resolver éstos y los futuros”

Así, una persona utilizará cualquier herramienta que le permita minimizar el tiempo que dedica a una tarea, aquí entra la *ley de los dos segundos*, hecha notar por ?, a partir del corolario que establece que “incrementar el esfuerzo involucrado en llevar a cabo una tarea, causará un decremento proporcional en el número de veces que ésta se lleva a cabo” prescribe que, si una herramienta presenta una ganancia *al menos de dos segundos* por sobre otras, será elegida como mejor alternativa. De esta manera, una interfaz de asistencia a la tarea principal de un usuario debe poder presentar la información con mayor rapidez que otras opciones, como el uso directo de motores de búsqueda, por ejemplo.

2.6.2. Interfaces para sistemas de asistencia al usuario

En el caso concreto de los sistemas del tipo propuesto por este proyecto (sistemas de recuperación oportuna de información), la tesis de doctorado de ? estudia a fondo la experiencia del usuario y cómo el diseño de la interfaz la afecta, y asegura que:

“Los *JITIRs* deben permitir a una persona concentrar su atención en su tarea primaria y a la vez dejar que la atención pueda ser dividida entre la tarea primaria y las recomendaciones.”

Para que esta aseveración sea realizable, se debe encontrar un equilibrio entre la *concentración de la atención* del usuario y presentar la interfaz de recomendaciones de una manera que permita distinguirla del ambiente de la tarea primaria o ignorarla cuando ya no se desee o necesite información; y, a la vez, la interfaz no debe ser tan disímil como para exigir un cambio mental de contexto tan grande que la tarea principal sea dejada de lado.

Para que este equilibrio sea conseguido, la transición entre la tarea principal y la revisión y uso de recomendaciones debe ser lo menos brusca posible.

Desde un punto de vista estético, esto se puede conseguir aprovechando el hecho de que *los usuarios exploran las páginas web desde arriba a la izquierda hasta abajo a la derecha* (?). Así, teniendo en cuenta que la interfaz de recomendación debe ser tanto accesible como no intrusiva, se recomienda ubicarla a la derecha del área de trabajo principal. También, se sugiere elegir colores en el rango cromático de la interfaz principal, de manera que se sepa que está relacionada y que, al mismo tiempo, no se distraiga la atención por ser demasiado distinta.

Desde el punto de vista funcional, ? sugiere el uso de *interfaces incrementales*. En las cuales:

“Cada etapa provee un poco más de información, al costo de requerir un poco más de atención para leerla y entenderla”

La idea de una interfaz de esta naturaleza es que permita al usuario decidir si vale la pena dividir su atención entre su tarea principal y las recomendaciones, reduciendo así el riesgo de afectar la concentración del usuario en vano. El mismo autor (?) muestra el ejemplo de seis etapas en una interfaz incremental en su sistema *Margin notes*, las cuales se reproducen a continuación

1. *Ninguna Acción*: el agente asistente decide qué hacer (no mostrar la interfaz o librarla de elementos distraentes).
2. *Existen recomendaciones*: la interfaz se muestra en la periferia.
3. *Gráfico de relevancia*: la interfaz muestra un histograma de los resultados.
4. *Tópicos*: El usuario puede ver los tópicos detectados por el asistente.
5. *Palabras Clave*: Para cada resultado, el usuario puede ver las palabras clave que hicieron que se diera.
6. *Visualizar el recurso*: El usuario decide leer la recomendación

Además de la interfaz incremental para este tipo de sistemas, el diseñador de interfaces habrá de tener en cuenta lineamientos generales de eficiencia de uso, como los siguientes (tomados de ?):

1. *Reducir el esfuerzo*: puesto que el propósito de un sistema de este tipo es asistir al usuario, es inconsecuente que el uso del mismo suponga una distracción considerable de la tarea principal.
2. *Facilidad de aprendizaje*: lo cual se logra siguiendo los estándares impuestos por sistemas similares familiares al usuario.
3. *Reducción de latencia*: dependiente de la implementación, debe estar a la altura de las expectativas del usuario en cuanto a tiempo de respuesta.
4. *Legibilidad y accesibilidad*: donde la elección de colores, fuentes y dimensiones entra en juego, de manera que la interfaz sea lo más *usable* posible.
5. *Comunicación*: dando retroalimentación al usuario donde sea necesario, de modo que éste sepa que la interfaz funciona correctamente.

3 Planificación

En este capítulo se definen las prácticas de gestión de proyecto aplicadas al presente: se identifican el ámbito y los recursos, se detalla la metodología de administración de proyectos detrás de la calendarización y se concluye con una estimación post-desarrollo. La estimación tardía se debe a la restricción de tiempo impuesta desde un principio: el proyecto sólo contaba con tres meses para el desarrollo, de manera que una estimación previa hubiera sido hasta cierto punto fútil.

3.1. Definición del ámbito e identificación de los recursos

Como hace notar ?, el primer paso en la planificación de proyectos de software es la definición del ámbito, el cual:

“Define las funciones y características que se entregarán a los usuarios finales, los datos que son entrada y salida, el ‘contenido’ que se presenta a los usuarios como consecuencia de emplear el software, así como el desempeño, las restricciones, las interfaces y la confiabilidad que *acotan* el sistema”

El autor debe recalcar que esta sección se ha escrito luego de completar el ciclo de desarrollo, por lo que a continuación se describe el ámbito actual -y no el inicial-, de ahí el nivel de detalle. Una descripción del ámbito inicial sería redundante. Siguiendo, pues, la práctica de narrar el ámbito de la aplicación, el de este proyecto se puede establecer de la siguiente manera:

Los implementadores obtienen una clave única de aplicación, así como credenciales de autenticación, al registrarse en el sitio del servicio. Acto seguido, pueden optar por registrar usuarios manualmente en el *panel de control* ofrecido en el mismo sitio del servicio, o utilizar llamadas a la *interfaz de programación de aplicaciones* para registrar usuarios individualmente o en grupo. Puesto que una adición múltiple de usuarios puede ser costosa, respecto al tiempo, los implementadores pueden optar por utilizar una llamada asíncrona al servicio, la cual confirmará mediante correo electrónico que los usuarios fueron agregados. En cualquier caso, para estas operaciones, los implementadores deberán autenticarse utilizando las credenciales provistas: clave de aplicación, nombre de usuario y contraseña. El servicio contestará mediante HTTP con una respuesta con código 401 si la autenticación falla o con una respuesta con código 200 y cuerpo codificado en JSON con un mensaje de éxito si la agregación resulta o fracaso si hubo un error interno o se excedió el límite de usuarios permitido: por cuestiones de almacenamiento, el servicio

establecerá un límite de usuarios a las aplicaciones que lo utilicen. Asimismo, el servicio ofrece una llamada para eliminar usuarios (que se comporta igual que la de agregación, incluso en cuanto a la necesidad de autenticación) y otra para ver los usuarios que se han registrado, para ésta, sin embargo, sólo la clave de aplicación es necesaria.

Los implementadores, a la hora de utilizar el servicio, pueden optar por utilizar el *widget* de javascript que se ofrece -sólo si la aplicación es web, evidentemente; si deciden utilizar éste, sólo habrán de agregarlo a la página donde aplicarán el servicio y proveer al constructor del *widget* las opciones que se deseen. Si no desean utilizar el widget -o no pueden, por no ser web la aplicación- deberán implementar métodos que hagan las llamadas al servicio de recomendación y lo interpreten: para obtener recomendaciones, la llamada correspondiente deberá incluir la clave de aplicación y el contexto local en forma de texto; el servidor contestará con una respuesta 404 (página no encontrada) si la clave provista no es válida o con una respuesta 403 (solicitud inválida) si, en caso de ser una llamada *AJAX*, la función de retorno (para utilizar la técnica JSONP) no es un identificador javascript válido. En cualquier otro caso, el servidor contestará con una respuesta con código 200 y un contenido codificado en JSON con un mensaje de error si el usuario no existe o el contenido no se provee o con un conjunto de términos clave encontrados y recomendaciones (incluyendo las *url* de la hipermedia, un resumen, código para retroalimentación y la relevancia) si la solicitud es correcta. Asimismo, se provee la llamada a actualizar el perfil, que se comporta de manera similar a la de obtener recomendaciones, mas recibe un parámetro más: los códigos de documentos elegidos como retroalimentación. La actualización del perfil es asíncrona, por lo que la llamada a esta función sólo devuelve que la tarea de actualización fue puesta en la cola de prioridad, y no necesariamente garantiza la actualización; esto se debe a que el proceso de evolución de perfil es relativamente largo y un procesamiento síncrono ralentizaría innecesariamente la aplicación implementadora (dado que es un componente secundario de la misma).

Establecido el ámbito, se proceden a identificar los recursos, una vez más, siguiendo los lineamientos de gestión de software presentados por ?:

Personal: Dada la naturaleza de este proyecto, solamente se cuenta con el autor para el desarrollo de software. Para la configuración inicial de recursos externos (como el servidor web y de base de datos), se habrá de solicitar apoyo a la facultad o el encargado del laboratorio asignado a los estudiantes de ingeniería en sistemas computacionales.

Entorno: este se detallará más adelante (entre los requisitos no funcionales de la etapa de análisis), mas se puede establecer que se requerirá de equipos con un sistema operativo linux tanto para desarrollo como para producción; asimismo, se requerirá de al menos 10 Gb de almacenamiento y al menos 1 Gb de memoria de acceso

aleatorio debido a la cantidad de datos en forma de texto libre a ser procesados. Por último, se requerirá de conexión a internet y suficiente ancho de banda como para procesar con el mínimo de latencia las solicitudes al servicio.

Software reusable: para el proyecto se utilizará un *framework* de desarrollo web que se encargue de tareas de bajo nivel como el direccionamiento de *urls* y la interpretación y codificación de la comunicación HTTP, luego, se requerirá software pre-existente para las siguientes tareas: indexación y búsqueda de documentos para recomendación, entrenamiento del clasificador de categorías y extracción de términos clave. Se deberá desarrollar software nuevo para el manejo de solicitudes al servicio, la implementación del widget, la obtención de la colección de software documental, el registro de aplicaciones, la creación, evolución y uso de los perfiles y la administración de las tareas de extracción automática, búsqueda, y gestión de los datos.

3.2. Calendarización

Como ya se apuntó, la restricción de tiempo estaba fija, de modo que lo que se describe a continuación se hizo con el lapso de tres meses en mente.

La metodología utilizada para la administración del proyecto fue una adaptación ágil a la tradicional (que incluye, como se describe en ?, las siguientes etapas: definición, planificación, iniciación, control y finalización o cierre). Como ya se explicó, se prefirió seguir un proceso evolutivo asimilable al desarrollo conducido por características, con iteraciones rápidas, cambios de los requerimientos y pruebas a nivel de característica. En cuanto a la etapa de planificación, sin embargo, se siguieron los lineamientos establecidos en el ya citado libro de ?. No obstante, la planificación, se debe confesar, se hizo a grandes rasgos (a nivel de objetivos e hitos) dejando los detalles *sobre la marcha*, debido a la naturaleza evolutiva de la metodología empírica que se tomó. A continuación, las etapas de la planificación:

División del trabajo: se consideraron los objetivos y se subdividieron en cuatro grandes *características* relativamente independientes entre sí y capaces de ser sujetas a pruebas, así como la inclusión de un período de pruebas formales y generales:

1. Obtención de colección documental y ontología
2. Implementación del componente de búsqueda
3. Implementación del componente de perfilado
4. Implementación del servicio
5. Pruebas generales

Estimación de tiempos: esta fue quizá la etapa más liberalmente ejecutada del proyecto: para cada característica se consideró un máximo de dos semanas, de modo que se pudieran cumplir los tres meses (dejando un par de semanas para cualquier contingencia). Tratando de que las tareas incluidas dentro de cada característica no excediesen una semana en su ejecución.

| Aplicación | Líneas de código | Esfuerzo (persona- meses) | Tiempo (meses) | Desarrolladores | Costo |
|--------------|------------------|---------------------------------|-------------------|-----------------|------------|
| Araña Web | 251 | 0.56 | 2.01 | 0.28 | \$ 944 |
| Servicio Web | 2,712 | 6.84 | 5.19 | 1.32 | \$ 11, 494 |

Cuadro 3.1: Estimación post-desarrollo

Identificación de hitos: la culminación y pruebas superficiales de cada característica se consideraron hitos.

Encadenamiento de actividades y planificación temporal: una vez identificados los hitos y las tareas necesarias para llevar a cabo cada objetivo, se utilizó un diagrama de Gantt para identificar qué tareas dependían de otras y así distribuir el tiempo de los tres meses de desarrollo.

Replanificación sobre la marcha: conforme se avanzó en el desarrollo, el diagrama se fue refinando conforme las tareas se cumplían -dado que muchas se lograron antes de lo estimado y otras después, dando como resultado el diagrama que se presenta en el anexo B.

3.3. Estimación

Para estimar el esfuerzo y recursos que se hubieran necesitado, se utilizó el modelo empírico COCOMO II¹ con un factor y exponente ajustados al perfil de un proyecto *orgánico*² aplicado con la herramienta *SLOCcount*³, una utilidad de línea de comandos para el sistema operativo linux. Se generaron dos paquetes independientes de software: *dmoz_ontology*, una araña web, utilizada para obtener y organizar la colección documental y *lebrixen*, el servicio en sí. El costo de personal se estableció en 8400 dólares anuales (700 mensuales), en base a entrevistas laborales que el autor tuvo en el año 2009. En la tabla 3.1 se presentan los resultados de la estimación.

¹http://sunset.usc.edu/csse/research/COCOMOII/cocomo_main.html

²Como se define en <http://www.dwheeler.com/sloccount/sloccount.html#cocomo>: “un pequeño grupo de programadores experimentados desarrollan software en un entorno familiar. El tamaño del software varía desde unos pocos miles de líneas (tamaño pequeño) a unas decenas de miles (medio)”

³<http://www.dwheeler.com/sloccount/>

4 Análisis y Diseño.

En este capítulo se documentan los distintos artefactos producidos a la hora de analizar y diseñar el sistema propuesto en este proyecto. Por cuestiones cronológicas, ambas etapas se han agrupado aquí, mereciendo la etapa de implementación y evaluación un capítulo aparte. Como se imaginará, el sistema no consiste en un sistema de información a gran escala, con una larga lista de requerimientos, casos de uso y escenarios de interacción; sin embargo, ninguna empresa de ingeniería debe hacerse sin ningún tipo de definición previa del derrotero a seguir.

El desarrollo clásico de sistemas de información es un proceso lineal que incluye seis etapas: compilación de requisitos, análisis, diseño, implementación, mantenimiento y retiro. Sin embargo, este proceso ideal ha sido desplazado en los últimos años por un enfoque iterativo: *el proceso unificado*. Éste consiste en cuatro fases: iniciación, elaboración, construcción y transición. En cada fase se llevan a cabo, en mayor o menor medida, los flujos de trabajo de revisión de requisitos, análisis, diseño, implementación y pruebas (?). Este enfoque iterativo es mucho más realista y efectivo que el enfoque tradicional, pero requiere de una gran cantidad de tiempo y una organización tales que se puede probar demasiado costoso para proyectos pequeños o de corta duración, características típicas del desarrollo web, que es la clase de desarrollo a llevar a cabo en este proyecto. Por ello, una metodología adaptada al desarrollo de aplicaciones web de este modelo iterativo, como la propuesta en ? y adaptada a la metodología ágil de orientación a las características se utilizará en el presente proyecto

4.1. Análisis

En esta etapa se definirán la motivación, alcance, audiencia y contribución del sistema desarrollado. El producto final de esta fase será un resumen de los requisitos funcionales y no funcionales, en la metodología elegida, la identificación de estos se hace en cuatro etapas: los análisis de contenido, de interacción, de función y de configuración. Por mor del orden, y para fundamentar sólidamente las etapas subsiguientes, cada análisis se discutirá brevemente en los apartados siguientes, finalizando con el listado de requerimientos. Antes de proseguir, sin embargo, se contestarán tres preguntas que servirán de trasfondo a estos análisis:

1. ¿Cuál es la principal motivación para la aplicación? Siendo la aplicación la implementación de un proyecto de apoyo a la producción de información, la principal fuerza tras el desarrollo de la misma es determinar si un sistema que pueda recomendar recursos útiles a un usuario que lleva a cabo una tarea de documentación

es verdaderamente una ganancia para los usuarios, tanto los autores de la información, que lo utilizarán directamente, como los consumidores que se beneficiarán de la calidad de la misma.

2. ¿Cuáles son los objetivos que debe satisfacer la aplicación? En suma, la aplicación debe trascender los sistemas de recuperación oportuna de la información equivalentes al basarse en una representación más fidedigna del contexto local del usuario, así como en la construcción y evolución de perfiles para evitar la ambigüedad en las recomendaciones.
3. ¿Quién utilizará la aplicación? Cualquier usuario del sistema anfitrión que emprenda una actividad de producción de información.

4.1.1. Sumario de requisitos

En base al objetivo y la audiencia del proyecto, se resumen a continuación los requisitos a tener en las siguientes etapas del análisis y diseño, redactados y organizados siguiendo las pautas propuestas en ?.

4.1.1.1. Requisitos funcionales

1. El sistema deberá tratar lo que el usuario escriba en la interfaz de documentación como el contexto local, representándolo como los demás documentos de la colección, con el fin de realizar una búsqueda.
2. El sistema deberá mantener un perfil del usuario con el fin de tener éste como base contextual para las recomendaciones.
3. El sistema deberá determinar cuándo es el momento idóneo de iniciar la recomendación o hacer una nueva, ya sea en base al tiempo pasado desde el inicio de la sesión, la cantidad de información aportada por el usuario o la diferencia entre el último contexto inmediato que creó una búsqueda y el actual.
4. El sistema deberá realizar una búsqueda y filtrado de documentos en base a las representaciones que haya hecho del contexto del usuario, entendiéndose por contexto tanto la base contextual provista por el perfil como el contexto inmediato de la tarea presente.
5. El sistema deberá determinar cuándo se utiliza un documento recomendado, con el fin de tomar esta acción como retroalimentación y así enriquecer el perfil de la base contextual.
6. El sistema deberá ser capaz de ofrecer recomendaciones y actualizar perfiles de usuario de la manera más flexible posible, sin obligar al implementador a usar el servicio más allá de su propósito primordial.

4.1.1.2. Requisitos no funcionales

1. El sistema deberá permitir reflejar la separación entre el sistema anfitrión, el que extrae los datos de entrada y presenta los de salida y el componente principal, el que realiza las búsquedas y filtrado e indexa y mantiene la colección documental. Tal separación se logrará mediante una interfaz de programación de aplicaciones que reciba y devuelva los datos en un formato serializado (json, de preferencia, o xml).
2. El sistema, en el lado del servidor, estará escrito en el lenguaje de programación python. En específico, se utilizará el *framework* de desarrollo web django¹.
3. El sistema, en el lado del cliente, estará escrito en el lenguaje de programación javascript, con la librería jquery².
4. El sistema deberá apegarse a los estándares de diseño de w3³, utilizando (x)html y css válidos.
5. El sistema utilizará, para el almacenamiento, el servidor de bases de datos postgresql⁴.
6. El sistema funcionará en un servidor Linux.
7. El sistema deberá devolver los primeros resultados de las recomendaciones en menos de diez segundos, tal cantidad de tiempo es aceptable, dada la naturaleza secundaria y no intrusiva del sistema (?).
8. El uso del sistema deberá estar libre de estado: toda la información necesaria para el uso del servicio deberá estar contenida en la solicitud, permitiendo así la escalabilidad mediante sistemas distribuidos.

4.1.2. Casos de uso

Se resumen a continuación los distintos casos de uso del sistema. Nótese que la escasez de éstos se debe a que la mayor funcionalidad del sistema radica en el trasfondo algorítmico de la indización, búsqueda y filtrado. Un diagrama UML que los ilustra se presenta en la figura 4.1.

Solicita Registro: el implementador llena un formulario de registro para el uso del sistema, donde provee la información de su aplicación y un correo electrónico de contacto válido, a la vez se valida que la solicitud sea hecha por un ser humano. El sistema responderá con un correo electrónico conteniendo la llave única correspondiente a la aplicación solicitante.

¹www.djangoproject.com

²<http://jquery.com/>

³<http://www.w3.org/>

⁴<http://www.postgresql.org/>

Obtiene interfaz: si el implementador opta por utilizar la interfaz provista por el servicio, se devuelve ésta serializada y configurada con los parámetros adecuados. El solicitante debe ser estar debidamente registrado como usuario del servicio. La interfaz, dado que es un agregado a la interfaz propia del sistema implementador, será conocida indistintamente como *widget* en el resto de este documento.

Solicita Recomendaciones: El sistema implementador provee el usuario y el contexto inmediato, el cual se analiza y se sintetiza en una consulta al componente de búsqueda, luego, el componente de filtrado analiza y pondera las recomendaciones en base al perfil del usuario y éstas son devueltas en un formato serializado de hipermedia. Nótese que, si el usuario es nuevo, se habrá de crear su perfil.

Actualiza Perfil: El sistema implementador provee el usuario, el último contexto inmediato y, opcionalmente, los documentos elegidos mediante retroalimentación; en base a esto, el servicio actualiza el perfil del usuario con las categorías/conceptos presentes en el contexto.

Provee Retroalimentación: el usuario final decide utilizar un recurso recomendado. En el caso de la interfaz provista por el servicio, es necesario tomar éste en cuenta a la hora de actualizar el perfil; el implementador también debería tener esto en cuenta, pero ello es responsabilidad exclusiva de él.

Registra/Elimina Usuarios: el implementador opta por agregar uno o varios usuarios al servicio (o eliminarlos). En cualquier caso, debe proveer el identificador único del usuario y sus credenciales de autenticación. El servicio responderá con un mensaje adecuado de fallo o éxito.

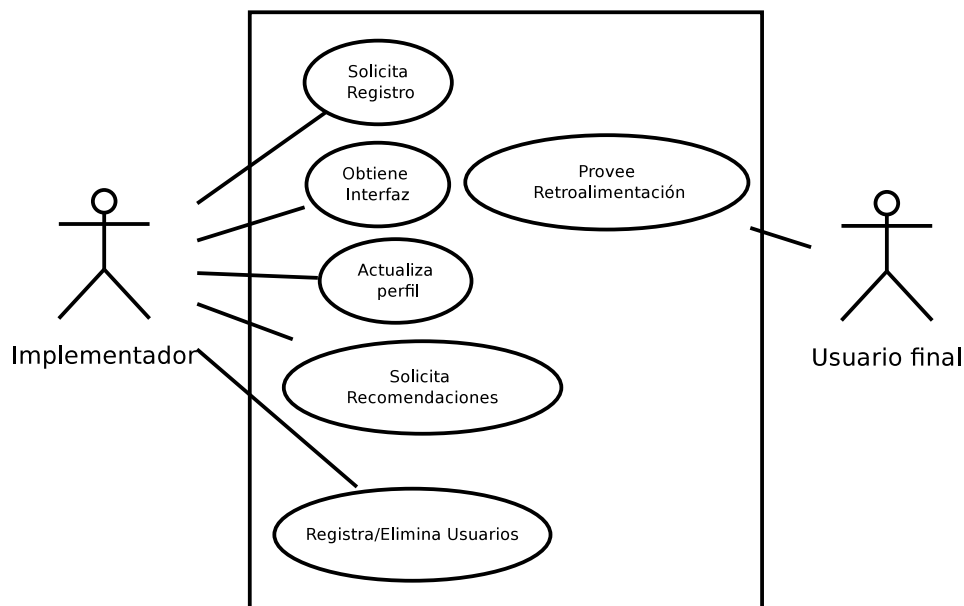


Figura 4.1: Diagrama de casos de uso

4.1.3. Análisis de contenido

4.1.3.1. Objetos del contenido

Como se puede deducir, el único contenido que el sistema mostrará al usuario son las representaciones de los documentos (en inglés, *code snippets*). Éstas consistirán en el título del documento, un sumario -de ser posible- y un enlace para visitar la localización original del documento (o descargarlo). Nótese que este contenido sólo se mostraría si el implementador decide utilizar la interfaz provista por el sistema.

4.1.3.2. Clases del análisis

Al analizar gramaticalmente los casos de uso, se pueden encontrar los siguientes objetos, cuyas relaciones se ilustran en el diagrama UML de la figura 4.2:

Preferencia: El grado de interés que un determinado usuario tiene en un concepto específico. Las preferencias de un usuario específico conforman su *perfil*.

Implementador: para efectos de almacenamiento, una aplicación que hace uso del servicio.

Usuario Final: individuos que utilizan las aplicaciones que se valen del servicio.

Recomendación: un resultado de la búsqueda y filtrado en la colección documental. Puesto que una recomendación apunta a un documento, se descarta como clase independiente.

Categoría: un concepto en la base de conocimiento del sistema.

Documento: una instancia de elemento de la colección documental.

4.1.4. Análisis de interacción y funcional

En esta etapa del análisis, se pretenden estudiar más a fondo las distintas interacciones entre el usuario y el sistema. Éstas ya fueron definidas a grandes rasgos en los casos de uso (4.1.2). Aquí se presentan diagramas de interacción profundizando en los casos de uso (figuras 4.3 y 4.4), diagramas de estado (figura 4.6) y un prototipo⁵ de la interfaz de usuario (figura 4.5).

Se consideran aquí dos enfoques posibles para la implementación del sistema -asumiendo el uso de la interfaz provista por el servicio: el primero (figura 4.6a) utiliza la información del perfil del usuario y la obtenida del contexto inmediato para construir la consulta -aquí se podría utilizar, por ejemplo, una ontología de dominio derivada del mismo perfil para la extracción de términos clave y la expansión con sinónimos. Por otro lado, el segundo enfoque (figura 4.6b) utiliza la información del perfil para filtrar y ordenar los resultados según su adecuación a éste, mientras que construye la consulta sólo en base al

⁵Creado con gomockingbird.com

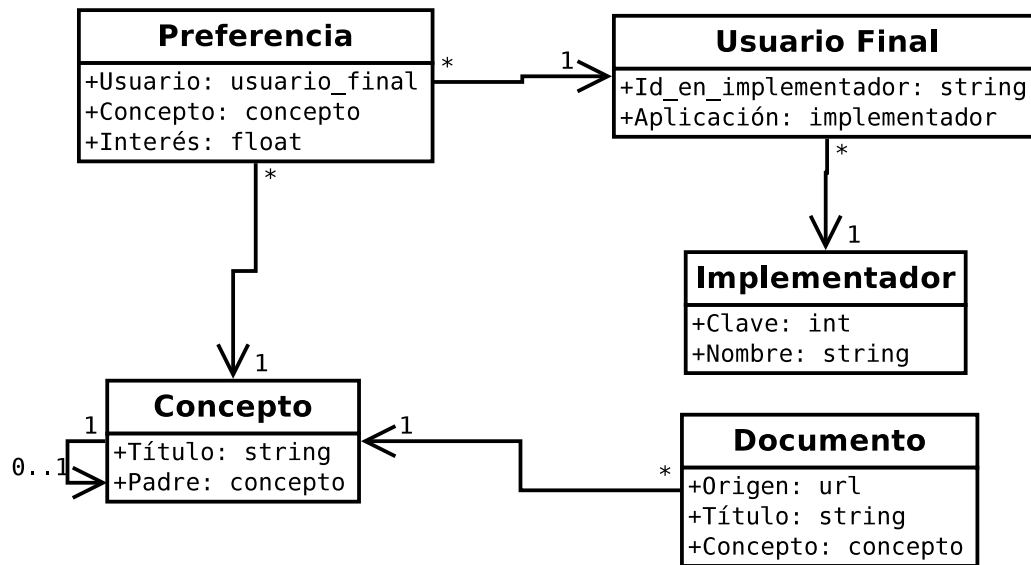


Figura 4.2: Diagrama de clases UML para las clases del análisis

contexto inmediato, extrayendo términos clave con algoritmos heurísticos, estadísticos o de procesamiento de lenguaje natural.

Aunque el análisis funcional merece un apartado aparte, aquí se ha obviado porque esta etapa de análisis ya ha presentado ciertos detalles de la funcionalidad interna y respecto al usuario; a la vez que una profundización mayor correspondería a la etapa de implementación.

4.1.5. Análisis de configuración

Aunque los requerimientos de configuración ya fueron mostrados en el apartado 4.1.1.2, las razones que los suscitaron se resumirán aquí.

Puesto que se ofrece la funcionalidad del sistema como un servicio solicitado por un implementador, la comunicación entre ambos debería ser tan independiente de las implementaciones como sea posible, para ello, se proponen dos notaciones de intercambio de datos ampliamente utilizadas en las aplicaciones web: *json* y *xml*. Se ha favorecido el primero por la facilidad de generación y análisis.

El lenguaje de programación python se ha elegido por su fácil sintaxis, extensa colección de librerías y eficiencia. El marco de desarrollo django se ha favorecido por la experiencia previa del autor con el mismo, su adhesión a filosofías de desarrollo efectivas, como el uso del modelo de arquitectura MVC, la orientación al desarrollo modular y la posibilidad de escalabilidad.

Puesto que se desea una interfaz de usuario dinámica e intuitiva, se hizo evidente que se necesitaría agregar funcionalidad al lado del cliente con javascript. Puesto que se desea que el sistema funcione en todos los navegadores web populares, se ha optado por la librería jquery, que se encarga de la compatibilidad y permite al desarrollador

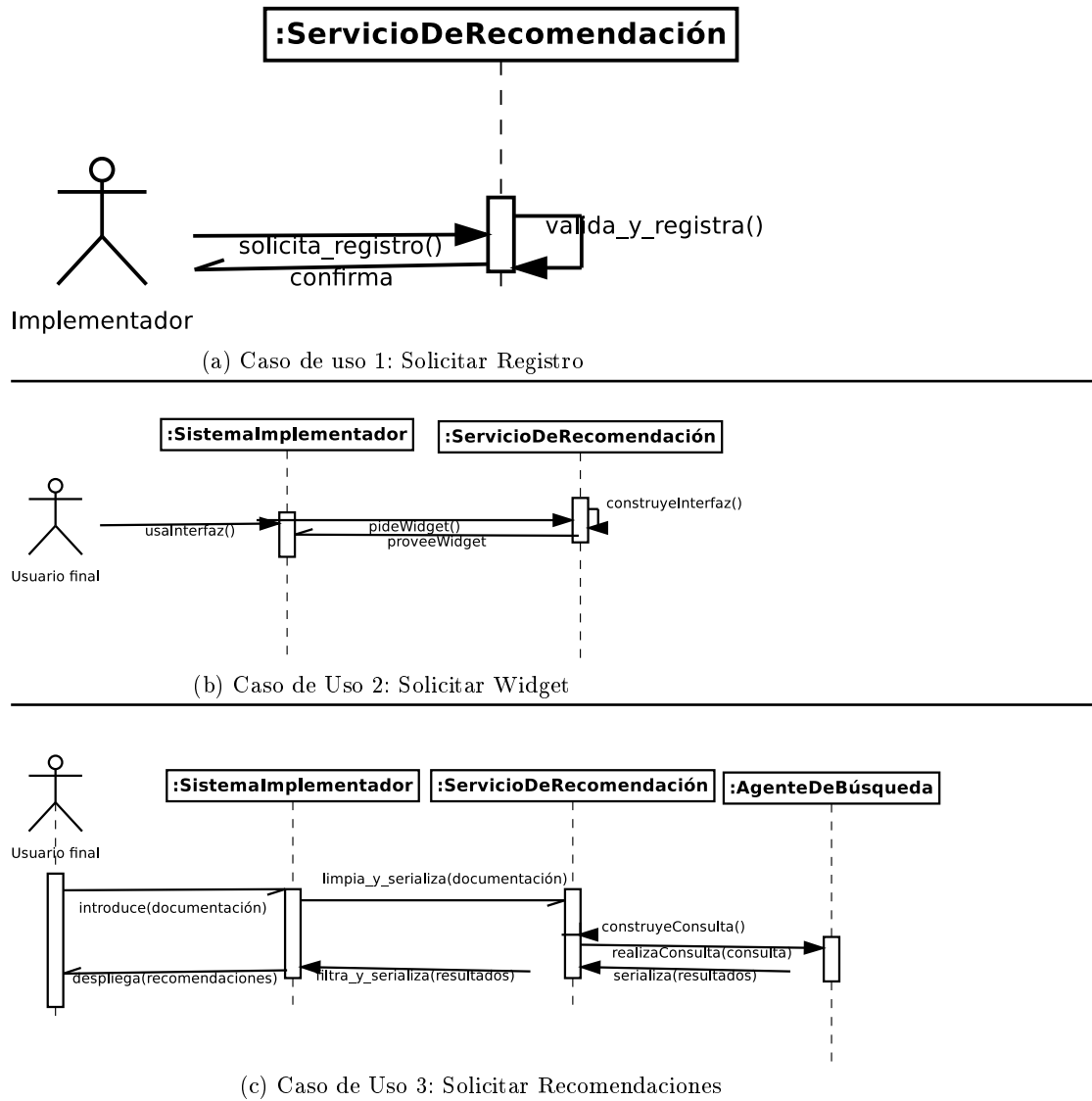


Figura 4.3: Diagramas de interacción para los casos de uso 1 al 3

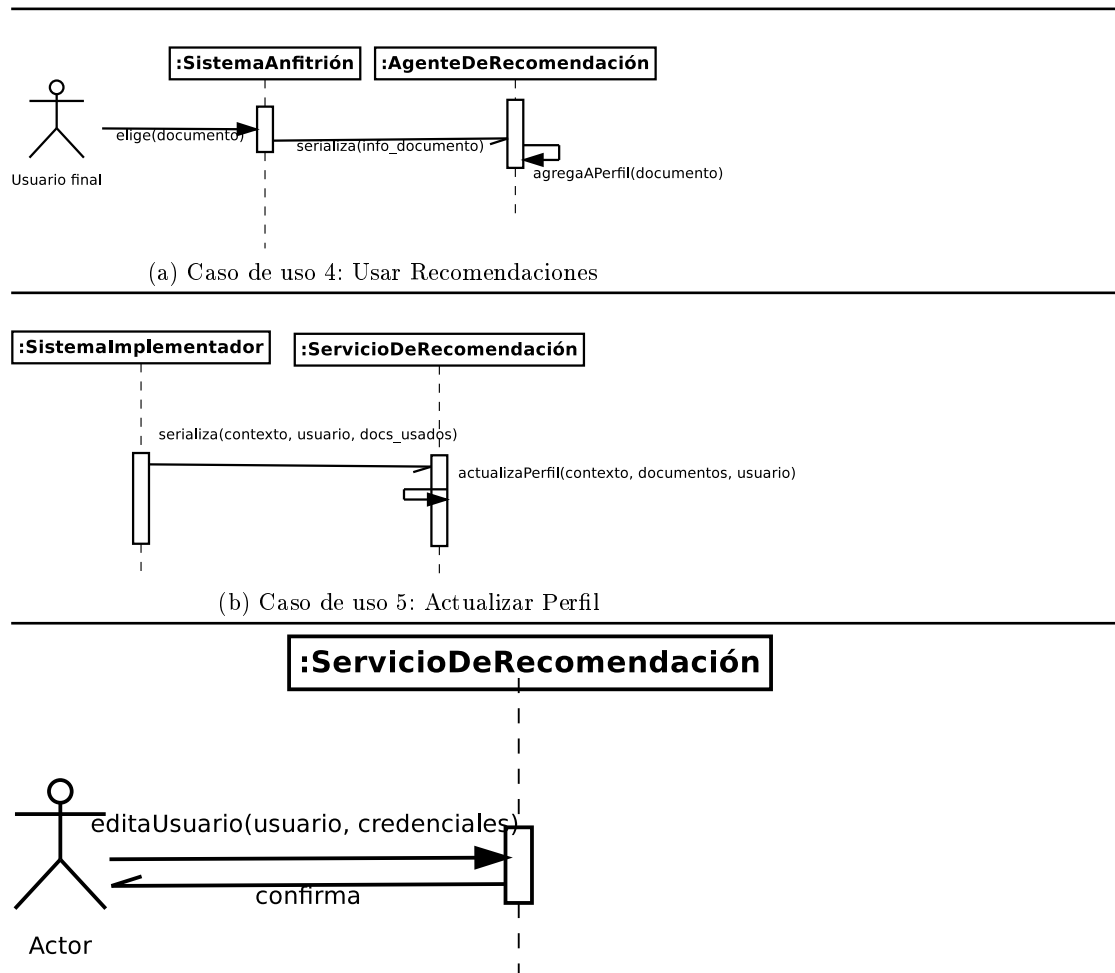


Figura 4.4: Diagramas de interacción para los casos de uso 4 al 6

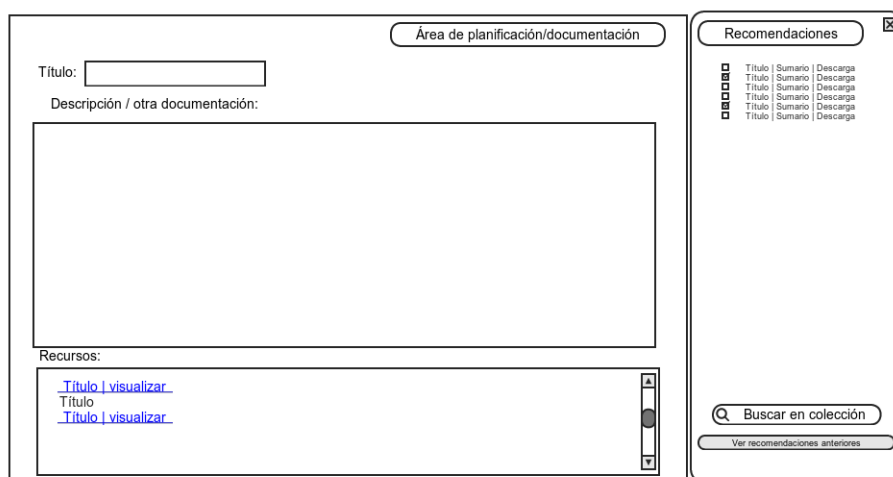


Figura 4.5: Prototipo de interfaz de usuario

preocuparse por escribir código funcional. A pesar de la existencia de diversas librerías javascript que comparten el mismo propósito (como prototype⁶ o mootools⁷), jquery tiene una sintaxis más sencilla, mayor uso en aplicaciones comerciales, soporte nativo a las técnicas JSONP y AJAX y eficiencia en la manipulación de (x)html⁸.

PostgreSQL es un motor de bases de datos eficiente en la manipulación de consultas, almacenamiento y procesamiento y es software libre.

Siguiendo la tradición del uso de servidores Linux para aplicaciones que pretendan escalar y ofrecer el mayor rendimiento y la compatibilidad óptima del *framework* de desarrollo con esta plataforma se ha optado un sistema operativo ubuntu linux para el almacenamiento y servicio.

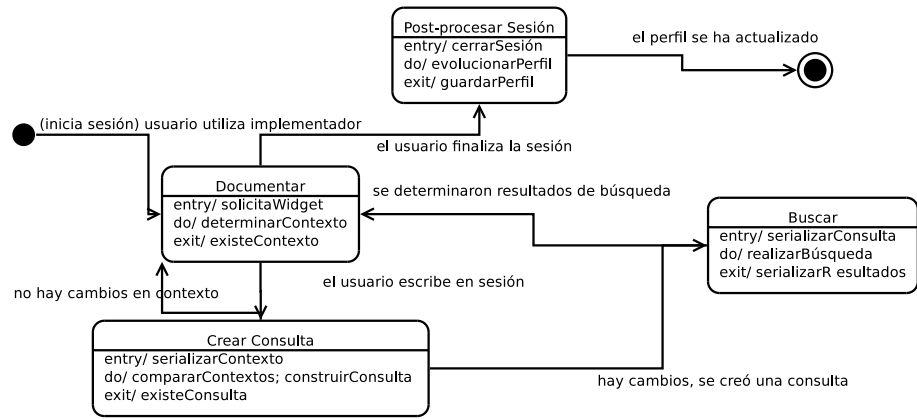
4.2. Diseño

Siguiendo la pauta sentada en la fase de análisis, se muestra aquí el diseño ordenado según las distintas facetas del mismo propuestas por ?, téngase en cuenta que la tercera etapa de ese flujo de diseño, el *diseño de navegación*, se obvia aquí porque el sistema carece de distintas interfaces a las cuales el usuario pudiese navegar; el usuario sólo verá la interfaz de recomendaciones.

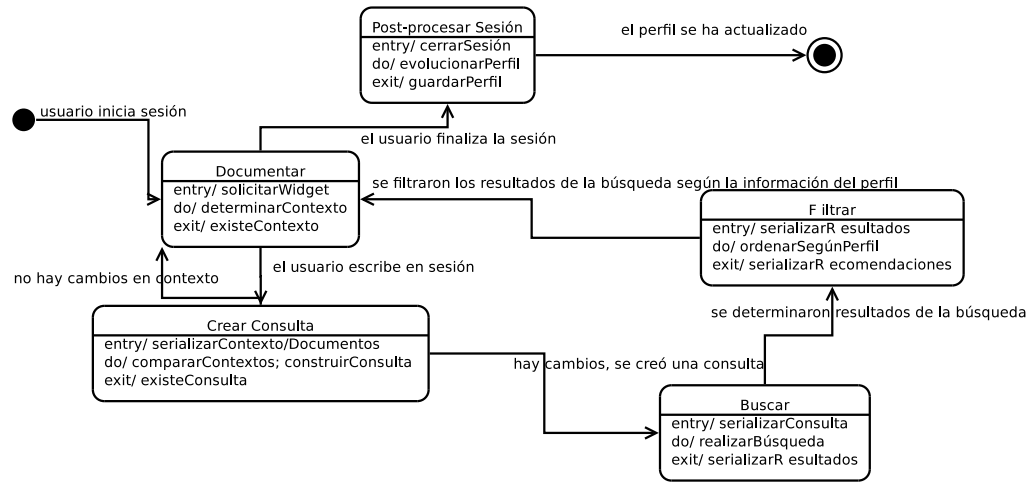
⁶<http://www.prototypejs.org/>

⁷<http://mootools.net/>

⁸Según la prueba comparativa de selectores css en <http://mootools.net/slickspeed/>



(a) Enfoque con *pre-filtrado*



(b) Enfoque de *post-filtrado*

Figura 4.6: Diagramas de estado para la interacción con el sistema

4.2.1. Diseño de la interfaz, estético y de contenido

Esta sección describe el diseño de la interfaz opcional que un implementador obtendrá si decide utilizar el *widget* de javascript provisto por el servicio.

Para no causar una división de atención contraproducente, se habrán de elegir colores neutros para el diseño de la interfaz. Y, tomando la idea de interfaz incremental propuesta por ? que presenta seis etapas del uso de una interfaz a la cual dirige paulatinamente su atención, se adaptaron éstas al presente proyecto de la siguiente manera:

1. *No hay interfaz visible*. En este momento el agente de recomendación está o decidiendo si la información en el contexto actual es suficiente como para llevar a cabo una búsqueda o realizando la búsqueda en sí. Puesto que no existen recomendaciones que mostrar, sería del todo contraproducente distraer al usuario con cualquier especie de contenido visual. El autor se había sentido tentado a utilizar una barra de progreso o una animación que mostrasen que el sistema estaba trabajando en las recomendaciones, pero esta decisión hubiera sido de mal diseño, porque, como asegura ?: “la interfaz [sólo] debe comunicar el estado de cualquier actividad que *el usuario haya iniciado*”⁹.
2. *Existen recomendaciones*. Cuando ya existan recomendaciones, se lo hará saber al usuario mediante una discreta pestaña a la derecha de la pantalla, de manera que sea notable en la visión periférica del usuario. Véase la figura 4.7 para un prototipo de cómo se vería la interfaz en este estado. Siguiendo los lineamientos de la ley de Fitt, se ha decidido ubicar esta pestaña en el borde de la pantalla, haciendo así la navegación a esta más rápida.
3. *Gráfico de relevancia*. Una vez que el usuario decida que es hora de evaluar las recomendaciones, dará clic en la pestaña de la interfaz. Esto dará paso a la ventana de recomendaciones, con un gráfico mostrando qué tan relacionadas están éstas, en promedio, al contexto inmediato actual. Así el usuario puede decidir regresar a su tarea sin continuar viendo el resto de la interfaz de recomendaciones. En todo momento el usuario debe poder esconder las recomendaciones.
4. *Palabras clave*. El usuario debe ser capaz de ver qué términos se extrajeron de su contexto local, y así juzgar si vale la pena seguir examinando los resultados.
5. *Muestras documentales ordenadas por relevancia*. Bajo el gráfico de relevancia se encontrarán las representaciones de los documentos encontrados, éstas estarán ordenadas por relevancia con el título resaltado y un corto resumen. Así el usuario puede juzgarlas someramente.
6. *Visualizar el recurso*. Una vez que el usuario ha decidido que un recurso puede serle útil, puede agregarlo inmediatamente a sus documentos de la sesión actual o visualizarlo en una ventana aparte. Nótese que en esta etapa el usuario ha pasado completamente de su tarea principal a la secundaria.

⁹Énfasis del autor

Además de las etapas de interfaz incremental de las recomendaciones en relación a la tarea principal antes discutidas, la interfaz de recomendación en sí debe obedecer lineamientos de diseño que aseguren una alta eficiencia cuando el usuario se decida a dividir su atención hacia ella. Los siguientes han sido tenidos en cuenta:

1. *Reducir el esfuerzo.* La interfaz de recomendación permitirá seleccionar fácilmente las recomendaciones y agregarlas con un botón que se encuentre en la muestra documental -o quizá agregar varias con un botón cercano a la lista de sugerencias.
2. *Facilidad de aprendizaje.* La interfaz de recomendaciones debe ser similar a lo que usuarios hayan usado antes, por ello, las muestras documentales se parecen a resultados de un motor de búsqueda. Además, claro, de lo discutido en el primer punto: las alternativas de uso permiten que tanto usuarios acostumbrados a interfaces dinámicas como neófitos puedan intuir sin mayor esfuerzo cómo usar las recomendaciones.
3. *Reducción de latencia.* Mediante el uso de solicitudes asíncronas al servidor (gracias a la técnica *ajax*¹⁰) a la hora de hacer las búsquedas de recursos a recomendar.
4. *Legibilidad y accesibilidad.* Mediante el uso de html apegado a los estándares de la w3, así como el *minimalismo* a la hora de mostrar los resultados: sólo lo necesario, sin elementos de distracción, con fuentes de tamaño y color legibles, y sumarios lo suficientemente cortos como para ser comprendidos pero no incurrir en una sobrecarga de información. Otra motivación para un diseño minimalista es lo poco deseable que, dada alguna configuración de pantalla relativamente estándar, el contenido tenga tan poco espacio que una barra de desplazamiento deba ser agregada por el navegador. Esto porque los usuarios encuentran indeseable este tipo de comportamiento (?).
5. *Comunicación:* Hacer saber al usuario cuándo, en base a un cambio de contexto, se han determinado nuevas sugerencias mediante un aviso conspicuo.

Para un prototipo de la interfaz de recomendación véase la figura 4.8, téngase presente que ésta estará a la derecha del área principal de trabajo (como en la figura 4.5), por lo que algunos detalles quizá sean distintos, dada la relativamente poca cantidad de espacio de la cual se dispone.

4.2.2. Diseño de arquitectura

Aunque ? identifica dos partes en esta etapa, a saber, la identificación y modelado de una *arquitectura de contenido* que refleje cómo las distintas páginas de una aplicación web están relacionadas en términos de navegación y una *arquitectura de aplicación* que se refiere al sistema en su totalidad. Dada la inexistencia de la primera arquitectura en el sistema propuesto, se hablará aquí solamente de la postrera.

¹⁰Del inglés *asynchronous javascript and xml* es una técnica que permite realizar solicitudes asíncronas al servidor sin necesidad de volver a cargar las páginas por completo a la hora de procesar la respuesta.

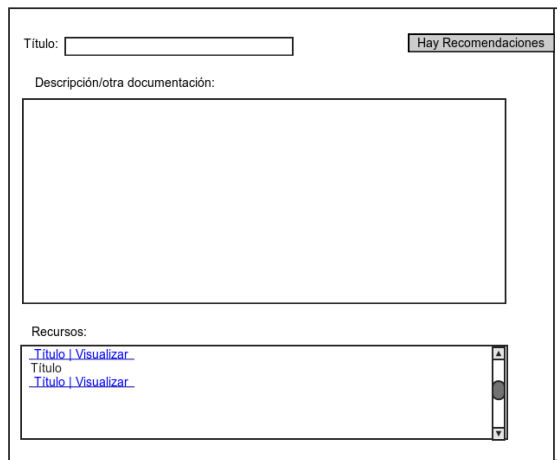


Figura 4.7: Estado oculto de la interfaz de recomendación

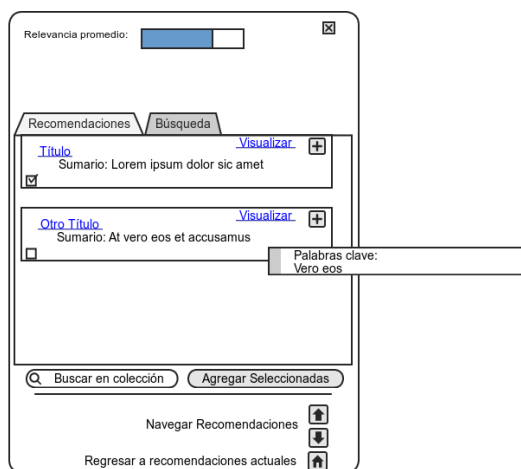


Figura 4.8: Detalle del prototipo de la interfaz de recomendación

4.2.2.1. Arquitectura en capas

La arquitectura en capas de la aplicación adoptará la propuesta por el marco de desarrollo *django*, una similar a la clásica modelo-vista-controlador: la arquitectura *modelo-plantilla-vista*¹¹ (MTV, por sus siglas en inglés). Todas las definiciones y descripciones que siguen están fundamentadas en la documentación oficial de *django*, que se puede encontrar -en inglés- en <http://docs.djangoproject.com/>.

Los componentes de esta arquitectura se describen someramente a continuación:

Modelo: un modelo en la arquitectura MTV es una clase que extiende a una súper-clase definida en el *framework* que abstrae tanto los atributos como la funcionalidad relacionadas a un objeto. Esta clase es independiente del sistema administrador de bases de datos que se utilice. Toda operación que se realice con el almacenamiento se hará a través de métodos implementados en la súper-clase.

Plantilla: (en inglés, *template*) representa la presentación de la aplicación, escrita en una combinación de html y un lenguaje para crear páginas dinámicas definido en el *framework* permite que desarrolladores y diseñadores puedan trabajar en la presentación y el comportamiento de la aplicación de manera separada y ordenada. Corresponde en parte a la *vista* del modelo MVC tradicional, porque define *cómo* se mostrará el contenido.

Vista: Corresponde en parte al controlador y la vista del modelo MVC tradicional; define *qué* contenido se presentará y es donde las interacciones del usuario con los datos almacenados se manejan. Cuando una URL¹² se solicita, *django* crea una instancia de una clase especial definida en el *framework*: la clase `HttpRequest`, que contiene meta-datos de la solicitud (como la ruta de la URL que se llamó y los encabezados) y la envía de parámetro a la vista, que es una función. La vista siempre devuelve una instancia de la clase `HttpResponse` que encapsula los datos que se mostrarán en una plantilla o se enviarán directamente al solicitante (como datos serializados en los formatos xml o json, una respuesta de error o una redirección a una página externa).

La interacción entre las distintas partes de esta arquitectura se puede ver así: el usuario solicita una página, identificada por una URL, el *framework* encuentra la *vista* correspondiente y la llama mediante una solicitud http, la vista puede o no interactuar con los *modelos* -creándolos, modificándolos o sencillamente obteniendo un conjunto de diversas instancias de éstos- y devuelve una respuesta que es mostrada en alguna *plantilla* o devuelta directamente al solicitante. Esta interacción está representada en la figura 4.9.

4.2.2.2. Despliegue

Entendida la separación en capas del sistema, se puede también considerar cómo se distribuirá físicamente la funcionalidad del servicio, para ello, se provee el diagrama

¹¹Comparada con la MVC en <http://docs.djangoproject.com/en/dev/faq/general/#django-appears-to-be-a-mvc-framework-but-you-call-the-controller-the-view-and-the-view-the-template-ho>

¹²del inglés, *uniform resource locator*

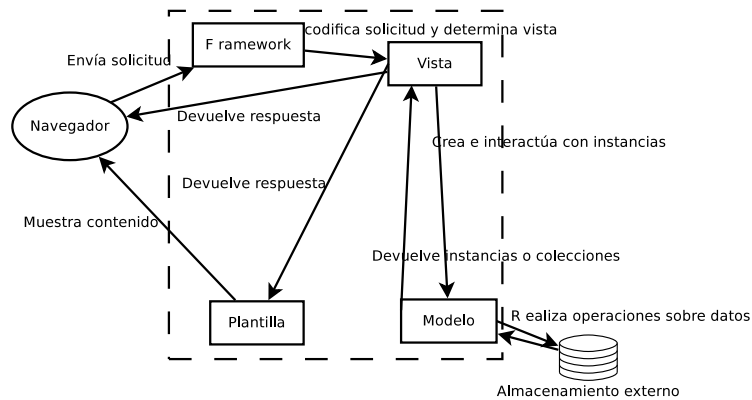


Figura 4.9: La arquitectura MTV

de despliegue de la figura 4.10. En ésta se puede observar que, dentro del servidor, se encuentra una instancia del *framework* django siendo ejecutada mediante la *interfaz de puerta de servidor web* (WSGI, por *web server gateway interface*) en un servidor apache; asimismo, se encuentra un servidor del administrador de bases de datos PostgreSQL, el cual contiene una base de datos con dos conjuntos semánticamente separados de tablas: las relativas a la información de aplicaciones implementadoras, con información de las mismas y los usuarios que éstas registran, incluidos los perfiles; el otro conjunto consiste en la información relativa a la ontología en uso y la colección documental. En el servidor, finalmente, se encuentra también el clasificador de categorías, un repositorio de datos administrado por *xapian* (una aplicación de indización y recuperación de información) utilizado por el servicio web para clasificar documentos y consultas. Fuera del nodo del servidor web, se encuentran los servicios web externos utilizados para la extracción de términos clave. Por último, se puede apreciar que no se supone nada del implementador, más que la comunicación con el servicio web mediante el protocolo http.

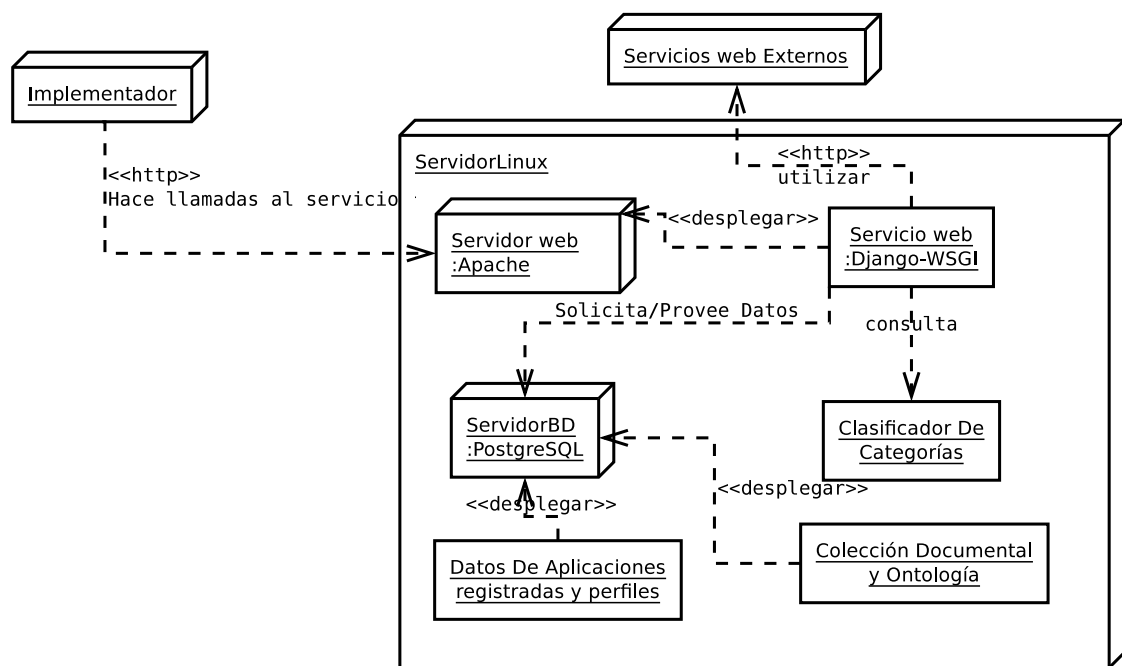


Figura 4.10: Diagrama de despliegue del servicio

5 Implementación

En este capítulo se discuten los aspectos teóricos y técnicos que llevaron a las decisiones de ingeniería involucradas en la implementación del sistema; el capítulo se organiza en base a los objetivos propuestos en la introducción general.

Para el desarrollo de este proyecto, dada su naturaleza académica y el poco tiempo disponible, se optó por construir el producto de software final a partir de la mayor cantidad de librerías re-utilizables ya existentes posible, de manera que tareas de considerable dificultad mas lo suficientemente comunes -como la indexación, búsqueda y clasificación- fueran realizadas mediante código ya probado e implementado, de manera que el implementador se pudiera dedicar a llevar a cabo los objetivos principales con éxito.

Como lo dice el título, el conocimiento sobre las preferencias de los usuarios se modelará mediante una ontología: una representación de conceptos relacionados mediante un grafo, donde los nodos son conceptos y las aristas las diversas relaciones entre ellos. En este proyecto, como se discutirá más adelante, se utiliza una ontología jerárquica, de modo que las relaciones entre conceptos son de *herencia*: una arista dirigida entre conceptos implica que el concepto de donde parte la arista es un *sub-concepto* del otro.

5.1. Obtención de la colección documental y creación de la base de conocimiento ontológica

Una colección documental suficientemente grande y bien organizada es fundamental para un sistema de recuperación de información, pues tendrá una mayor probabilidad de satisfacer cualquier consulta, no importa qué tan poco frecuente. Asimismo, un repositorio de conocimiento sobre el dominio elegido es crucial para un sistema de recomendación, ya que es mediante éste que se pueden hacer suposiciones acertadas sobre las preferencias del usuario.

Dado el carácter de este proyecto, ambas características son igualmente necesarias, y encuentran su síntesis en la siguiente necesidad concreta: poder, partiendo de suficiente evidencia documental, relacionar conceptos de una ontología (en este caso concreto) a un contexto de usuario, para poder, en última instancia, construir un perfil conceptual de los cursos asistidos por el sistema.

Como primera opción, se podría optar por construir una ontología a partir de una colección de documentos no clasificados, conseguidos de antemano o a medida los usuarios aporten recursos -un método interesante para ello se puede encontrar en ?. Aunque este enfoque es académicamente interesante e incluso podría resultar *a la larga* en una base de conocimiento exacta, dentro del contexto de la comunidad que la construya y utilice, no es factible para la implementación actual.

Una alternativa es partir de jerarquías conceptuales ya existentes, como proponen ?. La ventaja, evidentemente, radica en que el esfuerzo de obtener la ontología se minimiza, permitiendo ajustarla -en la mitad de lo posible- a las necesidades propias. Es este enfoque el que se ha adoptado en este proyecto. A continuación, se resumen las fases seguidas en esta etapa de la implementación.

5.1.1. Elección de una ontología

Con el advenimiento de la *web semántica*, muchos proyectos que tratan de organizar el conocimiento en la red han surgido, y otros que existían desde hacía varios años han encontrado nueva atención. Las necesidades consideradas para la elección de una fuente de ontología y, quizá, la misma colección documental, fueron:

- El fácil acceso a una estructura predefinida
- La posibilidad de actualización y la facilidad de la obtención de un sub-conjunto de la base entera de conocimiento
- La existencia de la misma en español (e inglés, de ser posible) y la oportunidad de extensión a otros lenguajes en un futuro.

*Wikipedia*¹, por ejemplo, se ha propuesto obtener y organizar el conocimiento humano para consulta. Pero, aunque existe cierta categorización en ésta y ciertamente se podría utilizar para crear jerarquías conceptuales (como lo han demostrado ?) la elección de un sub-conjunto del compendio de wikipedia, la coerción a una ontología jerárquica y la relativamente poca investigación académica realizada en ese respecto lograron que, aunque tentador, se optara por no utilizarla como fuente de la ontología y la colección documental. Una estructuración orientada a hacer más fácil para las aplicaciones obtener información estructurada de *Wikipedia* se propone en el trabajo de ?.

Otros proyectos, en la web, orientados a estructuras ontológicas (no necesariamente jerárquicas) o semánticas como *freebase.com* y *DBpedia.org* que se perfilan más acordes a las necesidades de este proyecto, mas lamentablemente éstas sólo están disponibles en inglés (aún cuando ambas aseveran la existencia de datos en varios idiomas, las ontologías sólo se encuentran en inglés).

Proyectos como CyC² y YAGO³, además de sólo estar disponibles en inglés, no ofrecen fácil acceso web.

La alternativa más sencilla de tomar -siendo la sencillez de implementación una necesidad informal, dada la poca cantidad de tiempo del que se dispone para el desarrollo de este proyecto- es la encontrada en el *Open Directory Project* (ODP)⁴: un proyecto relativamente más antiguo que los anteriores que pretende organizar en categorías los aportes de editores humanos, un enfoque similar al de *wikipedia* y *freebase.com*, pero con una estructura más rígida para organizar los conceptos. Las ventajas del uso del *ODP* son

¹wikipedia.org

²<http://researchcyc.cyc.com/>

³<http://www.mpi-inf.mpg.de/yago-naga/yago/>

⁴dmoz.org

varias: la estructura jerárquica, la disponibilidad en varios idiomas, las opciones de obtención de la ontología (por exploración directa de un web crawler o mediante la descarga de un documento estructurado, un *RDF*⁵), la posibilidad de obtener de una sola vez la colección documental y el hecho de haber sido sujeto de investigaciones académicas relacionadas a la presente (??).

5.1.2. Obtención de la ontología

Una vez elegida la ontología, se debe evaluar si se usará toda ésta o sólo un subconjunto de ésta. Como el *ODP* organiza a un nivel de especificidad varios tópicos de interés correspondientes a una gran cantidad de dominios del conocimiento, para fines de implementación, se decidió optar por el sub-árbol del tópico “*Science*”⁶ (ciencia) del *ODP* y los nodos paralelos a éste en español⁷.

En un principio se había considerado el uso de una araña web (*web crawler*) para obtener los árboles de ambas ontologías paralelas y luego los documentos que están listados en cada una. Esta consideración partió de un rápido vistazo a algunas páginas del sitio, que parecían dar un lugar específico a las sub-categorías y a los documentos que los usuarios han considerado relacionados con cada concepto. Pero la escritura de una pequeña araña⁸ que pusiera este concepto a prueba solamente extrayendo la información de recursos y sub-categorías -sin la descarga de los documentos listados entre los recursos- reveló que no sólo las páginas no son consistentes en el formato del html, sino que esta inconsistencia llevaría a un resultado inútil: muchos de los elementos extraídos eran información espuria o vínculos a otras categorías no necesariamente relacionadas.

Afortunadamente, el *ODP* ofrece su ontología entera en un archivo estructurado disponible para descarga en <http://rdf.dmoz.org/rdf/> (en concreto, en un archivo titulado *structure.rfd.tar.gz*) el cual es actualizado cada semana⁹. Se prosiguió entonces, a obtener el archivo, el cual, para el 23 de abril de 2010 tenía una magnitud de 76 megabytes comprimido (con *gzip*), y de aproximadamente 1.1 Gigabytes una vez extraído. Como se explicó antes, este archivo consiste en un xml que sigue el formato RDF, un formato creado con el fin de facilitar el acceso a información semántica sobre recursos.

Obtenido el *rdf* de la ontología del *ODP*, el siguiente paso consistió en la obtención de aquellos tópicos que se referían al sub-árbol elegido (el referente a ciencia y tecnología). Los pasos para la completación de esta tarea fueron:

1. La consulta a la documentación del *ODP* de las etiquetas para la construcción de este archivo: <http://rdf.dmoz.org/rdf/tags.html>. Y la inspección visual del mismo para determinar el orden de escritura de los tópicos: se descubrió que se parte del tópico más general y lo siguen inmediatamente sus categorías (encerradas en las etiquetas *narrow*, *narrow1* y *narrow2*) y los nodos equivalentes al mismo en los árboles ontológicos de idiomas alternos (en las etiquetas *altlang*).

⁵<http://www.w3.org/RDF/>

⁶<http://www.dmoz.org/Science/>

⁷http://www.dmoz.org/World/Espa~nol/Ciencia_y_tecnología/

⁸Basada en el ejemplo sencillo propuesto en <http://doc.scrapy.org/intro/tutorial.html>

⁹<http://www.dmoz.org/help/getdata.html>

2. La búsqueda del rango entre el cual se encontraban los tópicos referentes a la sub-ontología elegida, esto mediante el uso de expresiones regulares mediante el comando *egrep* <http://unixhelp.ed.ac.uk/CGI/man-cgi?egrep>. Encontrado el rango de líneas en el archivo donde se encontraban los tópicos deseados, se procedió a extraer éstas a un archivo *rdf* aparte mediante el comando *sed* <http://unixhelp.ed.ac.uk/CGI/man-cgi?sed>. El archivo de la sub-ontología de ciencia medía apenas 10 megabytes, cien veces menos que el original.
3. Del xml generado, se requirió obtener las urls de cada tópico (junto a la url del mismo en español, si existía), y esto se logró mediante un *script* escrito en *python* que se valió de la estructura del xml para encontrar las mismas y escribirlas en un archivo. Se encontraron 13,407 urls correspondientes a todos los tópicos en el sub-árbol de ciencia, y de éstas sólo 402 forman la sub-ontología paralela en español.

Así, una vez obtenidos los *url* que se refiriesen a los tópicos, se pudo proceder a obtener la colección documental.

5.1.3. Almacenamiento de la ontología

Por cuestiones de rendimiento, se optó por almacenar la ontología en la base de datos, como punto de partida para la creación de los perfiles de los cursos (ya que éstos, una vez almacenados, se referirían a categorías de la ontología), dado que, si se conservase solamente el RDF como referencia ontológica, se tendría que interpretar éste en memoria cada vez que se actualizase un perfil, incurriendo en gastos innecesarios de memoria y procesamiento.

Como primer paso, se expresó la ontología como un árbol de categorías, siendo cada nodo modelado una tupla relacional cuyos atributos, basados en los atributos establecidos en el *RDF* eran, para cada tópico:

Título: el título que el *ODP* asigna.

ID del tópico: la ruta única de la raíz de la ontología al presente tópico.

Código dmoz: el código numérico que el *ODP* asigna.

Última actualización: la última vez que el tópico fue editado en el *ODP*.

Descripción: un breve resumen

Padre: el tópico predecesor inmediato

Vínculos alternos: vínculos a los tópicos paralelos en otros idiomas (por ahora, sólo se almacena el vínculo al tópico en español).

Con el fin de facilitar el llenado y mantenimiento de la tabla, se creó un comando¹⁰ de *django* que permitiese, dado un *RDF* del *ODP* bien formado, crear o actualizar la representación relacional del mismo.

¹⁰Un comando de *django* es un script de *python* que tiene acceso a las variables de entorno y los módulos definidos en el proyecto

5.1.4. Obtención de la colección documental

El archivo con los *url* referentes a los tópicos de la sub-ontología sirvió como una entrada ideal para una araña web que pudiera visitar cada una de esas páginas y obtener los recursos listados en ellas, obteniendo a la vez, para cada recurso, información relevante para las siguientes etapas, como ser: el origen del mismo, la categoría donde se encontró, el lenguaje, un título descriptivo y una descripción.

La araña web se implementó en *Scrapy*¹¹, un *framework* dedicado precisamente a la escritura de arañas web y aplicaciones de extracción de información en páginas web. Dada la cantidad de almacenamiento, tiempo y ancho de banda requeridos para la ejecución de una aplicación de esta naturaleza, se utilizó una máquina virtual instalada en uno de los servidores del laboratorio de sistemas de UNITEC¹².

Como nota marginal, es importante notar que una araña web debe tener en cuenta los límites de los recursos que utiliza: una araña puede tomar dos caminos a la hora de hacer la exploración del sitio designado: un recorrido en anchura o en profundidad. Interesantemente, el árbol de exploración, en este caso, era conspicuamente ancho (13,407 *urls* iniciales) y poco profundo (por cada *url*, sólo se iba un nivel más abajo, el de los recursos, los cuales son en promedio 30 por cada *url* de la sub-ontología elegida). La primera vez que se ejecutó el proceso la memoria se vio tan sobrecogida por la cantidad de objetos en memoria (un objeto por cada *url* inicial) que, cuatro horas después del inicio, el sistema operativo se vio obligado a detener el proceso. Y esto porque la estrategia por defecto para una araña web en *scrapy* es el recorrido en anchura. Tomado esto en consideración, se optó por el recorrido en profundidad, con efectos dramáticos: cuatro horas después, la araña había logrado bajar varios cientos de documentos, cuando con la estrategia anterior, no logró bajar *ninguno*.

Para la colección documental a usar en la etapa de desarrollo y primera presentación en UNITEC, la araña web obtuvo una colección documental de 3.7 Gb, (con 100,546 archivos *html* y 1117 *pdf*) requiriendo aproximadamente quince horas-procesador (y veintitrés horas físicas) para ello. Para el ambiente de prueba -en la computadora personal del autor- se usó una sub-ontología de 18 categorías, con una colección documental de 17.1 Mb, compuesta por 482 documentos en *html* y 11 en *pdf*.

5.2. Implementación del componente de búsqueda

Una vez en poder de la colección documental, restan dos cuestiones: el almacenamiento estructurado de ésta y la elección de un módulo para buscar en la misma.

5.2.1. Elección de sustitutos documentales

Como se explicó antes¹³, es importante representar de manera informativa y compacta cada documento en la colección. En esta implementación, se optó por seguir el patrón

¹¹ scrapy.org

¹² Para ver una copia de la requisición del mismo, véase el apéndice A

¹³ consúltase 2.1.2

sentado por el *ODP*: almacenar un título, una descripción, la categoría del directorio donde se encontró (expresada como una ruta desde la raíz hasta el nodo donde se encuentra listado como recurso el sitio), una descripción, el *url* original, la fecha en que se obtuvo el documento y el contenido en sí.

La araña web que obtuvo los documentos se escribió de manera que preservase toda esta información serializada en archivos, de manera que facilitase el llenado de la base de datos de sustitutos documentales.

5.2.2. Elección de una implementación de recuperación de información

Puesto que el alcance del proyecto no incluye el demostrar la idoneidad de uno u otro modelo de IR ni mucho menos se centra en la implementación de alguno, se optó por elegir una implementación existente que pudiera ser integrada a un proyecto en *django* sin mayor problema y que pudiera obtener su colección documental de una base de datos relacional.

Se determinaron dos alternativas:

1. *Haystack*¹⁴: un proyecto que permite al desarrollador desentenderse de la implementación de búsqueda subyacente, mediante el uso de una interfaz de programación de aplicaciones. Actualmente, permite el uso de tres motores de recuperación de información: *solr*¹⁵, que implementa -en java- la indexación semántica latente; *whoosh*¹⁶, que implementa en python una versión extendida del modelo de espacio vectorial y *xapian*¹⁷, una implementación en C++ de un modelo probabilístico de recuperación de información.

Aunque *Haystack* es una solución idónea para proyectos comerciales, donde se requiere poca interacción, y a muy alto nivel, con la implementación de búsqueda, se consideró demasiado complejo para el proyecto actual por lo que se descartó, aunque los otros módulos del proyecto están escritos de manera que respeten el principio de bajo acoplamiento de modo que la posibilidad de uso de esta alternativa queda abierta.

2. *Djapian*¹⁸ Una interfaz entre *django* y *xapian*. Aunque es un proyecto menos flexible y maduro que *haystack*, *djapian* ofrece más control sobre los particulares de búsqueda, en especial la sensibilidad al idioma de las consultas y documentos, característica decisiva para su elección, ya que inicialmente el proyecto cuenta con una colección documental bilingüe.

¹⁴<http://haystacksearch.org/>

¹⁵<http://lucene.apache.org/solr/>

¹⁶<http://bitbucket.org/mchaput/whoosh/wiki/Home>

¹⁷<http://xapian.org/>

¹⁸<http://code.google.com/p/djapian/>

5.3. Implementación del componente de perfilado

En poder de la ontología, indexados los sustitutos documentales e implementado el componente de búsqueda automática, el siguiente paso fue resolver la cuestión de la representación, construcción y evolución del perfil.

5.3.1. Entrenamiento del clasificador de categorías

Antes de resolver el quid de esta etapa, hubo de sentar el fundamento para un enfoque ontológico de perfilado: para un documento o contexto local, se debe poder determinar a qué categoría o categorías pertenece. Aunque en un principio se pensó en seguir el ejemplo de ? y construir vectores acumulativos para cada categoría, siguiendo el modelo de espacio vectorial. Sin embargo, puesto que tal implementación habría trascendido el alcance del proyecto, se decidió hacer uso de una herramienta de propósito general: *xappy*, una interfaz en *python* para la librería de recuperación de información *xapian* escrita en C++.

Para cada categoría, se tomaron todos los documentos que pertenecían a la misma, según lo determinado en la etapa anterior, y se usaron para indexarla. En otras palabras, se consideró cada categoría como un gran documento compuesto del texto de todos los sub-documentos pertenecientes a ella, según el *ODP*; al indexar estos documentos/categorías, se consigue efectivamente que, dada una cadena de texto, se puedan encontrar las categorías a las cuales tiene más probabilidad de pertenecer. Aunque este enfoque es, en principio, similar al de ?, la mayor diferencia es el modelo de recuperación de información a usar: en este proyecto se eligió un modelo probabilístico en lugar del de espacio vectorial.

El lector quizá se pregunte por qué, dado que la misma librería subyacente se habría de usar, no se usó la misma interfaz que para el almacenamiento de documentos, *djapian*. La razón es simple: eficiencia de almacenamiento, *djapian* requiere el almacenamiento de *todo* el texto de cada documento en una tupla de una base de datos relacional para *luego* crear un índice de *xapian* (el cual, en el peor de los casos, es tan grande como la base de datos original), este precio se pagó con los sustitutos documentales porque se requería la integración con *django*; pero para la clasificación de categorías el almacenamiento redundante de los documentos para las categorías era sencillamente una pérdida. Con *xappy*, se puede obviar el paso intermedio de almacenamiento en una base de datos relacional y agregar los documentos inmediatamente al índice, con lo que se utiliza efectivamente menos espacio, ya que los documentos indexados no contienen todo el texto original, sino sólo lo que resulta luego del procesamiento de la librería: colecciones de-duplicadas de términos clave, libres de palabras vacías y, sobre todo, de redundancia.

Una vez creado el índice de categorías, se implementó una función que recibe una consulta y devuelve las diez categorías a las cuales tiene mayor probabilidad de pertenecer.

5.3.2. Ponderación de las categorías

Una de las razones por las cuales elegir el uso de una ontología, frente a la opción de una lista plana de categorías (o conceptos) es la posibilidad de descubrir, mediante las relaciones entre nodos de la ontología, conceptos que podrían ser de interés -aunque indirectamente- al usuario.

Para hacer este tipo de inferencia posible, se debe recurrir a un tipo especial de análisis de grafos llamado *análisis de redes ontológicas* (las bases teóricas de este tipo de análisis se pueden encontrar en ?). En específico, como sugiere el trabajo hecho con el sistema *ontocopi*, descrito por ?, se debe recurrir a un algoritmo de propagación, que permita encontrar otros nodos -o categorías- relacionados al inicial siguiendo las relaciones semánticas representadas como aristas en el grafo de la ontología. Este tipo de algoritmos se han usado en sistemas que utilizan como base de conocimiento una ontología, con distintos enfoques. En el previamente mencionado *ontocopi*, por ejemplo, debido a la existencia de diversas relaciones en la ontología, el algoritmo de propagación es equivalentemente complejo y flexible: puede ajustarse a sólo seguir ciertas relaciones hasta cierta profundidad.

En cambio, la ontología del *ODP* es un grafo más sencillo: consiste simplemente en una jerarquía, representable en un árbol, donde una categoría puede tener uno o ningún padre y ninguno o varios sucesores; siendo las relaciones de la ontología de un solo tipo: “es subcategoría de”. Con una ontología tan simple, el algoritmo de propagación se simplifica también.

El enfoque de ?, por ejemplo, simplemente propaga un medio del interés de la categoría inicial hacia su ancestro inmediato, luego un medio del interés de éste a su sucesor y así sucesivamente hacia la raíz. Esto permite que, aunque el usuario se haya interesado en una categoría específica, el perfil -y, por tanto, las recomendaciones- pueda tener en cuenta los antecesores más generales de ésta.

Por otro lado, el enfoque de ?, más fiel al análisis de redes ontológicas, pondera cada arista con el nivel de *inclusión* que una categoría tiene con su padre, es decir, dado el vector que representa a una categoría, el peso de la arista que une a ésta con su padre es proporcional a qué tantos de los términos del vector de ésta están contenidos en el vector que representa a su sucesor.

Siguiendo ese ejemplo, el enfoque de la ponderación de las aristas en este proyecto se hace en base a una medida estadística ligeramente distinta: la probabilidad de utilidad. Esta probabilidad empírica se deriva de la premisa que, entre más documentos tenga una categoría, más “prometedora” es en términos de futuras recomendaciones, por lo que su sucesor debería recibir menos atención; y, a la inversa, si una categoría carece de documentos, su sucesor debería tomarse más en cuenta a la hora de hacer recomendaciones, ya que es probable que la categoría no vuelva a producir resultados útiles.

Por consiguiente, la probabilidad de utilidad se puede expresar como el complemento de la razón entre la cantidad de documentos contenidos en una sub-categoría respecto a la cantidad de documentos total del sucesor de la misma. Donde la cantidad de documentos se define por la siguiente fórmula:

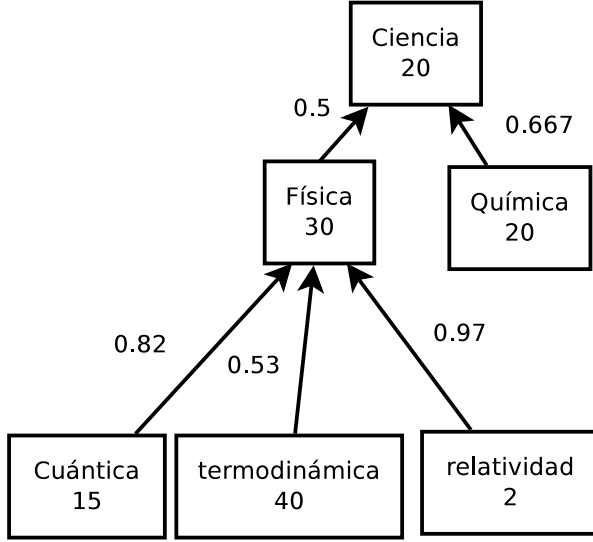


Figura 5.1: Ejemplo de ontología ponderada

$$D(C_i) = |docs(C_i)| + \sum_{c_j \in SubCats(C_i)} |docs(C_j)| \quad (5.1)$$

(donde $SubCats(C)$ se refiere al conjunto de sub categorías de un concepto C y $docs(C)$, al conjunto de documentos del mismo).

Así, la ponderación de una arista entre categorías se puede expresar como:

$$W(C_i, C_j) = \gamma * (1 - D(C_j)/D(C_i)) \quad (5.2)$$

En la implementación, la constante γ se ha establecido en la unidad, de manera que los pesos de las aristas entre categorías se encuentran en el rango $[0, 1]$.

La computación de estos pesos sólo se hace una vez para cada actualización de la ontología. Una representación gráfica de una ontología ponderada como la recién descrita se puede observar en la figura 5.1.

5.3.3. Evolución del perfil

Una vez construida y ponderada la ontología, la creación y evolución de los perfiles se convierten en tareas relativamente sencillas. Como ya se discutió en el marco teórico, existen dos enfoques principales para perfiles automáticos: utilizarlos para re-escribir las consultas del usuario, mediante conocimiento obtenido en sesiones anteriores; o usar ese mismo conocimiento para re-ordenar resultados de una búsqueda en una colección documental respecto a la similitud entre el perfil y los resultados. En la actual implementación se decidió utilizar el segundo enfoque, debido a que éste puede ser implementado mediante técnicas estadísticas puras, mientras que el primero debe ser implementado mediante

técnicas de procesamiento de lenguaje natural o construcción de consultas en un lenguaje estructurado.

El perfil del usuario se representa como un conjunto de tuplas, donde cada una es de la forma $(C, IS(C))$ donde C es un concepto (o categoría) de la ontología e $IS(C)$ es el *puntaje de preferencia* asignado a la categoría, este puntaje es un valor en el rango $[0, \infty]$, de manera que la preferencia demostrada por el usuario a una categoría en un momento de tiempo es proporcional a este puntaje.

Una vez que un usuario termina una sesión de uso del servicio, se utiliza el clasificador probabilístico entrenado con el *corpus* documental descrito en la sub-sección 5.3.1 para obtener las categorías a las cuales el contexto tenga mayor probabilidad de pertenecer, asignando a éstas un puntaje de preferencia equivalente a la similitud calculada por el clasificador entre cada categoría y el contexto, estando éste en el rango $[0, 1]$. Asimismo, los conceptos bajo los cuales están clasificados los documentos que el usuario pueda haber elegido como retroalimentación reciben un puntaje inicial de 1. Con los conceptos en este conjunto, los devueltos por el clasificador y los de los documentos, se tiene el *conjunto de conceptos base*.

Acto seguido, se utiliza un algoritmo de propagación (basado en el trabajo de ?) para agregar al conjunto de conceptos base los antecesores de éstos en la jerarquía ontológica, asignándoles una puntaje de preferencia calculado con la siguiente fórmula:

$$IS(C_i) = IS(C_j) * W(C_i, C_j) \quad (5.3)$$

partiendo de la premisa que, si un usuario muestra interés en un concepto, está mostrando un interés indirecto en los conceptos más generales que el mismo. Nótese que este interés está atenuado por la relación de pertenencia, previamente explicada, entre los conceptos; asignando más interés a los antecesores de conceptos cuyo cuerpo documental es pequeño en relación a otros al mismo nivel de la jerarquía.

Construido el conjunto expandido de conceptos, se procede al paso más crucial: la evolución propia del perfil. En esta etapa se hace una partición del conjunto de conceptos en base a la existencia previa de un puntaje de preferencia en el perfil: en este siguiente paso sólo se toman en cuenta los conceptos que tenían un puntaje previo, los otros no reciben más procesamiento y se agregan al perfil. De esta manera, para cada concepto previamente existente en el perfil, se calcula un nuevo puntaje de preferencia, utilizando la siguiente fórmula, adaptada de ?:

$$IS(C_k) \begin{cases} \beta * IS(C_k) & \text{si } C_k \notin CON \\ \beta * IS(C_k) + (1 - \beta) * IS(C_k) & \text{si } C_k \in CON \end{cases} \quad (5.4)$$

Donde CON es el conjunto de conceptos encontrados en la sesión actual y β es un factor de deterioro con un valor en el rango $[0, 1]$. Así, si un concepto anterior no vuelve a aparecer en la sesión actual, el interés es disminuido (se puede decir que el concepto envejece). En cambio, si el concepto vuelve a aparecer, aunque es deteriorado -para favorecer conceptos más nuevos- es aumentado mediante el inverso del factor de decadencia. En suma, el interés de los conceptos va decayendo y este deterioro sólo es contrarrestado si un concepto vuelve a ser favorecido. Para acelerar el proceso de deterioro y poder

favorecer los conceptos más recientes, se ha elegido un valor de 0.2 para el factor β . El proceso de evolución se puede ver resumido en el algoritmo 1.

Algoritmo 1 Evolución del perfil

Entrada: contexto (en forma de texto libre o lista de términos), el perfil del usuario y documentos (provistos como retroalimentación)

Salida: perfil de usuario actualizado

```

// categorizar devuelve un mapa <categoria:preferencia>
1: CON = categorizar(contexto) + categorizar(documentos)
2: for all  $\kappa \in \text{CON}$  do
3:    $c = \kappa$ 
4:    $p = c.\text{padre}$ 
5:   while  $p$  do
6:      $\text{CON.agregar}(\langle p:\text{max}(\text{CON.obtener}(p), \text{CON.obtener}(c)*W(p,c)) \rangle)$ 
7:      $c = p$ 
8:      $p = p.\text{padre}$ 
9:   end while
10: end for
11:  $\text{conceptosPreexistentes} = []$ 
12: for all  $\text{preferencia} \in \text{perfil}$  do
13:   if  $\text{preferencia} \notin \text{CON}$  then
14:      $\text{preferencia.puntaje} = \beta * \text{preferencia.puntaje}$ 
15:   else
16:      $\text{preferencia.puntaje} = \beta * \text{preferencia.puntaje} + (1 - \beta) * \text{preferencia.puntaje}$ 
17:   end if
18:    $\text{conceptosPreexistentes.agregar}(\text{preferencia.concepto})$ 
19: end for
20:  $\text{nuevosConceptos} = \text{Conjuntos.diferencia}(\text{CON}, \text{conceptosPreexistentes})$ 
21: for all  $\nu \in \text{nuevosConceptos}$  do
22:   if  $\text{CON.obtener}(\nu) \neq 0$  then
23:      $\text{perfil.agregar}(\nu, \text{CON.obtener}(\nu))$ 
24:   end if
25: end for

```

5.4. Implementación del servicio

Como se describió anteriormente (sección 1.6), el servicio debe ser lo suficientemente flexible como para ser adoptado por cualquier desarrollador en cualquier plataforma. Por ello, se decidió crear una interfaz de programación de aplicaciones que funcione sobre el protocolo HTTP, para que los implementadores sólo se preocupen de hacer las solicitudes correspondientes e interpretar las respuestas.

Para evitar que personas maliciosas abusen del servicio, se decidió restringirlo hasta cierto punto: en primer lugar se requiere que los implementadores se registren en la página

principal del sistema, para así obtener, mediante correo electrónico, una clave única para utilizar el sistema; esta clave es creada a partir de datos únicos del usuario mediante el algoritmo de cifrado RSA. Una segunda medida es limitar el número de usuarios¹⁹ que cada aplicación puede utilizar, además de requerir que, para registrar a un usuario, se utilice una solicitud autenticada -con una contraseña provista a la hora de registrar la aplicación- de manera que, si un usuario con malas intenciones decidiera crear usuarios espurios de una aplicación, tendría que obtener *dos* datos que sólo el implementador que registró la aplicación conoce: su clave única y su contraseña para registro. En un primer esbozo del diseño, se consideró que la clave única era suficiente para considerar auténtica una solicitud, por lo que el registro de usuarios no hubiera sido necesario (para crear un usuario sólo hubiera sido necesario pedir recomendaciones para un perfil aún no registrado), pero esta opción se consideró insegura al tener en cuenta que la clave única es visible por el usuario en el código del cliente si se decide utilizar una interfaz web; y aunque el código estuviese ofuscado, existen herramientas sencillas de obtener y utilizar que permiten ver todos los datos que una solicitud web contiene (como la consola javascript del navegador google chrome²⁰ o la extensión *firebug*²¹, del navegador firefox²²).

Pero la flexibilidad no debería afectar la facilidad de uso: aunque la solución más conveniente para desarrolladores experimentados sea una API que les permita utilizar el servicio como deseen, registrando los usuarios desde sus propias aplicaciones y luego utilizando los métodos de recomendaciones y actualización de perfil, implementadores menos versados en el desarrollo de aplicaciones deberían tener la oportunidad de aprovechar el servicio. Para este segmento de los usuarios potenciales se idearon dos soluciones: luego de registrarse y obtener las credenciales, se permite al usuario utilizar una interfaz web en el sitio del servicio para registrar usuarios, de manera que no se deben preocupar por crear su propio código para este efecto. La segunda solución, sólo para implementadores en aplicaciones web, es la interfaz opcional (o widget): para poder utilizar el servicio de recomendaciones, sólo deben incluir en la fuente de sus páginas web un pequeño *script* -diseñado para ser tan fácil de personalizar como lo es llenar un formulario- y el servicio les proporcionará el widget y éste hará las llamadas al servicio por ellos.

Ambas soluciones se describen con detalle en el anexo [manual de usuario]; en las siguientes secciones se detallan las decisiones de ingeniería que cada una de ellas implicó.

5.4.1. Interfaz de programación de aplicaciones

Para la implementación de la API se decidió seguir la tendencia actual e implementar principios del estilo de arquitectura REST. A continuación, se detalla cómo los principios

¹⁹Para la implementación actual, se cuentan con 30 Gb de almacenamiento, lo que implica que se pueden guardar aproximadamente 187 perfiles de usuario (teniendo en cuenta que una entrada de preferencia de usuario mide 12 bytes (dos llaves foráneas al concepto y el perfil, mas el puntaje de preferencia que utiliza el tipo *real* de postgresql; y, que en el peor de los casos, un perfil contendrá puntajes para las 14,000 categorías).

²⁰Disponible en <http://www.google.com/chrome>

²¹<http://getfirebug.com/>

²²<http://www.mozilla-europe.org/es/firefox/>

de esta arquitectura se ven cumplidos.

Distinción cliente-servidor: Característica fundamental de cualquier servicio web, el solo hecho de implementar éste como una API permite que tanto el cliente como el servidor puedan evolucionar independientemente, mientras cumplan con el protocolo HTTP y puedan interpretar lo que uno u otro transmite en las comunicaciones.

Sin estado: Se decidió que el implementador tendrá que enviar en cada solicitud el token único asignado a la aplicación así como el identificador del usuario final; aunque esto podría haberse resuelto con una llamada a alguna función de inicialización de sesión en la que estos datos se determinasen y guardasen en una sesión, se hubiera dependido de almacenamiento entre transiciones de estado en el servidor, obstaculizando la escalabilidad y la distribución de las tareas. Por ello, se prefirió hacer búsquedas en la base de datos cada vez que se reciba una solicitud, teniendo en cuenta claro, que existen facilidades del administrador de base de datos como el sistema de caché y el uso de índices para reducir la latencia.

Caché: puesto que las solicitudes se valen de mecanismos propios de HTTP, la facilidad de caché²³ del protocolo puede entrar en acción donde sea permitido por el mismo.

Identificación de recursos: mediante solicitudes a las URLs de la API, los implementadores reciben respuestas en hipertexto con los encabezados correspondientes (`{Content-Type:"application/json"}` para los datos relativos a recomendaciones y `{Content-Type:"text/html"}` para el widget opcional).

Manipulación de recursos mediante representaciones: el implementador hace solicitudes utilizando los métodos GET o POST y puede leer los recursos recomendados, devueltos en formato JSON o modificar el perfil de un usuario mediante una llamada a la URL correspondiente.

Hipertexto como motor del estado representacional: Al devolver recomendaciones, se incluye el URL de las mismas, para que el usuario final pueda acceder a ellas.

Mensajes auto-descriptivos: otra característica fundamental de las APIs, los mensajes de solicitud contienen todos los datos necesarios para que el servidor los procese, asimismo, las respuestas del servidor contienen los encabezados y códigos de estado estándar de HTTP para indicar cómo debería tratarlos el cliente: En el caso de la lectura de recursos recomendados, se requiere una solicitud via GET (aunque se permite POST debido a las limitaciones en la longitud de las solicitudes GET impuestas por algunos navegadores). En el caso del registro de nuevos usuarios, se requiere una solicitud via POST con autenticación http básica²⁴. Aunque podría parecer que la semántica de las solicitudes bajo los principios REST requeriría también que la actualización de los perfiles fuese una solicitud POST autenticada, nótese que es una tarea que no puede causar mayor daño a los datos de

²³<http://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html>

²⁴<http://www.w3.org/Protocols/HTTP/1.0/spec.html#AA>

usuarios de una aplicación: actualizaciones deliberadas de preferencias no consisten en sí amenazas a la seguridad, téngase también en cuenta que, si la actualización de perfil requiriese autenticación, la contraseña estaría expuesta en el código del lado del cliente en interfaces web que hagan solicitudes con la técnica AJAX.

Sistema en capas: inherente al *framework* elegido (django), separa el almacenamiento y el procesamiento de las solicitudes, de manera que todos los componentes del sistema del lado del servidor tienen el acoplamiento más bajo posible.

Code-on-demand: el *widget* opcional es provisto por el servidor como un script de javascript al cliente, de manera que se pueda expandir la funcionalidad.

5.4.2. Implementación de las interfaces opcionales

Como ya se mencionó, se decidió crear una interfaz que los implementadores de páginas web pueden incluir fácilmente. El diseño de ésta ya fue discutido en la sección de diseño de interfaz (4.2.1). Se decidió que la inclusión del widget se haría mediante un método que recibiese un objeto de javascript con los parámetros que el usuario desee (siendo éstos la clave de aplicación, el identificador único de usuario y las opciones de obtención de datos y retroalimentación descritas en el anexo [manual de implementador]).

En cuanto a la opción de registrar usuarios desde el sitio del servicio, se creó una pequeña interfaz que permite -luego de ingresar las credenciales de autenticación- introducir varios usuarios a registrar de una sola vez, así como ver qué usuarios están registrados y eliminar. (Cabe recalcar que toda esta funcionalidad está disponible a través de la API).

5.4.3. Detección de cambios en el contexto local

Para la interfaz opcional, el *widget*, se agregó un componente de detección de cambios en el componente de interfaz gráfica que el implementador elija (véase el manual de implementación para más detalles al respecto). Los cambios en el texto se monitorean cada 5 palabras -este factor se ha establecido en 5- y son analizados con un algoritmo de diferencia textual, en específico, el propuesto en ?. Acto seguido, se toman estas diferencias para calcular la distancia de levenshtein²⁵ entre el nuevo contenido y el anterior, y si esta distancia es mayor que un umbral aceptable (establecido para las pruebas preliminares en un 60 %), se considera que el texto ha cambiado lo suficiente como para contener nuevos conceptos que podrían resultar en nuevas recomendaciones. El algoritmo 2 resume este procedimiento.

5.4.4. Detección de términos clave

Un componente clave de la recomendación es la detección de la necesidad de información latente. Siguiendo el camino marcado por ?, se optó por un componente que extraiga términos clave del texto. Para este fin, se implementó un método que recibe

²⁵La cantidad de cambios -adiciones, borrados y sustituciones de caracteres individuales- que debe sufrir una cadena de texto para convertirse en otra, métrica propuesta en ?

Algoritmo 2 Detección de cambios en el contexto local

Entrada: contenido nuevo, contenido anterior,

Salida: si es necesario o no obtener nuevas recomendaciones

```
// eliminar espacios sobrantes y urls
1: contenidoNuevo = limpiarCadena(contenidoNuevo)
2: if abs(numPalabras(contenidoNuevo) - numPalabras(contenidoAnterior)) <  $\delta$  then
3:   return false
4: end if
5:  $\Delta$  = levenshtein(diff(contenidoAnterior, contenidoNuevo))
6: return ( $\Delta/\text{contenidoNuevo.longitud}$ )  $\geq 0,6$ 
```

el susodicho texto y el lenguaje en el que está escrito; luego, éste método utiliza una librería del lenguaje de programación python para extraer palabras clave²⁶ para documentos en inglés, y, para los demás lenguajes, (y opcionalmente para los que se pueden analizar con la librería previamente mencionada) se utiliza, por elección de quien utilice el sub-componente, uno de los siguientes servicios web:

1. Alchemy²⁷,
2. Yahoo term extraction²⁸
3. Wordsfinder²⁹.
4. Tagthe³⁰
5. OpenCalais³¹
6. Extractor³²

El sub-componente está abierto a la inclusión de más servicios web o librerías del lado del servidor a utilizar.

En la presente implementación, para construir una consulta se utiliza el método de extracción de términos clave previamente mencionado para obtener una lista que luego se une en una sola frase. Es importante notar que el estado presente del proyecto no es factible para un ambiente de producción, ya que los servicios web tienen límites de uso y/o pueden bloquear las solicitudes si se excede un umbral.

Para la implementación final, aunque se crearon módulos para comunicarse con todos los servicios antes mencionados, se decidió utilizar el servicio *tagthe* por defecto. Esto

²⁶la librería en cuestión es topia.termextract: <http://pypi.python.org/pypi/topia.termextract/>

²⁷www.alchemyapi.com/api/keyword/textc.html

²⁸<http://developer.yahoo.com/search/content/V1/termExtraction.html>

²⁹wordsfinder.com/api_Keyword_Extractor.php

³⁰<http://tagthe.net/fordevelopers>

³¹<http://www.opencalais.com/documentation/calais-web-service-api/>

³²<http://www.picofocus.com/DemoEngine/dbiExtractorDemoWebService.aspx?op=ProcessXtraction>

porque, en primer lugar, la calidad de los términos extraídos con éste es superior a los demás (puede lidiar con caracteres unicode y tiene una orientación al procesamiento de lenguaje natural), asimismo, el servicio no tiene límite de solicitudes. En caso de que una solicitud a este servicio fallase, el sistema tratará de utilizar el servicio de *yahoo*, éste, aunque más robusto que todos, utiliza un enfoque estadístico y no considera caracteres unicode (al menos no en las pruebas de evaluación); en último lugar, se recurre a una llamada al servicio de *alchemy*, que, aunque orientado al lenguaje natural, no presentó la misma calidad que *tagthe* de resultados en las pruebas de evaluación. Estas pruebas se hicieron comparando los términos que cada servicio extraía de textos en lenguaje natural con los que seres humanos extraían de los mismos. Aunque se diseñó una encuesta para este efecto, el factor tiempo no permitió que ésta se analizase, por lo que el criterio utilizado fue únicamente el del autor.

5.4.5. Re-ordenamiento de los resultados

El perfil del usuario cumple su papel principal a la hora de re-ordenar los componentes que el componente de búsqueda devuelve en respuesta a la consulta construida por el extractor de términos clave. Para cada resultado, el componente de búsqueda calcula un *puntaje de relevancia* con valores en el rango $[0, 1]$. El servicio re-ordena, en orden no-creciente, este conjunto de resultados asignándoles un nuevo puntaje de relevancia según la siguiente fórmula (adaptada de ?):

$$r.puntaje = \gamma * r.puntaje + (1 - \gamma) * IS(r.concepto) \quad (5.5)$$

Nótese que esta fórmula, de manera similar a la que evoluciona los perfiles, permite que los conceptos que tienen un puntaje de preferencia de parte del usuario logren llegar a posiciones con mayor puntaje de relevancia que aquellos que no, a la vez que todos son atenuados mediante un *factor de penalización* γ . Este factor se ha establecido en 0.3 . Cabe notar, también, que se presupone la existencia de la función $IS(perfil, concepto)$, que devuelve el puntaje de preferencia dado por el perfil al concepto dado, o un valor de cero si el perfil no tiene un puntaje de preferencia para el concepto. El re-ordenamiento se puede encontrar en el algoritmo 3

Algoritmo 3 Re-ordenación de los resultados de búsqueda según el perfil del usuario

Entrada: resultados con puntajes de relevancia, perfil de usuario actual

Salida: resultados con nuevos puntajes de relevancia según las preferencias del perfil

- 1: **for all** resultado \in resultados **do**
 - 2: resultado.puntaje = γ *resultado.puntaje + $(1 - \gamma) * IS(perfil, resultado.concepto)$
 - 3: **end for**
 - 4: resultados = reverse(sort(resultados))
-

6 Conclusión

En este capítulo se presentan, en la visión general, los resultados y aportes del proyecto; se evalúa críticamente el trabajo, identificando las debilidades y fortalezas del enfoque tomado, luego se discuten las posibilidades de trabajo futuro - incluidas alternativas de comercialización- y se concluye con algunas recomendaciones a futuras generaciones de estudiantes de ingeniería en sistemas computacionales y la facultad.

6.1. Visión General

El presente proyecto exploró el concepto de ofrecer un servicio de recomendación para enriquecimiento de tareas de producción de información. Dado que la mayor parte de las personas que crean datos textuales para ser leídos por otras lo hacen mediante herramientas informáticas y tienen acceso a internet, se implementó una interfaz de programación de aplicaciones (API) que permitiese a los desarrolladores de tales herramientas agregar esta funcionalidad a sus soluciones de software haciendo unas cuantas llamadas mediante el protocolo HTTP, en lugar de crearla por sí mismos: Los implementadores registran su aplicación en el sitio web del servicio y obtienen una clave única y credenciales de autenticación. Por razones de seguridad, y para ofrecer un servicio más confiable, la responsabilidad de registrar y administrar los usuarios para los cuales se crearán perfiles de personalización de recomendaciones se dejó a los implementadores, proveyéndoles una interfaz en el sitio del servicio para llevar a cabo esta tarea manualmente y llamadas a la API para hacerlo programáticamente (requiriendo ambos casos autenticación). Para aplicaciones web, se creó un archivo de javascript cuya inclusión permite la llamada a un método de inicialización que agrega a la página donde se incluya el archivo una interfaz donde se presentarán recomendaciones para el usuario actual. Y, alternativamente para aplicaciones web y como única opción para aplicaciones no basadas en hipermedia, la obtención de recomendaciones y actualizaciones de los perfiles se hace mediante llamadas a la API.

El software se desarrolló utilizando como componentes fundamentales algunas librerías re-utilizables para tareas como indexación, clasificación y recuperación de información, así como el uso de un *framework* de desarrollo web -concretamente, django- para eliminar la necesidad de manejar manualmente tareas de bajo nivel y permitir la separación en capas de la arquitectura. Para representar el conocimiento semántico del sistema se utilizó una ontología, en concreto, la sub-jerarquía de ciencia de la jerarquía ontológica provista por el *Open Directory Project*. De cada una de las sub-categorías en la rama elegida se obtuvo la colección documental, misma que se utilizó para entrenar un clasificador que permitiera identificar los conceptos involucrados en el texto introducido por usuarios finales. Los perfiles de usuario se representaron como instancias de ontologías anotadas con

puntajes de preferencia. Para ofrecer recomendaciones, se utilizaron librerías y servicios web que extrajesen los términos clave de los textos provistos por los usuarios finales y transmitidos al servicio mediante llamadas a la API, acto seguido, los cuales se utilizan para hacer búsquedas en la colección documental, para luego re-ordenar estos resultados en función de su similitud con los conceptos presentes en los perfiles de los usuarios. En la interfaz opcional que se puede obtener mediante la inclusión del *widget* de javascript se implementó una rutina de supervisión de cambios en el texto introducido por el usuario final, de manera que se renovaran las recomendaciones cuando evolucionase el contenido. La representación de la ontología se creó con la extensibilidad a otros lenguajes en mente utilizando el modelo de la ontología original del ODP: el idioma principal es inglés, y las categorías alternas se refieren a la misma instancia que la original en inglés. Esto permitió que, si un usuario introduce texto en un idioma, este texto tiene una influencia trans-lingüe en la evolución del perfil, de forma que lo aprendido del usuario en esa sesión será totalmente útil si en otra decide escribir en otro de los idiomas soportados.

Se considera que el presente proyecto contribuye a las áreas de investigación de los sistemas de recuperación de información oportuna y de recomendación al proveer un servicio sobre el cual nuevas soluciones de software se puedan fundamentar. Asimismo, es una implementación donde se prueba el concepto de recomendación orientada al conocimiento, donde éste se modela mediante ontologías y los perfiles del usuario evolucionan heurísticamente, de modo que los investigadores pueden utilizarlo para evaluar la utilidad práctica de este enfoque.

6.2. Observaciones y evaluación

En primer lugar, se debe hacer notar que, lamentablemente, no se lograron hacer experimentos con usuarios finales para descubrir qué tan útil es en realidad el enfoque de recomendaciones en tiempo real -en términos de reducción del esfuerzo e impacto en la usabilidad; el aprendizaje de perfiles trans-lingüe, y las elecciones algorítmicas correspondientes a la creación, uso y evolución automática de perfiles. Así, este tipo de estudio se habrá de relegar para trabajo futuro, para contribuir teóricamente a las áreas de investigación de recuperación oportuna de información y sistemas de recomendación.

La obtención de la ontología y la colección documental se hizo partiendo de la premisa, para esta primera incursión en el concepto, no se iba a necesitar actualizar o cambiar la ontología ni incrementar la colección documental, de manera que los componentes de software creados para manejar ésto fueran hechos para manejar una sola vez una gran cantidad de datos. Sin embargo, debió haberse hecho con la evolución de ambos repositorios de información en mente, en lugar de relegar esto a un trabajo futuro: manejando varias veces cantidades relativamente pequeñas de datos.

El componente de búsqueda incurre en un fallo conceptual: la librería de terceros que se utilizó está basada en documentos almacenados en bases de datos, de manera que los sustitutos documentales tuvieron que incluir el texto propio de los documentos, creando redundancia al tener el mismo texto en los documentos originales y los sustitutos documentales (además del texto que queda en los índices de la herramienta de recuperación

de información). Un enfoque mejor, aunque más caro en términos de desarrollo y tiempo, hubiera sido uno como el que se tomó para clasificar categorías: el súper-documento de categorías sólo existió a la hora de indexar, mas en realidad sólo se almacenan los datos necesarios para recuperación de información y los documentos originales.

Para la extracción de términos, sólo se utilizó una librería re-utilizable para el idioma inglés, para otros idiomas, dada la ausencia de una implementación en el lenguaje de programación elegido de una librería de extracción de términos, se optó por servicios web, esto, aunque práctico, agregó riesgos al proceso de extracción de términos dadas las posibles fallas de esos servicios y las restricciones que imponen (algunos limitan el tamaño del texto a analizar o el número de solicitudes al día). Asimismo, dado que las llamadas son a servidores remotos, el tiempo total de la obtención de recomendaciones para idiomas distintos al inglés tiene una latencia considerablemente mayor (en algunas pruebas preliminares, por ejemplo, una llamada al servicio para textos en inglés requería un total de 400 milisegundos, pero para textos en español se llegaron a requerir hasta dos segundos).

La librería de recuperación de información elegida era idónea a primera vista: permite ponderar campos específicos de los sustitutos documentales (como dar más peso al título, por ejemplo), almacenar documentos en distintos idiomas en una misma colección y ofrece una API de nivel medio flexible, sin embargo carece de documentación de calidad y algunas de las características que promete no están en realidad implementadas (la reducción a raíces para múltiples idiomas, aunque implementada en ciertas partes del código y prometida en la documentación no funciona, por ejemplo) y el mecanismo de cargado de índices es ineficiente: pretende cargar los índices en su totalidad a la memoria de acceso aleatorio, dependiendo del sistema operativo para esta tarea, causando errores en las aplicaciones que dependen de ella si no se dispone de suficiente memoria y el espacio que el sistema operativo asigna a paginación se acaba (en sistemas operativos linux, los procesos que abusan de memoria son sencillamente terminados).

Puesto que la etapa de re-ordenamiento de resultados para personalizar recomendaciones se ejecuta en tiempo real y existen restricciones prácticas que establecen que un servidor no debería tardar demasiado en responder a una solicitud (además que, como se mencionó en la teoría, para una tarea secundaria más de diez segundos de latencia se consideran una falla de comunicación desde el punto de vista del usuario) se decidió re-ordenar los resultados sólo tomando en cuenta la categoría de los mismos y el puntaje de preferencia en el perfil, obviando la similitud entre ésta y el contexto del usuario, que quizá daría resultados más exactos, puesto que este tipo de análisis requeriría más tiempo de respuesta al tener que categorizar el texto entero.

Para la evolución de perfiles, se consideró que la similitud de un documento y la categoría bajo la cual está clasificado es de un 100 %, esto debido a que se asumió los documentos son obtenidos del ODP, donde están clasificados por seres humanos, y el criterio automático del clasificador, aunque quizá hubiera reportado ciertas ganancias de exactitud, perdería las connotaciones subjetivas del criterio humano, valioso para simular conocimiento semántico.

La evolución de los perfiles es puramente automática, esto libra al implementador de tener que crear una interfaz para edición de perfil, mas priva al sistema en su totalidad

de la personalización humana por parte de usuarios finales y agrega todas las molestias de los sistemas que pretenden simular el criterio humano, como la sobre-generalización.

Para la agregación de múltiples usuarios y la evolución de perfiles se utilizó una cola de tareas, permitiendo el procesamiento asíncrono de éstas. Sin embargo, se introdujo el riesgo de que una tarea no fuese procesada, o el orden en que las evoluciones del perfil se haga incorrectamente, además de los problemas típicos introducidos por la concurrencia. Se debe recalcar que el software utilizado para este efecto está especializado en la tarea, por lo que ofrece protección básica para situaciones peligrosas en concurrencia y mecanismos de re-intento; pero riesgos de seguridad y confiabilidad subsisten, por ejemplo, en pruebas preliminares, se logró burlar el límite nominal de usuarios permitidos por aplicación al ejecutar una agregación múltiple (asíncrona) con el proceso de administración de tareas apagado, luego insertando más usuarios de manera sincronizada mediante la llamada de inserción individual hasta llegar al límite y por último activando el proceso de tareas: éste guardó el número de usuarios que había *antes* de las otras inserciones, efectivamente sobrepasando el límite.

El servicio se implementó apegándose donde fuera útil a la filosofía de diseño REST y la semántica inherente al protocolo HTTP, de ahí que operaciones que involucren cambios directos en la base de datos sólo se permitan mediante solicitudes autenticadas y mediante el método POST (en realidad, se debió haber utilizado PUT para la agregación de usuarios y DELETE para la eliminación, pero no todos los exploradores y librerías implementan estos métodos). Al requerir que las llamadas de obtención de recomendaciones y actualización de perfiles causaran siempre búsquedas en la base de datos (para obtener el usuario y la aplicación) en lugar de utilizar un sistema de sesiones y asumir persistencia de estados entre solicitudes (por ejemplo, sólo requiriendo que *una* llamada a la solicitud de recomendaciones tuviera presente los parámetros y suponiendo que, si faltasen en las subsiguientes, era porque la misma aplicación las solicitaba para el mismo usuario anteriores) quizá se sacrificó rapidez de respuesta, pero se hizo con el fin de permitir la escalabilidad en el futuro y proveer una interfaz más simple de implementar para los desarrolladores.

La confiabilidad se tuvo presente durante el desarrollo: para la extracción de términos en idiomas diferentes al inglés, si uno falla, se recae al uso de otros dos antes de responder, de manera que se reduzca la cantidad de solicitudes de recomendaciones fallidas (aunque este escenario se trató de evitar en la medida de lo posible, dada la latencia que introducen tal cantidad de llamadas a servicios web).

El lenguaje de programación utilizado, python, aunque práctico y muy poderoso, al ser de mayor nivel que otros lenguajes como C o C++, puede reducir el desempeño de aplicaciones como la desarrollada en este proyecto, que lidian con grandes cantidades de datos; de igual manera este lenguaje, al menos en la versión utilizada para el desarrollo -2.6; a diferencia de otros lenguajes utilizados para software similar (como java o C#), asume que las cadenas de caracteres están codificadas en ASCII por defecto, causando problemas serios -que en algunos casos retrasaron el desarrollo- cuando se procesa texto en lenguajes que requieren codificación UNICODE (como el español).

6.3. Trabajo futuro

El trabajo a futuro se puede dividir en tres áreas: desarrollo, análisis de implementaciones, comercialización. Las siguientes secciones detallan cada una de éstas.

6.3.1. Desarrollo

La implementación modular del servicio tiene la ventaja que cualquiera de los cambios propuestos a un componente a continuación tendrían poco o ningún impacto en los demás. El trabajo futuro se procura redactar aquí de manera que vaya de lo más importante en los primeros párrafos a lo meramente deseable al final.

El primer paso en desarrollo comprendería el diseño de un componente de evolución de la colección documental y la ontología. Respecto a la ontología, se podría mantener el modelo actual y mantenerla al día utilizando las actualizaciones semanales del *RDF* de la jerarquía ontológica del *ODP*. Se podrían, también, agregar nuevas fuentes de ontologías y permitir a los implementadores elegir el dominio de conocimiento u ontología de donde desean recibir recomendaciones. Asimismo, se podría cambiar radicalmente el modelo y depender de APIs o servicios de proyectos como freebase.com para la adquisición del conocimiento semántico. En cuanto a la colección documental, se podría seguir obteniendo documentos del *ODP*, lo cual tiene la ventaja de evitar una fase de clasificación, ya que en el *ODP* los documentos ya están agrupados bajo una categoría. De manera similar a la ontología, se podrían agregar nuevas fuentes parametrizables de documentos para dominios de conocimiento específicos, en este caso, se habría de construir un componente más especializado de clasificación para asignar categorías a los documentos.

Dado el polimorfismo de la información útil en la actualidad, el servicio debería poder ofrecer más hipermedia que la actual (HTML y PDF), como imágenes, video o archivos de sonido, siempre y cuando esto tuviera en cuenta que la filosofía fundamental del desarrollo es no distraer al usuario de su tarea principal, sino enriquecerla sin intrusión.

Se podría implementar un modelo de seguimiento histórico de sesiones, de manera que se sepa qué documentos han sido recomendados recientemente: así, se podría tomar en cuenta la última vez que un recurso se sugirió y dar preferencia a otros que no se hayan recomendado o que se hayan recomendado más allá de cierto umbral temporal; con esto se incrementaría la *serendipia* de los resultados.

Las políticas de registro de usuario podrían revisarse para normalizar los tipos de identificadores únicos aceptables (por ahora, cualquiera sirve, como códigos, nombres de usuario o direcciones de correo electrónico), sin embargo, quizá sería más sencillo para ambas partes (servicio e implementadores) que se exigiese uno solo -siendo el correo electrónico una opción recomendada.

La interfaz opcional (*widget*) se puede reescribir para ser aún más adaptable a las necesidades del implementador: múltiples campos de entrada de texto para el análisis de contexto local, cambios en los temas cromáticos y más opciones para provisión de retroalimentación y uso de las recomendaciones son algunos ejemplos.

Se podría construir un componente de clasificación más efectivo: en lugar de utilizar

un modelo simple de recuperación estadística de información para obtener categorías similares a un documento, aprovechar técnicas de aprendizaje de máquina para obtener clasificaciones cuyos criterios sean más efectivos que los que un modelado de lenguaje simple puede lograr.

Un modelo semántico de recuperación de información se podría utilizar para obtener los documentos a recomendar, de esta manera, en lugar de incurrir en el paso extra de la extracción de términos clave, se podrían encontrar documentos con un contenido relacionado al contexto local a nivel de significado, reportando -quizá- mejoras en sesiones donde el perfil aún no contiene suficientes datos (reduciendo aún más el problema de arranque en frío del cual sufren los sistemas de recomendación y que el uso de una ontología palia hasta cierto punto). Si se desarrollase un modelo semántico de recuperación de información, quizá hasta se podría obviar la etapa de personalización de los resultados teniendo como entrada el perfil del usuario: de ahí que este componente se pudiera llamar de *recuperación personalizada de información*.

Los enfoques de evolución de perfiles y personalización se podrían revisar: la personalización podría expandir las consultas a la colección documental mediante términos de sesiones anteriores o utilizando otras técnicas de re-escritura de consultas. La evolución podría utilizar técnicas de procesamiento de lenguaje natural para una identificación más exacta de las preferencias del usuario.

6.3.2. Análisis de implementaciones

Con el fin de descubrir la verdadera utilidad del servicio aquí descrito, se deberían conducir pruebas entre usuarios de diversas implementaciones, como ser sistemas de administración del contenido y *plug-ins* para procesadores de texto o navegadores web.

Con estos experimentos, se podrían responder las siguientes preguntas de hipótesis:

- ¿Cuál es la ganancia del uso de una ontología personalizada para el re-ordenamiento de resultados de búsqueda frente a los algoritmos convencionales de contexto local?
- ¿Qué tan efectivo es el enfoque trans-lingüístico para la evolución de los perfiles?
- ¿Existe una relación causal entre el tamaño de la colección documental -y la ontología básica- y la ganancia de utilidad en las recomendaciones a usuarios?
- ¿Es un enfoque heurístico/estadístico suficiente para la construcción automática de consultas de usuarios, o es necesaria la aplicación del procesamiento de lenguaje natural?
- ¿Cómo afecta una interfaz incremental la experiencia del usuario (frente a una interfaz más invasiva)?
- ¿Existe una ganancia de utilidad en la aplicación de un sistema de recomendación, frente a enfoques más convencionales, como la provisión de una interfaz de búsqueda o una búsqueda automática carente de memoria de uso?

- ¿Qué tanto se minimiza el esfuerzo frente al enfoque tradicional de escritura de *buscar-y-luego-escribir*?

6.3.3. Comercialización

Para obtener una ganancia monetaria de este servicio, se podría seguir el ejemplo de otros similares y tener en cuenta el verdadero costo de uso de los recursos que están involucrados en el mismo.

- Como muchas aplicaciones actuales, se podría imponer un límite de usuarios registrados para aplicaciones que hagan uso gratuito del servicio, si se requiere agregar más usuarios, habría que pagar una tarifa fija.
- Otra opción, complementaria quizá a la anterior, es restringir el número de llamadas al servicio por día para implementadores no-comerciales (el servicio de extracción de términos de yahoo, por ejemplo, establece un límite de 5,000 solicitudes por dirección IP¹).
- Una opción interesante de comercialización es la que implementa *zemanta*²: ciertos sitios pueden pagar por proporcionar contenido y ser recomendados y por el número de recomendaciones en las que aparecerán; sin embargo, el motor de recomendaciones de zemanta no da preferencia a éstos y los marca como “contenido promocionado”, lo cual demuestra respeto a la integridad de la filosofía inicial.

6.4. Recomendaciones

- En la actualidad, muchos desarrolladores de software, en especial aquellos con escasa experiencia, tienden a pensar que los recursos son cuasi-infinitos: los programas que uno escribe en sus tiempos de estudiante -en especial en lenguajes de alto nivel- casi nunca llevaron a su límite la memoria, el almacenamiento o el procesamiento; sin embargo, cuando se trabaja con grandes cantidades de datos que deben ser procesadas en períodos cortos de tiempo, estas limitaciones se vuelven evidentes, por lo que escribir código correcto, medirlo y luego, si es necesario, escribir código óptimo se debe volver una práctica personal arraigada y no solo una curiosidad de las ciencias computacionales.
- Para el tiempo en el que este documento se escribe, la carrera de ingeniería en sistemas computacionales carece de preparación formal para el desarrollo y despliegue de aplicaciones web. Existe, claro, una clase electiva llamada “desarrollo de aplicaciones web” que es, irónicamente, *mandatoria* para la carrera de ingeniería en sistemas electrónicos. A pesar de ello, al menos para cuando el autor curso las asignaturas, ciertos docentes se han esforzado en incentivar la investigación de esta área del desarrollo de software en clases como teoría de bases de datos y análisis y

¹<http://developer.yahoo.com/search/rate.html>

²<http://www.zemanta.com/faq/#faqid-79>

diseño de sistemas; sin embargo, es el parecer del autor que el diseño, desarrollo y despliegue de aplicaciones web y basadas en hipermedia aún se ve en la universidad como una más de “esas cosas que uno aprende por su cuenta” (al mismo nivel, quizá, de la creación de *scripts* en Bourne Shell y el uso de una interfaz de línea de comandos de sistemas *nix). Mas la tendencia actual es a la computación distribuida, la arquitectura orientada a servicios y las aplicaciones híbridas y puramente basadas en internet. La universidad, que se precia de ir a la vanguardia, debería incluir esto en la preparación de los ingenieros en sistemas computacionales, en lugar de permitir que unos pocos lo aprendan bien por su cuenta (gracias a la auto-disciplina y curiosidad o la simple necesidad laboral) y la mayoría terminen diseñando sitios empíricamente sin apegarse a estándares ni metodologías de desarrollo y despliegue.

- Asimismo, aunque existe una clase de ingeniería de software, el currículo actual lleva a los estudiantes a cursarla *después* de la asignatura de análisis y diseño de sistemas, por lo que actividades clave para el éxito de productos de software como la planificación y la estimación de costos se aprenden *a posteriori*: primero se analiza, diseña e implementa el primer proyecto formal de un alumno y *luego* se descubre cómo se debería haber hecho. Esto sólo perpetúa el desdén de los desarrolladores por las prácticas de gestión de proyectos. Como universidad que forma profesionales emprendedores, el modelo de desarrollador de software caótico e incapaz de tratar con otras personas o de crear documentación remotamente inteligible debería ser cosa del pasado; sin embargo, la asignación de proyectos individuales y la enseñanza tardía de métodos de planificación, análisis y diseño solamente perviven este estereotipo.
- Por último, en clases donde se desarrollen proyectos colaborativos se debería alentar al alumno (y no sólo mediante un consejo a *sotto voce*, sino, quizá, mediante laboratorios opcionales o algo de esa especie) a utilizar herramientas de control de versiones, para evitar malos entendidos y promover buenas prácticas de desarrollo de software (la frase “te paso la nueva versión del código por usb (o lo que sea), copiá y pegá esto y aquello” debería quedar proscrita de las conversaciones entre alumnos de la carrera).

A Correos electrónicos de comunicación con la facultad

A.1. Requisición de recursos de laboratorio

Para la obtención, procesamiento y uso del *corpus* documental, durante la etapa de desarrollo y la presentación, se solicitó el uso de un servidor del laboratorio destinado a la carrera de ingeniería en sistemas computacionales.

A.1.1. Copia del correo electrónico de solicitud

De: Luis Felipe Borjas <lfborjas@unitec.edu> para Jorge García <jorge.garcia@unitec.edu>, Daniel Martínez <dmartinezh@unitec.edu>

Fecha: 27 de abril de 2010 03:11

Asunto: Requisición de recursos de laboratorio enviado por unitec.edu ocultar detalles 03:11 (0 minutos antes)

Estimados,

Para mi proyecto de graduación necesito un cuerpo relativamente grande de documentos, los cuales pienso obtener de dmoz.org, un directorio abierto que categoriza sus recursos. Para ello, es necesario que ejecute una araña web y procesos que tomen lo que ésta obtenga, lo analicen y almacenen en una base de datos. Como es evidente, esto requiere una cantidad de procesamiento, memoria y almacenamiento tales que sería poco factible ejecutarlo en mi propia computadora sin retrasarme considerablemente en mis actividades laborales y académicas, por lo que solicito a ustedes el uso de uno de los servidores del laboratorio que está disponible para los alumnos de ingeniería en sistemas computacionales.

En dicho servidor, necesitaría de los siguientes aspectos:

- Un sistema operativo basado en unix (de preferencia ubuntu linux).
- Python 2.6 y permisos de escritura y ejecución en la carpeta que éste use para la instalación de librerías y acceso a ésta como súper-usuario (quizá con la presencia del administrador de laboratorio), o, en su defecto, el punto siguiente.
- Un usuario que tenga la potestad de leer, escribir y ejecutar archivos en una carpeta donde almacenaría todos los resultados del proceso de obtención de recursos y obtendría y ejecutaría los "scripts" que habrían de llevar a cabo el proceso.

- Garantía razonable de tiempo, flujo eléctrico y ancho de banda suficientes como para poder culminar la ejecución de los procesos sin demasiados inconvenientes.

Agradeciendo de antemano su asistencia en este asunto,
Luis Felipe Borjas

A.1.2. Copia de la respuesta de la facultad

De: Jorge Garcia <jorge.garcia@unitec.edu>

27 de abril de 2010 14:31

Para: Luis Felipe Borjas <lfborjas@unitec.edu>

Cc: dmartinezh@unitec.edu

Hola,

favor realizar la instalación requerida en un servidor virtual del laboratorio.
El servidor quedaría inhabilitado una vez haya tenido la terna.

Jorge.

A.2. Solicitud de acceso al servidor virtual desde fuera del laboratorio

Una vez provisto para el desarrollo del proyecto un servidor virtual, se encontró la dificultad de no poder acceder al mismo sino desde dentro del laboratorio para estudiantes de sistemas computacionales, dado que sería redundante y poco factible tener que volver a obtener y procesar el *corpus* documental en la computadora personal del autor para el día de presentación en la terna de graduación, se solicitó a la facultad acceso intra-universitario al mismo.

A.2.1. Copia del correo electrónico de solicitud

De: Luis Felipe Borjas <lfborjas@unitec.edu>

1 de mayo de 2010 13:53

Para: Jorge Garcia <jorge.garcia@unitec.edu>

Cc: Daniel Martínez <dmartinezh@unitec.edu>, Gerardo Romero <gromerom@unitec.edu>

Buenas tardes, Ingeniero

Mi proyecto de graduación, como le comenté en el correo de requisición de recursos de laboratorio, requiere cantidades considerables de datos (los datos que usaré en esta etapa y para la terna, por ejemplo, ocupan, en crudo -es decir, sin procesamiento alguno- 3.7 Gb de espacio) Y, luego del procesamiento que he de hacer, usaría, en el peor de los casos, unos 27 Gb de información en total (antes, claro, de una posible purga de datos redundantes). Con esto en

vista, mi mayor preocupación es que, para el día de mi terna de graduación, tenga que mover - ¡y volver a procesar!- esta ingente cantidad de datos a mi computadora personal, puesto que sólo puedo acceder al servidor virtual que me fue proporcionado desde dentro del laboratorio de sistemas.

Mi solicitud, entonces, consiste en que, al menos para la fecha de la terna (y, muy preferiblemente, desde antes, para pruebas), se me facilite acceso a ese servidor desde otros lugares de la universidad. Solicito esto porque estoy consciente que el acceso desde fuera de la universidad es una tarea más difícil -aunque sería definitivamente preferible.


Agradeciendo su apoyo,

Luis Felipe Borjas

A.2.2. Respuesta de la facultad

En respuesta a la solicitud, se procedió a asignar una dirección IP pública al servidor de virtualización del laboratorio de sistemas computacionales, acto seguido, se configuró un enrutador para hacer redirección de puertos hacia la dirección IP asignada al servidor virtual.

B Plan de proyecto

| |  | Nombre | Duración | Inicio | Terminado | D |
|----|--|---|-----------------|---------------------|--------------------|---|
| 1 | | <input type="checkbox"/> Obtención de colección documental y ontología | 18 days? | 04-07-10 ... | 04-30-10... | |
| 2 | | Elección de ontología | 1 day? | 04-07-10 0... | 04-07-10 ... | |
| 3 | | Diseño de modelos: sustitutos documentales y categorías | 2 days | 04-08-10 0... | 04-09-10 ... | |
| 4 | | Obtención de ontología y procesamiento | 3 days | 04-12-10 0... | 04-14-10 ... | |
| 5 | | Implementación de araña web | 3 days | 04-12-10 0... | 04-14-10 ... | |
| 6 | | Obtención de datos documentales y categorización | 7 days | 04-15-10 0... | 04-23-10 ... | |
| 7 | | Implementación y entrenamiento del clasificador de categorías | 3 days | 04-26-10 0... | 04-28-10 ... | |
| 8 | | Configuración inicial del servidor de producción | 1 day? | 04-29-10 0... | 04-29-10 ... | |
| 9 | | Pruebas de búsquedas y categorizaciones manuales | 1 day? | 04-30-10 0... | 04-30-10 ... | |
| 10 | | <input type="checkbox"/> Implementación del componente de búsqueda | 9 days? | 04-26-10 ... | 05-06-10... | |
| 11 | | Elección de librería de indexación | 2 days | 04-26-10 0... | 04-27-10 ... | |
| 12 | | Indexación de documentos | 2 days | 04-28-10 0... | 04-29-10 ... | |
| 13 | | Elección de servicios de extracción de términos | 1 day? | 04-26-10 0... | 04-26-10 ... | |
| 14 | | Implementación de módulo de extracción de términos | 5 days | 04-27-10 0... | 05-03-10 ... | |
| 15 | | Creación de interfaz de prototipo web para pruebas de extracción de términos y búsqueda | 2 days | 05-04-10 0... | 05-05-10 ... | |
| 16 | | Pruebas de búsquedas y extracción de términos en interfaz web | 1 day? | 05-06-10 0... | 05-06-10 ... | |
| 17 | | <input type="checkbox"/> Implementación de componente de perfilado | 8 days? | 05-07-10 ... | 05-18-10... | |
| 18 | | Creación de modelos de perfil | 1 day? | 05-07-10 0... | 05-07-10 ... | |
| 19 | | Elección de técnica de representación, creación y evolución de perfil | 1 day? | 05-07-10 0... | 05-07-10 ... | |
| 20 | | Implementación de módulo de evolución de perfiles | 4 days | 05-10-10 0... | 05-13-10 ... | |
| 21 | | Ponderación de ontología | 2 days | 05-10-10 0... | 05-11-10 ... | |
| 22 | | Creación de interfaz de pruebas (graficación de perfiles y simulación de búsquedas) | 2 days | 05-14-10 0... | 05-17-10 ... | |
| 23 | | Pruebas de perfilado | 1 day? | 05-18-10 0... | 05-18-10 ... | |
| 24 | | <input type="checkbox"/> Implementación del servicio | 15 days? | 05-14-10 ... | 06-03-10... | |
| 25 | | Diseño de urls | 1 day? | 05-14-10 0... | 05-14-10 ... | |
| 26 | | Implementación de módulo de obtención de perfil | 1 day? | 05-17-10 0... | 05-17-10 ... | |
| 27 | | Implementación de módulo de autenticación | 2 days | 05-18-10 0... | 05-19-10 ... | |
| 28 | | Implementación de módulo de interpretación de solicitudes | 4 days | 05-18-10 0... | 05-21-10 ... | |
| 29 | | Implementación de componente de re-ordenamiento de resultados | 3 days | 05-24-10 0... | 05-26-10 ... | |
| 30 | | Diseño de interfaz del widget | 3 days | 05-24-10 0... | 05-26-10 ... | |
| 31 | | Diseño de plug-in javascript | 3 days | 05-24-10 0... | 05-26-10 ... | |
| 32 | | Implementación de componente de detección de cambios contextuales y retroalimentación | 4 days | 05-27-10 0... | 06-01-10 ... | |
| 33 | | Creación de interfaz de registro y módulo de registro | 3 days | 05-24-10 0... | 05-26-10 ... | |
| 34 | | Implementación y pruebas en el servidor | 2 days | 06-02-10 0... | 06-03-10 ... | |
| | | | | | | |
| | | | | | | |

C Manual del implementador

Manual para Implementadores

T-Recs: un servicio de recomendación basado en ontologías
Versión 0.0.1

9 de junio de 2010

Índice general

| | |
|--|-----------|
| Introducción | 3 |
| Audiencia | 3 |
| 1. Para empezar | 4 |
| 1.1. Obtener clave de aplicación | 4 |
| 1.2. Comprender cómo puede responder el servicio | 5 |
| 1.3. Decidir método de administración de usuarios | 7 |
| 1.3.1. Administración mediante el panel de control | 7 |
| 1.3.1.1. Adición de usuarios | 7 |
| 1.3.1.2. Listado y eliminación de usuarios | 7 |
| 1.3.2. Administración mediante llamadas a la API | 9 |
| 1.3.3. Gestión de credenciales | 9 |
| 2. Uso básico | 10 |
| 2.1. Configuración preliminar | 10 |
| 2.2. Configuración del widget | 12 |
| 2.2.1. Solamente obtener recomendaciones | 12 |
| 2.2.2. Obtener recomendaciones y actualizar perfil con retroalimentación implícita | 14 |
| 2.2.3. Utilizar retroalimentación explícita | 15 |
| 2.3. Comprender el comportamiento del widget | 16 |
| 3. Uso avanzado | 21 |
| 3.1. Métodos de gestión de usuarios | 21 |
| 3.1.1. Adición de usuarios individuales | 21 |
| 3.1.2. Agregación de múltiples usuarios | 22 |
| 3.1.3. Listado de usuarios | 22 |
| 3.1.4. Eliminación de usuarios | 23 |
| 3.2. Servicio de recomendación | 23 |
| 3.2.1. Obtener recomendaciones | 23 |
| 3.2.2. Actualizar perfiles | 24 |
| 3.3. Ejemplos de llamadas a la API | 25 |
| 3.3.1. En javascript (con la librería jquery) | 26 |
| 3.3.2. En python | 26 |
| 3.3.3. En java | 26 |

Introducción

En la actualidad, la gente que produce información para ser consumida por un auditorio considerable se ha multiplicado gracias a la emergencia del internet, las redes sociales y generaciones cada vez más acostumbradas a herramientas informáticas. La costumbre más común de los escritores de contenido contemporáneos, para encontrar ideas o fundamento para lo que escriben es utilizar motores de búsqueda para investigar hasta donde el tiempo o la motivación lo permitan y *luego* dedicarse a escribir su propio trabajo. Esta costumbre, sin embargo, tiende a producir trabajos de baja calidad o alto riesgo de plagio y contradicciones, debido a que las ideas evolucionan y se desarrollan sólo cuando se está escribiendo.

Para ello, se han propuesto sistemas que asistan al proceso de escritura haciendo recomendaciones de referencias en tiempo real, de modo que, si los escritores lo desean, las pueden consultar cuando lo necesiten, ahorrándose el esfuerzo de formular consultas en motores de búsqueda y desperdiciar tiempo valioso en refinarlas hasta dar con material útil.

T-Recs es un servicio que permite agregar esta funcionalidad a cualquier sistema, ya sea una aplicación web o de escritorio, ahorrando a los implementadores la tarea de desarrollarla por su cuenta.

El servicio se provee mediante una interfaz de programación de aplicaciones (API) a la que se puede acceder mediante el protocolo HTTP.

Para desarrolladores web existe la opción de incluir unas cuantas líneas de código javascript a las páginas donde se desee implementar la funcionalidad, para así ahorrarse el paso de desarrollar una interfaz y métodos que manejen las recomendaciones; los implementadores en otro tipo de aplicaciones deberán implementar las llamadas a la API y proveer a sus usuarios de una interfaz que las presente, el desarrollo de la misma puede seguir algunos lineamientos aquí presentados, para aprovechar al máximo el servicio.

Audiencia

El presente documento está dirigido a desarrolladores de software tanto novatos como experimentados. Para utilizar el servicio no es necesario utilizar ningún lenguaje específico, lo único necesario es que el desarrollador sepa cómo hacer solicitudes a URLs e interpretar las respuestas mediante el protocolo HTTP. El contenido de las respuestas del servidor, en la versión actual, estará codificado utilizando la notación de intercambio de texto JSON (*Javascript Object Notation*) por lo que también se requiere que el implementador conozca una manera de extraer información de un texto en JSON en el lenguaje que elija. Los implementadores web que deseen incluir el *widget* de javascript, deben tener conocimiento básico de javascript, html y selectores en la sintaxis de css.

1 Para empezar

Aquí se detallan los pasos preliminares que cualquier implementador deberá seguir antes de usar el servicio.

1.1. Obtener clave de aplicación

Tanto el *widget* como el uso de la API requieren que se cuente con una *clave de aplicación*. Ésta es una cadena de caracteres que identifica a cada aplicación registrada, por cada aplicación que implemente el servicio, el desarrollador deberá proveer una clave válida.

Al acceder a <http://trececs.com/register/>, se encontrará un formulario como el mostrado en la figura 1.1. Contiene los siguientes campos:

1. Nombre o url de la aplicación: el implementador deberá proveer un nombre *único* para la aplicación, si es una aplicación web, se recomienda utilizar el URL con el que se accede a la misma. Si una aplicación con el mismo nombre ya está registrada para uso del servicio, se hará notar al usuario mediante un reporte de error en el formulario.
2. Correo electrónico de contacto: aquí se deberá cuidar de proporcionar una dirección de correo electrónico válida y activa: toda la comunicación subsiguiente con el desarrollador se hará mediante ésta.
3. Verificación: para evitar el abuso de aplicaciones maliciosas, se requiere que se llene

The registration form consists of the following elements:

- Your app's url/name:** A text input field.
- E-mail:** A text input field.
- Are you human enough?:** A reCAPTCHA challenge showing the text "I'm havarti" with a distorted background image.
- Ready!:** A green button.

Figura 1.1: Formulario de registro

un campo de verificación *recaptcha*¹.

Una vez completo con éxito el formulario, el usuario será redirigido a la página de instrucciones de uso y el sistema enviará un correo electrónico a la dirección provista con la clave de aplicación y las credenciales de autenticación correspondientes a la aplicación. Las credenciales de autenticación consisten en un nombre de usuario (el nombre de la aplicación) y una contraseña generada aleatoriamente, permiten el uso seguro de llamadas a métodos sensibles como adición y eliminación de usuarios.

Si el correo electrónico no se recibe -antes de asumir que no se recibió, se recomienda revisar en la carpeta que algunos clientes asignan a correo no deseado, el implementador no debe intentar registrarse de nuevo, sino contactar al servicio.

1.2. Comprender cómo puede responder el servicio

Para todas las llamadas a la API, el servidor habrá de contestar con una respuesta http. Todas las llamadas requieren que se provea la clave única de la aplicación. En caso de que la solicitud sea correcta, el servidor contestará con una respuesta con código de estado 200 (éxito) y el contenido codificado en el formato de intercambio de datos JSON con los atributos pertinentes a cada llamada. Si el servidor responde con un documento con un código de estado diferente a 200, refiérase a la tabla 1.1 para determinar la causa, todos los errores tratan de apegarse a la semántica estándar de códigos de estado definidos² por la IETF (*Internet Engineering Task Force*, Grupo de Trabajo en Ingeniería de Internet).

Para permitir llamadas a la API desde aplicaciones que utilicen la técnica AJAX (*Asynchronous Javascript and XML*, Xml y javascript asíncronos), todas las llamadas pueden recibir un parámetro opcional adicional: *callback*, que especifica una función de retorno. De esta manera, provisto este parámetro, la respuesta del servidor contendrá el objeto JSON de la contestación original puesto como parámetro de la función especificada en *callback*. Esta es la *única* manera en la que aplicaciones que utilicen la técnica AJAX pueden acceder a la API, porque la política de mismo origen³ impuesta por los navegadores modernos prohíbe, por seguridad, que intercambios entre dominios distintos alternen código del lado del cliente (mas permite el intercambio de *scripts*, que es como los navegadores reconocerán esta respuesta, en lugar de una asignación de un objeto en código cliente, que es lo que se hace con datos en JSON por lo general). Esta técnica se conoce como JSONP⁴ (*JSON with padding*, JSON con relleno). El usuario deberá saber interpretar el *script* devuelto como una llamada a una función que deberá definirse en el lado del cliente, es de notar que la mayor parte de librerías de javascript contemporáneas tienen la capacidad de manejar automáticamente solicitudes con la técnica JSONP⁵.

¹recaptcha.org

²<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>

³Para más detalles, véase lo documentado en https://developer.mozilla.org/en/Same_origin_policy_for_JavaScript

⁴Para más detalles, consúltense <http://bob.pythonmac.org/archives/2005/12/05/remote-json-jsonp/> y <http://developer.yahoo.com/common/json.html#callbackparam>

⁵Mootools <http://mootools.net/docs/more/Request/Request.JSONP> y jquery <http://api.jquery.com/jquery.getJSON/>, para citar un par de ejemplos, tienen métodos especializados para ello.

1 Para empezar

| Código | Posible Causa | Solución |
|--------------------------------|---|---|
| 400 (solicitud mal-formada) | Uno de los parámetros requeridos por la llamada no ha sido provisto o el parámetro opcional general de función de retorno javascript no es un identificador válido. | Revisar la llamada e incluir los parámetros necesarios: el contenido de la respuesta contendrá los parámetros requeridos que hayan faltado. |
| 401 (no autorizado) | Fallo de autenticación (por ausencia de credenciales o credenciales incorrectas en una llamada que requiere autenticación básica http ⁶). | Incluir las credenciales o revisarlas. Si se sospecha que un tercero las ha cambiado, contactar al servicio. |
| 403 (acceso prohibido) | Sólo es devuelto al tratar de registrar usuarios. Sucede cuando se ha excedido el límite permitido. | Eliminar usuarios inactivos o solicitar una expansión personalizada del límite (esta característica no es ofrecida en la versión actual del servicio). |
| 404 (no encontrado) | Si la solicitud es a una URL válida del servicio, la causa se debe a la provisión de una clave única inválida o un usuario no existente para la aplicación que llama. | Proveer una clave única de aplicación válida o un identificador de usuario existente. Si no se está seguro de qué usuarios se tiene registrados, utilizar la llamada del listado de usuarios o el panel de control. Si se ha perdido la clave única, contactar personalmente al servicio. |
| 500 (error interno) | Cuando el servidor ha tenido un error en alguno de sus componentes y no puede llevar a cabo el proceso. | Este es el único error que el usuario no puede enmendar y no requiere mayor acción de su parte. El servicio cuenta con un mecanismo de auto-reporte de errores internos a los desarrolladores. |

Cuadro 1.1: Posibles errores en llamadas a la API

1.3. Decidir método de administración de usuarios

Es responsabilidad del desarrollador registrar los usuarios que recibirán recomendaciones. Las llamadas al método de obtención de recomendaciones, además de la clave de aplicación, requieren un identificador único para el usuario actual, este identificador debe corresponder con alguno previamente registrado para la aplicación.

Se recomienda utilizar la dirección de correo electrónico como identificador único de los usuarios, pero cualquier cadena de caracteres que identifique unívocamente a cada usuario en la aplicación implementadora servirá.

Existen dos alternativas de agregación: mediante el panel de control provisto en el sitio web del servicio y mediante llamadas a la API de administración de usuarios. Asimismo, se provee una interfaz para que cualquier implementador pueda cambiar la contraseña asignada a su aplicación.

1.3.1. Administración mediante el panel de control

Esta alternativa se recomienda sólo para aplicaciones con un número pequeño y limitado de usuarios o donde la membresía se determina manualmente. Para acceder al panel de control, diríjase a trececs.com/dashboard/. Téngase en cuenta que si no se ha autenticado, se le solicitarán sus credenciales de acceso.

1.3.1.1. Adición de usuarios

La interfaz del servicio permite la agregación de varios usuarios de una vez, mientras no se exceda el límite de usuarios. Para agregar usuarios, se deben llenar los campos de texto deseados con los identificadores únicos que la aplicación proveerá cuando se utilice. Es responsabilidad del implementador la gestión de usuarios dentro del límite, y el servicio no se responsabiliza por la mala administración de los mismos ni por errores tipográficos en la adición; mas el servicio se compromete a no divulgar la información introducida y aplicar las mejores medidas de seguridad y confiabilidad en el almacenamiento de los datos. La interfaz de adición de usuarios se puede apreciar en la figura 1.2. Como se puede observar, en la parte superior se indica el estado del número de usuarios registrados respecto al límite impuesto por la aplicación. El número de cajas de texto para introducción de identificadores está limitado por el límite actual (que es el límite total menos el número de usuarios registrados).

1.3.1.2. Listado y eliminación de usuarios

El panel de control también permite revisar los usuarios ya registrados en la aplicación y da la facilidad de eliminarlos. Téngase presente que, al tratar de eliminar un usuario aparecerá un diálogo de confirmación para evitar borrados accidentales. Para identificar esta interfaz, véase la figura 1.3

1 Para empezar

You have 8 users registered. The current limit is 10: you can add 2 more

Register Users **All Users**

Fill this list with the unique identifiers your application will be passing to the recommendation service. With these, we'll know which user is performing a task and give her/him personalized recommendations

Register

Figura 1.2: Adición de usuarios en el panel de control

You have 8 users registered. The current limit is 10: you can add 2 more

Register Users **All Users**

| User | Date Added | |
|----------------|------------|--------|
| utahraptor | 2010-06-01 | Delete |
| dromiceiomimus | 2010-06-01 | Delete |
| trex | 2010-06-01 | Delete |
| ryan | 2010-06-01 | Delete |
| joey | 2010-06-01 | Delete |
| dino | 2010-06-01 | Delete |
| letzen | 2010-06-01 | Delete |
| user | 2010-06-01 | Delete |

Figura 1.3: Listado y eliminación de usuarios en panel de control

El formulario muestra el texto "Correo electrónico:" en verde, seguido de un campo de entrada rectangular. Debajo de este campo hay un botón rectangular verde con el texto "reset" en blanco.

Figura 1.4: Formulario de solicitud de restablecimiento de contraseña

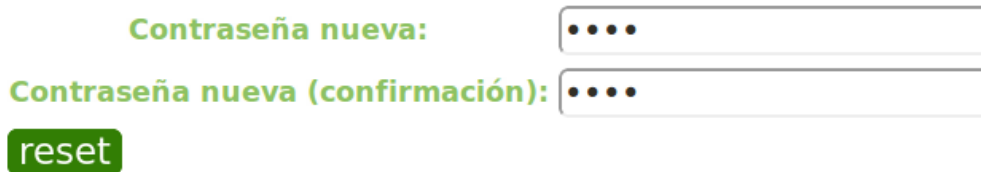
El formulario muestra el texto "Contraseña nueva:" en verde, seguido de un campo de entrada rectangular con tres puntos negros. Debajo de este campo hay el texto "Contraseña nueva (confirmación):" en verde, seguido de otro campo de entrada rectangular con tres puntos negros. En la parte inferior izquierda del formulario hay un botón rectangular verde con el texto "reset" en blanco.

Figura 1.5: Formulario de restablecimiento de contraseña

1.3.2. Administración mediante llamadas a la API

Se han provisto métodos de administración de usuarios en la API. El uso de estas llamadas es responsabilidad última del implementador, y él deberá saber manejar las credenciales de manera que sea difícil que usuarios mal-intencionados (o demasiado curiosos) las obtengan. Se recomienda no incluirlas en código del lado del cliente u ofuscarlas si fuera absolutamente necesario incluirlas en la aplicación final. Para una explicación detallada de estos métodos, refiérase a la sección 3.1.

1.3.3. Gestión de credenciales

Para acceder al panel de control o hacer llamadas de gestión de usuarios a la API, como ya se mencionó, la aplicación debe proveer credenciales de autenticación. Un nombre de usuario -que consiste en el nombre de aplicación usado para el registro- y una contraseña aleatoria son incluidas en el correo electrónico de confirmación de registro. Si por alguna razón el implementador pierde su contraseña o tiene razones para sospechar que ha sido robada, puede restablecerla accediendo a trececs.com/dashboard/password_reset/. Ahí se solicitará una dirección de correo electrónico que debe corresponder a la que se proveyó durante el registro (véase la figura 1.4). Acto seguido, el sistema enviará al correo provisto un mensaje conteniendo una URL única para el acto de restablecimiento. Siguiendo la dirección ahí provista, el implementador puede restablecer la contraseña para su aplicación (refiérase a la figura). Lamentablemente, si la dirección de correo electrónico de registro tuviese la seguridad comprometida, no existe aún un mecanismo alternativo de restablecimiento de contraseña.

2 Uso básico

La forma más expedita de comenzar a utilizar el servicio en una aplicación web es mediante la inclusión del widget. Éste es una interfaz que muestra las recomendaciones y hace las actualizaciones de perfiles automáticamente. Se requiere cierto conocimiento de javascript para configurarlo y de html para aprovecharlo.

Para demostrar cómo usarlo, se utilizará como referencia el documento html de la figura 2.1b (se puede ver cómo lo mostraría un navegador web en la figura 2.1a):

A continuación, se detallan los pasos a seguir para comenzar a utilizar el servicio en una aplicación web.

2.1. Configuración preliminar

1. Agregar las dependencias del widget. Antes de incluir el archivo que contiene la funcionalidad del widget, se requiere que, al menos, la librería *jquery* -se recomienda la versión 1.4.1 o una mayor- esté incluida (y opcionalmente *jqueryui* para efectos y animaciones, recomendada la versión 1.8.1). No es necesario bajarlas e incluirlas entre los archivos estáticos de la aplicación: es servida por google para este tipo de usos, y esta opción es la recomendada. Para incluirlas, entonces, se debe incluir, entre las etiquetas *head* del documento, las siguientes líneas de código:

```
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/
  jquery.min.js">
</script>
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jqueryui/1.8.1/
  jquery.min.js">
</script>
```

2. Agregar el archivo de javascript provisto por el servicio. Esto agregará la funcionalidad del *widget*. Agréguese mediante la siguiente línea de código entre las etiquetas *head* del documento:

```
<script type="text/javascript"
src="http://trece.com/static/js/service/recommender.js"
>
</script>
```

Editemos

Guardar

Referenciemos

- Se listan referencias aquí

(a)

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
2 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml">
4   <head>
5     <title>Cliente Simple </title>
6   </head>
7
8   <body>
9     <h2>Editemos</h2>
10    <form id="edition-form" action="someScript.php">
11      <textarea id="edition-area"></textarea>
12      <input type="submit" id = "edition-save" value="Guardar"/>
13    </form>
14    <h2>Referenciemos</h2>
15    <ul id = "edition-references">
16      <li>Se listan referencias aquí</li>
17    </ul>
18  </body>
19 </html>
```

(b)

Figura 2.1: Documento html de prueba

3. Luego de haber incluido los *scripts* necesarios, se puede proceder a configurar el widget.

2.2. Configuración del widget

La configuración del widget se hace mediante la inclusión de una etiqueta *script* dentro de la etiqueta *head* del documento. Esto permitirá que la interfaz se agregue al documento original mientras éste se carga. La tabla 2.1 muestra todas las opciones que la versión actual del *widget* ofrece. Téngase presente que la notación *objeto.propiedad* es sólo por hacer la tabla de configuración más sencilla de entender. A la hora de configurar se habrá de representar como *{objeto: propiedad}*, en la notación de objetos estándar de javascript. Las únicas opciones obligatorias son *appId* y *appUser*. Se recomienda que el implementador utilice una página generada dinámicamente del lado del servidor para poder obtener el usuario actual. Esto es posible mediante un script php que genere la configuración en tiempo de solicitud o mediante el sistema de plantillas para páginas dinámicas de *django*, para citar algunos ejemplos.

Por ejemplo, utilizando el sistema de plantillas de *django*, se podría incluir un script como el siguiente antes de la configuración:

```
<script>
    var app="{{appId}}";
    var user="{{user}}";
</script>
```

o, en php:

```
<script>
    //utilizando funciones ficticias que obtienen de alguna manera
    //la información:
    var app="<?php getCurrentApp();?>";
    var user="<?php getCurrentUser();?>";
</script>
```

En todo caso, por simplicidad, se asume que alguna variación de este script fue incluido antes de la configuración en los ejemplos que siguen (de modo que se cuenta con las variables *app* y *user*).

A continuación, se muestran algunas configuraciones de ejemplo.

2.2.1. Solamente obtener recomendaciones

Si se deseara utilizar el servicio sólo para ofrecer recomendaciones basándose en un contexto local (sin evolucionar perfiles y menos devolver retroalimentación), se podría incluir un script de configuración como el siguiente -basándose en la estructura de ejemplo mostrada en la figura 2.1b:

```
<script>
    $(function(){
```

| Opción | Función | Valor por defecto |
|-----------------------|---|-------------------|
| appId (obligatoria) | establece la clave única de la aplicación | null |
| appUser (obligatoria) | establece el id. único del usuario actual | null |
| lang | establece el lenguaje de la sesión actual | en |
| data.submit | selector que identifica unívocamente el elemento que causará la actualización de los perfiles | null |
| data.form | selector del formulario donde se encuentran el campo donde se introduce el texto | null |
| data.content | selector del área de donde se sacará el contexto local. | '#id_content' |
| feedback.mode | modo de provisión de retroalimentación. Las opciones son “ <i>follow</i> ” para que, si un usuario sigue un vínculo, se tome como retroalimentación y “ <i>select</i> ” que incluye un botón en cada recomendación para seleccionarla explícitamente. | 'follow' |
| feedback.container | selector del contenedor al cual se agregarán las recomendaciones elegidas (sólo válido para el modo <i>select</i>) | null |
| feedback.element | Elemento html dentro del cual se agregarán los vínculos de las recomendaciones elegidas | |

Cuadro 2.1: Opciones de configuración de la interfaz opcional (widget)

```

RECOMMENDER.init(
    {
        appUser: user,
        appId: app,
        data: {
            content: '#edition-area',
        },
    }
);
});
</script>

```

Nótese el uso de la notación de selectores¹ de css para referirse al área de texto de edición.

2.2.2. Obtener recomendaciones y actualizar perfil con retroalimentación implícita

Este es el escenario más probable: un usuario edita contenido y lo guarda de alguna manera. Asumiendo que el documento sigue los estándares semánticos del html, esta edición se logra mediante un formulario. El widget cambiará el comportamiento de la página: antes de ejecutar la acción original del formulario donde están los datos, actualiza el perfil del usuario actual. Asimismo, puesto que por defecto la retroalimentación es implícita, el código del widget supervisará los vínculos seguidos y utilizará esa información para aprender las preferencias del usuario. Un script como el siguiente podría lograr tal configuración (siempre en base al documento de ejemplo):

```

<script>
    $(function(){
        RECOMMENDER.init(
            {
                appUser: user,
                appId: app,
                data: {
                    content: '#edition-area',
                    form: '#edition-form',
                    submit: '#edition-save',
                },
            }
        );
    });
</script>

```

¹Si no se es familiar con esta sintaxis, refiérase a <http://www.w3.org/TR/CSS2/selector.html>

2.2.3. Utilizar retroalimentación explícita

Este escenario aún es experimental y puede cambiar en futuras versiones del servicio. La retroalimentación explícita permite que sólo documentos elegidos conscientemente por el usuario se consideren para aprender sus preferencias. Para este fin, a cada resultado de las recomendaciones se le agrega un botón, el cual, presionado, constituye retroalimentación.

Una funcionalidad extra es que, si en el documento existe algún contenedor para las referencias, cada documento elegido como retroalimentación se agregará como un *hijo* de éste. Si se decide utilizar esta característica, el implementador debe cuidar que la relación entre el componente donde se mantendrán las referencias elegidas y el elemento dentro del cual se *envolverán* estas estén relacionados semánticamente y produzcan html válido (por ello el valor por defecto asume que el contenedor es una lista y el componente para envolver documentos es un elemento de lista).

Una configuración como esta, siguiendo el documento de la figura 2.1b como ejemplo para los selectores, se puede lograr mediante un script como el siguiente:

```
<script>
    $(function(){
        RECOMMENDER.init(
            {
                appUser: user,
                appId: app,
                data: {
                    content: '#edition-area',
                    form: '#edition-form',
                    submit: '#edition-save',
                },
                feedback: {
                    mode: 'select',
                    container: '#edition-references',
                },
            }
        );
    });
</script>
```

El script anterior agregará la retroalimentación a la lista con identificador “edition-references”. Si, por ejemplo, “edition-references” no fuese una lista sino una tabla, y cada documento debiera ser una fila de esa tabla, se podría usar un script como este:

```
<script>
    $(function(){
        RECOMMENDER.init(
            {
                appUser: user,
                appId: app,
```

```

        data: {
            content: '#edition-area',
            form: '#edition-form',
            submit: '#edition-save',
        },
        feedback:{
            mode: 'select',
            container: '#edition-references',
            element: '<tr></tr>'
        }
    }
};
});
</script>

```

2.3. Comprender el comportamiento del widget

Una vez configurado el widget, la interfaz *antes* de comenzar una edición se verá como la de la figura 2.2 (la cual es una variación con estilos css del ejemplo de la figura 2.1a). Nótese que a la derecha de la pantalla se encuentra un icono de color negro: en este lugar no estorba la tarea principal del usuario y, puesto que no hay recomendaciones, el color es lo más inconspicuo posible (tiene incluso cierta transparencia).

Cuando el usuario comienza a editar y el *widget* determina que el contenido es suficiente como para hacer las primeras recomendaciones, cambiará de imagen y color, pulsando cinco veces. De esta manera se llama la atención del usuario -sin causar demasiada interrupción. Este cambio se ilustra en la figura 2.3.

Nótese que la pestaña de recomendaciones se encuentra en el borde de la pantalla, esto permite una navegación más fácil hacia ella cuando el usuario decida revisar las recomendaciones. Dar clic en esta pestaña abre la interfaz de recomendaciones ilustrada en la figura 2.4. Nótese que esta interfaz es incremental: aprovecha el hecho de que el usuario promedio la leerá de arriba hacia abajo para mostrar información cada vez más distrayente y en cada elemento el usuario puede decidir cerrar la interfaz y continuar trabajando o continuar inspeccionándolos:

- El primer elemento muestra un promedio de relevancia acompañado de una representación gráfica. Este promedio indica qué tan útiles considera el servicio las recomendaciones que encontró para el contexto actual. Si no las considera muy útiles (promedio menor a 30 %), el gráfico se torna rojo, si son medianamente relevantes, amarillo y si exceden un umbral (60 %), verde.
- A continuación, el usuario encuentra un cuadro de texto con los términos clave que el servicio encontró en su contexto local; si el usuario considera que estos términos reflejan la información que está produciendo, puede continuar.

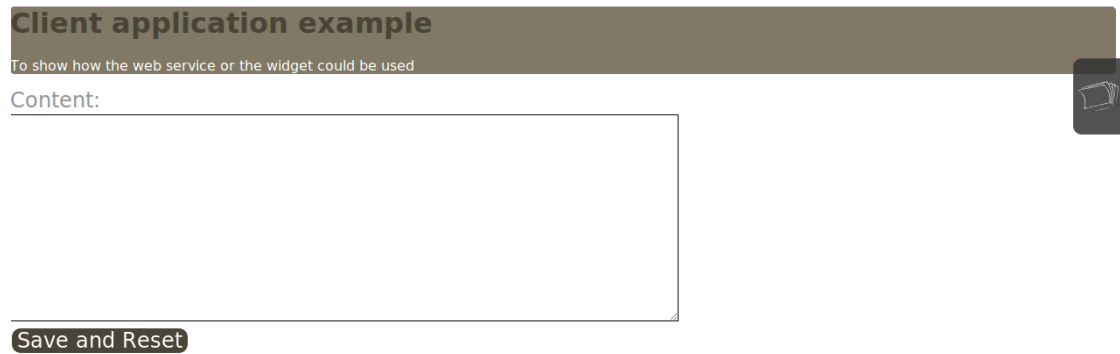


Figura 2.2: Widget inactivo: aún no hay recomendaciones

- El siguiente elemento contiene la lista de recomendaciones en sí. Cada una tiene un título que a la vez es un vínculo a la fuente original y está acompañada de las primeras diez palabras de un sumario. Si el usuario ha llegado a este punto, probablemente haya navegado con el *ratón* (mejor conocido como mouse o puntero) hacia esta zona de la interfaz. Si el puntero pasa por encima de una recomendación, el sumario se expande, mostrándose en su totalidad. En esta etapa de la inspección el usuario ha dedicado suficiente atención a la interfaz de recomendación como para pensar en utilizar alguna de las sugerencias.
- La última etapa es cuando un usuario decide seguir el vínculo de un documento para revisarlo. Aquí, si se está en modo implícito, se considera retroalimentación. Si se está en modo explícito, sólo cuando el usuario regrese a la interfaz y elija activamente el documento se considerará retroalimentación. Una interfaz en modo explícito se puede observar en la figura 2.5.

Mientras el usuario siga editando, el widget seguirá supervisando los cambios al contexto, ofreciendo nuevas recomendaciones cuando considere que éste ha cambiado lo suficiente.

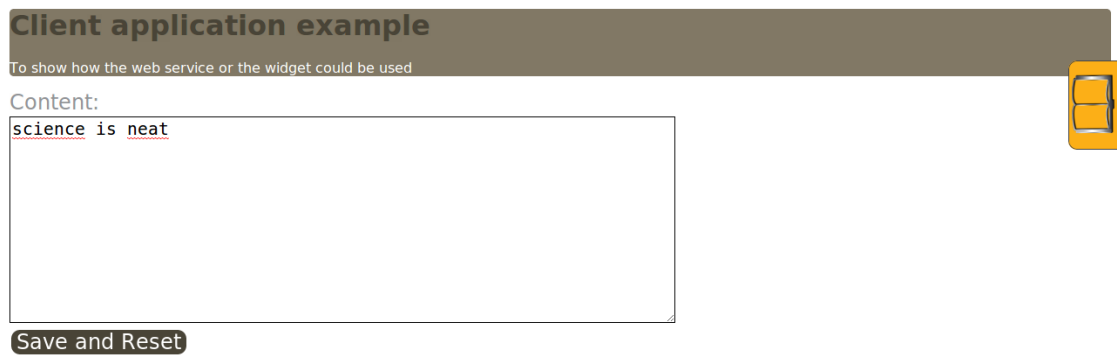


Figura 2.3: Widget activo: existen recomendaciones

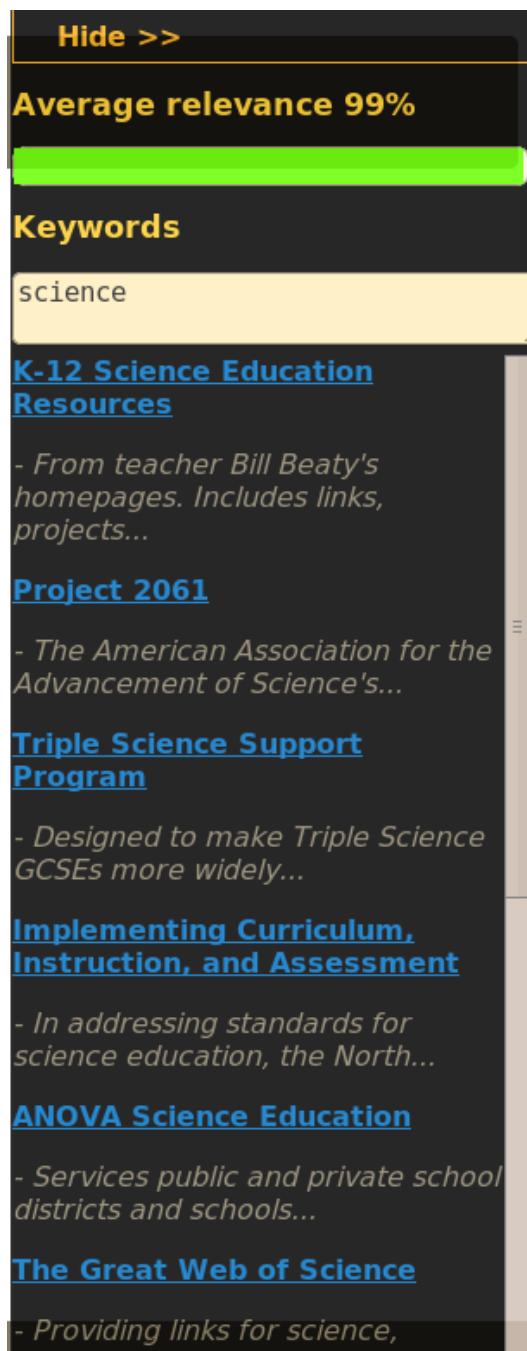


Figura 2.4: Interfaz de recomendaciones

Client application example

To show how the web service or the widget could be used

Content:

science is neat

Save and Reset

References

- [Project 2061](#)
- [K-12 Science Education Resources](#)

Hide >>

Average relevance 99%

Keywords

science

- [K-12 Science Education Resources](#)
- From teacher Bill Beaty's hompages. Includes links, projects...
- [Project 2061](#)
- The American Association for the Advancement of Science's...
- [Triple Science Support Program](#)
- Designed to make Triple Science GCSEs more widely...
- [Implementing Curriculum, Instruction, and Assessment](#)
- In addressing standards for science education, the North...
- [ANOVA Science Education](#)
- Services public and private school districts and schools...
- [The Great Web of Science](#)
- Providing links for science, astronomy, biology, chemistry...

Figura 2.5: Ejemplo de interfaz con retroalimentación explícita: dos documentos han sido elegidos

3 Uso avanzado

Para los desarrolladores que deseen aprovechar al máximo el servicio se creó una API que permite que, mediante solicitudes HTTP a métodos ubicados en URLs, se pueda obtener la funcionalidad ofrecida por el sistema. Como ya se aclaró, todas las respuestas a solicitudes bien formadas tendrán código de estado 200 (éxito) y el cuerpo será texto representando un objeto de javascript según la notación JSON. Consúltense la tabla 1.1 para una explicación de las solicitudes que devuelvan una respuesta cuyo código sea distinto a 200.

Se pueden distinguir dos conjuntos de métodos en la API: los que se utilizan para administrar usuarios y los utilizados para el servicio de recomendación en sí. En este capítulo se explican todos en detalle y se incluye un apartado con ejemplos de llamadas a algunos de éstos para dar ideas de implementación a los desarrolladores.

3.1. Métodos de gestión de usuarios

En esta sección se detallan los métodos de la API creados para permitir la gestión de usuarios. Es responsabilidad del implementador restringir el uso de éstos de manera que la utilización del servicio no se vea comprometida.

3.1.1. Adición de usuarios individuales

Esta llamada permite al implementador agregar los usuarios uno por uno (método recomendado para cuando se desee registrar un usuario al momento en que se cree en la aplicación implementadora o cuando el usuario elija utilizar la interfaz de recomendación) Sólo se requiere un usuario como parámetro, si se pasa una lista de usuarios, sólo el último se agregará y los demás se obviarán.

URL: `http://trece.com/api/registerUser/`

Método requerido: POST con autenticación básica HTTP.

Parámetros:

appld: la clave única de la aplicación

user: el identificador único a agregar.

Atributos de retorno:

created: un valor booleano que indica si la adición se hizo. Si el valor es inválido, se retornará un valor falso. Si el usuario es un duplicado, la adición no se hará y se retornará verdadero: no se permiten duplicados.

3.1.2. Agregación de múltiples usuarios

Esta llamada se proporciona para aplicaciones que deseen agregar varios usuarios ya existentes de una sola vez. Debido a que esto involucra una cantidad considerable de inserciones, la agregación se hace de manera asíncrona. Así, cuando se hace la llamada se agrega la solicitud a una cola de tareas para ser procesada cuando el administrador de carga del servidor lo considere idóneo. Un correo electrónico se enviará a la dirección de contacto registrada para la aplicación una vez terminada esta tarea.

URL: `http://trecs.com/api/bulkRegisterUsers/`

Método requerido: POST con autenticación básica HTTP.

Parámetros:

appld: la clave única de la aplicación

user: un arreglo de identificadores únicos a agregar. De preferencia, este arreglo se ha de representar en la solicitud como parámetros separados con el mismo nombre, por ejemplo:

`?user=trex&user=raptor&user=you`

Atributos de retorno:

queued: un valor booleano que indica si la tarea se agregó a la cola de trabajos.

3.1.3. Listado de usuarios

Esta llamada puede ser útil para la creación de interfaces de administración propias de la aplicación que se desarrolla, para detectar que personas maliciosas han agregado usuarios en contra de la voluntad del administrador o validar si se está cerca de sobrepasar el límite de usuarios permitido.

URL: `http://trecs.com/api/getUsers/`

Método requerido: GET

Parámetros:

appld: la clave única de la aplicación

Atributos de retorno:

users: una colección de objetos representando a los usuarios registrados de la aplicación, cada cual con los siguientes atributos:

id: el identificador único del usuario

added: la fecha en la cual se agregó el usuario.

3.1.4. Eliminación de usuarios

Sólo se permite para eliminar un usuario por vez. De la misma manera que la agregación individual de usuarios, sólo el último parámetro de usuario pasado será considerado. También requiere autenticación HTTP básica. Eliminar usuarios puede ser útil para cuando las aplicaciones dan la opción de desactivar usuarios o permiten desinstalación o renuncia a la aplicación del lado del cliente, es responsabilidad del implementador prever las consecuencias y tomar las medidas correspondientes en los casos donde se utilizará la eliminación de usuarios. La eliminación de un usuario implica la remoción irreversible del perfil que se haya mantenido del mismo.

URL: `http://trecs.com/api/deleteUser/`

Método requerido: POST con autenticación básica HTTP.

Parámetros:

appld: la clave única de la aplicación
user: el identificador único a eliminar.

Atributos de retorno:

created: un valor booleano que indica si la eliminación se hizo. Si el valor es inválido, se retornará falso.

3.2. Servicio de recomendación

En esta sección se detallan las funciones del servicio de recomendación. Es responsabilidad del implementador llamarlas cuando se considere necesario en la aplicación desarrollada y proveer una interfaz donde los usuarios puedan ver las recomendaciones y proveer retroalimentación.

3.2.1. Obtener recomendaciones

La llamada fundamental del servicio. Dados un texto libre y un usuario, el servicio encontrará documentos temáticamente relacionados al texto y los filtrará según las preferencias que se tengan registradas del usuario (el perfil). Si es la primera vez que se solicitan recomendaciones para un usuario, la etapa de filtrado sencillamente se ignorará. En la versión actual, solicitar recomendaciones para textos en lenguajes diferentes al inglés causará un período de latencia ligeramente mayor, debido al uso de servicios web externos para el análisis de los textos.

URL: `http://trecs.com/api/getRecommendations/`

Método requerido: GET o POST. Para reducir la latencia en textos cortos, se recomienda GET, pero si el texto excede los 2048 caracteres, se recomienda usar POST, para evitar problemas de librerías y navegadores que imponen este límite arbitrario.

Parámetros:

- appld (obligatorio):** la clave única de la aplicación.
- appUser (obligatorio):** el identificador único de un usuario registrado
- context (obligatorio):** el texto introducido por el usuario. Debe ser texto plano, sin ningún tipo de marcaje (HTML, XML, etc).
- lang (opcional):** el lenguaje en el que el texto está escrito. Si no se provee se asume que está en inglés.

Atributos de retorno:

- terms:** un arreglo de cadenas de caracteres con los términos clave encontrados en el texto.
- results:** un arreglo de objetos con las recomendaciones en sí. Cada objeto tiene los siguientes atributos:
 - id:** el código del documento en la colección documental.
 - title:** el título del documento.
 - summary:** un resumen del documento (puede estar vacío).
 - url:** el URL donde se puede encontrar el documento original.
 - percent:** el porcentaje de relevancia del documento respecto a los términos clave.
 - category:** el código de la categoría de la base de conocimiento del servicio bajo la cual está clasificado el documento.
 - weight:** el porcentaje de utilidad del documento respecto al perfil del usuario. Es una modificación del porcentaje.

Recomendaciones de uso: se exhorta al implementador a respetar el orden en el que se devuelven los documentos, ya que éste refleja las preferencias del usuario. Asimismo, la url se puede utilizar para permitir al usuario revisar el documento recomendado. No se recomienda mostrar datos como el porcentaje, el peso, la categoría o el código del documento, ya que pueden confundir a los usuarios finales. El implementador deberá crear un mecanismo que reciba retroalimentación de los usuarios de alguna manera: se debe saber qué documentos eligió el usuario en una sesión y almacenar los códigos de los mismos, puesto que la evolución del perfil se puede beneficiar de este tipo de retroalimentación.

La interfaz donde se muestren las recomendaciones se debería diseñar de manera que no invada la tarea principal del usuario (la edición de información), por lo que se alienta al programador a utilizar un tema cromático que no distraiga y a ubicarla a la derecha de la pantalla, puesto que esa es la zona a la cual los usuarios tienden a prestar menos atención (aunque esto depende de la aplicación).

3.2.2. Actualizar perfiles

Esta llamada es fundamental si se desea sacar el máximo provecho del sistema de recomendación. Permite a la aplicación implementadora indicar cuándo actualizar las

preferencias de los usuarios registrados, de modo que se mejoren el conocimiento sobre los mismos y la calidad de las recomendaciones.

URL: `http://trececs.com/api/updateProfile/`

Método requerido: POST o GET. Igual que la solicitud de recomendaciones, se recomienda POST para solicitudes muy largas.

Parámetros:

appld (obligatorio): la clave única de la aplicación

appUser (obligatorio): el identificador único de un usuario registrado.

context (opcional): texto libre representando el contenido introducido por el usuario a considerar para actualizar el perfil.

lang (opcional): el lenguaje del contexto. Inglés por defecto.

docs (opcional): un arreglo con los códigos de los documentos elegidos como retroalimentación. De preferencia, este arreglo se ha de representar en la solicitud como parámetros separados con el mismo nombre, por ejemplo:

`?docs=1117&docs=62&docs=1962`

Atributos de retorno: Ninguno. Si la tarea de actualización logró agregarse al administrador de trabajos para procesamiento asíncrono se contestará con una respuesta con código de estado 200.

Recomendaciones de uso: El escenario de uso ideal de esta llamada es cuando el usuario decide guardar su trabajo (o salir del sistema), aunque se puede llamar cuantas veces se quiera, mas no se asegura la actualización inmediata del perfil del usuario, pues esta es una tarea asíncrona que se ejecuta cuando el servidor tiene poca carga, dado que requiere procesamiento considerable. Como ya se mencionó, es altamente recomendado que se implemente un mecanismo para proveer retroalimentación, de forma que el servicio pueda aprender las preferencias del usuario con mayor exactitud y rapidez. Se puede proveer también un texto introducido por el usuario para el aprendizaje de las preferencias, se aconseja que éste sea la forma final de la edición. Si no se provee ni contexto ni retroalimentación, el perfil sencillamente *envejecerá*: se asumirá que el usuario ha perdido un poco de interés en *todas* las categorías que se sepa que ha preferido anteriormente.

3.3. Ejemplos de llamadas a la API

En esta sección se ejemplifica una llamada al método de obtención de recomendaciones en distintos lenguajes de programación, con el fin de dar un punto de partida a los desarrolladores. Todas las llamadas tienen como salida la simple impresión en pantalla de los títulos y urls de los resultados devueltos por el servicio.

3.3.1. En javascript (con la librería jquery)

```

1 $.getJSON('http://trecs.com/api/getRecommendations/?callback
  =?',
2 {context: 'Science is neat!', appId: '12345abc321', appUser:
  'someUser'},
3 function(data){
4     alert("Recommendations: ");
5     $.each(data.results, function(i,doc){
6         alert('Title: 'doc.title+', URL: '+ doc.url);
7     });
8 });

```

3.3.2. En python

```

1 from urllib import urlopen, urlencode
2 import json
3 raw_data = urlopen('http://trecs.com/api/getRecommendations/'
4
5                     ,
6                     urlencode({'appId': '12345abc321',
7                                   'appUser': 'someUser',
8                                   'context': "science is neat"})).read()
9 data = json.loads(raw_data)
10 print "Recommendations: "
11 for doc in data['results']:
12     print "Title: %s, URL: %s" % (doc['title'], doc['url'])

```

3.3.3. En java

```

1 import java.net.HttpURLConnection;
2 import java.net.MalformedURLException;
3 import java.net.URL;
4 import java.io.IOException;
5 import java.io.InputStream;
6 import java.io.BufferedReader;
7 import java.io.InputStreamReader;
8 import com.google.gson.Gson;
9 class Result {
10     public Doc[] results;
11     public String terms;
12     public Result(){}
13 }
14 class Doc {
15     String title;
16     String summary;

```

3 Uso avanzado

```
17         String id;
18         String url;
19         String percent;
20         String category;
21         String weight;
22         public Doc(){}
23     }
24     public class Demo {
25         public static String getContents(InputStream s)
26             throws IOException{
27             BufferedReader in = new BufferedReader(new
28                 InputStreamReader(s));
29             String inputLine;
30             StringBuffer outLine = new StringBuffer();
31             while ((inputLine = in.readLine()) != null)
32                 outLine.append(inputLine);
33             in.close();
34             return outLine.toString();
35         }
36         public static void main(String[] args) throws
37             IOException, MalformedURLException {
38             //build the request
39             URL serviceCall = null;
40             String call = "http://localhost:8000/api/
41                 getRecommendations/";
42             String appId = "12345abc321";
43             String appUser = "someUser";
44             String context = "Science+is+neat!";
45             String urlpath = String.format("%s?appId=%s&appUser=%s&
46                 content=%s",
47                 call, appId, appUser, context);
48             serviceCall = new URL(urlpath);
49             //get the response:
50             HttpURLConnection conn = (HttpURLConnection)serviceCall
51                 .openConnection();
52             //interpret the response:
53             InputStream is;
54             if(conn.getResponseCode() < 400){
55                 is = conn.getInputStream();
56                 Gson gson = new Gson();
57                 String json = getContents(is);
58                 Result data = gson.fromJson(json, Result.class);
59                 System.out.println("Recommendations");
60                 for(Doc doc: data.results){
```

3 *Uso avanzado*

```
55         System.out.printf("Title: %s, URL: %s \n", doc.title
56             , doc.url);
57     }
58 }else{
59     System.out.printf("Error in call: %s", conn.
60         getErrorStream().toString());
61 }
```