Universidad Zaragoza

Proyecto Fin de Carrera

Ingeniería en Informática

SISTEMA ANALIZADOR Y RECOMENDADOR DE BLOGS

Luis Latasa Yuste

Director: Gregorio de Miguel Casado

Departamento de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

Zaragoza, Septiembre de 2011

Resumen

La naturaleza libre de Internet ha propiciado que la búsqueda y seguimiento eficaces de información especializada se constituya en una tarea cada vez más compleja de abordar, debido esencialmente a su crecimiento masivo y sostenido.

Este proyecto aborda la creación de un prototipo software que facilita la suscripción del usuario a sitios web determinados para recibir de forma proactiva, notificaciones sobre textos objeto de interés. El sistema se entrena a partir de un conjunto de casos de entrenamiento y la retroalimentación progresiva producida por la incorporación de las nuevas lecturas que el usuario cataloga como de interés. El idioma de los textos a analizar y recomendar será el castellano, debido a su riqueza lingüística.

La presente memoria contiene el trabajo realizado en los siguientes aspectos: Una búsqueda de información relacionada con los aspectos básicos del proyecto, en la que se abordan aspectos lingüísticos propios del castellano que permiten explotar información invariante en el lenguaje, algoritmia para procesamiento de lenguaje natural en el contexto del análisis y catalogación de textos, aspectos relacionados con la escalabilidad para la gestión de grandes flujos de información, y cuestiones de paralelismo propias de la implementación de sistemas de información.

El estudio preliminar de estas cuestiones ha permitido elaborar una propuesta de arquitectura del sistema que organiza en bloques funcionales el tratamiento de la información, los flujos de datos y las posibilidades de paralelización. Esto ha conducido a una implementación de un prototipo del sistema, para el que se ha realizado una evaluación en cuanto a escalabilidad mediante un conjunto de experimentos ejecutados en plataformas computacionales heterogéneas. Como última sección de la parte principal de la memoria se proponen las conclusiones extraídas del trabajo. Finalmente, los anexos de la memoria recogen información específica sobre la algoritmia de Aprendizaje utilizada así como las decisiones concretas de diseño e implementación.

Agradecimientos

A Gregorio de Miguel, por haberme brindado la oportunidad de hacer este proyecto, por tu ayuda en todo momento, por tus consejos y tus charlas en Skype para resolver todas mis dudas.

Al Proyecto TIN2008-06582-C03-02 - "Secuencias Simbólicas: Análisis, Aprendizaje, Minería y Evolución", del Ministerio de Ciencia e Innovación, y al Grupo de Ingeniería de Sistemas de Eventos Discretos (GISED), por el soporte prestado para la realización del trabajo.

A Laura, mi bella doctora, por este año y medio, por estar segura en todo momento de que iba a poder con todo, por darme ánimos y porque cuando se está agobiado, una sonrisa amable es lo que más se agradece.

A mis amigos del CPS, por todos los buenos momentos, las risas y los cafés que han hecho de mi carrera los mejores años de mi vida.

A Elisa, Miguel Ángel y María, por los cafés en el Anika's, las cervezas en "Casa Chen" y tantos otros ratos de risas. Habéis sido mi momento de liberación en muchas tardes de estudio.

A mi familia, por vuestro ánimo, apoyo incondicional y confianza en que mi trabajo llegaría a buen puerto. Como no, a mi tío Fran por tus consejos y comentarios en la elaboración de esta memoria.

Muy en especial, a mis padres, porque aunque a veces hayáis sido un poco "pesadillas", me recordáis que las cosas no se hacen solas. Por vuestro cariño, por escucharme y aconsejarme, por todo lo que durante toda mi vida habéis hecho por mí.

Y, por supuesto, a todos los que olvido nombrar.

¡Muchas gracias a todos!

Índice general

Capítulo 1

Introducción

Los apartados siguientes resumen las motivaciones y los objetivos del proyecto BCatalog, que justifican la necesidad que conduce a la creación de este prototipo y estructuran y organizan el trabajo a realizar.

1.1. Motivación

La evolución de Internet a lo largo de sus escasas décadas de vida ha sido espectacular. En sus inicios apenas podían verse algunas páginas muy simples y pobres en diseño. Las empresas las utilizaban para ofrecer información sobre sus productos y servicios, a modo de catálogo en formato digital.

En la actualidad, la red se ha convertido en un medio de virtualización de la actividad humana. Utilizando su ordenador, el usuario puede comunicarse con sus amigos, compartir contenidos (música, vídeo, texto...), comprar y vender, informarse, formarse, o realizar gestiones bancarias y administrativas.

Para las búsquedas de información existen diferentes posibilidades:

El usuario puede utilizar buscadores que seleccionan y filtran los resultados de su búsqueda, mostrándolos por orden de relevancia. Un inconveniente que presentan estos buscadores es que muestran sitios web de todo tipo, tales como páginas de empresas, tiendas online, salas de chat, periódicos, o páginas personales. Si lo que necesita es centrarse en artículos de opinión, han de filtrarse manualmente los resultados propuestos por el motor de busqueda.

Algunas páginas web ofrecen al usuario un servicio de notificación que le informa cada vez que se publican nuevos contenidos, pero sin discriminar si son o no del interés

del usuario.

También existen subscripciones a búsquedas. Algunos buscadores envían avisos cuando aparecen nuevos contenidos relacionados con la búsqueda realizada, pero incluyendo avisos sobre sitios web de todo tipo, al igual que ocurre en la búsqueda realizada manualmente.

Estas opciones de búsqueda y notificación no permiten selecionar a la vez los temas y los sitios web de interés del usuario.

El crecimiento exponencial de la información presente en Internet se manifiesta en la ingente cantidad de sitios web que almacenan información especializada y organizada en temas que los propios usuarios incorporan a la red. En este sentido, es posible encontrar bibliografía científica que estudia tanto el crecimiento como el impacto de la que se ha dado a conocer como *blogosfera* ([1], [2]).

En este contexto, el proyecto propone la creación de un entorno software que permitirá al usuario subscribirse a sitios web de su elección y recibir notificaciones cuando aparezcan textos de su interés. Para determinarlo, el sistema seguirá un proceso de aprendizaje mediante la obtención de casos de ejemplo, el entrenamiento y la realimentación. Realizado este proceso, determinará el grado de relevancia de los textos y, si procede, se los recomendará al usuario.

El idioma de los textos que se van a recomendar será el español, debido a su riqueza lingüística, su gran cantidad de hablantes y abundancia de contenidos existentes.

En el siguiente apartado se introducen los objetivos concretos del proyecto así como la estructura de la memoria.

1.2. Objetivos del proyecto y estructura de la memoria

La motivación con la que nace este proyecto se concreta en una serie de objetivos específicos que estructuran el desarrollo del trabajo. La realización sucesiva de cada uno de ellos desemboca en la creación de un primer prototipo funcional de mi analizador y recomendador de textos. Estos objetivos, que definen la estructura de la memoria, son:

1. **Búsqueda de información relacionada con el proyecto:** Se realiza un estudio en profundidad centrado en aspectos lingüísticos, técnicas para el análisis y clasificación de textos, y localización de herramientas ya existentes que faciliten la realizacion del trabajo. Todo ello se expone en el apartado 2 de la memoria.

2. **Propuesta de la arquitectura del sistema:** En el apartado 3 se concretan las funcionalidades que ofrece el sistema, así como las estrategias seguidas en las tareas de procesamiento lingüístico, aprendizaje, gestión de los flujos de información y paralelización de tareas internas.

3. **Prototipo del sistema:** El apartado 4 explica la estructura final del prototipo a construir, concretando las tecnologías elegidas para la fase de implementación.

4. **Implementación del prototipo:** La fase de implementación se expone en el apartado 5, en el que se detallan los rasgos principales de cada bloque funcional construido.

5. **Experimentos y evaluación del prototipo:** El apartado 6 contiene los experimentos realizados sobre el sistema, profundizando en la influencia de la paralelización de las distintas tareas internas del mismo. Se comentan y comparan los resultados obtenidos.

6. **Conclusiones:** En el apartado 8 se exponen los principales problemas encontrados a lo largo de la realización del proyecto y se extraen conclusiones sobre el funcionamiento del prototipo y el alcance del proyecto.

Capítulo 2

Estado del arte

Este apartado contiene un resumen de la información encontrada sobre los aspectos relevantes relacionados con el proyecto. Se introducen conceptos necesarios para entender el funcionamiento del mismo y se comentan algunas herramientas ya existentes.

2.1. Análisis lingüístico

El sistema debe identificar el grado de relación del contenido del texto con los temas de interés del usuario por lo que es necesaria la revisión de los siguientes conceptos que se emplean en la clasificación de textos:

- **Campo conceptual:** Conjunto de palabras asociadas a una misma idea o concepto. No es necesario que las tengan un origen común, puesto que en lo que se centra un campo conceptual es en el significado de las palabras y no en su forma.
- **Lexema:** Unidad mínima con significado propio. También llamado raíz, porque sirve como base para generar palabras, añadiéndole prefijos y/o sufijos.
- **Morfema:** Unidad sin significado propio, que limita y concreta el significado de un lexema. También llamado desinencia, aporta información de género, número, tiempo, finalidad, etc... Se distingue entre prefijo y sufijo, dependiendo de si es colocado antes o después de la raíz.
- **Derivación:** Proceso en el que, a partir un lexema y un conjunto de morfemas, se obtienen todas las palabras que se pueden formar combinándolos. Al conjunto de palabras obtenido se le llama familia sintáctica, pues todas ellas provienen de una raíz común.
- **Lematización:** Proceso inverso a la derivación. Dada una palabra, se obtiene su lexema.

Repasados estos conceptos clave, se pasa a comentar algunas de las herramientas para el procesamiento del lenguaje natural que se han encontrado:

- **Diccionarios online:** De acceso público, facilitan mucha información sobre una palabra: significado, sinónimos, familia léxica a la que pertenece, información sobre su etimología, e incluso foros de discusión en los que se discute sobre su correcto uso. Algunos ejemplos de estos diccionarios son el ofrecido por la Real Academia Española [3], o WordReference [4], un sitio web en el que se pueden encontrar diccionarios de traducción entre múltiples idiomas, y que contienen además un foro para resolver dudas entre sus usuarios.

- **Diccionario ideológico de Zirano:** También de acceso público y gratuito, Zirano [5] ofrece la posibilidad de buscar palabras tanto para obtener su significado como para obtener su campo conceptual. Dada una palabra, sugiere una lista de ideas a la que puede estar asociada, y, tras elegir una de ellas, proporciona un conjunto de palabras relacionadas.

2.2. Tecnologías para la creación de sitios Web

Debido a la relación del proyecto con Internet ha sido necesario de un estudio de las distintas alternativas existentes para la creación de sitios web.

Se definen a continuación algunos términos relacionados con las tecnologías que se asocian a la gestión de información que son objeto de interés del proyecto.

- **Sistema de gestión de contenidos:** (CMS, del inglés Content Management System). Proporciona una interfaz de administración en la que el propietario del sitio puede añadir contenidos, gestionar los menús laterales, insertar enlaces a otras páginas, cambiar la apariencia de la web, etc. Los CMS están creados para ser utilizados sin necesidad de poseer conocimientos de informática. El propietario puede modificar el sitio web con un simple clic y añadir contenidos como si de un procesador de textos se tratase.

- **Blog:** Es un tipo de CMS en el que los textos publicados aparecen estructurados uno tras otro en la página principal. Se utiliza frecuentemente cuando lo que se pretende es dar continuidad en el tiempo, dar a conocer hechos que pueden estar relacionados y mostrar esa relación. Wordpress [6] y Blogspot [7] son algunos de los servicios gratuitos de creación de *blogs* más importantes en la actualidad.

- **Post:** Coloquialmente se llama *post* a un texto que se publica en un *blog*. De esta forma se deja que el término *página* se refiera a una sección independiente,

mostrada en exclusividad (sin otros posts ni páginas debajo ni encima). Así, una misma web puede tener páginas independientes, estar estructurada como *blog* (un post debajo de otro), o utilizar una estructura mixta.

En cuanto a las herramientas para la creación de sitios web, existe una gran variedad de alternativas gratuitas. Se comentan a continuación algunas de ellas, orientadas tanto a usuarios sin conocimientos técnicos de informática como a desarrolladores.

- **Herramientas de usuario:** Un usuario sin conocimientos técnicos informáticos puede crear un *blog* registrándose en Blogspot [7] o en Wordpress [6]. Tras registrarse en el servicio y obtener un nombre de usuario y contraseña, puede entrar en su panel de administración y empezar a publicar.

- **Herramientas para usuarios intermedios:** Para un nivel mayor de personalización, existen CMS disponibles para descarga que pueden instalarse en el servidor web que el usuario tenga contratado. Este tipo de instalación proporciona al usuario la posibilidad de modificar el código fuente, crear sus propias extensiones, o instalar otras existentes para añadir funcionalidades que no vienen por defecto. Algunos de estos CMS son Xoops [8], Joomla [9], PHP-Nuke [10] o la versión instalable de Wordpress [11], todos ellos gratuitos.

- **Herramientas para programadores:** En ocasiones un programador puede necesitar desarrollar su propio CMS. Existen frameworks gratuitos que pueden ser utilizados como punto de partida para el desarrollo. Un ejemplo de este tipo de software es CodeIgniter [12], escrito en PHP y basado en el patrón de diseño MVC (Model-View-Controller). Incluye, entre otras utilidades, las de criptografía, compresión de archivos etc.

2.3. Técnicas en aprendizaje orientadas a la clasificación de textos

Buscando documentación acerca de las distintas técnicas en aprendizaje orientadas a la clasificación de textos, se ha encontrado abundante documentación al respecto.

Existen técnicas de *Data Mining* mediante las cuales se obtiene información no trivial a partir del análisis exhaustivo de muchos datos de ejemplo disponibles. El *Data Mining* puede aplicarse a diferentes tipos de datos como enteros, números en coma flotante, cadenas de texto y otros objetos más avanzados, como se explica en el libro *Data Mining: Concepts and Techniques* [13].

Entre las muchas técnicas de *Data Mining*, se encuentra el uso de las Máquinas de Vectores de Soporte, (SVM, del inglés *Support Vector Machine*) basadas en la separación de los datos de ejemplo a través de hiperplanos calculados mediante las llamadas funciones Kernel. Algunos de los textos de interés en los que se ha inspirado este proyecto son los siguientes:

1. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond* [14]: Introduce conceptos necesarios para comprender el funcionamiento de las máquinas SVM, profundizando en los fundamentos matemáticos subyacentes y aportando demostraciones, ilustraciones y ejemplos.

2. *Kernels for Structured Data* [15].

3. *Pairwise Classification as an Ensemble Technique* [16]: En este artículo se detalla la construcción de un clasificador SVM multiclase a partir de un conjunto de clasificadores SVM binarios.

4. *Text Classification using String Kernels* [17]: Profundiza en el comportamiento de los Kernels de comparación de cadenas de texto e introduce optimizaciones que pueden aplicarse para obtener mejores resultados.

5. *Lambda pruning: an approximation of the string subsequence kernel for practical SVM classification and redundancy clustering* [18]: Este artículo expone las mejoras obtenidas al incluir en el algoritmo tradicional de análisis de secuencias de caracteres un sistema de poda, que disminuye el tiempo necesario para procesar dos cadenas de texto.

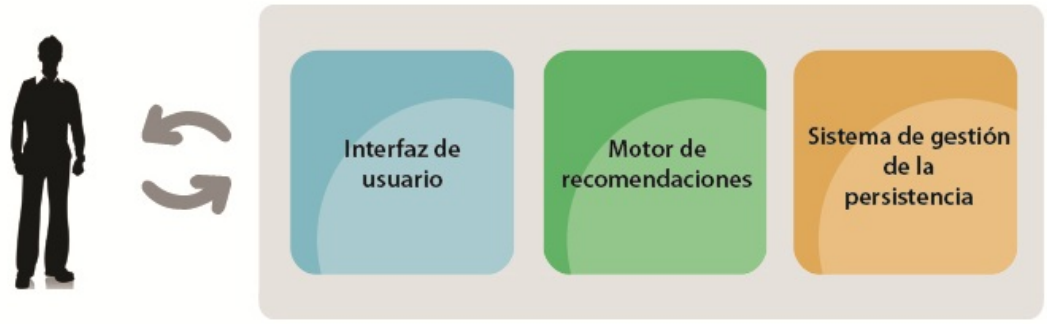
Como se comenta en apartados posteriores, todos ellos han sido utilizados como punto de partida en la implementación de los módulos que componen el sistema.

Capítulo 3

Propuesta de la arquitectura del sistema

El sistema se estructura en tres bloques funcionales principales que permiten separar la interfaz de usuario, el motor de recomendaciones y el sistema de gestión de la persistencia, de forma que abordar cada uno de ellos separadamente facilite la construcción del prototipo. La figura 3.1 muestra un esquema con dichos bloques funcionales, cuya función se define a continuación.

Figura 3.1: *Esquema simplificado de la arquitectura del sistema.*



El bloque de interfaz interactúa con el usuario, ofreciéndole las siguientes funcionalidades:

- Gestión de subscripciones a *blogs* de interés, incluyendo alta, listado y cancelación.
- Gestión de las categorías de los textos. Los nuevos textos serán recomendados si el sistema determina que pertenecen a alguna de ellas.

- Gestión de los textos de ejemplo. Añadir uno nuevo supone asociarlo a una de las categorías citadas.
- Entrenamiento del sistema para que sea capaz de predecir la categoría a la que pertenece el texto nuevo.
- Visualización y validación de las recomendaciones de textos ofrecidas al usuario.

El motor de recomendaciones es el bloque principal del sistema y se encarga de:

- Recoger y validar los datos procedentes de la interfaz de usuario.
- Obtener los textos para su posterior análisis.
- Comprobar periódicamente si, en los *blogs* de interés del usuario, hay textos nuevos y recomendárselos en caso de estar relacionados con sus temas elegidos.
- Administrar la máquina de aprendizaje que, tras ser entrenada, será capaz de predecir la categoría a la que pertenecen los nuevos textos encontrados.
- Interactuar con el sistema de gestión de la persistencia para el almacenamiento de la información utilizada y necesaria para el sistema.

Por último, el sistema de persistencia permite guardar:

- Los resultados de los análisis.
- El estado de la máquina de aprendizaje.
- Toda la información de los textos que se analicen.
- Otra información que pueda ser interesante conservar por temas de rendimiento.
- Almacena datos de configuración necesarios para el arranque del sistema.

3.1. Procesamiento lingüístico

La naturaleza del proyecto requiere que el sistema sea capaz de procesar textos atendiendo a sus contenidos e identificar los temas que tratan. No resulta práctico determinar la similitud entre dos textos considerando número de caracteres que tienen en común. Los campos conceptuales son de utilidad en este contexto, puesto que son conjuntos de palabras con significados relacionados que pueden ser utilizados para detectar semejanzas en los temas tratados en los textos.

En este primer prototipo los campos conceptuales se calculan en el momento de creación de una nueva categoría. El sistema proporcionará un campo conceptual asociado a las palabras que forman el nombre de la categoría. De esta forma, "política" llevaría asociadas palabras como partido, ideología, parlamento y otras que el usuario podrá añadir según sus preferencias.

El proceso de obtención del campo a partir de una determinada lista de palabras es muy importante, ya que el conjunto obtenido se utilizará como base durante la fase de entrenamiento. Hay que tener en cuenta que la inclusión de palabras poco relacionadas así como un número reducido de ellas distorsionarán las predicciones. A continuación se detalla el proceso de creación de una nueva categoría y el cálculo del campo conceptual:

1. El usuario introduce las palabras clave.

2. Se obtienen los campos de cada una de ellas por separado. Para ello pueden emplearse diccionarios online, bases de datos o cualquier otra herramienta disponible.

3. Cada una de las palabras que forman los campos obtenidos pasa por un lematizador, que se encarga de eliminar los morfemas de la palabra, para quedarse con la raíz o lexema. Así se dispone de la esencia de cada palabra, invariante a los mecanismos de composición y derivación inherentes a la Lengua Castellana.

4. Con los campos conceptuales lematizados de cada palabra se calcula el campo resultante. Este proceso no es trivial porque si el sistema es demasiado restrictivo se obtiene un conjunto final muy reducido, y por tanto poco relevante. Siendo demasiado flexible, por el contrario, se obtiene un resultado con excesivas palabras que, además disminuir el rendimiento durante el proceso de análisis posterior, introduce palabras poco relacionadas. En este prototipo el campo total se calcula como la unión de los campos individuales, aunque cada contexto de aplicación puede requerir un criterio diferente para mejorar los resultados en la predicción.

El campo conceptual constituye un caso de entrenamiento para la categoría creada y será utilizado como un ejemplo más a la hora de realizar el aprendizaje y las predicciones. De esta forma se parte de una base adecuada que permite el análisis de contenidos de interés, pues se dispone de un conjunto de palabras relacionadas que pueden encontrarse en los textos relacionados con dicha categoría.

3.2. Algoritmia de aprendizaje

El sistema a construir requiere un método para distinguir diferentes tipos de textos atendiendo a su contenido. Este prototipo hará uso de algoritmos de aprendizaje supervisado.

Estos algoritmos completan una serie de operaciones necesarias para poder realizar predicciones:

1. **Recepción de datos de ejemplo:** El algoritmo requiere que le sean proporcionados datos de ejemplo de las diferentes clases que va a reconocer. La cantidad de estos datos, así como su relevancia, condicionará considerablemente la precisión de las predicciones.

2. **Fase de entrenamiento:** Una vez le son facilitados los datos de ejemplo, se realiza una serie de ajustes internos, que permiten a la máquina de aprendizaje aprender a distinguir nuevos elementos.

3. **Fase de predicción :** Finalizado el entrenamiento, el algoritmo es capaz de determinar la clase a la que más se asemeja un nuevo dato.

Como puede verse, los algoritmos de aprendizaje supervisado son muy adecuados para este proyecto, en el que el usuario asociará un conjunto de textos a sus respectivas categorías, y el sistema se encargará de recomendarle nuevos textos cuando determine que están relacionados con los temas de su interés.

El problema abordado en este proyecto encaja perfectamente en este tipo de algoritmos puesto que el usuario proporciona al sistema los textos de ejemplo y las categorías a las que pertenecen, el sistema se entrena en función de los datos recibidos, y a partir de ese momento predice la categoría de los nuevos textos publicados para recomendarlos, si procede, al usuario.

3.3. Gestión de flujos de información

El motor de recomendaciones se encarga de, ante la necesidad de disponer de datos que todavía no haya manejado, obtener dicha información del mundo exterior (Internet).

Las comunicaciones con el mundo exterior requieren tiempo para solicitar los datos y esperar a que sean proporcionados. Este tiempo, para nada despreciable, hace necesario almacenar toda la información recibida que pueda ser utilizada de nuevo si la penalización para obtenerla es elevada.

También es aconsejable almacenar los resultados del cálculo de relación entre dos textos, ya que es un dato invariante que se empleará repetidamente a lo largo del tiempo.

Para abordar el problema, el motor de recomendaciones enviará al sistema de gestión de la persistencia los textos, los campos conceptuales obtenidos y el resultado de los análisis. Todo este proceso se detalla en el apartado 4.

Otra ventaja importante del almacenamiento de datos es la disponibilidad de la información inmediatamente después del arranque del sistema tras un cese en su funcionamiento. El sistema puede dejar de funcionar temporalmente por un fallo en la red o en la máquina donde se ejecute. Disponer de los datos almacenados evita la necesidad de calcularlos de nuevo.

3.4. Paralelización de tareas

Las tareas internas que el sistema realiza para analizar los textos y predecir el grado de relevancia de los nuevos contenidos publicados requieren cálculos intensivos y tiempo para ejecutarlos. Por ello resulta ventajoso paralelizar estas operaciones. Después de un estudio de la naturaleza de cada una de ellas, se han encontrado los siguientes puntos en los que un trabajo en paralelo mejorará los tiempos de ejecución.

- **Obtención del campo conceptual:** En esta tarea la consulta de un diccionario de gran tamaño puede puede requerir mucho tiempo. También es posible que lo que se quiera sea calcular el campo asociado a varias palabras para luego determinar el conjunto total. Puesto que la obtención de cada campo individual es una operación independiente, pueden ejecutarse las consultas individuales en paralelo y luego fusionar los resultados, reduciendo así el tiempo total de ejecución de la tarea.

- **Entrenamiento y predicción en la máquina de aprendizaje:** Aun siendo la operación de entrenamiento bastante poco frecuente, lleva consigo una gran penalización. Con muchos casos de ejemplo y muchas clases de datos, puede requerir varios minutos, o incluso horas. En cuanto a la predicción, desgraciadamente es

bastante más utilizada que la anterior, ya que se ejecuta cada vez que haya contenidos nuevos en cualquiera de los *blogs* en los que el usuario se haya subscrito. Éste es pues otro contexto en que existe la posibilidad de introducir mejoras en cuanto a la paralelización de la algoritmia asociada al entrenamiento y a la predicción.

Capítulo 4

Prototipo del sistema

En este apartado se presenta el prototipo del sistema, comentando en primer lugar las decisiones previas tomadas para transformar la figura 3.1 en el modelo a implementar.

El proceso de refinamiento se lleva a cabo en varias fases. Primero se introducen algunas consideraciones que determinan la ubicación de los bloques del sistema, después se elige el algorimo de aprendizaje y posteriormente las tecnologías a utilizar.

4.1. Primera aproximación

El esquema inicial, mostrado en la figura 3.1 encaja perfectamente con el patrón de diseño MVC (del inglés, Model-View-Controller). Éste será pues el punto de partida para el desarrollo del prototipo, ya que separa la interfaz de usuario, el control y el sistema de gestión de la persistencia de forma que es posible su desarrollo de forma independiente.

Debido a la potencia de cálculo necesaria para llevar a cabo las tareas de entrenamiento y predicción de la máquina de aprendizaje, se prevé que, aunque el desarrollo y las pruebas básicas se lleven a cabo en un ordenador personal, la implantación real de un sistema de estas características requerirá de un entorno de procesamiento mucho más potente como, por ejemplo, una granja de servidores.

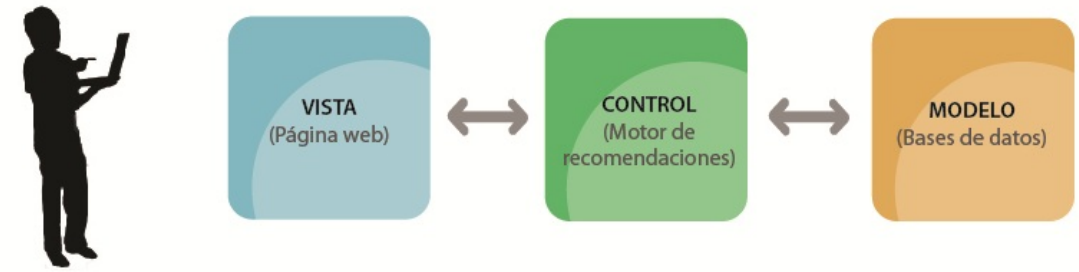
Así, se plantea desde un primer momento la escalabilidad como requerimiento en todos los aspectos de diseño e implementación que se puedan considerar. En este sentido, se prevé que en lugar de una aplicación de escriterio, el usuario accederá a una página web que se comunicará con el motor de recomendaciones. Esta estructuración hace que el sistema sea accesible desde cualquier lugar, requiriendo solamente un navegador web instalado, en lugar de otro software especializado.

Cabe destacar que dicha separación deja abierta la posibilidad de implementaciones futuras en las que un programa de escritorio o una aplicación para un dispositivo móvil realicen la misma función que la página web. Es una decisión de diseño muy interesante debido a la gran expansión que están protagonizando las nuevas tecnologías, como por ejemplo las aplicaciones para dispositivos móviles.

Como medio de almacenamiento se utilizará un servidor de bases de datos que podrá estar alojado en la misma máquina que el motor de recomendaciones o en otra cualquiera, al igual que el servidor web donde se aloja la página. Este planteamiento refuerza la escalabilidad y robustez del prototipo ya que, ante un fallo en el funcionamiento de cualquiera de los módulos, el sistema puede seguir funcionando con normalidad con sólo mover el bloque averiado a una máquina diferente.

La figura 4.1 muestra la nueva estructura del sistema tras tener en cuenta las consideraciones anteriores.

Figura 4.1: *Esquema refinado de la arquitectura del sistema.*

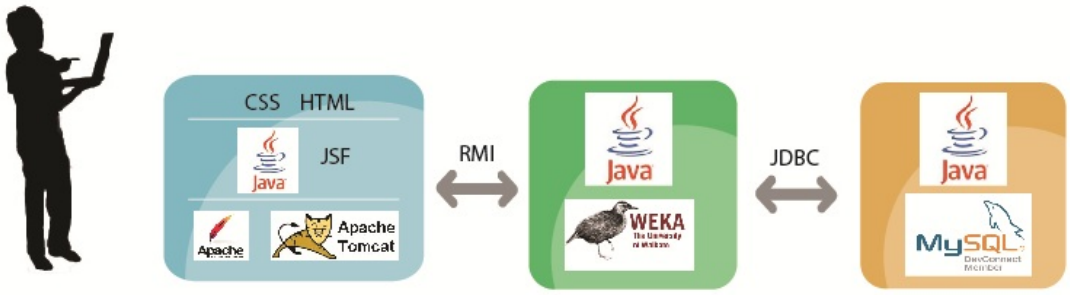


La vista se convierte en una página web que ofrecerá al usuario todas las operaciones disponibles. Se comunicará con el motor de recomendaciones (el control), que validará los datos, realizará los cálculos correspondientes, y almacenará, mediante el sistema de gestión de la persistencia (el modelo), los datos que se precise conservar. El motor de recomendaciones incluirá un módulo actualizador, encargado de revisar periódicamente los *blogs* a los que el usuario se haya subscrito y recomendarle, si procede, la lectura de los nuevos contenidos publicados.

4.2. Elección de las tecnologías

De entre todos los lenguajes de programación disponibles, Java [19] ofrece utilidades que permiten enlazar todos los componentes que forman el sistema, como puede verse en la imagen 4.2. A continuación se detallan cada una de las tecnologías a utilizar.

Figura 4.2: *Estructura del sistema a construir.*



1. **La página web:** Se utiliza el servidor web Apache [20] con su extensión Tomcat [21], que permite el trabajo con Java en el desarrollo de páginas web. La página se implementa con codigo HTML (del inglés, HyperText Markup Language) [22] y hojas de estilo en cascada (CSS, del inglés Cascade Style Sheet) [23]. Los campos de los formularios HTML se relacionan con objetos *Java Bean*[24] mediante el framework Java Server Faces (JSF) [25], para poder ser procesados por el servidor.

2. **El motor de recomendaciones:** Se trata de un objeto RMI (del inglés, Remote Method Invocation) [26] que implementa servidores en forma de objetos remotos, cuyos métodos pueden ser llamados por el cliente como si de un objeto local se tratase. Esta herramienta ofrecida por Java es muy interesante, pues ofrece comunicación entre ambos procesos de forma transparente, sin necesidad de diseñar un protocolo de comunicación.

3. **La máquina de aprendizaje:** Se utilizan máquinas SVM (del inglés, Support Vector Machine). Para la explicación de su funcionamiento, ver Anexo C. En cuanto a la implementación, se parte como punto de partida de la desarrollada por Weka [27], que ofrece un buen número de clasificadores, sistemas internos de cache, y optimizaciones necesarias para la obtención de buenos resultados. Para su uso en este proyecto, se requieren algunas modificaciones sobre el algoritmo original implementado por Weka, que se detallan en el apartado 5.2.

4. **La gestión de la persistencia:** Se implementa con bases de datos de MySQL [28], por su comodidad de uso, eficiencia en las operaciones y por la existencia en Java del driver JDBC (del inglés, Java Data Base Connection) [29], que proporciona una interfaz para la interaccion con la base de datos.

5. **El actualizador:** Se implementa con objetos de la clase Thread de Java [30], y se ejecuta en paralelo al motor de recomendaciones una vez arrancado el sistema.

Capítulo 5

Implementación

Este apartado contiene un resumen de los componentes más relevantes de los tres grandes bloques funcionales del sistema: la vista, el motor de recomendaciones y la capa de acceso a datos. Para una explicación más extensa pueden consultarse los Anexos D, E y F donde además pueden encontrarse otros componentes que no aparecen en esta memoria por motivos de extensión.

5.1. Implementación de la vista

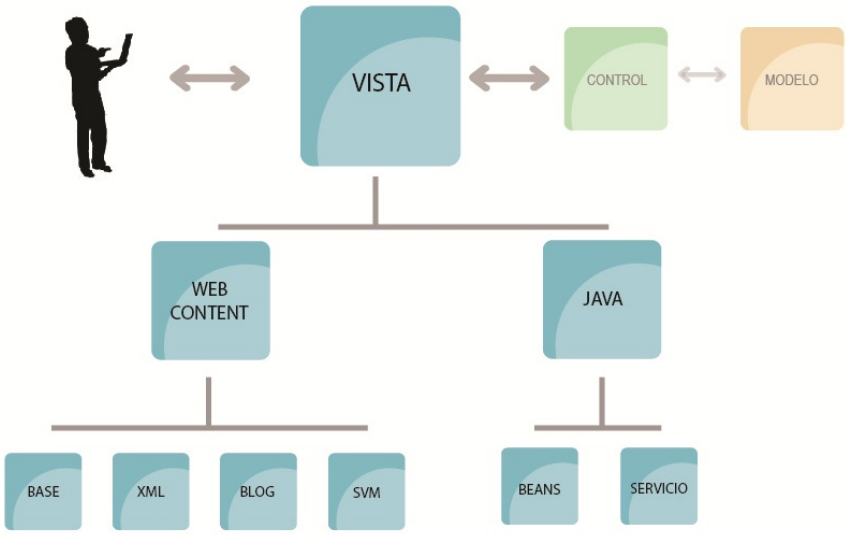
La vista está compuesta por dos bloques diferenciados, la página web y el módulo Java que recoge los datos, comunica con el motor de recomendaciones y devuelve los resultados para ser mostrados al usuario. En los siguientes subapartados se introducen dichos bloques. Una explicación más extensa sobre cada uno de ellos, así como el detalle de otros paquetes no mencionados en esta memoria, pueden consultarse en el Anexo D.

5.1.1. La página web

La página web se implementa con código HTML, hojas de estilo CSS y utilidades ofrecidas por JSF. Está estructurada en carpetas según la funcionalidad de las subpáginas, quedando así separadas las relacionadas con el manejo de los *blogs* de las relacionadas con la gestión de la SVM.

JSF proporciona herramientas para crear plantillas que definen la estructura de la pantalla. En el código HTML de las subpáginas se insertan etiquetas de JSF para incluir los contenidos propios de cada una. De esta forma se separa el contenido del estilo, pudiéndose implementar de forma independiente.

Figura 5.1: *Estructura del bloque de vista.*



5.1.2. El control de página web

Para el manejo de los datos introducidos por el usuario existe un módulo Java estructurado en capas. Los datos pasan de una capa a la otra, viajan al motor de recomendaciones, y los resultados vuelven al módulo Java, atravesando las capas en sentido contrario, para que el servidor genere a continuación la página que incluye los resultados y sea devuelta al usuario.

El paquete contenido en la capa del nivel superior, llamado *Beans*, contiene los objetos donde JSF almacena los datos introducidos por el usuario. Tras comprobar que todos los datos necesarios han sido proporcionados, el paquete envía los datos al nivel inferior, llamado *Servicio*, y espera resultados o errores devueltos por el motor de recomendaciones.

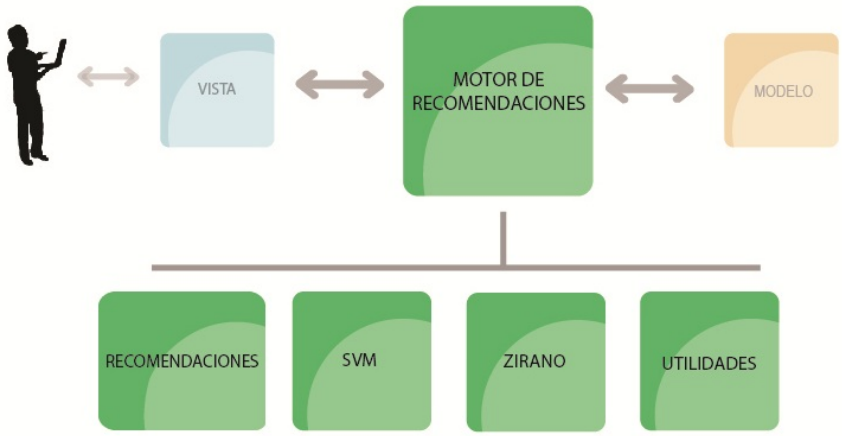
El paquete *Servicio*, en el nivel inferior, contiene el objeto remoto del motor de recomendaciones. Tras recibir los datos procedentes del usuario, que le proporciona el paquete *Beans*, invoca el método remoto correspondiente y devuelve los resultados al nivel superior.

5.2. Implementación del control: Motor de recomendaciones

Se resumen a continuación los componentes más relevantes del motor de recomendaciones, que aparecen en la figura 5.2. Puede consultarse el Anexo E para una explicación

en mayor profundidad, donde además se aportan más detalles sobre la implementación de este módulo.

Figura 5.2: *Estructura del motor de recomendaciones.*



5.2.1. El recomendador

Es el componente principal del bloque de control. Implementa la interfaz del objeto remoto RMI que contiene los métodos que se ofrecen al bloque de vista. Se ejecuta al arrancar el sistema y, tras realizar el ajuste inicial de parámetros internos, queda a la espera de conexiones entrantes.

Cada uno de los métodos remotos implementados valida los parámetros de entrada, comprueba la existencia de los datos a manejar en la base de datos, realiza la operación solicitada y devuelve los resultados, o si procede, una excepción que será tratada en el bloque de vista.

5.2.2. La máquina SVM

Este paquete implementa el clasificador SVM cuyo funcionamiento se expone en el Anexo C. Parte de la implementación de Weka, pero introduce algunas modificaciones para adecuarlo al problema a resolver por el sistema.

El clasificador SVM de Weka implementa el clasificador multiclase expuesto en el artículo *Pairwise Classification as an Ensemble Technique* [16]. Predice a qué clase se asemeja más un nuevo dato desconocido, pero no determina si el dato no pertenece a ninguna de las clases. Por tanto ante un nuevo dato que no pertenezca a ninguna de las clases de entrenamiento, el algoritmo original determinará que pertenece a alguna de las existentes.

La modificación consiste en crear un clasificador binario encargado de reconocer cada una de las clases de entrenamiento. Hay pues tantos clasificadores binarios como clases. Si cualquiera de estos clasificadores determina que el nuevo dato pertenece a la clase que reconoce, el nuevo texto se recomienda al usuario. En caso contrario se descarta, pues no es un texto de interés.

Adicionalmente se añade soporte para el trabajo en paralelo en las tareas de entrenamiento y predicción, que se explica en el apartado 5.4.

5.2.3. El lematizador

El lematizador extrae la raíz de una palabra, como se ha comentado en el apartado 2.1. Es una clase obtenida en SourceForge [31] que implementa el algoritmo de Porter [32]. No se han requerido modificaciones en su algoritmo original.

5.2.4. El módulo de comunicación con el diccionario de Zirano

Para la obtención de los campos conceptuales se ha implementado un módulo que interactúa con el diccionario de Zirano. Su tarea principal consiste en simular la navegación que realizaría un visitante en su página web para obtener una lista de todas las palabras relacionadas con la introducida por el usuario.

El proceso simulado consta de las siguientes fases:

1. El usuario introduce la palabra de su interés.

2. La la página de Zirano sugiere una lista de ideas o acepciones relacionadas con esa palabra.

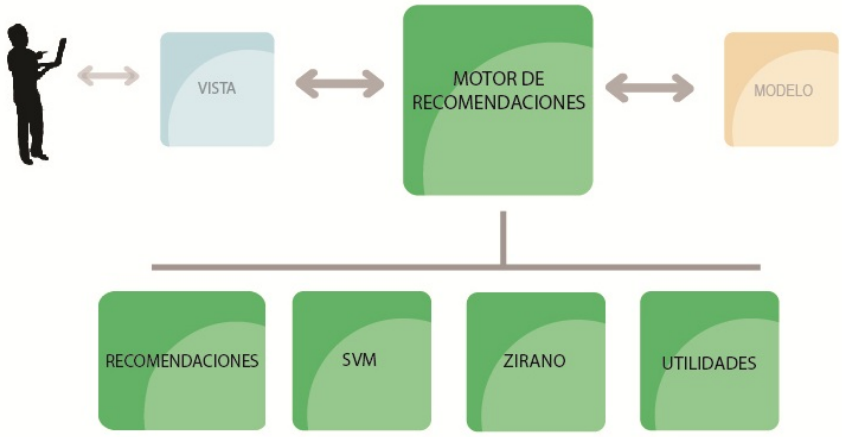
3. El usuario navega por las diferentes acepciones sugeridas y, para cada una de ellas, obtiene un conjunto de palabras relacionadas.

Este módulo realiza un recorrido por todas las ideas sugeridas, hasta extraer un número suficiente de palabras relacionadas.

5.3. Implementación del modelo: Capa de acceso a datos

En este subapartado se resume la implementación de la capa de acceso a datos.

Figura 5.3: *Estructura del módulo de acceso a datos.*



Cabe resaltar la penalización derivada de la descarga de un texto desde internet. Esta operación requiere un tiempo considerable y por ese motivo los textos descargados no se eliminan de la base de datos, sino que permanecen en el sistema, a modo de cache. Una recomendación o un caso de ejemplo harán pues referencia a un texto ya guardado. Pueden borrarse las recomendaciones pero no los textos a los que hacen alusión, de forma que si se vuelve a añadir de nuevo una recomendación o un caso de ejemplo no es necesario descargarlo.

La figura 5.4 muestra las distintas tablas existentes en la base de datos, así como las relaciones entre ellas.

En el paquete DAO existen ocho clases encargadas de interactuar con las tablas de la base de datos con las que están relacionadas. Las clases son las siguientes:

- **BlogDao:** Para inserción, listado y borrado de *blogs* de interés.
- **ClaseDao:** Gestiona el almacenamiento de las clases de entrenamiento.
- **ClasificadorDao:** Ofrece métodos para el almacenamiento en disco de la SVM.
- **ConceptosDao:** Para la gestión de palabras, raíces y campos conceptuales.
- **EntrenamientoDao:** Gestiona el almacenamiento los datos de entrenamiento del sistema.
- **InfoDao:** Para el manejo de la tabla de información del sistema.
- **PostDao:** Ofrece operaciones de inserción de nuevos textos en la base de datos así como de listado. Por motivos comentados al principio de este subapartado no se ofrecen operaciones de eliminación.

Figura 5.4: *Esquema de la base de datos.*

- **RecomendaciónDao:** Gestiona la tabla de recomendaciones.

Para una explicación más detallada, ver Anexo F.

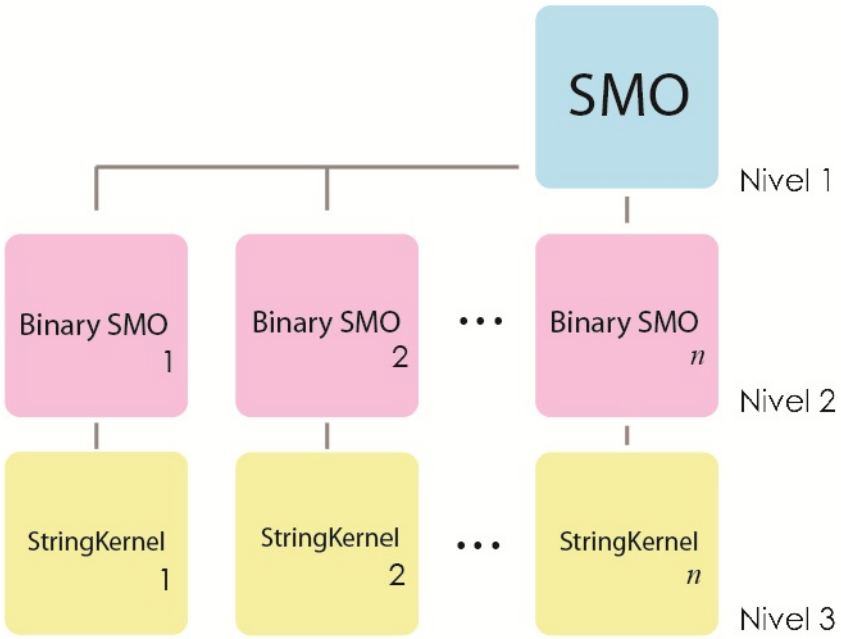
5.4. Paralelización de tareas

Como se ha mencionado en el apartado 3.4, existen varios actividades en el sistema que pueden realizarse en paralelo para obtener mejoras en tiempo. Para este primer prototipo se decidió centrarse en las relacionadas con la máquina de aprendizaje, más concretamente en las operaciones de entrenamiento y predicción.

El trabajo en paralelo puede introducirse en cualquiera de los tres niveles del clasificador SVM implementado, tal y como ilustra la figura 5.5. Pueden hacerse cálculos simultáneos en la función de comparación del Kernel SSK, en las operaciones de entre-

namiento y predicción de las máquinas SVM binarias, o en el clasificador final. Cada una de estas alternativas se explica en los párrafos siguientes.

Figura 5.5: *Niveles de paralelización de la máquina SVM.*



En la función de comparación de dos cadenas, en la clase StringKernel de Weka, hay un punto concreto donde pueden lanzarse cálculos en paralelo. Como se explica en el apartado G.5 del Anexo G, para calcular el grado de semejanza normalizado, se realizan tres llamadas a la función Kernel que son independientes entre ellas. Los resultados obtenidos se multiplican y dividen entre ellos, pero nada impide realizar los cálculos en paralelo guardando los resultados en variables temporales para operar después con ellas.

En el nivel superior al Kernel se encuentran las máquinas SVM binarias. El método de entrenamiento realiza iteraciones, en las que se invoca repetidamente al Kernel y se ajustan coeficientes internos antes de volver a iterar. Es imposible lanzar en paralelo las diferentes iteraciones, pues una iteración necesita los resultados de todas las anteriores. Sin embargo, en cada iteración se realizan varias llamadas a la función Kernel independientes, que sí se pueden ejecutar al mismo tiempo. Lo mismo ocurre con el método de predicción, que realiza comparaciones (invocaciones al Kernel) entre el dato nuevo y cada uno de los vectores de soporte, para construir un resultado. Pueden lanzarse dichas comparaciones en paralelo e ir acumulando los resultados en una variable temporal, para luego realizar cálculos con ella y generar el resultado que será devuelto.

Éstos son los puntos paralelizables a este nivel.

Por último, las diferentes máquinas SVM binarias pueden trabajar en paralelo ya que cada una dispone de su copia individual del conjunto de datos de entrenamiento. Cada una de ellas genera, tanto en el proceso de entrenamiento como en el de predicción resultados independientes de las demás, y por tanto pueden ejecutarse simultáneamente.

La decisión de qué puntos paralelizar y cuales no, no es trivial. Tratar los tres niveles supone la creación de un número elevado de tareas. Implementar sólo los dos inferiores proporciona mejores tiempos al aumentar el número de datos de ejemplo, y sólo el superior proporciona mejores tiempos al aumentar el número de clases diferentes. El grado de paralelismo utilizado en cada nivel debería ser ajustado teniendo en cuenta la cantidad y variedad de textos que el usuario vaya a manejar.

En este prototipo se decidió implementar los dos niveles superiores. Al entrenar y predecir todas las máquinas SVM binarias realizan sus cálculos en paralelo y crean, según sea necesario, nuevas tareas para ejecutar las invocaciones al Kernel SSK. Queda pendiente para ampliaciones futuras la inclusión de trabajo en paralelo en el nivel inferior.

Para la implementación de estas mejoras, se han añadido objetos de la clase ExecutorService [33], incluída en el paquete Concurrent de Java, al clasificador SVM. Estos objetos administran conjuntos de tareas (implementaciones de la interfaz Runnable [34]), con la ventaja de que los hilos de ejecución creados no se destruyen al finalizar la tarea, sino que son reutilizados según quedan libres. Se evita así la creación continua de hilos de ejecución, con su correspondiente penalización de tiempo.

Capítulo 6

Evaluación del prototipo. Experimentos

La naturaleza masiva del cálculo asociado al entrenamiento y a la predicción utilizando máquinas SVM ha planteado como requerimiento, desde un primer momento, la necesidad de diseñar el sistema con la máxima escalabilidad posible.

Así, el estudio del comportamiento paralelo de los procesos más intensivos en cálculo se ha materializado en un conjunto de experimentos que se han ejecutado en distintas plataformas computacionales.

6.1. Descripción del experimento

El almacenamiento de un texto de prueba introducido por el usuario requiere que el sistema descargue su código y lo procese para eliminar etiquetas HTML. Pero, debido a la gran cantidad de formatos de página web que existen y a que los lectores pueden escribir sus comentarios, resulta muy difícil eliminar completamente la información ajena al texto de interés. Esto influye negativamente en la calidad de las predicciones y requiere una mejora del algoritmo de filtrado.

Por ese motivo los experimentos realizados se centran en la influencia de la paralelización sobre las tareas del sistema, concretamente en las operaciones internas de la máquina SVM. Las mejoras en tiempo observadas son independientes de la calidad de los textos de ejemplo y por tanto pueden extraerse conclusiones de mayor interés.

6.1.1. Conjunto de datos de entrada

Se han creado 4 clases de entrenamiento, que son las categorías de las que el usuario desea mantenerse informado. Éstas son: *política*, *deportes*, *economía* y *tecnología*.

Para cada una de ellas, existen 5 artículos de ejemplo relacionados extraídos de las versiones online de los periódicos Heraldo de Aragón[35], Marca[36] y El País[37]

El experimento consiste en un entrenamiento y una predicción de un artículo nuevo, obteniendo tiempos de ejecución secuencial y de ejecuciones paralelas con un número variable de procesadores. Se pretende estudiar la mejora en tiempo obtenida al ejecutar el código en tres entornos distintos, tanto en su versión secuencial como empleando el máximo número de procesadores disponibles.

6.1.2. Presentación del hardware

Para el lanzamiento del experimento se dispone de tres máquinas diferentes que se detallan a continuación.

En primer lugar, mi ordenador portatil, al que en adelante se llamará "Portatil", que tiene las siguientes características:

- Procesador AMD Athlon 64 X2 dual-core QL-60.
- Tipo de máquina: x86.
- Sistema Operativo: Windows 7.
- Número de CPUs: 2.
- Frecuencia de procesador: 1.9 GHz.
- Memoria total: 4GB.

En segundo lugar, se han realizado pruebas en Cluster Hermes [38] del Instituto de Investigación de Ingeniería de Aragón (I3A) de la Universidad de Zaragoza. Concretamente, se ha utilizado la máquina Selene2 bajo Condor[39], con las siguientes características:

- Nodo: selene2.hermes.cps.unizar.es
- Tipo de máquina: x86.
- Sistema Operativo: Linux.
- Versión del Sistema Operativo: 2.6.18-238.5.1.el5.
- Número de CPUs: 48
- Frecuencia de procesador: 2200 MHz.

- Memoria total: 99004784.000 KB
- Tamaño de Swap total: 102399984.000 KB

Por último, Gregorio de Miguel me ofreció la posibilidad de lanzar el experimento en su ordenador. Esta máquina se llamará "Goyo" en adelante y tiene las siguientes caracteristicas.

- Procesador: Intel Core i7 920
- Tipo de máquina: x86.
- Sistema Operativo: Windows 7 64 bits.
- Número de CPUs: 4x2 (2 hilos por CPU).
- Frecuencia de procesador: 3.95 GHz.
- Memoria total: 6GB.

Se ha preparado un *script* para ejecutar los experimentos en las distintas máquinas. En primer lugar se lanza un servidor que carga los datos necesarios para su funcionamiento y queda a la espera de conexiones entrantes. A continuación se lanza un cliente que solicita el entrenamiento del sistema y una predicción de un texto nuevo. Finalizadas ambas tareas, cliente y servidor finalizan su ejecución.

Para el caso del nodo Selene2, los técnicos que dan soporte a los usuarios de Hermes han preparado un *script .sub* para Condor, que puede consultarse en el Anexo H.

6.2. Descripción de los resultados

Las tablas 6.1 y 6.2 muestran los tiempos obtenidos para cada una de las operaciones por separado. Cabe resaltar las diferencias entre las máquinas que han intervenido en el proceso de pruebas de escalabilidad, lo que dificulta la extracción de conclusiones.

Cuadro 6.1: *Tiempo del entrenamiento ejecutado en las distintas máquinas.*

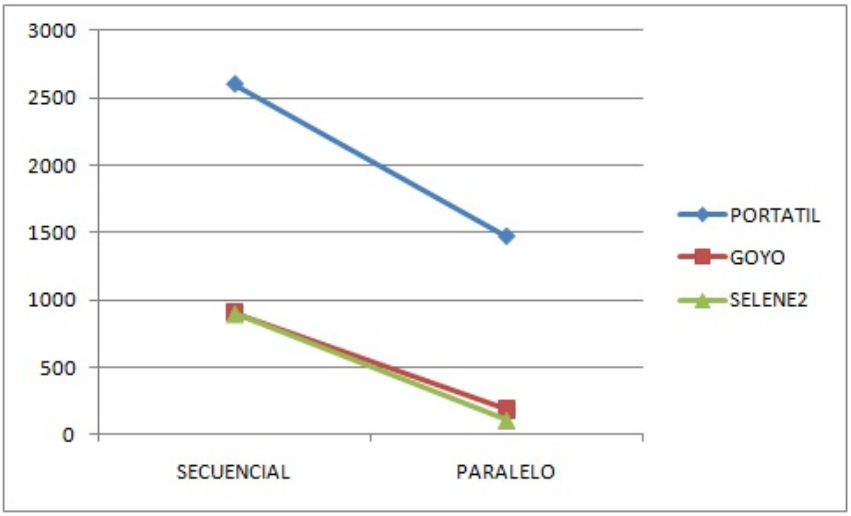
	PORTATIL (2 CPUs)	GOYO (8 CPUs)	SELENE2 (48 CPUs)
SECUENCIAL	2601	909	900
MAXIMO CPUs	1470	186	109

Las figuras 6.1 y 6.2 ilustran gráficamente los resultados de las tablas anteriores. En ambas se representa, para cada máquina, los tiempos obtenidos en ejecucions secuenciales y paralelas.

Cuadro 6.2: *Tiempo de la predicción ejecutada en las distintas máquinas.*

	PORTATIL (2 CPUs)	GOYO (8 CPUs)	SELENE2 (48 CPUs)
SECUENCIAL	164	59	145
MAXIMO CPUs	171	57	134

Figura 6.1: *Resultados de operacion de entrenamiento para cada una de las máquinas.*



En el entrenamiento, el tiempo de cálculo va disminuyendo conforme aumenta el número de procesadores disponibles, especialmente en el caso de mi ordenador portátil en el que duplicar el número de procesadores disminuye el tiempo de cálculo casi a la mitad.

Por otra parte, la operación de predicción obtiene peores resultados. En mi ordenador portátil, trabajar con dos procesadores produce tiempos de ejecución mayores que trabajar secuencialmente. Las otras dos máquinas obtienen mejoras poco significativas de tiempo.

La figura 6.3 muestra las mejora en tiempo obtenida en cada una de las tres máquinas en las operaciones de entrenamiento y predicción.

La mejora en tiempos obtenida es menor a lo que se esperaba durante la implementación del paralelismo. Esto de debe a diversos factores, entre los que se pueden encontrarse los siguientes:

1. **Compartición de unidad de punto flotante:** Si un núcleo de la máquina sólo dispone de una unidad de punto flotante que los distintos hilos de ejecución pueden necesitar utilizar al mismo tiempo, se producen esperas hasta que dicha unidad

Figura 6.2: *Resultados de operacion de predicción para cada una de las máquinas.*

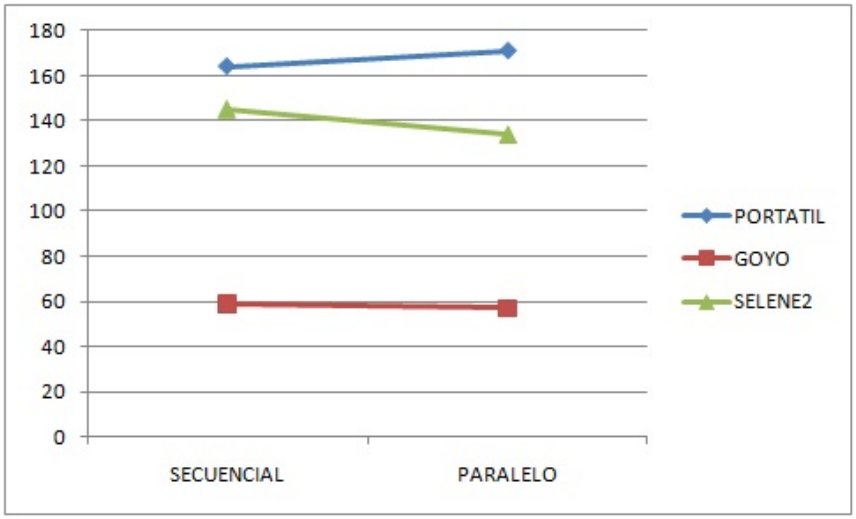
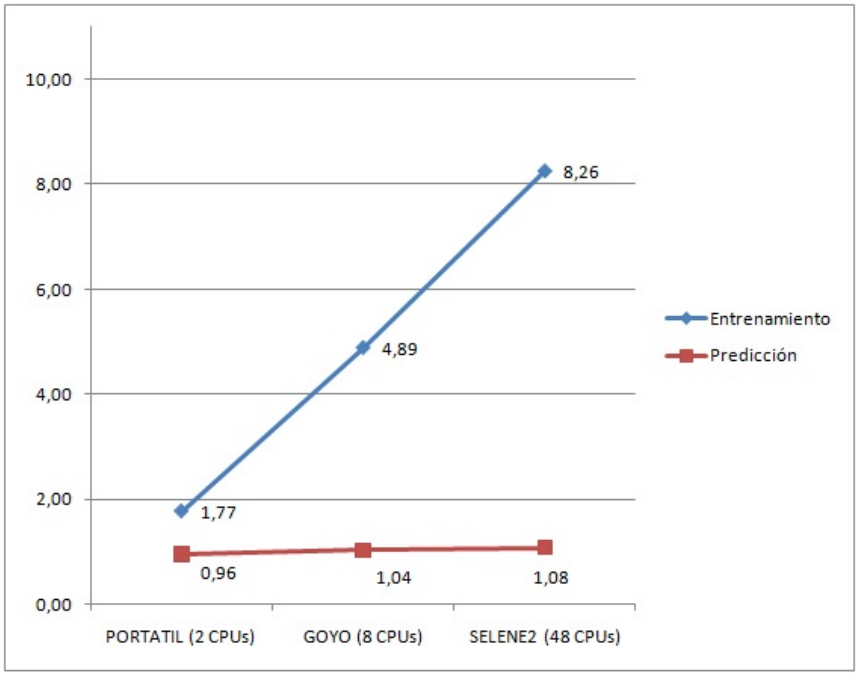


Figura 6.3: *SpeedUp obtenido en las diferentes máquinas.*



queda libre. Este caso se da en procesadores multihilo, en los que se comparten algunas unidades funcionales. En este proyecto, en el que el cálculo con números reales es intenso, este hecho puede producirse frecuentemente.

2. **Compartición de memoria cache:** Los hilos de ejecución comparten algún nivel de cache, como el L3, de manera que al haber varias tareas en paralelo

trabajando con grandes cantidades de información, pueden producirse excesivos movimientos entre memoria principal y cache, con sus correspondientes penalizaciones en tiempo.

3. **Excesivo tamaño de los datos:** Los datos manejados desde la aplicación Java son cadenas de caracteres muy grandes. La utilización de memoria RAM durante la ejecución ha alcanzado en algunos casos los 2Gb. Constructores de copias que manejen cantidades demasiado grandes de memoria introducen también penalizaciones en tiempo.

4. **El alto nivel del código:** La implementación de este prototipo se sustenta en la máquina virtual de Java y su funcionamiento interno, en la librería Weka, en otras estructuras de datos ofrecidas por Java y depende además del planificador de tareas del sistema operativo. En el caso del nodo Selene2, el planificador Condor también influye en los resultados. Todos estos niveles de abstracción a los que no he tenido acceso total, impiden conseguir los resultados óptimos así como predecir las razones exactas que causan los tiempos obtenidos.

Para un desarrollo posterior debería hacerse un estudio más intensivo de las causas que influyen en los resultados. No obstante, el prototipo ofrece una visión adecuada de cómo el sistema se comporta ante la variación de los recursos disponibles.

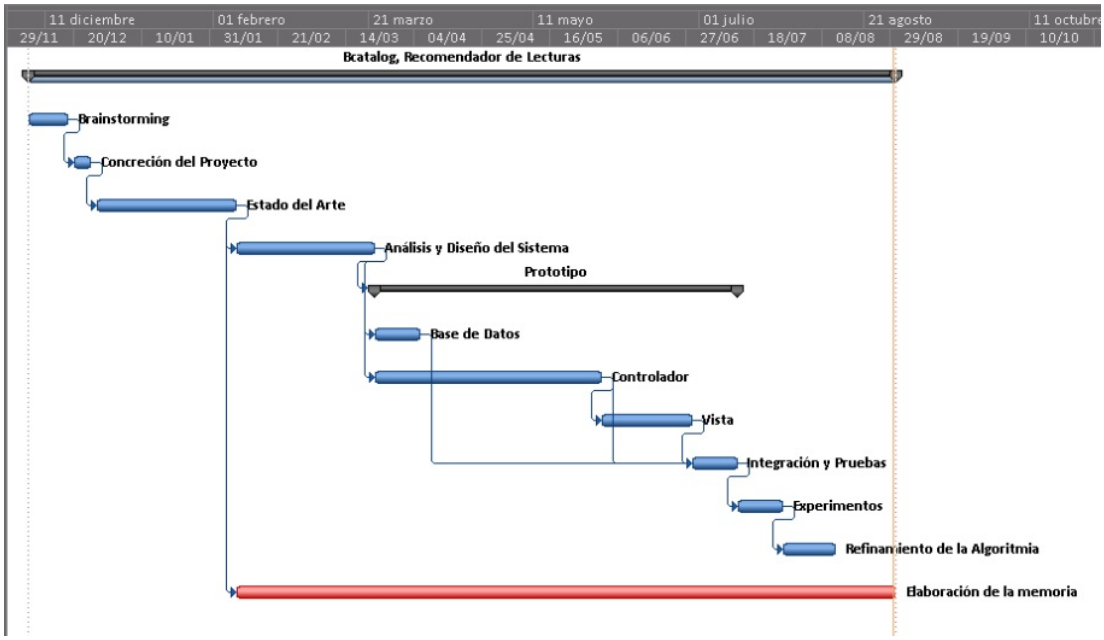
Capítulo 7

Organización del proyecto

En este apartado se expone cómo se ha organizado el desarrollo del proyecto, como puede verse en el diagrama de Gantt de la figura 7.1.

La duración total del trabajo realizado ha sido de 192 días, a lo largo de los cuales se ha ido redactando y refinando el contenido de esta memoria.

Figura 7.1: *Diagrama de Gantt que muestra la organización del proyeecto.*



Las primeras dos semanas se dedicaron a una lluvia de ideas para delimitar el problema a resolver. A continuación se realizó un estudio de los aspectos básicos necesarios para abordarlo. Este estudio requirió un mes de trabajo en el que se leyó abundante documentación al respecto.

Figura 7.2: *Detalle de las fechas de inicio, finalización y duración de las tareas a realizar.*

	Modo de	Nombre de tarea	Duración	Comienzo	Fin
1		**Bcatalog, Recomendador de Lecturas**	**192 días**	**lun 06/12/10**	**mar 30/08/11**
2		Brainstorming	10 días	lun 06/12/10	vie 17/12/10
3		Concreción del Proyecto	5 días	lun 20/12/10	vie 24/12/10
4		Estado del Arte	31 días	lun 27/12/10	lun 07/02/11
5		Análisis y Diseño del Sistema	31 días	mar 08/02/11	mar 22/03/11
6		**Prototipo**	**80 días**	**mié 23/03/11**	**mar 12/07/11**
7		Base de Datos	10 días	mié 23/03/11	mar 05/04/11
8		Controlador	50 días	mié 23/03/11	mar 31/05/11
9		Vista	20 días	mié 01/06/11	mar 28/06/11
10		Integración y Pruebas	10 días	mié 29/06/11	mar 12/07/11
11		Experimentos	10 días	mié 13/07/11	mar 26/07/11
12		Refinamiento de la Algoritmia	12 días	mié 27/07/11	jue 11/08/11
13		Elaboración de la memoria	146 días	mar 08/02/11	mar 30/08/11

La fase de construcción se completó en aproximadamente 80 días, repartidos como puede puede verse en las figuras 7.1 y 7.2.

Terminado el prototipo, se realizaron experimentos y refinó la algoritmia utilizada.

Capítulo 8

Gestión del Proyecto y Conclusiones

Nueve meses de trabajo han permitido la construcción de este primer prototipo. En este apartado se comentan los principales problemas encontrados durante el proceso de desarrollo y se extraen las conclusiones.

El módulo de vista se ha construido sin problemas destacables. Una vez obtenida y revisada toda la documentación necesaria para poner en marcha el desarrollo, éste se ha completado con éxito. Se ha obtenido una página web sencilla, de manejo intuitivo y que cumple perfectamente su función.

En el módulo de control, el motor de recomendaciones, el trabajo ha sido más complicado, fundamentalmente debido a que se ha trabajado con herramientas de terceras partes y a que el proyecto requiere el uso de máquinas muy potentes, de las que no se ha dispuesto.

Se ha conseguido obtener campos conceptuales amplios pero concisos, si bien convendría revisar el algoritmo utilizado y adaptarlo para obtener conjuntos más reducidos pero con mayor relevancia.

El algoritmo de aprendizaje, la máquina SVM, funciona correctamente. Las predicciones realizadas por el sistema son correctas en buen número de casos, aunque es importante recalcar que la precisión de las mismas depende en gran medida de la calidad de los textos de ejemplo.

Conseguir estos casos de ejemplo no es trivial, pues hay que filtrar el código HTML recibido por el sitio web propietario del texto para eliminar, además de las etiquetas

HTML, la información que no pertenece al texto propiamente dicho. Sería recomendable, de cara a próximas ampliaciones, la revisión del algoritmo de filtrado, para conseguir eliminar más información no relevante, como comentarios de los lectores. Filtrar correctamente todas las páginas web es complicado, pues los diseñadores no siguen ningun estándar de nombrado de los diferentes bloques que componen la página, lo que dificulta el análisis preciso del codigo.

Las pruebas de paralelización son, en mi opinión, un punto mejorable en el proyecto. Realizar un buen entrenamiento para obtener recomendaciones precisas obliga a introducir un número de ejemplos lo suficiente grande para que el sistema sepa distinguir los textos de las diferentes categorías. No se ha dispuesto de máquinas con la potencia de cálculo necesaria para poder realizar pruebas reales y por tanto, ha sido imposible realizar pruebas con grandes volúmenes de datos para poder asegurar que la escalabilidad del sistema es buena.

Respecto a la paralelización, tampoco ha sido posible obtener resultados tan favorables como se esperaba, pues el sistema trabaja con demasiadas capas (máquina física, sistema operativo, máquina virtual de Java, librería Weka, mi propio código) y no se ha podido acceder a todas para estudiar las causas de los resultados obtenidos. Por tanto no he podido concluir hasta qué punto mi trabajo de paralelización es mejorable, que seguramente lo es, ni hasta qué punto depende de la máquina, del algoritmo de Weka, ni de ningun otro componente utilizado.

Sin embargo, me gustaría agradecer de nuevo a Gregorio de Miguel, mi director de proyecto, así como al proyecto TIN2008-06582-C03-02 - “Secuencias Simbólicas: Análisis, Aprendizaje, Minería y Evolución”, del Ministerio de Ciencia e Innovación, y al Grupo de Ingeniería de Sistemas de Eventos Discretos (GISED), haberme facilitado el acceso a máquinas de mayor potencia, gracias a las cuales he podido realizar pruebas más interesantes que las que podría haber ejecutado utilizando sólo mi ordenador portátil.

Siendo crítico, veo algunos puntos mejorables en el resultado visible del proyecto, como ya se ha comentado en los párrafos anteriores. A pesar de ello, mi valoración personal del trabajo realizado es buena. Pienso que el desarrollo se ha realizado de forma ordenada después de una etapa de obtención y lectura de la documentación disponible. Esto ha permitido que el trabajo de desarrollo sea más fluido y con menos imprevistos por un diseño precipitado.

ANEXOS

Anexo A

Manual de usuario

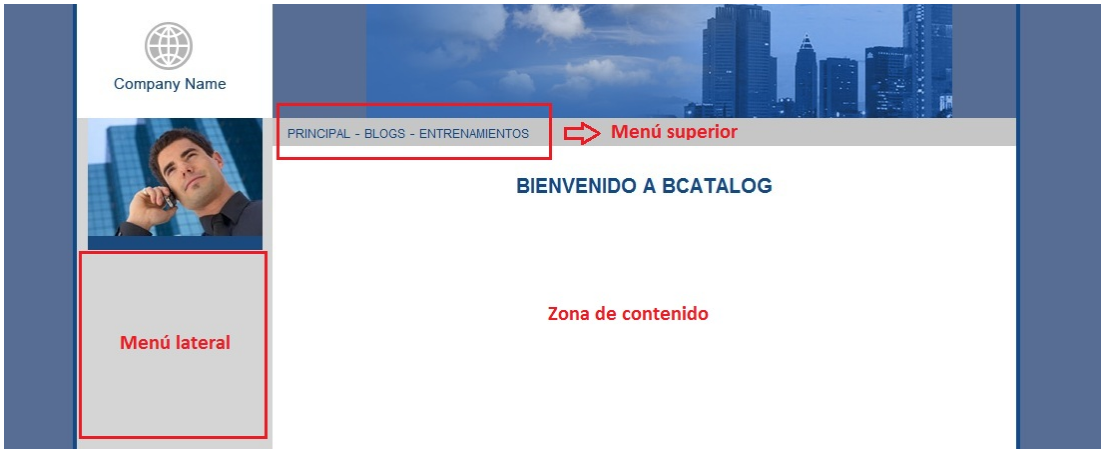
BCatalog es un sistema de recomendación de *blogs* que notifica al usuario cuando alguno de sus sitios favoritos publica contenidos relacionados con temas de su interés. Para el manejo del sistema el usuario dispone de una página web muy intuitiva. De esta forma no se requiere ningún software instalado en su ordenador a excepción de un navegador web, pudiendo así mantenerse informado en casa, en el trabajo o en cualquier lugar donde se encuentre.

En los apartados siguientes se resume toda la información necesaria para empezar a obtener recomendaciones y se muestran capturas de pantalla que facilitarán la comprensión del lector y su familiarización con la aplicación.

A.1. Estructura de la página

La página web está formada por tres zonas principales con las que el usuario puede interactuar, tal como se ilustra en la figura A.1:

Figura A.1: *Estructura de la página.*



1. El menú de navegación superior presenta las secciones de la página web. A través de este menú el usuario puede visualizar las recomendaciones propuestas, gestionar sus *blogs* favoritos y entrenar al sistema para que empiece a recomendarle nuevos contenidos.

2. El menú izquierdo presenta las subsecciones existentes en la sección del menú superior seleccionada.

3. La zona de contenido muestra el texto y los formularios correspondientes a cada sección.

A.2. Gestión de blogs

Seleccionando la opción *Blogs* del menú superior se accede a la pantalla de gestión de las subcripciones. Todas las operaciones disponibles aparecen juntas en esta sección, para la cómoda manipulación de los *blogs* favoritos del usuario. Las acciones que pueden realizarse aparecen reflejadas en la figura A.2, y son las siguientes:

Figura A.2: *Gestión de subscripciones a blogs.*

1. Añadir una subscripción a un *blog*

2. Ver el listado de subscripciones en curso.

3. Cancelar una subscripción. BCatalog dejará de recomendar el *blog* seleccionado.

A.3. Entrenamiento del sistema

En la sección *Entrenamientos* están disponibles todas las acciones necesarias para que el sistema aprenda las preferencias del usuario en cuanto a temas de interés se

refiere. El usuario proporcionará a BCatalog un conjunto de textos de ejemplo para cada uno de los temas de los que desee recibir notificaciones.

El proceso de entrenamiento del sistema se lleva a cabo en dos fases tras las cuales BCatalog estará listo para empezar a analizar los *blogs* seleccionados en busca de nuevos contenidos relacionados. Estas tareas son accesibles desde el menú lateral de la página web, como ilustra la figura A.3. Es importante que el sistema vuelva a entrenarse a lo largo del tiempo añadiendo nuevos textos de ejemplo que pueden ser introducidos manualmente o bien ser recomendaciones que el usuario considere acertadas. A continuación se enumeran las distintas fases del entrenamiento, que se describen en detalle en los subapartados siguientes:

1. Creación de las categorías de interés del usuario (ej: economía).
2. Introducción de textos de ejemplo, especificando la categoría a la que pertenecen.
3. Entrenamiento del sistema.

Figura A.3: *Entrenamiento del sistema.*

A.3.1. Gestión de categorías de textos

La gestión de categorías puede realizarse desde la pantalla *Administrar clases de entrenamiento*, en el menú lateral izquierdo, una vez seleccionada la opción *Entrenamientos* del menú superior.

Esta pantalla es muy similar a la de gestión de blogs, introducida en el apartado A.2 y ofrece las siguientes acciones a realizar, que aparecen en la figura A.4:

1. Añadir una clase de entrenamiento (categoría a la que puede pertenecer un texto).

Figura A.4: *Gestión de clases de entrenamiento.*

2. Ver el listado de las clases existentes

3. Eliminar una clase de entrenamiento.

A.3.2. Gestión de textos de ejemplo

Una vez definidas las clases de textos que el sistema va a manejar, el siguiente paso es añadir casos de ejemplo de cada una de las clases. Pueden pertenecer, o no, a los *blogs* a los que el usuario se ha subscrito. Con estos textos BCtalog aprenderá a identificar las preferencias del usuario, y por lo tanto es necesario que los ejemplos introducidos sean lo más relevantes posible para asegurar la calidad de las recomendaciones.

Figura A.5: *Inserción de un nuevo texto de ejemplo.*

Para la introducción de los datos de ejemplo la página web proporciona la pantalla *Nuevo caso de entrenamiento*. En ella, el usuario deberá introducir la dirección URL donde se encuentra el texto y seleccionar la categoría a la que pertenece, de entre las ya creadas anteriormente. Este proceso se ilustra en la figura A.5.

La pantalla *Administrar casos de entrenamiento* ofrece un listado de todos los ejemplos almacenados hasta el momento mostrados por categorías, como puede verse en la figura A.6. Junto a cada caso de ejemplo se ofrece la opción de borrarlo si el usuario decide que no le interesa que esté asociado a esa cagegoría.

Figura A.6: *Gestión de textos de ejemplo.*

Es importante comentar que en este listado aparecen tanto los ejemplos introducidos por el usuario como aquellas recomendaciones que el usuario haya aceptado.

A.3.3. Entrenamiento del sistema

Completadas las fases anteriores el usuario puede entrenar al sistema desde la pantalla *Gestión de SVM*, mostrada en la figura A.7, que es accesible desde el menú lateral. BCatalog iniciará el proceso de entrenamiento de su máquina de aprendizaje. Este proceso puede requerir un tiempo para ser completado.

A partir de ese momento, BCatalog visitará periódicamente los *blogs* favoritos del usuario, descargará y analizará los nuevos contenidos y recomendará, si procede, su lectura al usuario.

Esta pantalla, ofrece adicionalmente la posibilidad de introducir la URL de un texto cualquiera, y solicitar a BCatalog que lo analice y añada la correspondiente recomendación. Con esta utilidad el usuario puede comprobar el funcionamiento del sistema y la calidad de las predicciones.

Figura A.7: *Entrenamiento del sistema y predicción.*

A.4. Recomendaciones

Cuando BCatalog determine que los nuevos textos están lo suficientemente relacionados con las preferencias del usuario, añadirá nuevas recomendaciones que el usuario podrá ver cada vez que ingrese en la página y hasta que las acepte o rechace.

En la pantalla *Principal*, que aparece en la figura A.8 y es accesible desde el menú de navegación superior, aparecerá un listado con las recomendaciones no pendientes de verificación. Cada recomendación tiene su correspondiente botón para informar a BCatalog de si es o no del agrado del usuario.

Figura A.8: *Ejemplo de recomendaciones propuestas por el sistema.*

En caso de que el usuario esté satisfecho con las recomendaciones, pasarán a formar parte de los textos de ejemplo y serán tenidas en cuenta cuando el sistema vuelva a entrenarse. En caso contrario la recomendación desaparecerá.

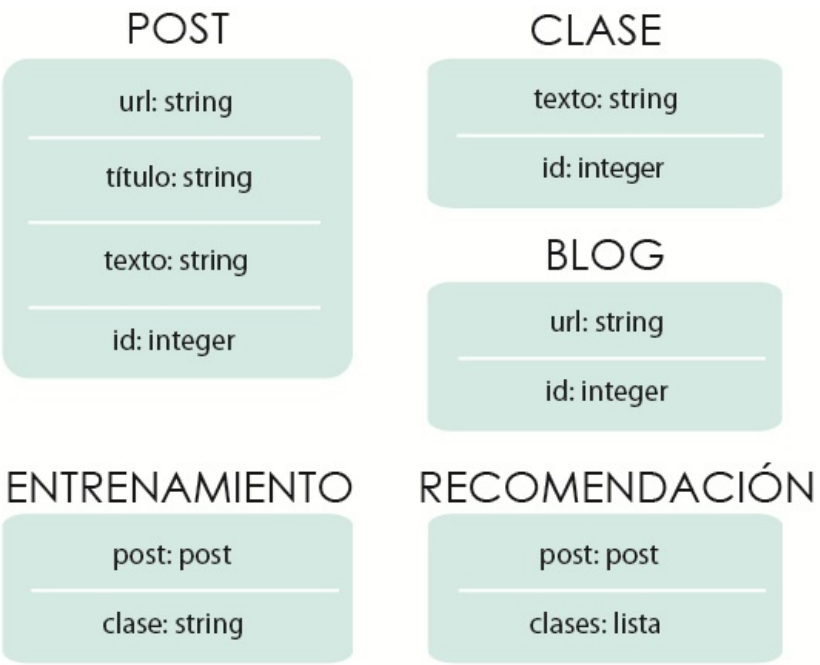
Anexo B

Relación de clases del sistema

En este apartado se detallan las clases que el sistema maneja para realizar las tareas de análisis y recomendación de textos publicados en *blogs* de interés del usuario. Todas ellas se incluyen en el paquete *Entities*, presente tanto en la vista como en el motor de recomendaciones.

La figura B.1 ilustra las cinco clases y sus atributos. El significado de cada uno de ellos se detalla a continuación:

Figura B.1: *Clases almacenadas por el sistema.*



Blog: Representa un *blog* de cuya publicación de nuevos contenidos se notificará al usuario. Está formado por los siguientes atributos:

- *URL*: Una cadena de texto que almacena su dirección URL (del inglés, Uniform Resource Locator).
- *id*: Un entero que almacena un identificador único del *blog* para uso interno.

Clase: Representa una clase de entrenamiento, que en este protipo es una categoría a la que los textos pueden pertenecer. Sus atributos son:

- *nombre*: Una cadena de texto que contiene las palabras que forman el nombre de la categoría.
- *id*: Un entero que almacena un identificador único de la clase para uso interno.

Post: Representa un texto de internet, y contiene toda su información relevante para el sistema:

- *URL*: Una cadena con su dirección URL.
- *id*: Un entero que almacena un identificador único del *post* para uso interno.
- *titulo*: Una cadena que almacena el titulo del texto.
- *texto*: Una cadena que almacena el texto de interés.

Entrenamiento: Representa un caso de ejemplo que será utilizado por la máquina de aprendizaje. Está formado por los siguientes atributos:

- *post*: Un objeto de la clase Post, que contiene toda la información del texto.
- *clase*: La clase o categoría a la que pertenece.

Recomendación: Representa una recomendación de un texto, que se ofrecerá al usuario tras haber determinado que es de su interés. Sus atributos son:

- *post*: Un objeto de la clase Post con la información sobre el texto.
- *clases*: Una lista de cadenas que representan las clases con los que el *post* está relacionado.

El significado de los atributos resulta obvio, ya que simplemente representan la información que el sistema necesita manipular. Cada clase ofrece únicamente métodos para la lectura y escritura de sus atritubos.

Anexo C

La Máquina de Vectores de Soporte

Las Máquinas de Vectores de Soporte (en adelante SVM, del inglés Support Vector Machine) son un conjunto de algoritmos de aprendizaje supervisado, mediante los que se puede enseñar a un sistema a diferenciar elementos de diferentes clases.

Para ello, es necesario entrenarlo proporcionándole un conjunto significativo de casos de ejemplo. Tras el proceso de aprendizaje, el sistema será capaz de predecir a qué clase de las que ha aprendido a reconocer pertenece un nuevo dato desconocido.

En los siguientes apartados se revisan algunas nociones necesarias para entender cómo funcionan las SVM (apartado C.1) y se introducen las funciones Kernel (apartado C.2), centrando el interés en la funcion SubString Kernel (apartado C.3) que es la utilizada en este proyecto.

C.1. Introducción

El funcionamiento detallado de este tipo de herramientas excede los objetivos de esta memoria, puesto que se requieren conocimientos matemáticos avanzados. Por tanto se va a explicar, de forma intuitiva, en qué consisten los procesos de entrenamiento y de predicción. Para información más detallada, puede consultarse cualquiera de los libros en los que se basa el desarrollo del proyecto: *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond* [14], *Pairwise Classification as an Ensemble Technique* [16] o *Kernels for Structured Data* [15].

Es importante tener en cuenta que la resolución de sistemas de ecuaciones lineales es computacionalmente muy eficiente ya que para ello pueden usarse productos escalares

entre vectores, idea en la que se apoyan las SVM. En cambio, la resolución de sistemas no lineales obliga a emplear otro tipo de métodos más complejos y menos eficientes.

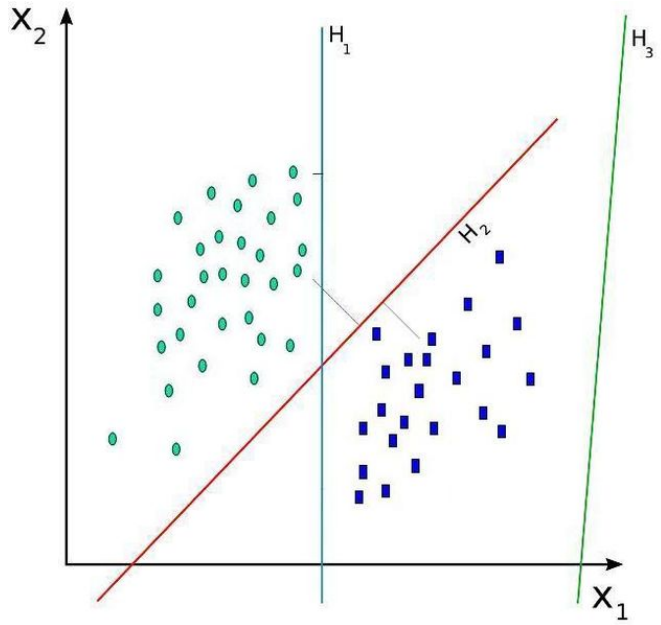
Para ilustrar los procesos de entrenamiento y predicción, se van a utilizar objetos con dos propiedades, y dos clases de entrenamiento a las que dichos objetos pueden pertenecer.

Sea X el conjunto de datos de ejemplo, X_1 y X_2 las propiedades de interés de dichos datos. Un dato $x \in X$ se representa de la forma $x = (x_1, x_2)$, tal que $x_1 \in X_1$ y $x_2 \in X_2$.

El objetivo de la máquina SVM cuando se entrena, es determinar un hiperplano que separe los objetos de ambas clases, de forma que además la distancia entre el objeto más cercano al hiperplano en cada una de las clases sea máxima.

En la figura C.1 se muestra cómo se vería representado un conjunto de datos de ejemplo conocidos, separados por el hiperplano calculado por la SVM.

Figura C.1: *Representación de objetos e hiperplanos de separación.*



Existen infinitos hiperplanos que separan las clases de entrenamiento, como pueden ser H_1 y H_2. La SVM elige H_2, pues, como se ha comentado, interesa quedarse con el que permita un margen máximo entre ellas. H_3 no separa las dos clases, y por tanto no es un hiperplano válido. En este caso, y dada la naturaleza del ejemplo, los hiperplanos son 1-dimensionales.

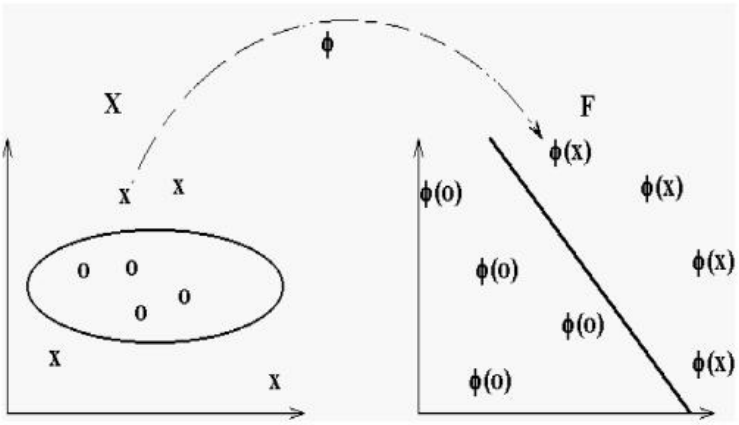
C.2. Las funciones Kernel

En la mayoría de los casos reales, como casos con más de dos propiedades (variables predictoras), no es posible conseguir que todos los elementos de cada clase queden separados por un hiperplano, sino que se consiguen curvas de separación. La solución más eficiente en este tipo de problemas consiste en proyectar la información a un espacio de dimensión superior de forma que, con esta nueva representación, sea posible encontrar un hiperplano y obtener un problema lineal, que puede resolverse eficientemente.

Para realizar esta proyección se utiliza una función de mapeo Φ que asocia a cada elemento del conjunto original un vector de dimensión superior, como puede verse en la figura C.2.

Sea F el conjunto formado por los mapeos de los datos de entrada. Formalmente, $F = \{\Phi(x), \forall x \in X\}$. Determinado dicho conjunto, la máquina SVM sí va a ser capaz de encontrar un hiperplano que separe los mapeos de los objetos de las distintas clases, como ilustra la figura.

Figura C.2: *Ejemplo de función de mapeo* Φ.



La dimensión del conjunto F puede ser muy elevada. Puesto que internamente la SVM trabaja calculando productos escalares de los datos representados como vectores, almacenar vectores de tan altas dimensiones resulta poco práctico en casos no triviales.

Es aquí donde cobran importancia las funciones Kernel, de las que todavía no se ha hablado. Estas funciones reciben dos vectores y devuelven el producto escalar de sus mapeos Φ.

Además, las funciones Kernel cumplen la igualdad C.1, donde $\langle \ldots \rangle$ representa el producto escalar entre dos vectores.

$$K(x_1, x2) = \langle \phi(x_1), \phi(x_2) \rangle = \langle x_1, x_2 \rangle, x_1, x_2 \in X \tag{C.1}$$

La igualdad C.1 significa que el producto escalar de los datos de entrada coincide con el producto escalar de sus mapeos Φ. Este producto es el que calcula la función Kernel y se utiliza como medida de similitud entre dos vectores.

Una característica importante de este tipo de funciones es que no necesitan calcular y almacenar los mapeos completos para realizar el producto escalar. En su lugar, van calculando cada componente, multiplicándolas y acumulando los resultados.

En ocasiones es útil que el resultado de la evaluación de la función Kernel esté comprendido entre 0 y 1 o, lo que es lo mismo, que esté normalizado. La ecuación C.2 muestra el cálculo del producto escalar normalizado.

$$k_{norm} = \frac{K(x_1, x_2)}{\sqrt{K(x_1, x_1) \cdot K(x_2, x_2)}}, k_{norm} \in [0, 1] \tag{C.2}$$

Existen numerosas funciones Kernel diferentes y la elección de la más adecuada depende del problema de clasificación a resolver. En este proyecto se utiliza el SubString Kernel cuyo funcionamiento se describe en el apartado C.3.

El proceso de entrenamiento de la máquina SVM consiste, como se ha mencionado, en la obtención de un hiperplano de separación que ofrezca un margen de error máximo. Los vectores de soporte son los que forman la base de dicho hiperplano, y son calculados en este proceso.

El proceso de predicción se realiza una vez se han obtenido los vectores de soporte. Mediante productos escalares, calculados por la función Kernel, se determina a qué lado del hiperplano se encuentra el nuevo dato del que se quiere conocer su clase. Cada clase queda separada a un lado del hiperplano, por lo tanto, el lado donde se determine que el nuevo objeto se encuentra determinará la clase a la que pertenece.

C.3. El SubString Kernel

La función SubStringKernel (en adelante SSK) es el Kernel más conveniente en la tarea de análisis de textos que es objeto del proyecto y por eso ha sido la alternativa elegida de entre las distintas funciones Kernel. Los cálculos realizados son complejos

y pueden consultarse en detalle en el artículo *Text Classification using String Kernels*[17]. En este apartado se introduce su funcionamiento intuitivamente para facilitar la comprensión del lector.

Se puede consultar los libros *Text Classification using String Kernels*[17] y *Lambda pruning: an approximation of the string subsequence kernel for practical SVM classification and redundancy clustering*[18] para una explicación en mayor profundidad acerca del kernel SSK y técnicas para su optimizacion.

El Kernel SSK realiza un cómputo de las subsecuencias comunes entre dos textos. Una subsecuencia es un conjunto de caracteres consecutivos, o no, dentro de una cadena de texto.

Algunos ejemplos de subsecuencias de la palabra *trampolín* son *tram*, *tapon*, *trín*, *t* o *tn*.

Este Kernel asigna más importancia a las subsecuencias de mayor longitud y penaliza a las formadas por caracteres no consecutivos. Cuanto más separados estén los caracteres de la subsecuencia, más penalización sufrirán y por tanto menos valor aportarán al resultado de la comparacion.

A modo de ejemplo, sean $t_1 =$ *"Trampolín"*, $t_2 =$ *"rampa"* y $t_3 =$ *"Tan"* tres cadenas cortas. El Kernel SSK determinará que t_1 está más relacionada con t_2 que con t_3, puesto que *rampa* tiene 3 caracteres comunes consecutivos con *Trampolín*, además del caracter *a*, también común. En cambio, *Tan* tiene sus 3 caracteres en común pero no son consecutivos, de modo que serán penalizados en el cómputo.

Los valores devueltos por esta función dependen de la longitud de las cadenas. Si se comparan dos cadenas muy largas y poco relacionadas es frecuente obtener un resultado mayor que si se comparan dos cortas muy similares. Una de las razones es que los caracteres individuales son también considerados en el cómputo, con lo que aportarán valor al resultado aun siendo poco relevantes.

Por este motivo conviene que la función devuelva valores normalizados, teniendo en cuenta la longitud de las cadenas que procesa. De esta forma se obtienen valores entre 0 y 1, con lo que se puede medir y comparar mejor el grado de semejanza entre cadenas de longitudes muy dispares, así como comparar los resultados en distintas comparaciones.

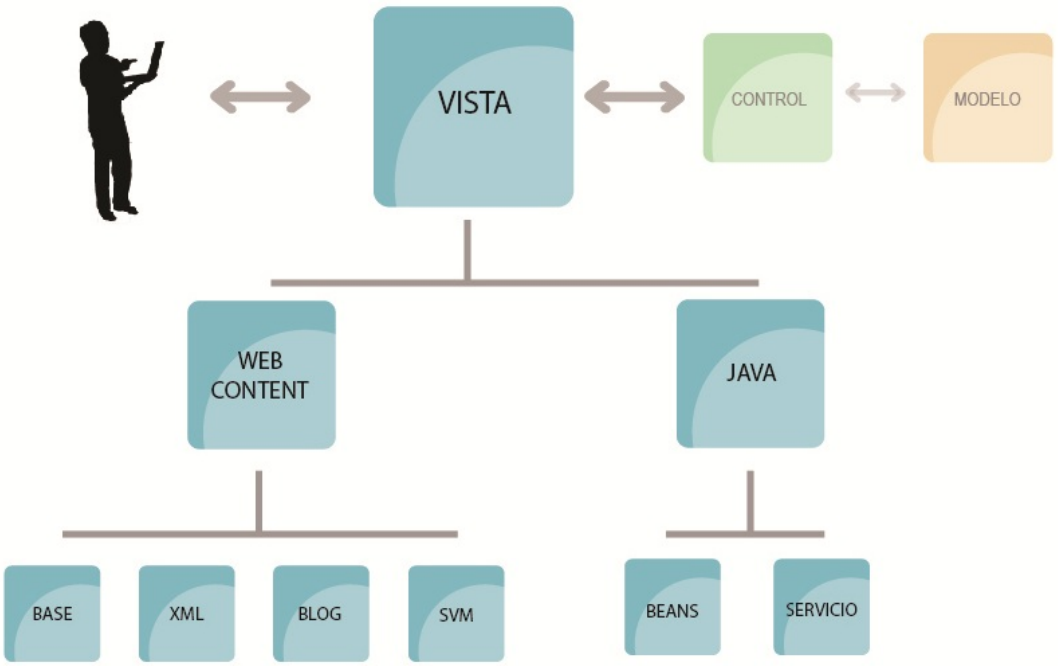
Anexo D

La página web

La página web constituye la interfaz de comunicación entre el usuario y BCatalog. El módulo de vista está pues formado por dicha página y una serie de herramientas que recogen los datos proporcionados, contactan con el motor de recomendaciones y muestran al usuario la información devuelta por éste.

La figura D.1 ilustra la estructura interna del módulo vista, y los componentes que forman tanto la página web como el bloque Java que procesa los datos.

Figura D.1: *Estructura del módulo de vista.*



La carpeta *Web Content* se estructura en subcarpetas y contiene todos los archivos necesarios para el funcionamiento de la página web. El capítulo D.1 detalla cada uno de estos elementos, entre los que se incluyen los siguientes:

1. La carpeta *base*, que contiene la plantilla de las pantallas que componen la web.
2. Distintos archivos XML[40] necesarios para la configuración de la página.
3. La carpeta *blog*, que incluye las pantallas asociadas a la gestión de *blogs*.
4. La carpeta *svm*, en la que se encuentran las pantallas relacionadas con el aprendizaje del sistema.

El módulo Java incluye dos paquetes que forman una jerarquía por niveles, explicada en mayor profundidad en el apartado D.2. Estos paquetes son:

1. El paquete *beans*, que recoge los datos procedentes de la página web y almacena la información devuelta por motor de recomendaciones y que será mostrada al usuario en respuesta a sus peticiones.
2. El paquete *servicio* que recibe los datos del paquete *beans*, envía la petición al motor de recomendaciones y devuelve los resultados al nivel superior.

D.1. La carpeta Web Content

La carpeta *Web Content* contiene los archivos HTML de la página web, la plantilla que define la estructura de las pantallas serán mostradas al usuario, una serie de archivos XML que configuran el comportamiento del servidor y el archivo *index.xhtml* que es el punto de entrada por defecto a la página.

D.1.1. La carpeta base

En el desarrollo de páginas web resulta muy útil separar el estilo de la página del contenido dinámico generado como resultado a las peticiones del usuario. Con ello se consigue una división de tarea que facilita la depuración de errores, el trabajo entre miembros del equipo de desarrollo y las posibles ampliaciones del sistema a construir.

Ésta es la razón de la existencia de la carpeta *base*, en la que se especifican por separado la plantilla de la página web (fichero *template.xhtml*), los estilos que personalizan su apariencia (carpeta *css*) y las imágenes que aparecerán en la página a mostrar al usuario (carpeta *images*).

El fichero *template.xhtml* define la estructura de la página, las zonas de la pantalla donde se pueden colocar contenidos dinámicamente. Es en él donde se especifica que la pantalla estará formada por una cabecera con el logotipo de la aplicación, un menú superior de navegación, uno lateral y una zona central principal para mostrar información al usuario.

El contenido de cada una de las zonas no aparece en este archivo, sino que se incluirá posteriormente dependiendo de la página solicitada. Esta forma de trabajo es similar a la utilizada en los lenguajes de programación, en las que determinados ficheros con código pueden incluirse en otros donde vayan a ser utilizados.

Los colores, el tipo y tamaño de la fuente, las dimensiones de las zonas que componen la pantalla y todas las demás propiedades relacionadas con el estilo del sitio web se implementan por separado en el archivo *style.css*, ubicado en la carpeta *css*.

La carpeta *images* contiene las imágenes mostradas en la página resultado, como son el logotipo de la aplicación, la imagen de la cabecera superior y otras que se utilizan como fondo en las distintas zonas de la pantalla.

D.1.2. Los archivos XML

Para el correcto funcionamiento de la página web se requieren fundamentalmente dos archivos que configuran el comportamiento del sitio ante la actividad del usuario. Estos archivos se encuentran en la carpeta *WEB-INF*, y se comentan a continuación:

1. *faces-config.xml*: Especifica los objetos Java que manejarán la información intercambiada entre el usuario y el sistema. Este fichero indica a JSF qué campos de los formularios HTML ha de asociar a qué objetos Java para su manipulación posterior. Incluye además reglas de navegación que especifican a qué página se ha de redirigir al usuario cuando realice cada acción de las ofrecidas en la página web.

2. *web.xml*: Aporta información al servidor web, como la página principal por defecto, la extensión en que acaban las direcciones URL de cada una de las secciones (seccion.html, seccion.jsf o cualquier otra que se decida) y otras indicaciones de utilidad.

D.1.3. Las carpetas blog y svm

Por último, las carpetas *blog* y *svm* contienen archivos con el código HTML propio de cada sección existente en la página web. Es importante destacar que el código que

aparece en cada uno de estos archivos define únicamente el contenido de cada una de las zonas de la pantalla especificadas en el fichero *template.xhtml*. Para obtener codidigo final, el servidor realizará una serie de operaciones que se comentan en el apartado D.1.4

La carpeta *blog* contiene un único archivo llamado *gestion.xhtml*, en el que se encuentra el código para generar los formularios para añadir, listar y borrar los *blogs* a los que el usuario quiere subscribirse.

La carpeta *svm* contiene los siguientes archivos que implementan las pantallas para gestionar el aprendizaje y entrenamiento de BCatalog:

1. *index.xhtml*: Es el punto de entrada a la sección de gestión de aprendizaje del sistema.

2. *gestion_clases.xhtml*: La pantalla que ofrece al usuario la creación de una categoría para sus textos de ejemplo.

3. *nuevo_entrenamiento.xhtml*: La pantalla para la inserción de textos de ejemplo.

4. *gestion_entrenamiento.xhtml*: La pantalla que muestra el listado de textos de ejemplo agrupados por categorías.

5. *svm.xhtml*: La pantalla desde la que se puede entrenar al sistema.

D.1.4. Proceso de construcción de una pantalla

Como se ha mencionado repetidas veces a lo largo de este apartado, el estilo, la estructura de la página web, las imágenes y el contenido se implementan por separado. La generación del código HTML final que será enviado de vuelta al usuario y que el navegador web interpretará para mostrarle la pantalla pertinente, se realiza en los siguientes pasos:

1. Cargar el código HTML contenido en el archivo *template.xhtml*.

2. Incluirle el archivo de estilos que se especifica en el.

3. Completar el contenido de cada zona de la pantalla con el código que aparece en el archivo correspondiente a la sección solicita.

4. Una vez ensambladas las piezas que componen el código final, se envía el resultado al usuario.

D.2. El módulo Java de gestión

El módulo Java de gestión está formado por dos paquetes que trabajan como un sistema por niveles que la información va atravesando en su recorrido de ida y vuelta entre el ordenador del usuario y el motor de recomendaciones.

El paquete *beans* constituye el nivel superior. Contiene dos clases, *BlogVista* y *SvmVista*. Cada una de ellas incluye objetos donde JSF almacena los datos procedentes del usuario y los devueltos por el motor de recomendaciones. Este paquete comprueba la existencia de los datos que recibe y los pasa al nivel inferior.

El nivel inferior lo implementa el paquete *servicio*, que es el encargado de interactuar con el motor de recomendaciones. Incluye las clases *BlogServicio* y *SvmServicio*, que ofrecen métodos que sus homólogas del nivel superior pueden invocar. Estas clases se encargan de enviar la petición al motor de recomendaciones y esperar los resultados o una excepción en caso de haber algún error en los parámetros proporcionados.

Si se recibe una excepción, ésta se propaga hasta el nivel superior, donde se construye un mensaje de error que será enviado al usuario para informarle del problema.

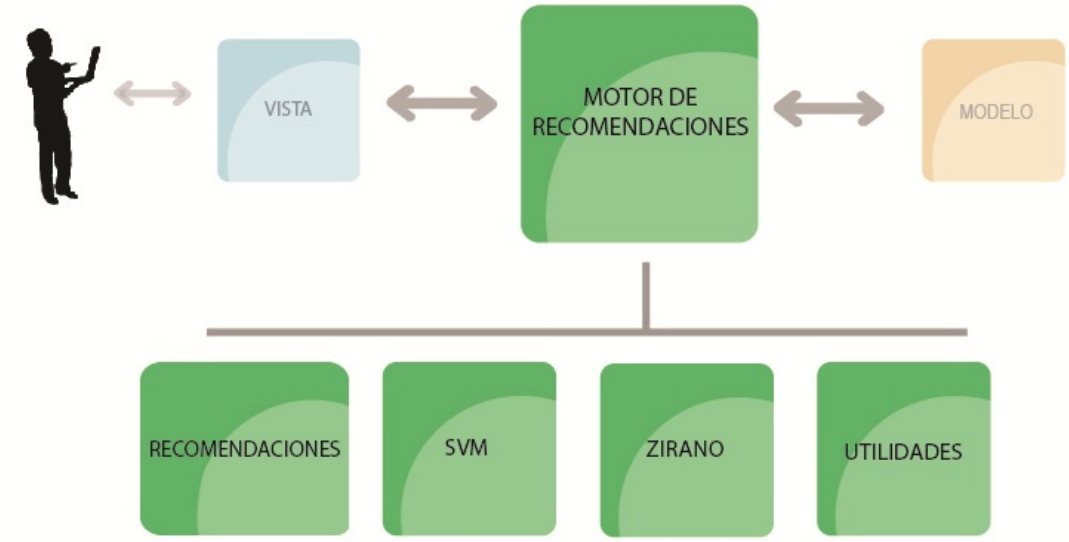
Por último, el módulo Java de gestión incluye un paquete de utilidades para la validación de datos, la interfaz del servidor remoto RMI que contiene los métodos que ofrece el motor de recomendaciones y un paquete con las clases que maneja el sistema, ya comentadas en el apartado B.

Anexo E

El motor de recomendaciones

El motor de recomendaciones es el bloque principal del sistema. Se implementa como un objeto remoto RMI, puesto que con esta herramienta de Java se construyen servidores como objetos cuyos métodos pueden ser invocados por el cliente. Esta idea se ha introducido ya en el apartado 4.2, Elección de las tecnologías.

Figura E.1: *Estructura del motor de recomendaciones.*



Para que un objeto pueda ser utilizado como un objeto remoto, es necesario que cumpla una serie de requisitos, que son los siguientes:

1. Los métodos que ofrezca al cliente tienen que estar definidos en una interfaz, y la clase a la que pertenece el objeto debe implementarlos todos.

2. Esta interfaz debe heredar de la clase Remote, contenida en el paquete RMI.

3. El cliente debe disponer de una copia de la interfaz, para conocer los métodos que puede invocar. No necesita su implementación, ya que es información privada del servidor.

4. Para que el objeto sea accesible por otras máquinas hay que darlo de alta en el registro RMI.

Este bloque contiene además módulos auxiliares, que se ilustran en la figura E.1.

En los siguientes apartados se profundiza en cada uno de estos módulos, los paquetes Java que los implementan, las clases que contienen y los métodos más relevantes de cada una de ellas.

E.1. El paquete recomendaciones

Este apartado contiene la interfaz remota, la clase *Recomendador*, la excepción propia *BCatalogException* y la clase *Actualizador*, encargada de la revisión de los *blogs* en busca de nuevos contenidos.

E.1.1. La interfaz remota

La interfaz remota contiene solamente los prototipos de los métodos que el recomendador debe implementar. El listado completo de estos métodos puede verse en el bloque de código E.1. Puesto que el código fuente es de fácil comprensión y está suficientemente comentado, no son necesarias explicaciones adicionales.

Código E.1: *CatalogInterface.*

```
1  public interface CatalogInterface extends java.rmi.Remote {
2
3      // Gestion de blogs
4      void nuevoBlog(String url) throws Exception;
5      public ArrayList<Blog> listadoBlogs() throws Exception;
6      public void borrarBlog(int id) throws Exception;
7
8      // Gestion de clases de entrenamiento
9      void nuevaClase(String urul) throws Exception;
10     public ArrayList<Clase> listadoClases() throws Exception;
11     public void borrarClase(int id) throws Exception;
12
13     // Gestion de textos de ejemplo (datos de entrenamiento)
14     public void nuevoEntrenamiento(String url, int clase) throws
           Exception;
```

```
15      public ArrayList<Entrenamiento> listadoEntrenamientos()
            throws Exception;
16      public void borrarEntrenamiento (int postId, int claseId)
            throws Exception;
17
18      // Operaciones sobre la SVM
19      public void entrenar() throws Exception;
20      public void predecir(String url) throws Exception;
21
22      // Gestion de recomendaciones
23      public void aceptarRecomendacion(int postId) throws Exception
            ;
24      public void rechazarRecomendacion(int postId) throws
            Exception;
25      public ArrayList<Recomendacion>listadoRecomendaciones()
            throws Exception;
26  }
```

E.1.2. El recomendador

La clase Recomendador es el punto de entrada al programa y por tanto incluye su propio método *main*. Cuando el programa inicia su ejecución, crea el objeto del servidor proporcionándole los parámetros recibidos por línea de comandos. Dicho servidor queda bloqueado a la espera de nuevas conexiones entrantes. El bloque de código E.2 muestra un resumen del método *main*.

Código E.2: *Método main del recomendador.*

```
1   public static void main(String[] args) {
2
3     // Crea el servidor remoto
4     try {
5       new Catalog(args);
6
7     } catch (Exception re) {
8       System.out.println(re);
9       System.exit(1);
10    }
11
12    // Pone el servidor a la espera
13    Object sync = new Object();
14    synchronized(sync) {
```

```
15      try { sync.wait(); } catch(Exception ie) {}
16    }
17  }
```

El constructor recibe los parámetros y los procesa pasándolos al método *inicializar*. También da de alta el servidor en el registro de RMI para que sea accesible desde otras aplicaciones, locales o remotas, crea los objetos DAO para la interacción con la base de datos y pone en marcha el proceso actualizador para que compruebe las actualizaciones en los *blogs* de interés. El código E.3 es un resumen del código del constructor de la clase.

Código E.3: *Constructor del recomendador.*

```
1  public Catalog(String[] args) throws RemoteException,
       MalformedURLException {
2    super();
3
4    // Captura de parametros
5    inicializar(args);
6
7    // Alta del servidor en el registro de RMI
8    try {
9      java.rmi.registry.LocateRegistry.createRegistry(1099);
10     Naming.rebind("rmi://localhost/Catalog", this);
11   } catch (Exception e) {
12     System.exit(2);
13   }
14
15   // Creacion de objetos DAO
16   try {
17     blogDao = new BlogDao(servidorBD, BD, usuarioBD, passBD,
         puertoBD);
18     ...
19   } catch (CatalogException e) {
20     e.printStackTrace();
21     System.exit(2);
22 }
23
24   // Arranque del actualizador
25   actualizador = new Actualizador(blogDao, infoDao,
26     svm, recomendacionDao, conceptosDao, debug);
```

```
27      actualizador.iniciar();
28  }
```

Para el procesamiento de los parámetros se utiliza la utilidad JArgs [41],que ofrece métodos para su fácil manipulación. A continuación se detallan los pasos necesarios en este proceso:

1. Se crea un objeto de la clase CmdLineParser, la utilidad que se va a utilizar.
2. Se especifican las opciones disponibles desde la línea de comandos y su tipo. Por ejemplo, se puede especificar que se espera un parámetro entero, opcional, y con valor 0 en caso de no ser proporcionado.
3. Se ejecuta el método de análisis.
4. Se recogen los parámetros, almacenándolos en las variables que interesen.

El código E.4 muestra un ejemplo cómo capturar un parámetro de tipo booleano y almacenarlo en una variable llamada *debug*. En caso de no estar presente, el valor por defecto sera *false*.

Código E.4: *Captura de parámetros por línea de comandos.*

```
1  public void inicializar(String[] args) {
2
3    // Crea el analizador de parametros
4    CmdLineParser parser = new CmdLineParser();
5
6    // Define las opciones disponibles, su tipo y su
7    // representacion en la linea de comandos
8    CmdLineParser.Option debug = parser.addBooleanOption('v', "
        verbose");
9    ...
10
11   // Analiza los parametros
12   try {
13   parser.parse(args);
14   }
15   catch ( CmdLineParser.OptionException e ) {
16   printUsage();
17   System.exit(2);
18   }
```

```
19
20    // Comprueba los parametros e inicializa variables del programa
21    this.debug =  (Boolean)parser.getOptionValue(debug, false);
22  }
```

Un ejemplo de cómo funciona la utilidad JArgs puede verse en el código E.5. Los dos primeros ejemplos provocarían que la variable *debug* del programa se inicializara con valor *true* mientras el tercero provocaría un valor *false*.

Código E.5: *Ejemplos de captura de parámetros.*

```
1   $ bcatalog -v
2   $ bcatalog --verbose
3   $ bcatalog
```

La clase Recomendador incluye además la implementación de los métodos definidos en la interfaz remota. En cada uno de ellos se realizan comprobaciones de control, tales como la validez de los parámetros, la existencia en la base de datos de la información a insertar o borrar y otras que sean necesarias. En caso de errores en los parámetros, se lanza una excepción BCatalogException, que se explica a continuación. Una vez validados los datos de entrada, cada método realiza las operaciones correspondientes y devuelve los resultados.

E.1.3. La excepción BCatalogException

La excepción BCatalogException es una excepción propia que maneja la aplicación. Contiene una lista de cadenas donde se almacenan los errores que la han provocado, un método para añadir uno nuevo a la lista y uno para obtenerlos. El código E.6 incluye la implementación de la excepción.

Código E.6: *CatalogException.*

```
1  public class CatalogException extends Exception {
2
3    private ArrayList<String>errores;
4
5    private static final long serialVersionUID =
        8151911863184279687L;
6
7    public CatalogException() {
```

```
8       super();
9       errores = new ArrayList<String>();
10    }
11
12    public void addError(String error) {
13      errores.add(error);
14    }
15
16    public ArrayList<String> getErrores() {
17      return errores;
18    }
19  }
```

E.1.4. El actualizador

La clase Actualizador implementa el componente que comprueba periódicamente la existencia de nuevos textos en los *blogs* de interés del usuario. Contiene un bucle que ejecuta una tarea y espera una hora antes de volver a ejecutarla. La tarea incluye las siguientes operaciones:

1. Recoger la fecha de la última comprobación, almacenada en la base de datos.
2. Recoger de la base de datos la lista de *blogs* a los que el usuario está subscrito.
3. Para cada *blog*, obtener un listado de textos publicados con fecha posterior a la de la última comprobación.
4. Predecir la clase de cada uno de los textos obtenidos e insertar en la base de datos, si procede, la nueva recomendación pertinente.

E.2. El paquete svm

Este paquete contiene la implementación de la Máquina de Vectores de Soporte de Weka adaptada para su utilización en este proyecto.

Antes de detallar las modificaciones que se han realizado es necesario introducir el funcionamiento del paquete original de Weka. El apartado E.2.1 muestra los segmentos más significativos del código a modificar.

En el apartado E.2.6 se explican las modificaciones llevadas a cabo para convertir la svm de inicial en la herramienta final que este proyecto requiere.

E.2.1. El paquete original de Weka

La máquina SVM implementada por Weka se encuentra en el paquete *weka.classifiers.functions*, y recibe el nombre de clase *SMO*. Sus componentes más importantes para comprender los cambios introducidos son los siguientes:

1. Dataset: Conjunto de datos de entrenamiento.

2. Matriz de clasificadores SVM binarios: Matriz de tamaño nxn de objetos de la clase *BinarySMO*, donde n corresponde al número de clases de entrenamiento.

3. Subclase *BinarySMO*: Clasificador SVM binario cuyo funcionamiento y misión se explica en el apartado E.2.3.

4. Métodos *BuildClassifier* y *DistributionForInstance*: Métodos de entrenamiento y predicción.

Dataset es el conjunto de datos de ejemplo de los que el clasificador dispone en el momento de realizar las tareas de entrenamiento y predicción. No es necesario un apartado para su explicación en profundidad. La utilidad del resto de componentes se detalla en los apartados siguientes.

E.2.2. Matriz de clasificadores binarios

La clase *SMO* trabaja internamente con clasificadores SVM binarios. Estos clasificadores, cuyo funcionamiento se detalla en el apartado E.2.3, son entrenados para distinguir objetos de dos clases de entrenamiento.

La matriz de clasificadores binarios se emplea pues para diferenciar, dos a dos, todas las clases a las que los datos de ejemplo pertenecen.

E.2.3. La clase BinarySMO

La clase *BinarySMO* implementa los clasificadores SVM binarios. El detalle del algoritmo de estas máquinas excede los objetivos de esta memoria, por lo que únicamente se introducen los métodos de aprendizaje y predicción.

El método *BuildClassifier* es es el método que entrena el clasificador *BinarySMO*. Recibe los datos de las dos clases que se encarga de reconocer. Realiza ajustes iniciales, la obtención de los vectores de soporte y otra serie de ajustes finales.

El método *SVMOutput* es el método de predicción. Dada una nueva instancia, realiza llamadas al Kernel SSK, que Weka implementa en la clase *StringKernel*. Realiza comparaciones entre el nuevo dato y cada uno de los vectores de soporte, y utiliza los resultados obtenidos para generar el resultado devuelto.

E.2.4. El método *BuildClassifier*

El método *BuildClassifier* de la clase *SMO* recibe los datos de ejemplo que el clasificador utiliza. Tras una serie de comprobaciones iniciales crea los distintos objetos *BinarySMO*, proporciona a cada uno los datos de las dos clases con las que trabaja y los entrena. El bloque de código E.7 ilustra este proceso de entrenamiento.

Código E.7: *Método BuildClassifier.*

```
1  public void buildClassifier(Instances insts) throws Exception {
2
3    // Comprobaciones iniciales
4    ...
5
6    m_classIndex = insts.classIndex();
7    m_classAttribute = insts.classAttribute();
8
9    // Genera los subconjuntos que representan
10   // a cada clase
11   Instances[] subsets = new Instances[insts.numClasses()];
12   for (int i = 0; i < insts.numClasses(); i++) {
13     subsets[i] = new Instances(insts, insts.numInstances());
14   }
15   ...
16
17   // Crea la matriz de BinarySMO
18   m_classifiers =
19     new BinarySMO[insts.numClasses()][insts.numClasses()];
20
21   // Asigna a cada BinarySMO su conjunto de datos
22   for (int i = 0; i < insts.numClasses(); i++) {
23     for (int j = i + 1; j < insts.numClasses(); j++) {
24       m_classifiers[i][j] = new BinarySMO();
25       Instances data = new Instances(insts, insts.numInstances())
             ;
26       for (int k = 0; k < subsets[i].numInstances(); k++) {
27         data.add(subsets[i].instance(k));
28       }
```

```
29        ...
30
31        // Entrena cada BinarySMO
32        m_classifiers[i][j].buildClassifier(data, i, j,
33        m_fitLogisticModels,
34        m_numFolds, m_randomSeed);
35      }
36    }
37  }
```

E.2.5. El método *DistributionForInstance*

El método *DistributionForInstance* es el encargado de predecir la clase a la que pertenece un nuevo dato desconocido. Para ello invoca el método *SVMOutput* de cada clasificador *BinarySMO*.

Cada objeto *BinarySMO* se encarga de predecir a cuál de las clases que compara es más probable que pertenezca el nuevo dato. Por tanto, cada invocación al método *SVMOutput* sirve para asignar votos a cada una de las clases.

El resultado que se devuelve es el porcentaje de votos que cada una de las clases existentes ha obtenido en la predicción. Este proceso de asignación de votos puede verse en el bloque de código E.8.

Código E.8: *Método DistributionForInstance.*

```
1  public double[] distributionForInstance(Instance inst) throws
       Exception {
2
3    // Comprobaciones iniciales
4    ...
5
6    // Crea un vector donde almacenar los votos para cada clase
7    double[] result = new double[inst.numClasses()];
8
9    // Para cada BinarySMO, ejecuta el metodo SVMOutput
10   for (int i = 0; i < inst.numClasses(); i++) {
11     for (int j = i + 1; j < inst.numClasses(); j++) {
12       if ((m_classifiers[i][j].m_alpha != null) ||
13       (m_classifiers[i][j].m_sparseWeights != null)) {
14         double output = m_classifiers[i][j].SVMOutput(-1, inst);
```

```
15
16          // En funcion del resultado, asigna el voto
17          // a una u  otra clase.
18          if (output > 0) {
19            result[j] += 1;
20          } else {
21            result[i] += 1;
22          }
23        }
24      }
25    }
26
27    // Normaliza el resultado para que
28    // el resultado quede expresado en %
29    Utils.normalize(result);
30
31    return result;
32  }
```

E.2.6. Modificaciones realizadas al algoritmo original

En este apartado se describen las modificaciones que se requieren para adaptar el código implementado por Weka al problema tratado en este proyecto.

La implementación original presenta un inconveniente importante. Dado un conjunto de clases de entrenamiento, el método de predicción devuelve un conjunto con las probabilidades de que el nuevo texto pertenezca a cada una de las clases, pero presupone que pertenece a alguna de ellas.

Según este modelo, BCatalog no puede determinar que un nuevo texto **no** pertenece a ninguna de las clases y que por tanto no es del interés del usuario. A continuación se detallan cada una de las decisiones tomadas para la modificación del código inicial:

Primero, los objetos *BinarySMO* ya no comparan dos clases de entremiento. En su lugar, cada uno de ellos se especializa en reconocer los objetos de una clase. Por este motivo la matriz de *nxn* clasificadores binarios se convierte en un vector de n de estos objetos.

En cuanto al proceso de entrenamiento, los *BinarySMO* reconocen dos clases ficticias. Una clase es la llamada “SI”, que contiene los elementos del conjunto de datos inicial que pertenecen a la clase de la que se encargan. La otra clase es “NO”, que contiene

todos los demás. Esto quiere decir que todos los clasificadores binarios manejarán todos los datos de entrenamiento, pero cada uno de ellos los tratará de manera diferente.

El bloque de código E.9 ilustra esta nueva forma de entrenamiento.

Código E.9: *Método BuildClassifier adaptado.*

```
public void buildClassifier(Instances insts) throws Exception {

  // Comprobaciones iniciales
  ...

  m_classIndex = insts.classIndex();
  m_classAttribute = insts.classAttribute();

  // Extrae los textos
  ArrayList<String> textos = new ArrayList<String>();

  for (Instance inst : insts) {
    String nueva = inst.stringValue(0);
    textos.add(nueva);
  }

  ArrayList<Attribute> atributos = crearAttributes(textos);

  // Crea los datasets para cada clasiffier
  Instances[] subsets = new Instances[insts.numClasses()];

  for (int i=0; i<subsets.length; i++) {
    ArrayList<String> clases = new ArrayList<String>();
    // Recorremos el conjunto de datos inicial, utilizando los
    // valores de las clases SI y NO como corresponda
    for (Instance inst : insts) {
      if (inst.value(1)==i) clases.add("SI");
      else clases.add("NO");
    }
    subsets[i] = crearDataSet(atributos, textos, clases);
  }

  // Crea los clasificadores binarios
  m_classifiers = new MiBinarySMO[insts.numClasses()];

  for (int i = 0; i < subsets.length; i++) {
```

```
37      m_classifiers[i] = new MiBinarySMO();
38    }
39
40    // Entrena los clasificadores binarios
41    for (int i=0; i<subsets.length; i++) {
42      m_classifiers[i].buildClassifier(subsets[i], 0, 1,
43        m_fitLogisticModels,
44        m_numFolds, m_randomSeed);
45    }
46  }
```

Por su parte, el proceso de predicción también es ligeramente distinto al original. Cada clasificador binario determina si el nuevo objeto pertenece a su clase ficticia "SI". El nuevo resultado contendrá tantas componentes con valor 1.0 como categorías con las que el nuevo texto esté relacionadas. Si el texto no está relacionado con ninguna de las clases de entrenamiento, el vector contendrá n valores 0.0.

El bloque de código E.10 ilustra el nuevo método de predicción:

Código E.10: *Método DistributionForInstance adaptado.*

```
1   public double[] distributionForInstance(Instance inst) throws
        Exception {
2
3     // Comprobaciones iniciales
4     ...
5
6     double[] result = new double[inst.numClasses()];
7
8     for (int i = 0; i < inst.numClasses(); i++) {
9       if ((m_classifiers[i].m_alpha != null) ||
10      (m_classifiers[i].m_sparseWeights != null)) {
11        double output = m_classifiers[i].SVMOutput(-1, inst);
12
13        // Si el texto esta relacionado, se incrementa la
              componente
14        // correspondiente en el vector de resultados.
15        if (output <= 0) {
16          result[i] = 1;
17        } else {
18          result[i] = 0
```

```
19          }
20        }
21      }
22
23      return result;
24  }
```

E.3. Clase Zirano

La clase Zirano ofrece un único método que, dada una palabra, obtiene una lista de palabras relacionadas que extrae de la página web de Zirano. Cuando un usuario quiere obtener esta lista de ideas a través de la página web, tiene que realizar una serie de pasos para localizar las palabras que le interesan. Los pasos son los siguientes:

1. Introduce la palabra de su elección.
2. La web le ofrece un conjunto de ideas con las que se puede relacionar su palabra.
3. El usuario navega entre esas opciones, obteniendo para cada una un conjunto de palabras relacionadas.

El método *obtenerIdea* simula esta navegación, de forma que la web de Zirano devuelva los resultados que devolvería a un usuario que recorriera todas las ideas relacionadas con su palabra. Para ello, realiza las siguientes operaciones:

1. Descarga la página inicial de Zirano, y almacena la *cookie* que ésta le envia. Dicha *cookie* identifica al usuario cuya navegación se está simulando, y por tanto será enviada en cada transacción a modo de identificador.
2. Se envía a la web la palabra de la que se quiere obtener el campo conceptual.
3. La web responde con un listado de ideas a las que puede estar relacionadas. Se analiza y procesa el código HTML de la respuesta para obtener un listado de enlaces a los que enviar las siguientes peticiones.
4. Cada uno de los enlaces obtenidos constituye una petición a la web de un listado de palabras relacionadas con una idea en concreto.
5. Ante una de estas peticiones, la web responde con un listado de palabras que el método va almacenando. El código HTML de estas nuevas respuestas se filtra para quedarse únicamente con el listado de palabras.

6. Una vez completadas las peticiones para todas las ideas, o alcanzado un número máximo de palabras almacenadas, la navegación termina y el método devuelve los resultados.

E.4. El paquete útil

Por último se introducen algunas utilidades creadas para facilitar la implementación del resto de módulos.

E.4.1. Interfaz para el manejo de blogs

En este apartado se explica cómo el sistema obtiene los textos de Internet. Para poder implementarse esta funcionalidad se precisa de alguna herramienta que permita conocer qué textos forman la página y cuando fueron publicados.

Una herramienta útil que existe para este cometido son los archivos XLM llamados Sitemap.xml [42], que incluyen la información, como su dirección URL, su título, su frecuencia de actualización o su fecha de publicación. Un ejemplo del contenido de dichos archivos es el que aparece en el código E.11.

Código E.11: *Ejemplo de sitemap.xml.*

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset
    xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <url>
    <loc>http://www.example.com/</loc>
    <lastmod>2005-01-01</lastmod>
    <changefreq>monthly</changefreq>
    <priority>0.8</priority>
  </url>
</urlset>
```

La existencia de este archivo no es obligatoria aunque sí muy recomendable, pues facilita la accesibilidad de la página web y mejora el posicionamiento en los buscadores. A pesar de la gran variedad de sitios web que el usuario puede estar interesado en leer, este proyecto se centra en los desarrollados utilizando el CMS Wordpress, ya que todos ellos contienen un Sitemap y se puede localizar con facilidad aunque pueda estar guardado con otro nombre.

Por esta razón se ha creado una interfaz Java que permite la obtención del Sitemap y su manipulación, y una implementación concreta para las páginas desarrolladas con Wordpress. Los métodos ofrecidos por esta interfaz aparecen en el código E.12.

Código E.12: *ManejoBlogInterface.*

```
1  public interface ManejoBlogInterface {
2
3    // Devuelve true si el blog tiene un Sitemap valido
4    public boolean isValido();
5
6    // Obtiene una lista de Posts publicados despues de la fecha "
       limite"
7    public ArrayList<Post> obtenerPosts(Date limite);
8
9    // Descarga de nuevo el archivo sitemap.xml
10   public void actualizarSiteMap();
11 }
```

E.4.2. Manejo de código HTML

Para el manejo de código HTML se ha creado una clase que ofrece métodos que otras clases pueden necesitar, y que se comentan a continuación:

La clase Zirano, que se detalla en el apartado E.3 necesita simular la navegación efectuada por un usuario que desea obtener un campo conceptual. En el proceso, un navegador web intercambiaría una *cookie* con el servidor de Zirano. Este intercambio se implementa con el método *obtenerCookie*.

La descarga del código HTML de una página web se realiza mediante el método *obtenerHTML* que recibe como parámetros, además de su dirección URL, un número máximo de intentos de descarga tras los cuales se aborta la operación y se devuelve un error.

El método *obtenerPagina* devuelve un objeto de tipo *Post*, con el título, la dirección URL y el texto filtrado de la página descargada.

El método *filtrarHTML* se utiliza para eliminar las etiquetas HTML y líneas en blanco de la cadena que recibe como parámetro. El resultado es una cadena de texto que contiene el texto plano de la página, y que es utilizada por la máquina de aprendizaje.

De ahí la importancia de la eficacia de este método, pues es necesario que el texto a analizar contenga la mínima cantidad de información no relacionada con él.

El bloque de código E.13 contiene los prototipos de cada uno de los métodos:

Código E.13: *Clase HTML.*

```
1  public class HTML {
2
3    public static Post obtenerPagina(String url) throws
        CatalogException { ... }
4
5    protected static String obtenerHTML(
6        String direccionUrl, String cookie, int intentos) { ... }
7
8    protected static String obtenerCookie (String direccionUrl, int
        intentos) { ... }
9
10   public static String filtrarHTML(String html) { ... }
11 }
```

E.4.3. Utilidades de conversión

La clase *Conversiones* incluye los métodos para convertir datos de distintos tipos que han sido necesarias en algunos módulos del sistema. El código E.14 muestra los prototipos de dichos métodos.

Código E.14: *Clase Conversiones.*

```
1  public class Conversiones {
2
3    // Devuelve el double representado por la cadena s
4    public static double atof(String s) { ... }
5
6    // Devuelve el entero representado por la cadena s
7    public static int atoi(String s) { ... }
8
9    // Devuelve la representacion en forma de cadena de la fecha d
10   public static String datetostring (Date d) { ... }
11   // Devuelve la fecha representada por la cadena s
12
```

```
13    public static Date stringtodate (String s) { ... }
14  }
```

E.4.4. La clase es_Stemmer

La clase Stemmer implementa el lematizador que calcula la raíz de una palabra. Se basa en el algoritmo de Porter[32], y es utilizada por el sistema cuando se calculan los campos conceptuales.

Es importante mencionar que la clase se descargó de un foro de internet en el que alguien preguntaba donde podía conseguir una implementación de un lematizador en Java. En al código fuente no aparece la página web del autor, así que no puedo incluir su referencia. Según se explicaba en aquel foro, se trata de una traducción de la implementación escrita en PHP[43] que puede descargarse de SourceForge [31].

Anexo F

El sistema de gestión de la persistencia

En este apartado se explica en detalle la estructura del sistema de gestión de la persistencia, que incluye una base de datos donde se almacena la información necesaria para el funcionamiento del sistema y un conjunto de clases Java que ofrecen una interfaz de comunicación con ella.

El apartado F.1 contiene la explicación sobre la estructura de la base de datos y una serie de consideraciones previas que justifican la existencia de sus tablas. En el apartado F.2 se introducen las clases que manejan la base de datos y se concreta con qué tablas interacciona cada una de ellas.

F.1. La base de datos

La base de datos se aloja en un servidor MySQL, y contiene 9 tablas donde se almacenan los *blogs*, los textos manejados por el sistema, la información referente a palabras y campos conceptuales, las clases de entrenamiento, los textos de ejemplo, las recomendaciones y otra información necesaria para el funcionamiento del sistema.

Su estructura completa puede verse en la figura F.1, que muestra todas las tablas existentes, los campos que las forman y cómo se relacionan unas tablas con otras. Para obtener esta estructura final ha sido necesario tener en cuenta algunas consideraciones que se comentan a continuación.

La descarga de un texto desde internet requiere conectarse a un servidor web externo para solicitarle el código HTML de la página donde aparece dicho texto. El código HTML recibido hay que filtrarlo para eliminar información que no sea relevante y

Figura F.1: *Estructura de la base de datos.*

almacenarlo en la base de datos. Todo este proceso lleva un tiempo de ejecución asociado que no puede despreciarse.

El usuario puede añadir textos de ejemplo al sistema y borrarlos si más tarde considera que ya no son relevantes. En un momento dado puede decidir que un texto ya no está relacionado con una de sus categorías, sino con otra diferente. Si no se adopta ninguna medida ese texto se descargará varias veces.

Por otra parte una recomendación se convertirá en caso de ejemplo si el usuario la aprueba. Podrá ocurrir, por tanto, que posteriormente el usuario pueda actuar como se comenta en el párrafo anterior, cambiando la categoría con la que un texto está relacionado.

Con el fin de evitar las repetidas descargas de un mismo texto, se ha decidido que la tabla *Post* sea un almacén al que sólo puedan añadirse textos nuevos y obtener un listado de ellos. Esta decisión implica que las tablas *Recomendación* y *Entrenamiento* contendrán referencias a la tabla *Post*, en lugar de sus datos completos.

En cuanto al almacenamiento de los campos conceptuales, conviene tener en cuenta lo siguiente. En primer lugar varias palabras pueden tener la misma raíz, como por ejemplo ocurre con *mesa* y *mesilla*. Almacenar raíces repetidas supone un gasto innecesario de espacio.

Una palabra puede aparecer en varios campos conceptuales distintos. Este hecho se produce muy frecuentemente debido a la gran cantidad de acepciones que las palabras pueden tener en la lengua española. Cada palabra debería ser almacenada sólo una vez, para evitar información redundante en la base de datos.

Por todo ello se dispone de tres tablas para manipular los campos conceptuales. En la tabla *Palabra* se almacenan las palabras completas que el sistema haya manejado, independientemente de que pertenezcan al nombre de una categoría o a un campo conceptual. La tabla *Raíz* contiene las raíces de palabras obtenidas en el proceso de lematización. Por último, la tabla *Campo Conceptual* relaciona pares de elementos de las dos tablas anteriores.

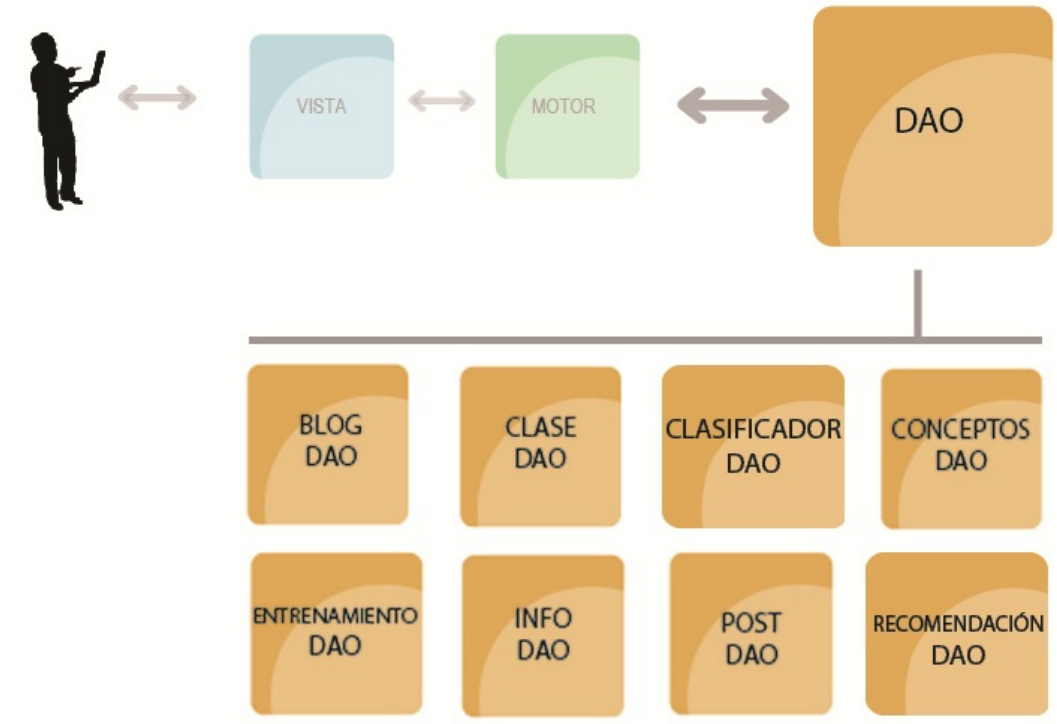
Las tablas *Post*, *Palabra*, *Raíz* y *Campo Conceptual* sólo ofrecen operaciones para añadir nuevos datos y obtener un listado. No ofrecen la posibilidad de eliminar datos, ya que constituyen un almacén que conviene conservar por motivos de rendimiento.

En ampliaciones del sistema que incluyan la posibilidad de que existan diferentes usuarios las ventajas de estas decisiones serán más notables, pues cada uno añadirá información al sistema que estará disponible al resto de los usuarios. Cuantos más usuarios utilicen el sistema, a mayor velocidad aumentará la cantidad de información en los almacenes y menos descargas nuevas habrá que realizar.

F.2. El paquete de acceso a datos

El paquete de acceso a datos, al que en adelante se llamará DAO (del inglés, Data Access Object) contiene diferentes clases que se encargan de la interacción con la base de datos. La figura F.2 ilustra las clases que incluye dicho paquete.

Figura F.2: *Estructura del paquete DAO.*



A continuación se introduce la función las clases del paquete, especificando las tablas a las que accede cada una.

La clase BlogDao manipula la tabla *Post*. Ofrece métodos para añadir, listar, comprobar si existe y borrar *blogs* en la base de datos.

La clase ClaseDao manipula la tabla *Clase*. Gestiona las clases de entrenamiento, que representan las categorías a las que los textos pueden pertenecer.

La ClasificadorDao no manipula ninguna tabla. Ofrece métodos para el almacenamiento en disco de la SVM. Puesto que la información de la máquina de aprendizaje ocupa demasiado espacio para introducirse en una base de datos, se opta por utilizar el sistema de ficheros.

La clase ConceptosDao controla el acceso a las tablas *Palabra*, *Raíz* y *Campo Conceptual*, permitiendo añadir palabras, almacenar sus raíces y campos conceptuales. No ofrece métodos de listado ya que los datos que maneja son de uso interno del sistema. Tampoco ofrece operaciones de borrado, con el fin de almacenar a modo de cache todos los datos que se inserten.

La clase EntrenamientoDao manipula la tabla *Entrenamiento*. Gestiona los datos de entrenamiento del sistema, ofreciendo operaciones de inserción, listado, comprobación de existencia y eliminación.

La clase InfoDao manipula la tabla *Info*, que contiene información necesaria para el funcionamiento del sistema. En este prototipo dicha información consiste únicamente en la fecha y hora a la que se realiza la última comprobación de los blogs de interés en busca de nuevos textos que recomendar al usuario.

La clase PostDao interactúa con la tabla *Post*. Ofrece operaciones de inserción de textos en la base de datos, así como de listado. Por motivos comentados al principio de este subapartado no se ofrecen operaciones de eliminación.

La clase RecomendaciónDao manipula la tabla *Recomendación*. Gestiona las recomendaciones que serán ofrecidas al usuario, y ofrece métodos de inserción, listado, borrado y comprobación de existencia.

Anexo G

La paralelización de tareas en detalle

Este apartado describe las modificaciones realizadas para incorporar trabajo en paralelo en algunas tareas internas del sistema. En este prototipo las mejoras introducidas están relacionadas con la máquina SVM.

Un estudio del funcionamiento interno de la implementación de Weka, tras su adaptación para su utilización en este proyecto, ha permitido encontrar tres alternativas en las que puede incluirse la ejecución de tareas en paralelo, que se ilustran en la figura G.1.

La primera de ellas se encuentra en el Kernel SSK, implementado en la clase *StringKernel* que se encuentra en el paquete *weka.classifiers.functions.supportVector*. Las posibilidades de paralelización en esta clase se detallan en el apartado G.1.

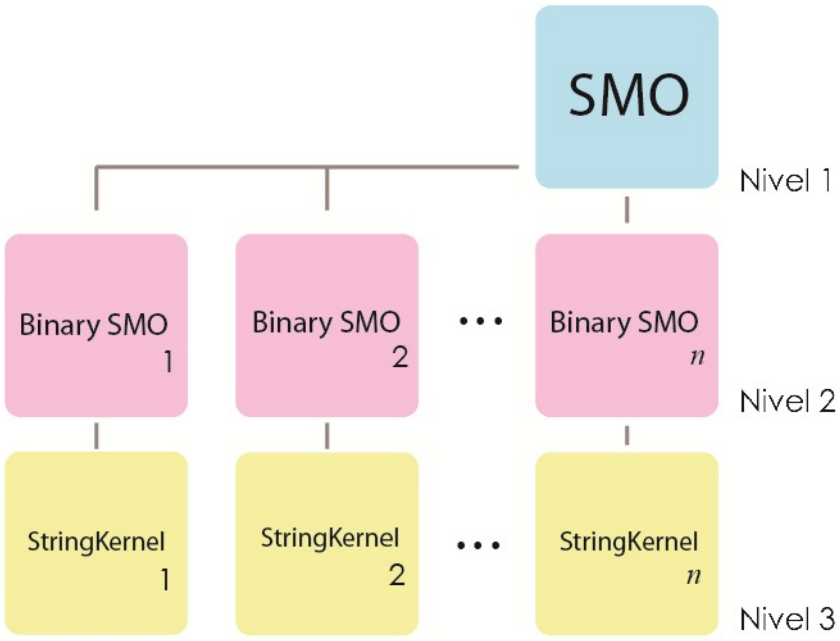
La segunda alternativa consiste en la modificación de los métodos de entrenamiento y predicción de los clasificadores binarios (clase *BinarySMO*), y se comenta en el apartado G.2.

Por último, es posible que todos los clasificadores binarios realicen en paralelo las tareas de entrenamiento y predicción, como se explica en el apartado G.3.

G.1. Paralelismo en el Kernel SSK

El método de comparación de dos cadenas ofrecido por la clase StringKernel, que implementa el Kernel SSK, realiza tres cálculos consecutivos. Uno de ellos compara las

Figura G.1: *Niveles de paralelización de la máquina SVM.*



dos cadenas, y los otros dos se utilizan para normalizar el resultado, como ilustra la ecuación C.2 en el anexo C, *La Máquina de Vectores de Soporte*.

El bloque de código G.1 contiene el código Java que compara las dos cadenas y normaliza el resultado obtenido para que esté comprendido entre 0 y 1.

Código G.1: *Código paralelizable en la clase StringKernel*

```
1  public double normalizedKernel(char[] s, char[] t){
2
3    // Compara cada cadena consigo misma
4    double k1 = unnormalizedKernel(s, s);
5    double k2 = unnormalizedKernel(t, t);
6
7    // Calcula el factor de normalizacion
8    double normTerm = Math.sqrt( k1*k2 );
9
10   // Compara una cadena con otra y devuelve
11   // el resultado normalizado
12   return unnormalizedKernel(s, t) / normTerm;
13 }
```

Estudiando el método *unnormalizedKernel* en profundidad puede comprobarse que en ningún momento modifica las cadenas que recibe como parámetros. Por ello, y dado que las tres invocaciones a este método utilizan datos que no dependen unos de otros, pueden ejecutarse en paralelo. El bloque de código G.2 muestra una ligera modificación del código anterior, a la que ya se le puede introducir el trabajo en paralelo:

Código G.2: *Código paralelizable en la clase StringKernel*

```
1  public double normalizedKernel(char[] s, char[] t){
2
3    // Compara cada cadena consigo misma
4    double k1 = unnormalizedKernel(s, s);
5    double k2 = unnormalizedKernel(t, t);
6
7    // Compara una cadena con otra
8    double k3 = unnormalizedKernel(s,t);
9
10   // LAS TRES LLAMADAS ANTERIORES
11   // PUEDEN LANZARSE EN PARALELO.
12   // CUANDO TODAS TERMINEN PUEDE
13   // EJECUTARSE EL CODIGO A CONTINUACION
14
15   // Calcula el factor de normalizacion
16   double normTerm = Math.sqrt( k1*k2 );
17
18   // Devuelve el resultado de la comparacion
19   // normalizado.
20   return k3 / normTerm;
21 }
```

Tras esta modificación menor en el código original pueden crearse tres objetos de la clase Thread de Java, asignarles a cada uno una de las llamadas al método *unnormalizedKernel* y esperar a que todos terminen para generar el resultado a devolver.

G.2. Paralelismo en la clase BinarySMO

En este apartado se introducen las modificaciones que se pueden realizar en el código de la clase *BinarySMO* para implementar la paralelización de sus tareas de entrenamiento y predicción.

Es la parte más complicada de este proceso, pues se trata de trabajos iterativos y no siempre puede asegurarse de que las diferentes iteraciones sean independientes entre ellas, de forma que el orden de ejecución no altere los resultados.

En el apartado G.2.1 se profundiza en el estudio del código del método *buildClassifier* realizado. Por otra parte, el estudio del método *SVMOutput* se comenta en el apartado G.2.2.

G.2.1. El método buildClassifier

En el método *buildClassifier* se realiza una búsqueda de los vectores de soporte que el clasificador utilizará en la fase de predicción. Esta búsqueda se lleva a cabo de forma incremental, utilizando en cada iteración los vectores encontrados hasta ese momento y utilizándolos para encontrar otros nuevos.

El bloque de código G.3 muestra el bucle principal del método. Como puede verse, no hay un número fijo de iteraciones, sino que en función de los resultados se van asignando valores a las variables *numChanged* y *examineAll*. Por ese motivo las iteraciones no pueden ejecutarse en paralelo.

Código G.3: *Resumen del método BuildClassifier*

```
1  public synchronized void buildClassifier(Instances insts...)
       throws Exception {
2
3    // Ajustes iniciales
4    ...
5
6    // Bucle de busqueda
7    while ((numChanged > 0) || examineAll) {
8      numChanged = 0;
9
10     if (examineAll) {
11       for (int i = 0; i < m_alpha.length; i++) {
12         if (examineExample(i)) {
13           numChanged++;
14         }
15       }
16     } else {
17       ...
18     }
19   }
```

```
20
21    // Ajustes finales
22    ...
23  }
```

Dentro de cada iteración, el método *examineExample* se ejecuta un número fijo de veces, pues el valor de *m_alpha.length* no se modifica ni en ese bucle ni dentro del método. Por tanto las iteraciones de este bucle interno podrían ejecutarse en paralelo, si no fuera porque dentro del método se utilizan variables cuyos valores se van modificando a lo largo de las iteraciones del bucle principal. Por ese motivo se descarta esta segunda opción de paralelización.

El método *examineExample* contiene ajustes iniciales y una llamada al método *takeStep*. El bloque de código G.4 muestra las acciones más relevantes ejecutadas por cada uno de ellos.

Código G.4: *Resumen de los métodos examineExample y TakeStep*

```
1  protected boolean examineExample(int i2) throws Exception {
2
3    // Comprobaciones iniciales
4    ...
5
6    return takeStep(i1, i2, F2);
7  }
8
9
10 protected boolean takeStep(int i1, int i2, double F2) throws
       Exception {
11
12   // Comprobaciones iniciales
13   ...
14
15   // Llamadas al Kernel que son
16   // paralelizables
17   k11 = m_kernel.eval(i1, i1, m_data.instance(i1));
18   k12 = m_kernel.eval(i1, i2, m_data.instance(i1));
19   k22 = m_kernel.eval(i2, i2, m_data.instance(i2));
20
21   // Mas ajustes
22   ...
```

```
23
24    // Llamadas al metodo de prediccion
25    // paralelizables
26    f1 = SVMOutput(i1, m_data.instance(i1));
27    f2 = SVMOutput(i2, m_data.instance(i2));
28
29    // Ajuste de vectores de soporte
30    if (a1 > 0) {
31      m_supportVectors.insert(i1);
32    }
33    ...
34
35    // Otros ajustes
36    for (int j = m_I0.getNext(-1); j != -1; j = m_I0.getNext(j)) {
37      if ((j != i1) && (j != i2)) {
38        m_errors[j] +=
39          y1 * (a1 - alph1) * m_kernel.eval(i1, j, m_data.instance(
                i1)) +
40          y2 * (a2 - alph2) * m_kernel.eval(i2, j, m_data.instance(
                i2));
41      }
42    }
43
44    return true;
45  }
```

Las invocaciones al Kernel y al método *SVMOutput* pueden ser ejecutadas en paralelo puesto que sólo reciben parámetros de entrada y no hay dependencias de escritura. En cambio, los ajustes iniciales y la actualización de los vectores de soporte dependen entre iteraciones.

Del párrafo anterior se puede deducir que no pueden realizarse invocaciones paralelas al método *examineExample*, pero sí es posible ejecutar al mismo tiempo algunas de sus operaciones internas.

G.2.2. El método SVMOutput

El método *SVMOutput* es el método que ejecuta la tarea de predicción en un objeto *BinarySMO*. En su interior se realizan una serie de comprobaciones iniciales y, tras ellas, se invoca al Kernel SSK para que compare el nuevo dato con cada uno de los vectores de soporte conocidos. El resultado de cada comparación se utiliza para ir acumulando

el resultado devuelto. El resumen de este método puede verse en el bloque de código G.5

Código G.5: *Resumen del método SVMOutput*

```
public double SVMOutput(int index, Instance inst) throws
    Exception {

  // Comprobaciones iniciales
  ...

  double result = 0;

  // Compara con cada vector de soporte
  // y acumula el resultado

  for (int i = m_supportVectors.getNext(-1); i != -1;
          i = m_supportVectors.getNext(i)) {
    result += m_class[i] * m_alpha[i] * m_kernel.eval(index, i,
        inst);
  }

  // Ultimo ajuste del resultado y devolucion
  result -= m_b;

  return result;
}
```

Las iteraciones del bucle son independientes entre sí. La variable *result* va acumulando el resultado en cada iteración y la unica precaución necesaria es que la consulta de su valor y la actualización se hagan en exclusión mutua. Teniendo en cuenta ese detalle no hay ningún inconveniente en que las invocaciones al Kernel se hagan en paralelo.

G.3. Paralelismo en la clase SMO

La clase *SMO* es el clasificador SVM que BCatalog utiliza para aprender las preferencias del usuario y recomendarle lecturas de su interés.

Durante la operación de entrenamiento del clasificador se crean tantos clasificadores binarios como clases de entrenamiento, proporcionando a cada uno una copia del conjunto de datos de ejemplo debidamente etiquetados.

Entrenar el sistema consisiste pues en entrenar cada uno de los objetos *BinarySMO*. Puesto que reciben una copia propia del conjunto de datos pueden entrenarse en paralelo sin necesidad de ningún mecanismo de control ni otras medidas para garantizar la corrección de los entrenamientos individuales.

La fase de predicción se realiza de la misma manera. Una vez entrenados dichos objetos, el método *distributionForInstance* ofrecido por el clasificador SVM invoca al método *SVMOutput* de los clasificadores binarios y opera con los resultados. Todas estas invocaciones también pueden realizarse en paralelo para obtener mejoras en los tiempos de ejecución.

G.4. Consideraciones sobre las alternativas

Es importante conocer los efectos que cada una de las alternativas de paralelización producirán en el rendimiento del sistema. A continuación se introducen los beneficios estimados de cada una de ellas:

La paralelización de tareas a nivel más bajo, en el método de evaluación del Kernel SSK, supone mejoras tanto en operaciones de entrenamiento como de predicción. Como se ha comentado, con esta alternativa se realizan tres cálculos en paralelo, y por tanto el tiempo de ejecución de una evaluación queda en teoría acotado por el máximo de los tiempos de estos tres cálculos. Por contra, si se implementan otras alternativas se crean demasiadas tareas en paralelo y es posible que los efectos producidos por esta alternativa se vean reducidos al crear y destruir los hilos de ejecución.

La segunda alternativa de paralelización, los métodos ofrecidos por los objetos *BinarySMO* obtiene mejores resultados al aumentar el número de datos de ejemplo, puesto que se ejecutan en paralelo buena parte de las invocaciones al Kernel SSK. Un inconveniente de esta alternativa es que si varios de estos objetos trabajan al mismo tiempo el número hilos de ejecución aumenta significativamente. Este inconveniente se agrava si se combina con la alternativa anterior.

La tercera alternativa de paralelización consiste en que los objetos *BinarySMO* trabajen en paralelo, como se ha comentado en el apartado G.3. Esta alternativa resulta ventajosa si el número de clases de entrenamiento (clases de interés del usuario) aumenta. Cuantas más clases, mayor número de entrenamientos y predicciones se ejecutan al mismo tiempo.

La elección de qué niveles de paralelismo implementar no es trivial, pues depende del número de clases de entrenamiento, de la cantidad de textos de ejemplo y de la potencia de cálculo disponible en la máquina en la que se ejecuta el sistema. La solución optima requiere un estudio del servicio que se desea ofrecer y de las infraestructuras disponibles.

En la construcción de este prototipo se ha decidido implementar los dos niveles superiores, dejando la paralelización de la función de evaluación del Kernel SSK para implementaciones futuras.

G.5. Implementación de las mejoras

Para la implementación de las mejoras propuestas se ha añadido un objeto *CachedThreadPool* a los objetos *SMO* y *BinarySMO*.

La clase *CachedThreadPool* [44] implementa un gestor de tareas concurrentes. Cuando se le solicita que ejecute un trabajo en paralelo crea un hilo de ejecución y le asigna la tarea. Cuando éste termina se encarga de destruirlo de forma transparente al programador. Una ventaja importante que proporciona es la reutilización de hilos de ejecución una vez terminan su trabajo de forma que no sea necesario crearlos de nuevo ante una nueva solicitud de trabajo.

Esta característica ayuda a reducir la penalización en tiempo derivadas de la creación y destrucción de nuevos hilos de ejecución puesto que sólo se crean cuando no hay ningun hilo disponible, y se destruyen cuando pasado un tiempo un hilo no recibe tareas a ejecutar.

Los objetos *CachedThreadPool* pueden recibir cualquier objeto que implemente la interface *Runnable*, como por ejemplo objetos de la clase *Thread*, *Task* o *FutureTask*[45].

La clase *FutureTask* ejecuta un bloque de código en paralelo y proporciona un método con el que se puede recuperar el resultado de su tarea. Es la implementación elegida en este prototipo puesto que el código a ejecutar en paralelo consiste en métodos que devuelven un resultado.

Introducidas las herramientas a utilizar, se pasa a describir el proceso de modificación del código fuente. El bloque de código G.6 muestra la versión secuencial inicial y, a continuación, el código que resulta de la inclusión de paralelismo.

Código G.6: *Ejemplo de paralelización de código*

```
// INVOCACION SECUENCIAL
  k11 = m_kernel.eval(i1, i1, m_data.instance(i1));
  k12 = m_kernel.eval(i1, i2, m_data.instance(i1));
  k22 = m_kernel.eval(i2, i2, m_data.instance(i2));

// INVOCACION EN PARALELO

  // Crea FutureTasks que devuelven
  // valores de tipo double
  FutureTask<Double> t11, t12, t22;

  // Asigna a cada FutureTask la tarea
  // a realizar
  t11 = new FutureTask<Double>(
  new TareaEval(m_kernel, i1, i1,
  m_data.instance(i1)));
  t12 = new FutureTask<Double>(
  new TareaEval(m_kernel, i1, i2,
  m_data.instance(i1)));
  t22 = new FutureTask<Double>(
  new TareaEval(m_kernel, i2, i2,
  m_data.instance(i2)));

  // Envia las tareas al gestor
  executor.submit(t11);
  executor.submit(t12);
  executor.submit(t22);

  // Recoge los resultados
  k11 = t11.get();
  k12 = t12.get();
  k22 = t22.get();
```

En primer lugar se crean los objetos *FutureTask*, que ejecutarán tareas que devuelven datos de tipo Double con los resultados de la operación. Estas tareas se envían al gestor de tareas, llamado *executor* en el código. Por último se almacenan los resultados conforme las tareas van terminando.

Anexo H

Evaluación del prototipo. Experimentos

En este apartado se incluye información de interés relacionada con los experimentos realizados sobre el sistema. Esta información incluye:

1. **El *script* que lanza los experimentos:** Lanza el servidor y el cliente y, tras la ejecución de la tarea, finaliza ambos procesos. El servidor se lanza tanto en modo secuencial como en paralelo. Ver bloque de código H.1.
2. **El *script .sub* necesario para la ejecución bajo Condor**. Ver bloque de código H.2.
3. **Salida producida por Condor:** Las estadísticas generadas por Condor en la ejecución del experimento en la máquina Selene2. Ver bloque de código H.3.
4. **Salida producida por BCatalog:** Información generada por BCatalog para el trazado de ejecución. Ver bloque de código H.4.
5. **Uso de memoria y CPU en Windows:** Capturas de pantalla en ejecuciones secuencial y paralela en la máquina de Gregorio de Miguel. Ver figuras H.1 y H.2.

Código H.1: *Script que lanza los experimentos.*

```
1
2  # Lanza los experimentos en modo secuencial
3  # (sin parametro -o)
4
5  $ java -jar serv.jar --server loixiyo.com
6      --user loixiyoc_catalog --db loixiyoc_catalog
7      --pwd bcatalog12345 -v --log &
8  $ java -jar cliente.jar
9
10 # Lanza de nuevo en modo paralelo
11
12 $ java -jar serv.jar --server loixiyo.com
13     --user loixiyoc_catalog --db loixiyoc_catalog
14     --pwd bcatalog12345 -v --log -o &
15 $ java -jar cliente.jar
```

Código H.2: *Script para ejecución bajo Condor.*

```
1   Universe = vanilla
2   Executable  = /bin/bash
3
4   should_transfer_files = YES
5   when_to_transfer_output = ON_EXIT
6  # transfer_input_files = foo1.cfg,foo2.cfg
7
8   Getenv = TRUE
9  # notify_user = pepe@unizar.es
10 # notification = Complete
11 # priority = 10
12
13 # input   = test.data
14  output  = experimento.out
15  error   = experimento.err
16  log     = experimento.log
17
18 +BigJob = true
19
20 request_cpus=48
21 request_memory=48000
22
23  arguments = /home/gised/gmiguel/Experimento/run
24  Queue
```

Código H.3: *Salida producida por Condor.*

```
1
2  000 (1768988.000.000) 08/22 10:01:02
3        Job submitted from host: <172.16.4.100:47970>
4  ...
5  001 (1768988.000.000) 08/22 10:01:24
6        Job executing on host: <172.16.6.50:46806>
7  ...
8  005 (1768988.000.000) 08/22 10:05:35 Job terminated.
9    (1) Normal termination (return value 0)
10   Usr 0 00:36:49, Sys 0 00:00:02  -  Run Remote Usage
11   Usr 0 00:00:00, Sys 0 00:00:00  -  Run Local Usage
12   Usr 0 00:36:49, Sys 0 00:00:02  -  Total Remote Usage
13   Usr 0 00:00:00, Sys 0 00:00:00  -  Total Local Usage
14   706  -  Run Bytes Sent By Job
15   801528  -  Run Bytes Received By Job
16   1706  -  Total Bytes Sent By Job
17   801528  -  Total Bytes Received By Job
18 ...
```

Código H.4: *Salida producida por BCatalog.*

```
1  ********************************************
2  Catalog: Creado objeto RMI
3  Catalog: Numero de procesadores disponibles: 48
4  Catalog: Cliente esperando a arranque del servidor
5  Catalog: Creados objetos DAO
6  Catalog: Creada SVM
7  Catalog: Arrancando predictor automatico
8  Predictor: Iniciando...
9
10 Catalog: Sesion iniciada
11 Catalog: Nuevo entrenamiento
12 Catalog: Obteniendo datos de entrenamiento. 4 clases
13 Catalog: Iniciando entrenamiento en paralelo
14 Catalog: Entrenamiento terminado en 109 segundos,
15           guardando SVM en disco
16
17 Catalog: Prediciendo URL simple
18 Catalog: Post cacheado http://www.heraldo.es/noticias ...
19 Catalog: Iniciando prediccion en paralelo
20
```

```
21 Resultados para 0 = 1.0329057196343712
22 Resultados para 1 = -0.13558198210978978
23 Resultados para 2 = 0.9730040019371566
24 Resultados para 3 = 0.8660084505334502
25
26 Catalog: Prediccion terminada en 134 segundos.
27 Catalog: Pertenencia a clases de
28          http://www.heraldo.es/noticias ...
29
30 Catalog:    - tecnologia: 0.0
31 Catalog:    - economia: 1.0
32 Catalog:    - deportes: 0.0
33 Catalog:    - politica: 0.0
34
35 Catalog: Finalizando servidor a peticion del usuario
```

Figura H.1: *Uso de memoria y CPU en ejecución secuencial.*

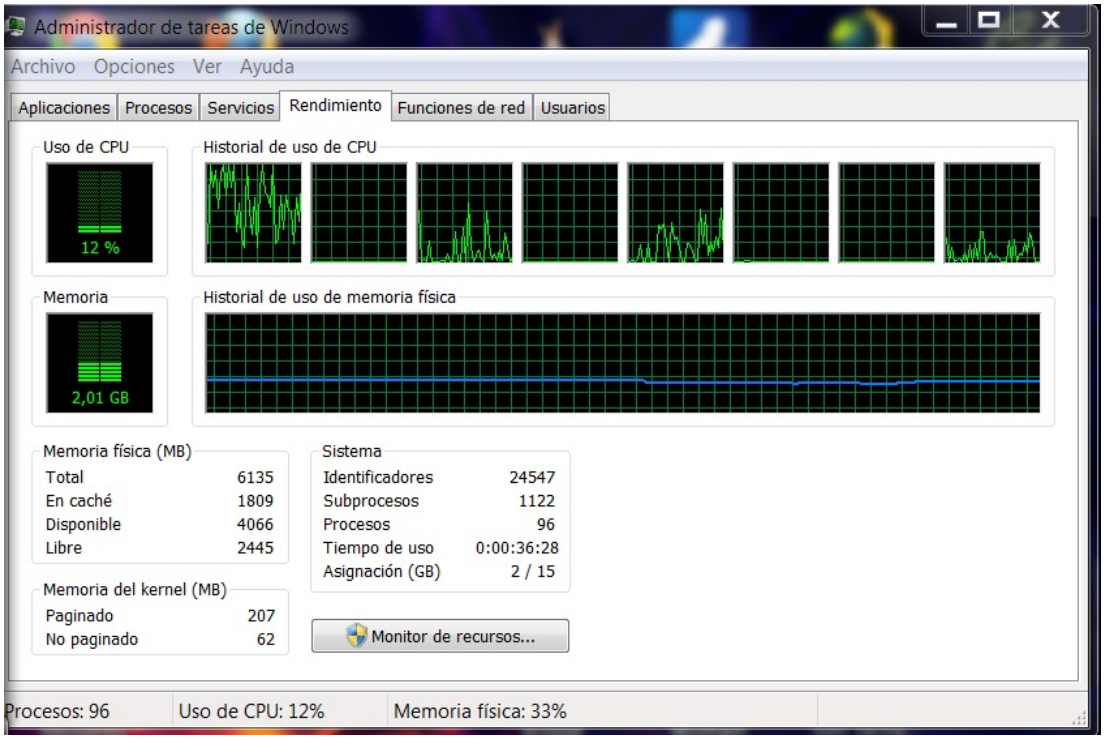
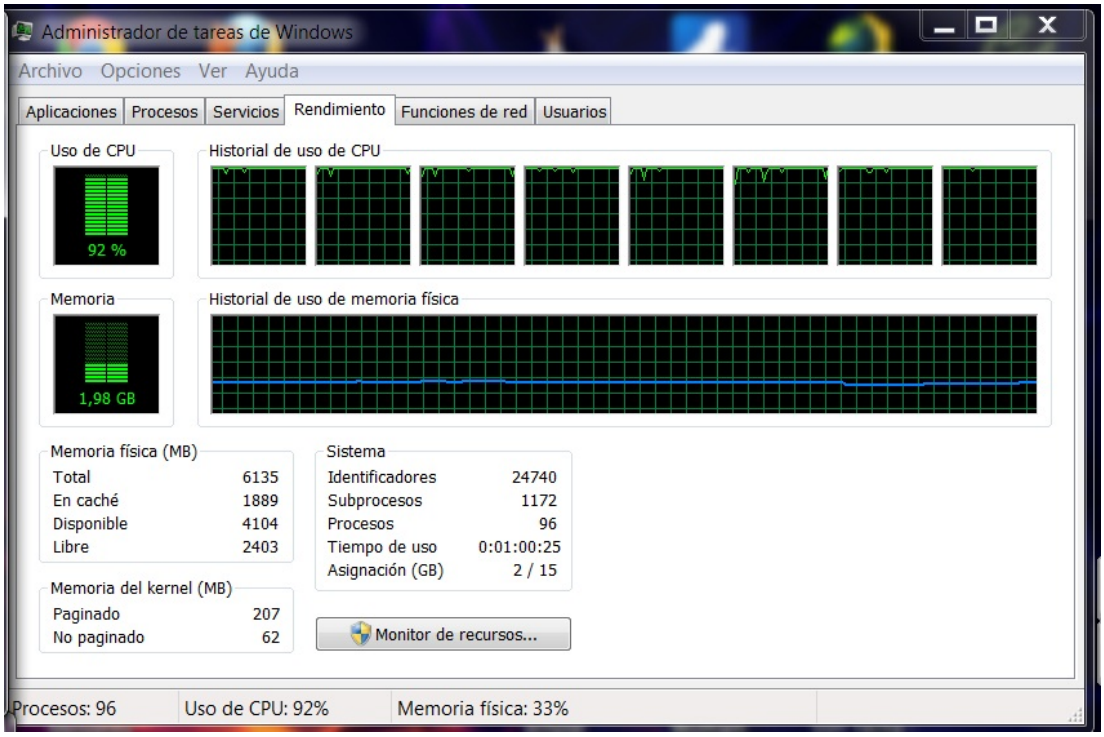


Figura H.2: *Uso de memoria y CPU en ejecución paralela.*



Índice de figuras

Bibliografía

[1] J. J. Merelo and F. Tricas, "Métrica de la blogosfera. algunas medidas y relaciones en la blogosfera hispana," *Telos: Cuadernos de comunicación e innovación*, vol. 65, pp. 101–104, 2005.

[2] H. Liu, P. S. Yu, N. Agarwal, and T. Suel, "Guest editors' introduction: Social computing in the blogosphere," *IEEE Internet Computing*, vol. 14, no. 2, pp. 12–14, 2010.

[3] "Real Academia Española." http://www.rae.es/rae.html.

[4] "Word Reference." http://www.wordreference.com.

[5] "Diccionario Ideológico de Zirano." http://www.zirano.com.

[6] "Servicio de Blogs Wordpress." http://www.wordpress.com.

[7] "Servicio de Blogs BlogSpot." http://www.blogspot.com.

[8] "CMS Xoops." http://www.xoops.org.

[9] "CMS Joomla." http://www.joomla.org.

[10] "CMS PHP-Nuke." http://www.phpnuke.org.

[11] "WordPress, versión de descarga." http://www.wordpress.org.

[12] "FrameWork MVC CodeIgniter." http://www.codeigniter.com.

[13] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.

[14] B. Scholkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, 2001.

[15] T. Gartner, J. W. Lloyd, and P. A. Flach, "Kernels for structured data," *SIGKDD EXPLORATIONS*, vol. 5, pp. 49–58, 2002.

[16] J. Fürnkranz, "Pairwise classification as an ensemble technique," in *Proceedings of the 13th European Conference on Machine Learning (ECML-02)* (T. Elomaa, H. Mannila, and H. Toivonen, eds.), vol. 2430 of *Lecture Notes in Artificial Intelligence*, (Helsinki, Finland), pp. 97–110, Springer-Verlag, 2002.

[17] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, C. Watkins, and B. Scholkopf, "Text classification using string kernels," *Journal of Machine Learning Research*, vol. 2, pp. 563–569, 2002.

[18] A. K. Seewald and F. Kleedorfer, "Lambda pruning: an approximation of the string subsequence kernel for practical svm classification and redundancy clustering," *Adv. Data Analysis and Classification*, vol. 1, no. 3, pp. 221–239, 2007.

[19] "Lenguaje de programación Java." http://java.com/es/.

[20] "Servidor Web Apache." http://www.apache.org.

[21] "Extensión Tomcat para Apache." http://tomcat.apache.org.

[22] "Estándar HTML." http://www.w3.org/html.

[23] "Estándar CSS." http://www.w3.org/css.

[24] "Java Beans, Java Entreprise Edition." http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138795.html.

[25] "Java Server Faces." http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html.

[26] "Java RMI." http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html.

[27] "Clasificadores Weka." http://www.cs.waikato.ac.nz/ml/weka/.

[28] "Bases de datos MySQL." http://http://www.mysql.com/.

[29] "Driver Java DataBase Connection." http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136101.html.

[30] "Clase Thread de Java." http://download.oracle.com/javase/1.4.2/docs/api/java/lang/Thread.html.

[31] "Lematizador para idioma Español." http://stemmer-es.sourceforge.net/.

[32] "Algoritmo de lematización de Porter." http://tartarus.org/~martin/PorterStemmer/.

[33] "Interface ExecutorService de Java." http://download.oracle.com/javase/1,5.0/docs/api/java/util/concurrent/ExecutorService.html.

[34] "Interface Runnable de Java." http://download.oracle.com/javase/1,5.0/docs/api/java/util/concurrent/ExecutorService.html.

[35] "Diario Heraldo de Aragón." http://www.heraldo.es.

[36] "Periódico deportivo Marca." http://www.marca.es.

[37] "Diario El País." http://www.elpais.com.

[38] "Cluster Hermes. Universidad de Zaragoza." http://web.hermes.cps.unizar.es/ganglia/.

[39] "Proyecto Condor." http://www.cs.wisc.edu/condor/.

[40] "Estándar XML." http://www.w3.org/XML/.

[41] "Librería JArgs." http://jargs.sourceforge.net.

[42] "Archivo SiteMap.xml." http://www.w3.org/WAI/sitemap.html.

[43] "Lenguaje PHP." http://www.php.net.

[44] "Clase CachedThreadPool de Java." http://download.oracle.com/javase/1.5.0/docs/api/java/util/concurrent/Executors.html#newCachedThreadPool().

[45] "Clase FutureTask de Java." http://download.oracle.com/javase/1.5.0/docs/api/java/util/concurrent/FutureTask.html.