Minería de texto para la categorización automática de documentos

M. Alicia Pérez Abelleira y Carolina A. Cardoso [*]

aperez@ucasal.net

Resumen

La clasificación de documentos de texto es una aplicación de la minería de textos que pretende extraer información de texto no estructurado. Su interés se justifica porque se estima que entre el 80% y el 90% de los datos de las organizaciones son no estructurados. Por otro lado, la búsqueda semántica permite al usuario especificar en una consulta no solamente términos que deben aparecer en el documento, sino conceptos y relaciones, que pueden detectarse mediante el análisis de texto. El objetivo de este trabajo es implementar un buscador semántico que aproveche el resultado de algoritmos de aprendizaje automático para la clasificación de documentos.

El dominio de aplicación es un corpus de más de 8000 documentos que contienen nueve años de resoluciones rectorales de la Universidad Católica de Salta en distintos formatos (Microsoft Word, texto plano, PDF). El sistema aprovecha las ventajas de la arquitectura UIMA sobre la que se han implementado analizadores que extraen meta-datos (fecha y número de resolución, unidad académica, personas, etc.) Asimismo se han explorado una variedad de algoritmos de aprendizaje semi-supervisado aplicados a la categorización de documentos, comparándolos experimentalmente entre sí y con algoritmos supervisados. Estos últimos precisan una gran cantidad de ejemplos etiquetados, algo generalmente costoso en la práctica en el caso de la clasificación de documentos. Los algoritmos semi-supervisados en cambio son capaces de aprovechar ejemplos no

[*] Alicia Pérez es Licenciada en Informática por la Universidad Politécnica de Madrid y PhD in Computer Science por Carnegie Mellon University. Actualmente se desempeña en la Facultad de Ingeniería e Informática de la UCS como docente de Sistemas Expertos y de Compiladores y como Jefa del Departamento de Investigación y en la Universidad Carlos III de Madrid (España) como docente de Inteligencia Artificial (2004, 2006) de la carrera de Ingeniería Informática – grupo bilingüe.
Carolina Cardoso es Licenciada en Ciencias de la Computación, se desempeña como Ayudante Docente de las asignaturas Estructuras de Datos y Algoritmos y Lenguaje I y Adjunta Interina de Compiladores.

etiquetados. En particular, en los experimentos en nuestro dominio el algoritmo de *co-training* ha demostrado tener buenas propiedades, incluso a pesar de la restricción teórica de que los atributos deben ser redundantes e independientes. No obstante el algoritmo supervisado SMO que entrena SVMs es superior.

Nuestro objetivo final es construir un buscador semántico que utilice los metadatos obtenidos automáticamente por los anotadores implementados en UIMA y las categorías asignadas automáticamente por los algoritmos de aprendizaje.

Palabras claves: buscador semántico, aprendizaje semisupervisado, categorización de documentos, minería de texto, UIMA

1. Introducción

El conocimiento es cada vez más un recurso de importancia estratégica para las organizaciones y su generación, codificación, gestión, divulgación aportan al proceso de innovación. Todos estos aspectos se incluyen en lo que se ha dado en llamar la *gestión del conocimiento*. La cantidad de documentos de diversos tipos disponibles en una organización es enorme y continúa creciendo cada día. Estos documentos, más que las bases de datos, son a menudo un repositorio fundamental del conocimiento de la organización, pero a diferencia de éstas la información no está estructurada. La minería de textos tiene como objetivo extraer información de texto no estructurado, tal como entidades (personas, organizaciones, fechas, cantidades) y las relaciones entre ellas. Por otro lado, la búsqueda semántica permite al usuario especificar en una consulta no solamente términos que deben aparecer en el documento, sino esas entidades y relaciones extraídas mediante el análisis de texto.

La categorización de documentos de texto es una aplicación de la minería de texto que asigna a los documentos una o más categorías, etiquetas o clases, basadas en el contenido. Es un componente importante de muchas tareas de organización y gestión de la información. El enfoque tradicional para la categorización de textos en que los expertos en el dominio de los textos definían manualmente las reglas de clasificación ha sido reemplazado por otro basado en técnicas de aprendizaje automático, o en combinaciones de éste con otras técnicas.

Nuestro trabajo se centra en desarrollar técnicas para la categorización automática de documentos según su contenido que

avancen el estado del arte en nuestro medio, aplicando el automático a la minería de texto. El objetivo final es implementar un buscador semántico que aproveche el resultado de algoritmos de aprendizaje automático para la clasificación de documentos. El dominio de aplicación es un corpus de más de 8000 documentos que contienen nueve años de resoluciones rectorales de la Universidad Católica de Salta en distintos formatos (Microsoft Word, texto plano, PDF).

Este artículo comienza describiendo la información no estructurada, repositorio fundamental del conocimiento de una organización, y las arquitecturas para la gestión de información no estructurada. La Sección 3 muestra nuestra instanciación del modelo general para el problema de la clasificación y búsqueda de resoluciones rectorales. En la Sección 4 se exploran diferentes algoritmos para la categorización de documentos y se describen los experimentos realizados para determinar su adecuación a nuestro dominio. Concluye el trabajo con el funcionamiento del motor de búsqueda semántica (Sección 5) y presentando algunas conclusiones.

2. Información estructurada y no estructurada

La información estructurada se caracteriza por tener un significado que pretende no tener ambigüedad y que está representado explícitamente en la estructura o formato de los datos. El ejemplo típico es una base de datos relacional. De la información no estructurada podría decirse que su significado no está implicado en su forma y por tanto precisa interpretación para aproximar o extraer el mismo. Los documentos en lenguaje natural o hasta de voz, audio, imágenes, etc. entran en esta categoría. El interés por extraer significado de la información no estructurada se debe a que se estima que entre el 80% y el 90% de los datos de las organizaciones son no estructurados (Moore, 2002). Aunque muchas organizaciones han invertido en tecnologías para minería de datos estructurados procedentes de sus bases de datos y sistemas transaccionales, en general no han intentado capitalizar sus datos no estructurados o semi-estructurados.

Una aplicación de gestión de la información no estructurada (UIM por sus siglas en inglés) típicamente es un sistema de software que analiza grandes volúmenes de información no estructurada con el fin de descubrir, organizar y entregar conocimiento relevante al usuario final. La información no estructurada puede ser mensajes de correo electrónico, páginas web o documentos generados con una variedad de procesadores de texto, como en el caso de las resoluciones rectorales de nuestra universidad. Estas aplicaciones utilizan para el análisis una

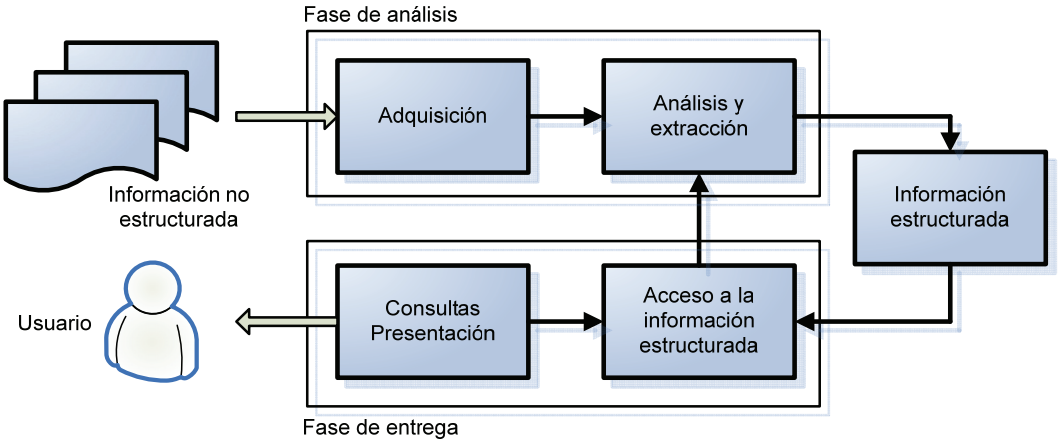


Figura 1: Esquema de una arquitectura UIM (Ferrucci & Lally, 2004)

variedad de tecnologías en las áreas del procesamiento del lenguaje natural, recuperación de la información, aprendizaje automático, ontologías y hasta razonamiento automático.

El resultado del análisis generalmente es información estructurada que representa el contenido de la entrada no estructurada y que se hace accesible al usuario mediante aplicaciones adecuadas. Un ejemplo puede ser la generación de un índice de búsqueda y la utilización de un buscador que facilita el acceso a documentos de texto por tema, ordenados según su relevancia a los términos o conceptos procedentes de la consulta del usuario.

Existen diversas arquitecturas para el desarrollo de aplicaciones UIM. Para nuestro trabajo hemos utilizado UIMA (Ferruci & Lally, 2004), una arquitectura software basada en componentes que surgió como proyecto de investigación de IBM y fue puesta a disposición de la comunidad como software libre.

3. Arquitectura del sistema

Conceptualmente suele verse a las aplicaciones de UIM con dos fases: una de análisis y otra de entrega de la información al usuario. En la fase de análisis se recogen y analizan colecciones de documentos. Los resultados del análisis se almacenan en algún lenguaje o depósito intermedio. La fase de entrega hace accesible al usuario el resultado del análisis, y posiblemente el documento original completo mediante una interfaz apropiada. La Figura 1 muestra ambas fases de manera esquemática.

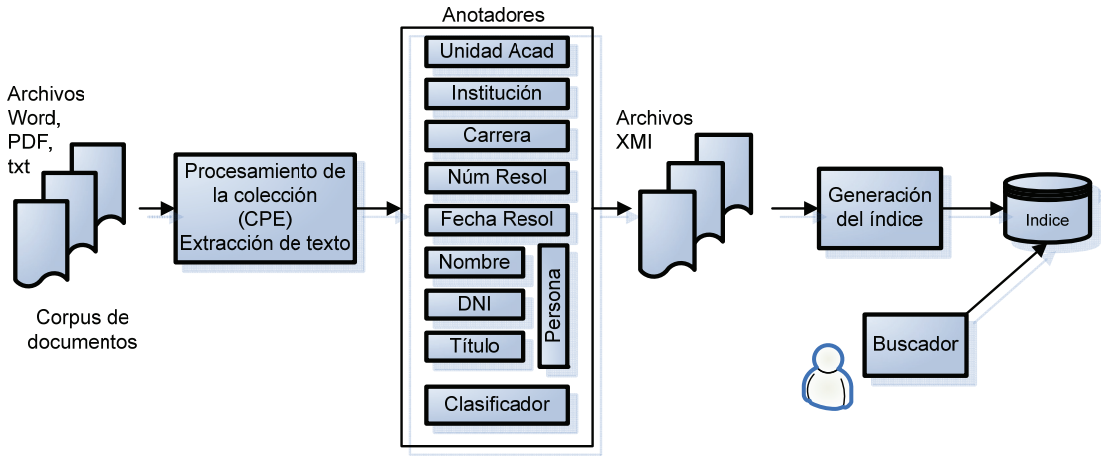


Figura 2: Arquitectura del sistema

La Figura 2 muestra la aplicación de este esquema a nuestro dominio, en el que partimos de más de 8000 resoluciones rectorales en archivos de texto de distinto tipo: Word, PDF, texto plano. Previo al análisis, se procede a la extracción del texto de cada archivo utilizando las herramientas de software libre POI (poi.apache.org) y tm-extractors (www.textmining.org). Las vocales acentuadas y otros caracteres especiales son reemplazados por los caracteres ASCII correspondientes (ej. *á* es reemplazado por *a*). También se divide en partes la resolución extrayendo el encabezado (texto que contiene el número y la fecha de la resolución) y el cuerpo con la mayor parte de la información, y descartando en lo posible el texto “de forma”.

La fase de análisis incluye tokenización y detección de entidades en documentos individuales tales como personas, fechas, organizaciones, unidades académicas y datos sobre la resolución (fecha y número). Además con la ayuda de un clasificador aprendido automáticamente del corpus de resoluciones, como se explica en la Sección 4, se anota cada documento con una categoría. Existen 21 categorías que fueron obtenidas del personal especializado en la elaboración de resoluciones. Algunos ejemplos son: designación de planta docente, convenio de pasantías, convenio de colaboración, llamado a concurso docente, o designación de tribunal de concurso.

El resultado de la fase de análisis es un conjunto de archivos en formato XMI (Sección 3.3). Estos archivos contienen, además de las partes relevantes del texto original, metadatos en forma de anotaciones correspondientes a las entidades y a la categoría de documentos. Estos archivos serán procesados para construir el índice de un motor de búsqueda que contiene los tokens (en nuestro caso, las palabras que

aparecen en el texto) así como las entidades y categorías extraídas automáticamente.

En la fase de entrega existe una interfaz para hacer consultas de búsqueda en el índice de forma que el usuario pueda buscar documentos que contengan combinaciones booleanas de tokens, entidades y categorías mediante un motor de búsqueda semántica.

3.1. Análisis a nivel de documento

UIMA es una arquitectura especialmente pensada para combinar los distintos componentes de la fase de análisis de texto y facilitar su disponibilidad para diversas aplicaciones en la fase de entrega. En UIMA, el componente que contiene la lógica del análisis se llama anotador. Cada anotador realiza una tarea específica de extracción de información de un documento y genera como resultado anotaciones, que son añadidas a una estructura de datos denominada CAS (*common analysis structure*). A su vez, esas anotaciones pueden ser utilizadas por otros anotadores. Los anotadores pueden ser agrupados en anotadores agregados.

La mayoría de los anotadores de nuestro sistema realizan reconocimiento de entidades con nombre (*named entity recognition* o NER), tarea principal de los sistemas de recuperación de información, que busca localizar y clasificar elementos atómicos del texto que pertenecen a categorías predefinidas. Las entidades extraídas por nuestro sistema son: personas, unidades académicas, carreras, instituciones, fechas, número y año de las resoluciones. Cada una de las entidades es extraída por un anotador. Además, para detectar entidades correspondientes a personas se utilizan entidades como nombres propios, DNIs y títulos, obtenidas por los anotadores correspondientes.

Las técnicas utilizadas para el reconocimiento de entidades son (Alias-i, 2008; Feldman & Sanger, 2007):

- Equiparación con expresiones regulares que capturan el patrón que siguen las entidades (ejemplos son la detección de DNIs, fecha y número de las resoluciones).
- Equiparación con diccionarios y *gazetteers* (ejemplos son las carreras, unidades académicas, instituciones, títulos y nombres propios). El diccionario de nombres propios consta de más de 1300 nombres y fue extraído automáticamente del sistema de gestión de alumnos. El enfoque basado en componentes de UIMA nos ha permitido adaptar el *Gazetteer Annotator* de Julie Lab (Tomanek &

Wermter, 2008), que está basado en la implementación que hace Lingpipe del algoritmo Aho-Corasick (Alias-i, 2008).
- Equiparación con plantillas: para detectar entidades correspondientes a personas se utiliza una plantilla que describe a la persona mediante los siguientes atributos: *nombre1, nombre2, apellido(s), DNI, título*. Sólo *nombre1* y *apellido(s)* son obligatorios. Todos los atributos son a su vez entidades detectadas por anotadores, excepto *apellido(s)* que se detecta mediante una expresión regular.

Además de los anotadores mencionados, se utiliza el anotador de UIMA que detecta tokens usando una sencilla segmentación basada en los espacios en blanco, y crea anotaciones con tokens y sentencias.

Aparte de todos estos anotadores, existe otro que asigna la categoría de documento en base al modelo aprendido automáticamente, como se describe en la Sección 4.

3.2. Análisis a nivel de colección

Mientras que el análisis descripto en la sección anterior se realiza en cada documento, existen casos en que el análisis debe ser realizado al nivel de una colección de documentos. Un caso particular es un bucle de realimentación, que produce recursos estructurados a partir del análisis de una colección de documentos y luego usa esos recursos para permitir el subsiguiente análisis de documentos (Ferrucci & Lally, 2004). Tal estructura de realimentación se ha utilizado para implementar el aprendizaje automático de categorías (Figura 3). Esta aplicación se puede dividir en dos fases: la fase de entrenamiento (parte superior de la figura) y la fase de clasificación (parte inferior).

En la fase de entrenamiento, un motor de procesamiento de colecciones (*Collection processing engine* o CPE) analiza una colección de documentos, denominada conjunto de entrenamiento. Este CPE invoca a un analizador que utilizando algoritmos de aprendizaje automático produce un modelo. Este modelo, basado en los atributos del documento, es capaz de asignarle una categoría. El modelo forma parte de un nuevo anotador, el clasificador de la Figura 2, que se encarga de aplicar el modelo. En la fase de clasificación, el modelo es consultado por el clasificador al analizar un nuevo documento y asignarle una categoría, que se convierte en una anotación más sobre el documento. La Sección 4 describe los algoritmos de aprendizaje utilizados para generar el modelo.

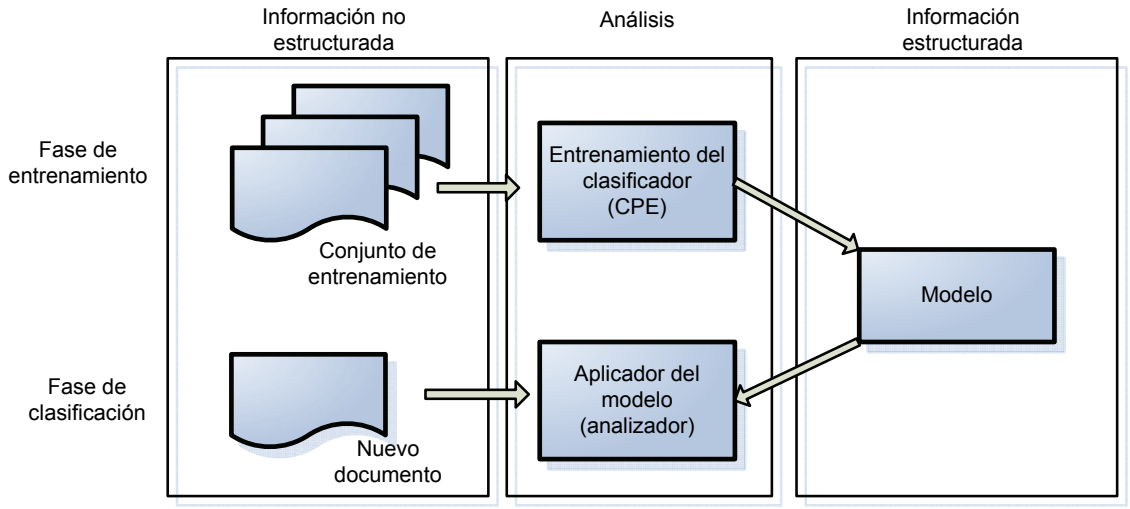


Figura 3: Implementación del aprendizaje automático en la arquitectura de UIMA

3.3. Formato de la información estructurada

Un importante principio de arquitectura en UIMA es que todos los analizadores operan sobre una estructura de datos estándar, el CAS. El CAS incluye el texto y las anotaciones. El CAS es el principal repositorio de información estructurada y utiliza como lenguaje UIMA el estándar XMI (XML Metadata Interchange) (OMG, 2007). XMI está basado en UML (Unified Modeling Language) y proporciona un formato en XML para el intercambio de modelos UML, lo que además hace posible la interoperabilidad con otras aplicaciones. En XMI el texto se incluye en el SofA (*Subject of Analysis*) del CAS como atributo sofaString y las anotaciones en el CAS tienen prefijos particulares en el espacio de nombres de XMI, como por ejemplo <*mio:Carrera*>. El espacio de nombres queda definido al declarar un sistema de tipos como parte del dominio en UIMA. La Figura 4 muestra un ejemplo del resultado del proceso de análisis. Dos *SofAs* recogen el encabezado, del que sólo se extrae la fecha y el número de resolución, y el cuerpo de la misma. La primera anotación del tipo *SourceDocument Information* guarda información sobre el archivo, para poder recuperarlo posteriormente, por ejemplo, para mostrarlo al usuario como resultado de una búsqueda. A continuación aparecen las anotaciones de Unidad Académica y de Carrera con su principio y fin en el texto del documento. La anotación Fecha de Resolución tiene varios campos: el texto original (*FechaResolCompleta*) y el día, mes y año extraídos del mismo.

Finalmente la anotación *Clase* contiene la categoría *DesigPlanta* asignada a esta resolución por el modelo aprendido.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xmi:XMI … xmi:version="2.0">
…
  <cas:Sofa xmi:id="1" sofaNum="2" sofaID="encabezado"
mimeType="text" sofaString="ResoluciOn N 470/08 En el Campo
Castanares, sito en la Ciudad de Salta, Capital de la Provincia del
mismo nombre, Republica Argentina, sede de la Universidad Catolica
de Salta, a los veintisiete dias el mes de mayo del ano dos mil ocho:" />
  <cas:Sofa xmi:id="13" sofaNum="1" sofaID="_InitialView"
mimeType="text" sofaString="la presentacion efectuada por las
autoridades de la Escuela Universitaria de Educacion Fisica,
dependiente de la Facultad de Artes y Ciencias en virtud de la cual se
propone las modificaciones de designaciones docentes Planta
Transitoria, para la carrera Licenciatura en Educacion Fisica…" />
…
  <examples:SourceDocumentInformation xmi:id="25" sofa="13"
begin="0" end="0" uri="file:/D:/UIMA/docs/RES.%20%20Nº0470-08.txt"
offsetInSource="0" documentSize="2280" lastSegment="false" />
  <mio:UA xmi:id="48" sofa="13" begin="177" end="208"
confidence="30.0"
componentId="de.julielab.jules.lingpipegazetteer.GazetteerAnnotator"
specificType="UnidadAcademica" />
  <mio:Carrera xmi:id="72" sofa="13" begin="398" end="430"
confidence="0.0"
componentId="de.julielab.jules.lingpipegazetteer.GazetteerAnnotator"
specificType="Carrera" />
  <mio:NumeroResol xmi:id="33" sofa="1" begin="0" end="19"
nroResol="RESOLUCION N 470/08" numero="470" anio="2008" />
  <mio:FechaResol xmi:id="40" sofa="1" begin="196" end="248"
anio="2008" mes="MAYO" dia="27"
fechaResolCompleta="VEINTISIETE DE MAYO DE DOS MIL OCHO" />
   <mio:Clase xmi:id="28" sofa="1" begin="0" end="0" valor="
DesigDocPlanta" />
…
</xmi:XMI>
```

Figura 4: Ejemplo de texto anotado.

4. Aprendizaje automático para la categorización de documentos

Se define la categorización (o clasificación) de textos como la actividad de etiquetar textos en lenguaje natural con categorías temáticas de un conjunto predefinido (Sebastiani, 2005). Si a esto se añade la identificación automática de dichas categorías, el problema se denomina *clustering* de textos.

El enfoque dominante actualmente para el problema de categorización de textos se basa en técnicas de aprendizaje automático: se construye automáticamente un clasificador mediante aprendizaje inductivo de las características o atributos de las categorías a partir de un conjunto de documentos previamente clasificados que sirven como conjunto de entrenamiento. Se trata por tanto del llamado aprendizaje supervisado. Los algoritmos de aprendizaje generalmente utilizados para la (NaiveBayes, SMO, etc) pueden ser entrenados para clasificar documentos dado un conjunto suficientemente grande de ejemplos de entrenamiento, cada uno de los cuales ha sido etiquetado previamente con la categoría correspondiente. Una importante limitación de estos algoritmos es que precisan una cantidad grande de ejemplos (documentos) etiquetados para poder alcanzar una precisión apropiada. El etiquetado de hasta miles de documentos ha de ser realizado por una persona, especialista en el área de interés de los documentos, y por tanto es un proceso muy costoso.

Esto ha dado pie a nuevas familias de algoritmos de aprendizaje denominado semi-supervisado, capaces de aprender de un número limitado de ejemplos de entrenamiento etiquetados utilizando adicionalmente documentos no etiquetados, generalmente fácilmente disponibles. En un trabajo anterior (Pérez & Cardoso, 2009) hemos experimentado en una variedad de dominios con tres enfoques representativos de este tipo de algoritmos: la aplicación de *expectation maximization* (EM) al uso de datos etiquetados y no etiquetados; los algoritmos de *co-training*; y el algoritmo *co-EM* que combina características de los dos anteriores. En nuestros experimentos la implementación de *co-training* ha dado los mejores resultados para la categorización de textos, por lo que enfocamos en ella nuestra descripción.

La categorización de texto suele desarrollarse en tres etapas: pre-procesamiento de los datos, construcción del clasificador y categorización de los nuevos documentos, A continuación se describen los dos primeros pasos. El tercero es realizado mediante un anotador (ver Sección 3).

4. 1. Pre-procesamiento de los datos

En esta fase se transfiere el texto original, resultante de la tokenización, a un formato compacto que será utilizado en las fases siguientes. Incluye:

- Lematización (*stemming*), que reduce una palabra a su raíz (stem). Este proceso ayuda al recall, que es una medida sobre el número de documentos que se pueden encontrar con una consulta. Se ha utilizado el algoritmo propuesto e implementado en Snowball (snowball.tartarus.org).
- Eliminación de palabras de función (stopwords: artículos, preposiciones, conjunciones, etc. o) y otras dependientes del dominio. En el caso de las resoluciones rectorales, pueden ser 'Universidad Católica de Salta', 'rector', 'resuelve', etc. Esta eliminación está basada en un diccionario y ocurre después de la lematización.
- Selección de atributos, para reducir la dimensionalidad (número de atributos) del espacio de datos eliminando los que parezcan irrelevantes, como por ejemplo palabras que aparezcan en todos los documentos.
- Asignar distintos pesos a los atributos (o el mismo peso). Es común utilizar tf–idf (*term frequency–inverse document frequency*) para evaluar la importancia de una palabra en el corpus de documentos. Estamos realizando experimentos para elegir la opción más adecuada en nuestro corpus.

Con los pasos anteriores cada documento de la colección se convierte a una representación compacta adecuada para los algoritmos de aprendizaje. En nuestro caso dicha representación es el formato arff y el filtro *StringToWordVector* de Weka (www.cs.waikato.ac.nz/ml/weka/) puede ser configurado para aplicar la mayoría de las transformaciones anteriores, exceptuando la conversión de formatos de documentos.

4.2. Construcción del clasificador

Los documentos, transformados en conjuntos de atributos y valores por la etapa anterior, se utilizan para construir un clasificador que asignará categorías a nuevos documentos. Hemos evaluado una variedad de algoritmos. La limitación del aprendizaje supervisado es la disponibilidad de una cantidad importante de ejemplos de entrenamiento ya categorizados. Esta categorización o etiquetado es un proceso manual y laborioso. Para disminuir este esfuerzo surgen los algoritmos semi-supervisados, que son capaces de aprender a partir de

un conjunto de ejemplos etiquetados y no etiquetados. A continuación se describen brevemente algunos de los algoritmos utilizados en los experimentos.

4.2.1. SMO

El aprendizaje de máquinas de vectores soporte es un método supervisado que ha demostrado buenas propiedades para la categorización de documentos. El algoritmo SMO (*sequential minimal optimization*) que hemos utilizado es la implementación en Weka del algoritmo de optimización minimal secuencial para entrenar máquinas de vectores soporte usando un kernel polinomial (Witten & Frank, 2005).

4.2.2. Co-training

Blum y Mitchell (1998) introdujeron la idea de *co-training*, al reconocer que los atributos de algunos conjuntos de datos pueden descomponerse naturalmente en dos subconjuntos, y cada uno de ellos sirve efectivamente para clasificar cada documento, es decir, son redundantemente predictivos. Cada uno de esos subconjuntos actúa como una perspectiva diferente de la tarea de clasificación. La Tabla 1

Entradas: colecciones iniciales de documentos etiquetados *E* y sin etiquetar *N.*
•Dividir el conjunto de atributos en dos, *A* y *B.*
Repetir mientras existan documentos en *N*:
•Construir un clasificador *CA* usando los atributos *A* de los documentos de *E*.
•Construir un clasificador *CB* usando los atributos *B* de los documentos de *E*.
•Para cada clase *C*, elegir el documento de *N* que ha sido etiquetado por *CA* como perteneciente a *C* con mayor confianza, sacarlo de *N* y añadirlo a *E.*
•Para cada clase *C*, elegir el documento de *N* que ha sido etiquetado por *CB* como perteneciente a *C* con mayor confianza, sacarlo de *N* y añadirlo a *E.*
Salida: dos clasificadores *CA* y CB cuyas predicciones son combinadas multiplicando las probabilidades y renormalizándolas.

Tabla 1: El algoritmo de *co-training*

describe el algoritmo. Nuestra implementación genera los modelos en cada iteración con el algoritmo SMO.

Co-training depende de que las dos perspectivas (conjuntos de atributos A y B) sean redundantes e independientes - para cada instancia, los atributos de A son condicionalmente independientes de los de B dada la clase de la instancia. Aunque parezca una suposición poco realista en la práctica, hay resultados que muestran que funciona bien en ciertos conjuntos de datos que no satisfacen completamente estos requisitos, y esta consideración sigue siendo una pregunta abierta (Blum & Mitchell, 1998; Nigam & Ghani; 2000). Nuestros experimentos han estado dirigidos a entender el comportamiento en la categorización de textos en nuestro problema particular.

4.2.3. Expectation maximization (EM)

EM está basado en una técnica sencilla pero efectiva para clasificar textos, Naive Bayes, que sólo aprende de datos etiquetados (Witten & Frank, 2005). La idea es utilizar Naive Bayes para aprender clases para un pequeño conjunto etiquetado y después ampliarlo a un conjunto grande de datos no etiquetados utilizando el algoritmo EM de clustering iterativo (Nigam et al, 2000; Witten & Frank, 2005). El procedimiento tiene dos pasos: primero entrenar un clasificador Naive Bayes usando los datos etiquetados. Segundo, aplicarlo a los datos no etiquetados para etiquetarlos con las probabilidades de clase (el paso *E* o *expectation*). Tercero, entrenar un nuevo clasificador usando ahora las etiquetas de todos los datos (el paso *M* o maximización). Cuarto, repetir estos pasos hasta llegar a la convergencia (cuando el cambio en las probabilidades asignadas es menor que un cierto umbral). Nuestra implementación en Weka tomó como punto de partida la de Ray Mooney (disponible en www.cs.utexas.edu/users/ml/risc).

4.3. Configuración de los experimentos

Para evaluar nuestro enfoque hemos utilizado un corpus de 1000 resoluciones rectorales pertenecientes al año 2007 a las que se han asignado una de 21 categorías manualmente. De este corpus hemos extraído conjuntos de entrenamiento de diversos tamaños para experimentar comparando el comportamiento de los algoritmos supervisados (SMO) y no supervisados (*co-training*, EM) en función del número de ejemplos disponibles. Estos algoritmos fueron seleccionados en base a resultados de experimentos anteriores (Pérez y Cardoso, 2009).

El corpus ha sido preprocesando utilizando las utilidades de filtrado de Weka con los pasos descritos en la sección 4.1. En este preprocesamiento también hemos variado los valores de diversos parámetros, especialmente las maneras de seleccionar atributos y de asignar pesos a los atributos.

- Para reducir la dimensionalidad se ha realizado la selección de los 100 atributos más relevantes. Se ha experimentado con dos maneras de seleccionar estos atributos: los 100 más relevantes en la colección completa para todas las clases, y los 100 más relevantes en la colección completa para cada una de las clases. Esta segunda opción selecciona en la práctica entre 700 y 900 atributos.
- Los dos mecanismos más populares para asignar pesos a los atributos son binario y tfidf. En el primer caso, los pesos son 0 o 1 indicando ausencia o presencia de una palabra (el atributo) en el documento. Tfidf asigna mayor peso a los atributos que aparecen más frecuentemente, es decir, los considera representativos del contenido del documento, pero además equilibra esto reduciendo el peso de un atributo si aparece en muchos documentos, es decir, es menos discriminante.

Para evaluar la calidad de la clasificación de cada modelo en cada variación del corpus, algoritmo, y parámetros de preprocesamiento hemos examinado dos medidas: precisión y F1. La precisión (*accuracy* o porcentaje de documentos correctamente clasificados) es la más utilizada para evaluar algoritmos de aprendizaje, pero por si sola puede no ser la más representativa en la categorización de documentos ya que cada clase típicamente tiene muchos más ejemplos negativos que positivos. Por ello se utiliza F1 que combina con igual peso *precision* (número de documentos correctamente categorizados como pertenecientes a una determinada clase dividido por el número total de documentos etiquetados como pertenecientes a tal clase) y *recall* (número de documentos correctamente categorizados como pertenecientes a una determinada clase dividido por el número total de documentos que realmente pertenecen a esa clase, es decir, incluyendo los que el algoritmo no haya conseguido identificar como tales). F1 se calcula promediando las F1 de cada una de las clases (*macro-averaging*) (Yang & Joachims, 2008).

4.4. Resultados y discusión

La Figura 5 muestra los valores de la medida F1 en función del número de ejemplos de entrenamiento disponibles. F1 toma valores entre 0 y 1. Los resultados usando como métrica la precisión—

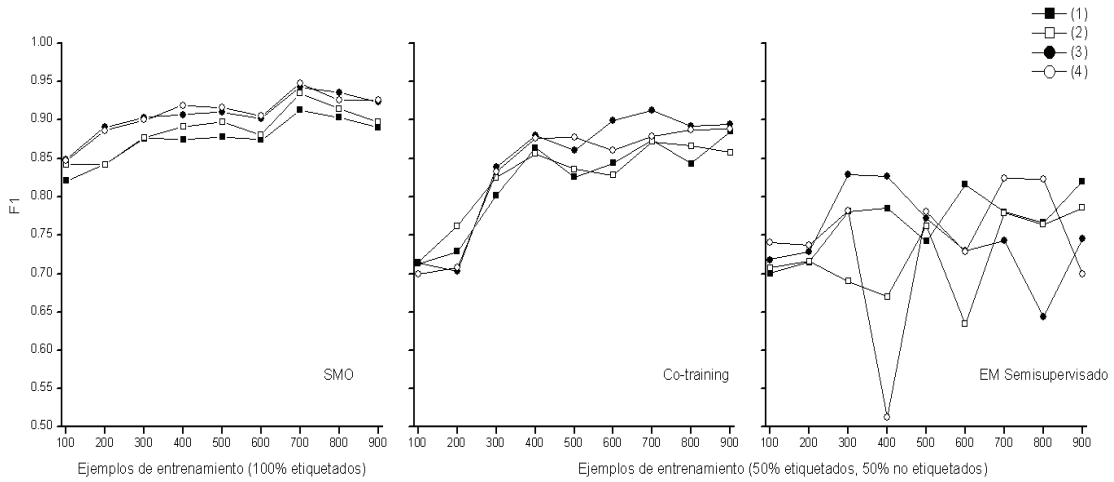


Figura 5: Variación de F1 con el número de ejemplos de entrenamiento según los mecanismos de reducción de la dimensionalidad y de asignación de pesos a los atributos.

accuracy—son muy similares y no se muestran aquí. SMO precisa ejemplos etiquetados; *co-training* y EM, al ser semi-supervisados, pueden aprovechar ejemplos no etiquetados también, por lo que en el experimento la mitad del conjunto de entrenamiento estaba etiquetada y la otra no. Para cada algoritmo modificamos los mecanismos de reducción de la dimensionalidad y de asignación de pesos a los atributos. En los casos (3) y (4) se seleccionaron los 100 atributos más relevantes para cada una de las 21 clases, resultando en un total de entre 614 y 933 atributos (dependiendo del conjunto de entrenamiento); en (1) y (2) los 100 atributos más relevantes para todas las clases. Solamente en los casos (2) y (4) se aplica la transformación tfidf.

En general los resultados son mejores (de 1% a 6% mejores en el caso de SMO y *co-training*) cuando se eligen los atributos más relevantes para cada una de las 21 categorías (líneas 3 y 4). Esta selección lleva a un conjunto de atributos mucho mayor, lo cual aumenta el tiempo de aprendizaje del modelo considerablemente en el caso de *co-training*, aunque no así en el caso de SMO. No obstante, dado que el modelo sólo se construye una vez, éste no es un factor tan importante a la hora de elegir el algoritmo. Por otro lado, la utilización de tfidf (gráficos 2 y 4) supone algo de mejora solamente cuando se utiliza SMO.

La Figura 6 (a) compara los valores de F1 para los tres algoritmos en el caso (3), es decir, seleccionando los 100 atributos para cada una de las 21 clases y sin usar tfidf. Puede verse cómo SMO produce

mejores resultados, especialmente con menos ejemplos de entrenamiento disponibles. Como se indicó, el conjunto de entrenamiento de los algoritmos semisupervisados estaba sólo parcialmente etiquetado. Por lo tanto, en cierto modo SMO tenía a su disposición más información (más ejemplos etiquetados). Por ello además comparamos en la Figura 6 (b) los tres algoritmos solamente frente al número de instancias de entrenamiento etiquetadas. (Los algoritmos semi-supervisados reciben además un número igual de instancias no etiquetadas). En este caso no hay tanta diferencia en el rendimiento de SMO y *co-training*, pero vemos que el uso de los ejemplos no etiquetados no supone una ventaja en este dominio (a diferencia de otros dominios como los explorados en (Pérez y Cardoso, 2009)).

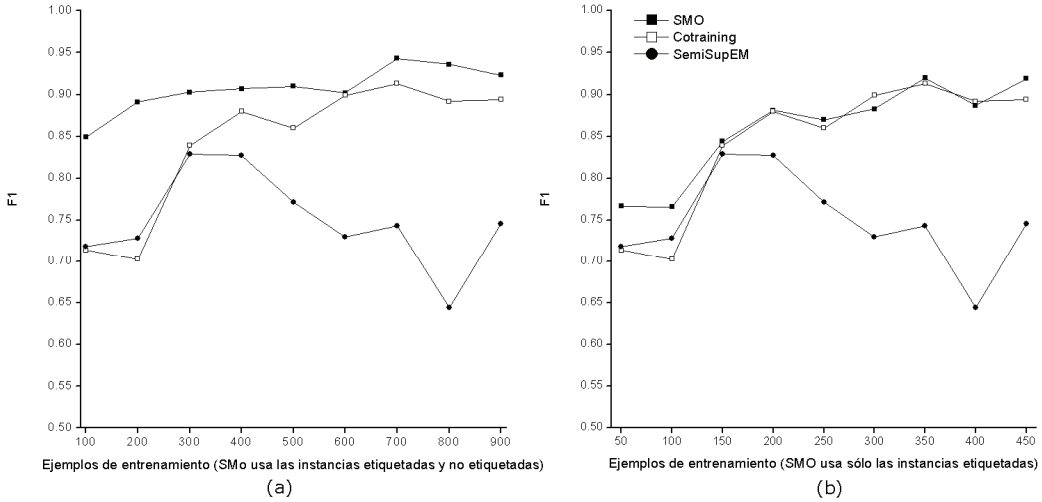


Figura 6: Comparación de los tres algoritmos.

5. Búsqueda semántica

Nuestro objetivo final es construir un buscador semántico (Guha et al, 2003) que utilice los meta-datos obtenidos automáticamente por los anotadores implementados en UIMA y las categorías asignadas automáticamente por los algoritmos de aprendizaje. Para ello hemos experimentado adaptando dos APIs de soltare libre diferentes para implementar motores de búsqueda: SemanticSearch y Lucene. Mientras que la segunda es más ampliamente utilizada para construir motores de búsqueda en general, la primera ha sido especialmente construida para su integración con UIMA, ya que como ésta última fue desarrollada por IBM. Nuestro prototipo inicial ha sido por ello construido sobre SemanticSearch y ofrece una interfaz básica para el buscador semántico. Una segunda implementación con Lucene es la base del

trabajo de un alumno de grado que ha colaborado en nuestro proyecto y está construyendo una interfaz amigable y más completa basada en web para el buscador.

Esta sección comienza exponiendo los fundamentos de la búsqueda semántica y de los motores de búsqueda. A continuación, en la sección 5.3 se describe el lenguaje *XML Fragments* que utilizamos para realizar consultas. Las secciones 5.4 y 5.5 describen SemanticSearch y el prototipo que hemos construido utilizando esta herramienta. Las secciones 5.6 y 5.7 describen Lucene y las pruebas que hemos hecho como base de un motor de búsqueda basado en esta herramienta.

5.1. Búsqueda semántica

Los motores de búsqueda tradicionales indexan las palabras que aparecen en los documentos y procesan consultas que son combinaciones booleanas de palabras. Después devuelven una lista de documentos que contienen esa combinación de palabras ordenada según diversos parámetros.

Por otro lado, muchas aplicaciones UIM pueden beneficiarse de más inteligencia en esta búsqueda, con consultas que buscan documentos no solamente basándose en las palabras que aparecen en el documento, sino en el contexto en que estas aparecen y en conceptos derivados de la interpretación del texto por los anotadores y que han sido asociados al documento en forma de anotaciones. El motor de búsqueda semántica es capaz de construir su índice no sólo con las palabras del texto sino también con las anotaciones. Obviamente la interfaz del motor de búsqueda debe permitir al usuario elaborar consultas que utilicen esas anotaciones.

La Figura 7, extraída de (Hampp & Lang, 2005), ubica la búsqueda semántica en el contexto de los paradigmas de búsqueda. Cada una de las tecnologías está construida sobre las que aparecen bajo ella. La búsqueda semántica se apoya en los elementos de la búsqueda tradicional (*keywords*) y sus operadores (+ y -, AND, OR, etc) y añade la facilidad para buscar conceptos o entidades y las relaciones entre ellos. Para ello introduce nuevos elementos, como los lenguajes *XML Fragments* (ver sección siguiente) o *XPath* para expresar las consultas. Las aplicaciones suelen ocultar esta sintaxis de los usuarios. La búsqueda en lenguaje natural trata de encontrar documentos en respuesta a una pregunta directa del usuario final. Así se evita la complejidad de la consulta de la búsqueda semántica, pero puede

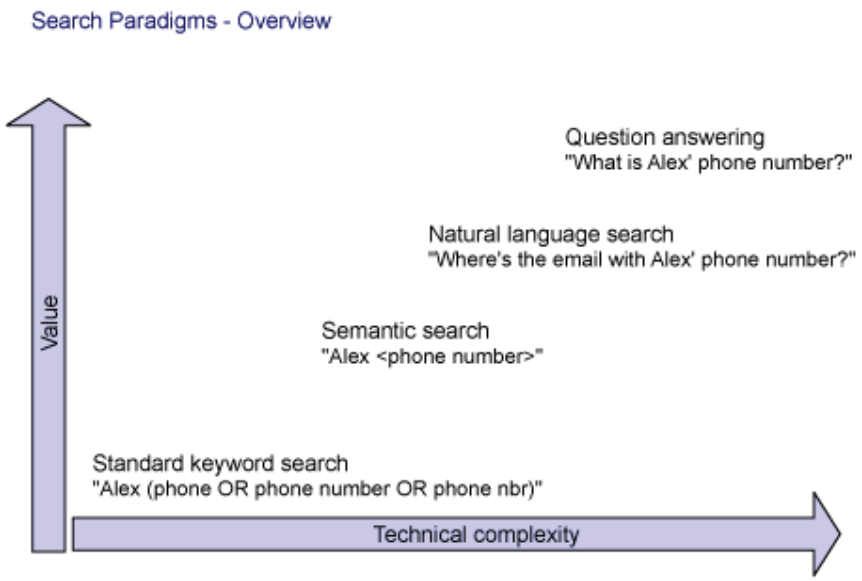


Figura 7: Comparación de paradigmas de búsqueda (Hampp & Lang, 2005).

interpretarse incorrectamente la pregunta del usuario. Suele consistir en el análisis textual de la consulta, presentada en lenguaje natural por el usuario, y su traducción automática en una consulta semántica en uno de los lenguajes mencionados. El resultado es una colección de documentos relevantes a la consulta. Finalmente, la respuesta a preguntas busca devolver la parte de un documento que incluye la respuesta. Puede también consolidar hechos procedentes de diversos documentos en una sola respuesta. Suele construirse basándose en la búsqueda en lenguaje natural, que devuelve un conjunto de documentos relevantes, con un postprocesamiento para extraer la respuesta precisa de esos documentos.

5.2. Fundamentos de un motor de búsqueda

Un motor de búsqueda almacena la información sobre una colección de documentos o páginas web que ha visitado previamente. En el caso de la web estas páginas son previamente obtenidas por un *Web crawler* (araña o gusano), navegador automatizado que va siguiendo todos los vínculos que encuentra en las páginas visitadas.

Si el motor de búsqueda permite diversos formatos, deben prepararse primero los documentos para la tokenización. Muchos documentos contienen información de formato además del contenido textual. Si esta información, por ejemplo etiquetas HTML, fuera incluida

en el índice, podría producir resultados espurios. A veces se denomina análisis de formato a la identificación y manejo del contenido de formato que está incluido en un documento de texto y que controla la forma en que el documento es presentado en una pantalla o navegador o interpretado por una aplicación (ej. un procesador de texto). Otros nombres de estas tareas son análisis de la estructura, eliminación de etiquetas, normalización del texto, limpieza de texto, o preparación del texto. Algunos formatos son propietarios y hay muy poca información disponible al respecto (ej. los utilizados por Microsoft Word o Excel), mientras que otros están ampliamente documentados.

Posteriormente los contenidos textuales de cada documento o página son analizados para determinar cómo indexarlos. Por ejemplo, podrían extraerse información especial de títulos, nombres de archivo, o campos especiales llamados meta-etiquetas, además del cuerpo del documento. Esta información se almacena en una base de datos o índice para ser utilizada por las consultas. El objetivo es recuperar la información relevante lo más eficientemente posible. Sin el índice, el motor de búsqueda tendría que escanear cada documento del corpus en cada consulta, lo que requeriría considerable tiempo y recursos. Por ejemplo, mientras que se puede consultar un índice de 10.000 documentos en milisegundos, recorrer cada palabra de 10.000 archivos llevaría minutos y hasta horas. Algunos motores de búsqueda almacenan en el índice todo o parte del documento.

Cuando un usuario realiza una consulta, el motor examina el índice y devuelve una lista de los documentos o páginas que mejor la satisfacen. Frecuentemente se devuelve también un breve resumen del documento con el título y partes del texto, resaltando las que responden a la consulta. La mayoría de los motores permiten operadores booleanos (AND, OR y NOT) en la consulta

5.3. El lenguaje XML Fragments

Para construir las consultas hemos adoptado el lenguaje de consultas *XML Fragments* (Chu-Carroll et al, 2006; Carmel at al, 2003) debido a su expresividad y a la disponibilidad de un motor de búsquedas como parte del componente *SemanticSearch* de la arquitectura UIMA (Hampp & Lang, 2005). La interfaz de búsqueda de *SemanticSearch* permite los operadores estándar de búsqueda, tales como búsqueda de texto libre y los operadores AND, OR, NOT y comodines para combinar las consultas.

Una consulta en el lenguaje *XML Fragments* consta de una estructura XML sub-especificada que combina consultas de palabras con consultas de información anotada. Esto permite buscar conceptos más específicos, e incluso relaciones entre objetos (por ejemplo, "la persona y la unidad académica deben aparecer en la misma frase" o "una persona que pertenece a cierta unidad académica" o "una unidad académica vinculada a una institución" porque aparecen cerca en el documento).

Algunos ejemplos sencillos de consultas serían:

- *ingenieros*: devuelve cualquier resolución que contenga la palabra "ingenieros"

- *<Institución/>*
  devuelve cualquier resolución que contenga una anotación de Unidad Académica

- *<Institución> Ingenieros </Institución>*
  devuelve cualquier resolución que contenga una anotación de Institución que contenga la palabra *ingenieros* (efectivamente encontrando las referencias a COPAIPA, el Consejo Profesional de Ingenieros, Agrimensores y Profesiones Afines)

- *<Institución> Ingenieros </Institución> pasantía*
  devuelve los documentos que respondan a la consulta anterior que además contengan la palabra *pasantía*

- *<UnidadAcadémica>ingen </UnidadAcadémica> <FechaResol año=2007> <FechaResol>*
  devuelve todas las resoluciones de la Facultad de Ingeniería e Informática del año 2007

- *<Institución> Ingenieros </Institución> pasantía <UnidadAcadémica>ingeniería </UnidadAcadémica>*
  resoluciones en que aparezcan la Facultad de Ingeniería e Informática, COPAIPA y la palabra *pasantía.*

- *<UnidadAcadémica> ingeniería </UnidadAcadémica> <clase categ=ConvenioPasantía </clase>*
  resoluciones en que aparece la Facultad de Ingeniería e Informática que han sido etiquetadas como convenios de pasantía por el clasificador aprendido.

Estos ejemplos se corresponden con las operaciones que se pueden realizar con *XML Fragments* definidas por Chu-Carroll et al (2006):

- conceptualización, que generaliza un literal (string) a un concepto apropiado del sistema de tipos. Por ejemplo, la consulta "*institución*" devuelve documentos en que aparezca esa palabra, mientras que <*Institución/*> busca ocurrencias de la anotación institución, aunque la palabra "*institución*" misma no aparezca en el documento.

- restricción: restringe las ocurrencias de etiquetas XML indicando qué palabras deben aparecer. Por ejemplo <*Institución*> *ingenier* <*/Institución*> devuelve ocurrencias del Consejo Profesional mientras que <*UnidadAcadémica*> *ingenier* <*/UnidadAcadémica*> devuelve documentos que se refieren a la Facultad de Ingeniería.

- relación: una anotación representa la relación entre los términos que aparecen en la consulta. Por ejemplo <*FechaResol*>*2007 Septiembre* <*FechaResol*> encuentra las resoluciones de septiembre del 2007 (y no simple resoluciones de 2007 en que aparezca la palabra septiembre).

Así *XML Fragments* permite aumentar la expresividad de las consultas de los usuarios y obtener resultados más focalizados en la búsqueda. Obviamente no se espera que el usuario final exprese sus consultas directamente en este lenguaje, sino proporcionarle una interfaz que le asista a formular la parte estructurada de las consultas, por ejemplo mediante formularios HTML donde las entradas están asociadas a elementos predefinidos correspondientes a las anotaciones, complementada con la búsqueda tradicional de *keywords*.

5.4. *SemanticSearch*

El paquete *SemanticSearch* 2.1 (http://www.alphaworks.ibm.com/tech/uima) añade a UIMA un motor de búsqueda semántica. Incluye un CAS Consumer que llena un índice con el contenido del documento así como las anotaciones generadas añadidas por los anotadores implementados en UIMA. Para consultar el índice puede usarse el lenguaje *XML Fragments* descrito en la sección anterior. Este paquete es software libre aportado por IBM y para acceder al índice existe una API para ser utilizada en una aplicación en Java.

Para construir un índice utilizando UIMA y *SemanticSearch* primero hay que correr un CPE (Collection Processing Engine) que incluye los anotadores y un CAS Consumer que toma los tokens y anotaciones, y además las oraciones del texto. Este CAS Consumer está disponible con la distribución de SemanticSearch y se llama SemanticSearchCasIndexer. Entre los anotadores debe haber uno que produce las anotaciones de tokens y oraciones.

```
<indexBuildSpecification>
  <indexBuildItem>
    <name>org.apache.uima.examples.tokenizer.Token</name>
    <indexRule>
      <style name="Term"/>
    </indexRule>
  </indexBuildItem>
  <indexBuildItem>
    <name>org.apache.uima.examples.tokenizer.Sentence</name>
    <indexRule>
      <style name="Breaking"/>
    </indexRule>
  </indexBuildItem>
  <indexBuildItem>
    <name>org.apache.uima.mio.Dni</name>
    <indexRule>
      <style name="Annotation">
            <attributeMappings>
                  <mapping>
                        <feature>dniNum</feature>
                        <indexName>dniNum</indexName>
                  </mapping>
            </attributeMappings>
        </style>
      </indexRule>
  </indexBuildItem>
……
 </indexBuildSpecification>
```

Figura 8: Ejemplo parcial del archivo de configuración de SemanticSearch.

Es necesario proporcionar un archivo de configuración (Index Build Specification) que indica cómo van a usarse las anotaciones para construir el índice. La Figura 8 muestra parte de este archivo para nuestra aplicación. Este archivo está formado por una colección de elementos (IndexBuildItems). Cada uno se refiere a un tipo de anotación y a un estilo. El primer item del ejemplo especifica que el tipo de anotación token debería ser indexado con el estilo Term. Esto significa que cada anotación token será considerada un token individual a la hora de la búsqueda. En el caso de las anotaciones relativas a nuestro dominio, todos los ítems serán del tipo *annotation*, que indica

que cada anotación será almacenada en el índice como un segmento de texto con nombre igual al nombre de la anotación y abierto a que se pueda buscar dentro de ese segmento. Además puede indicarse un campo (*feature*) específico de la anotación como el que se va a indexar.

Tras el procesamiento de la colección, el indexador construye el índice que puede ser consultado con tokens o con etiquetas XML según el lenguaje *XML Fragments*. Para más detalles sobre SemanticSearch y su integración con UIMA puede consultarse (Apache UIMA Development Community, 2008a, 2008b, 2008c; IBM, 2006).

5.5. Implementación del buscador semántico utilizando *SemanticSearch*

Nuestra primera aplicación del buscador semántico se ha desarrollado en el lenguaje de programación Java. Sus principales procesos son clasificar documentos, crear el índice para exploración de documentos y realizar búsqueda de resoluciones a partir de anotaciones y/o palabras. La Figura 9 muestra la sencilla interfaz del prototipo. El menú “Opciones” cuenta con los comandos “Crear Indice” y “Salir”. El primero recorre todos los documentos de la colección, obtiene las anotaciones y construye el índice de búsqueda. Cada vez que se incorporen nuevas resoluciones, el índice debe ser construido de nuevo.

El anotador UIMA utilizado, AnotadoresMasClase, es un anotador agregado que incluye los anotadores que reconocen personas, unidades académicas, carreras, instituciones, número y fecha de resolución y un nuevo anotador que utiliza el modelo generado en la etapa de entrenamiento y que determina a que categoría pertenece el documento que se analiza. Incluye también un anotador estándar de UIMA que produce palabras (tokens) y oraciones (*SimpleTokenAndSentenceAnnotator*).

La aplicación realiza también la búsqueda de resoluciones mediante consultas que pueden combinar palabras (*keywords*) y anotaciones. Para ello pueden utilizarse tres listas desplegables, donde aparecen los criterios de búsqueda y tres cuadros para introducir el texto a buscar. Las posibles formas de realizar consultas son:

- Con todas las palabras introducidas en el cuadro correspondiente
- Con algunas palabras

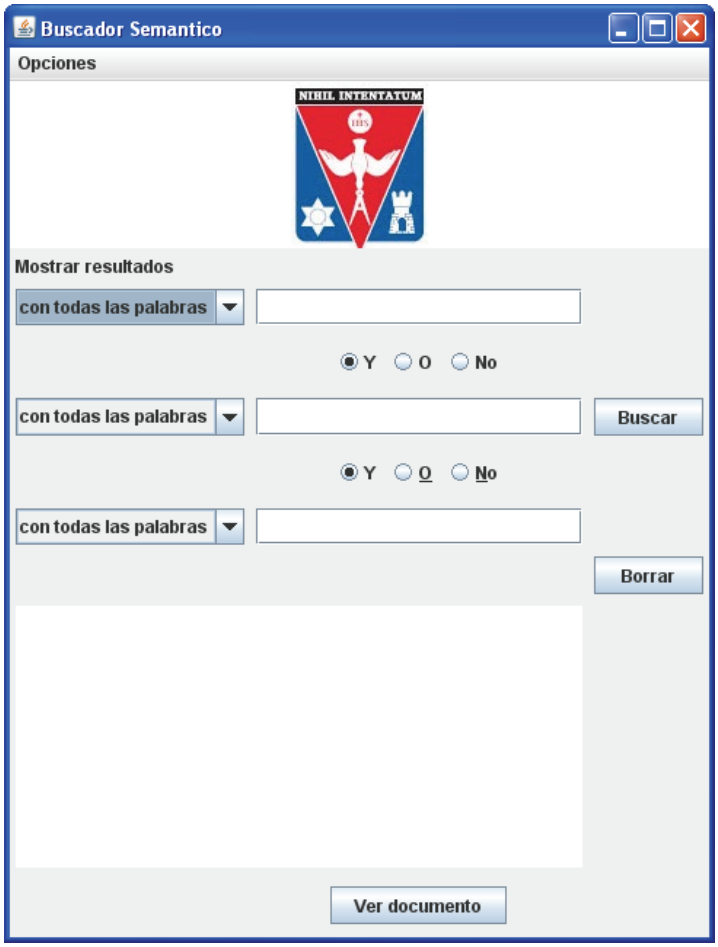


Figura 9: Interfaz del prototipo del buscador semántico.

- Con una frase exacta
- Sin determinadas palabras
- Sin determinadas palabras
- Por Nombre, Apellido, Institución, Unidad Académica, DNI, Título, Año de Resolución, Número de Resolución o Categoría de Resolución.

En el caso de que se realice búsqueda por tipo de resolución (Clase) se despliega otra lista con las categorías disponibles. Además los botones de opciones entre los cuadros de textos permiten realizar consultas más avanzadas, mediante operadores que combinan las anteriores. La Figura 10 muestra algunos ejemplos de consultas con los resultados correspondientes. Las consultas generadas por la interfaz son traducidas al lenguaje *XML Fragments*.

Para buscar documentos que simplemente contengan un cierto tipo de anotación, basta colocar "." en el cuadro correspondiente. Por ejemplo, si se quiere buscar resoluciones de cualquier Unidad Académica, se elige esta opción y en el cuadro correspondiente se escribe "."

La búsqueda se activa presionando el botón "Buscar" y los nombres de los documentos encontrados que cumplan con las combinaciones se mostrarán en el panel inferior. El botón "Ver documento" abrirá el documento seleccionado de esta lista. Las Figuras 11, 12 y 13 muestran ejemplos de consultas con el resultado obtenido y uno de los documentos propuestos abierto.

5.6. El proyecto Lucene

Apache Lucene es una librería escrita en Java para implementar motores de búsqueda en texto de alto rendimiento y con una gran gama de opciones. Se trata por tanto de una alternativa a SemanticSearch. Está siendo utilizada en múltiples proyectos y aplicaciones que precisan búsqueda en texto completo, especialmente portable a múltiples plataformas (Paul, 2004; Gospodnetić, 2004). Apache Lucene es un proyecto de software libre y abierto bajo la Licencia Apache que permite su uso en programas tanto abiertos como comerciales.

El centro de la arquitectura lógica de Lucene se encuentra el concepto de Documento (*Document*) que contiene Campos (*Fields*) de texto. Cada campo representa información sobre el texto que se desea indexar, y puede también incluir metadatos sobre el documento, tales como su autor, o la fecha de creación, aunque no sean campos en los que se va a realizar la búsqueda.

Esta flexibilidad permite a Lucene ser independiente del formato del archivo. Por ello cualquier archivo de texto (PDF, HTML, documentos de Microsoft Word, etc) puede ser indexado siempre que se pueda extraer la información textual del mismo.

Antes de la indexación, un campo de texto se convierte en la representación final del índice que es una colección de términos. Para ello primero se divide el texto en instancias de tokens de Lucene, que pueden después ser modificados mediante una serie de filtros (por ejemplo, para eliminar *stopwords).*

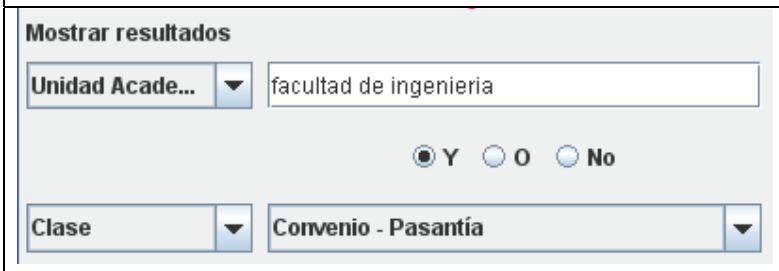
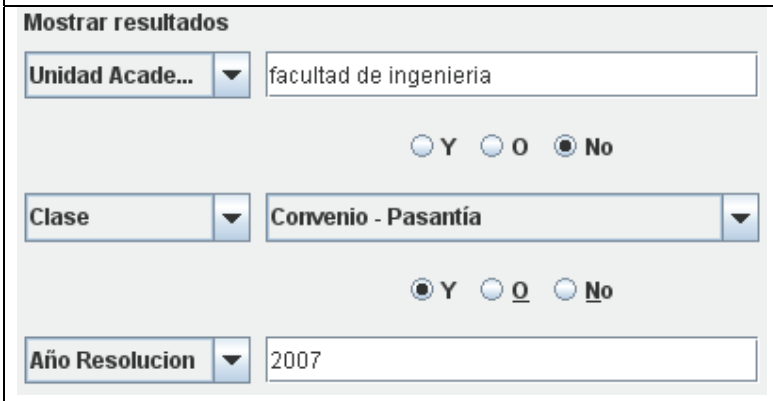
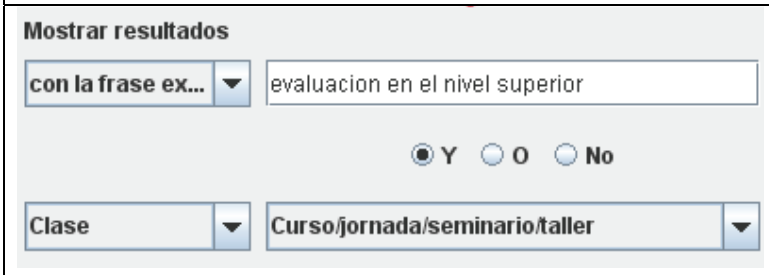
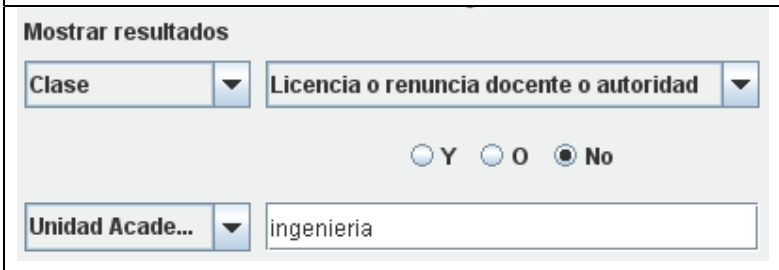
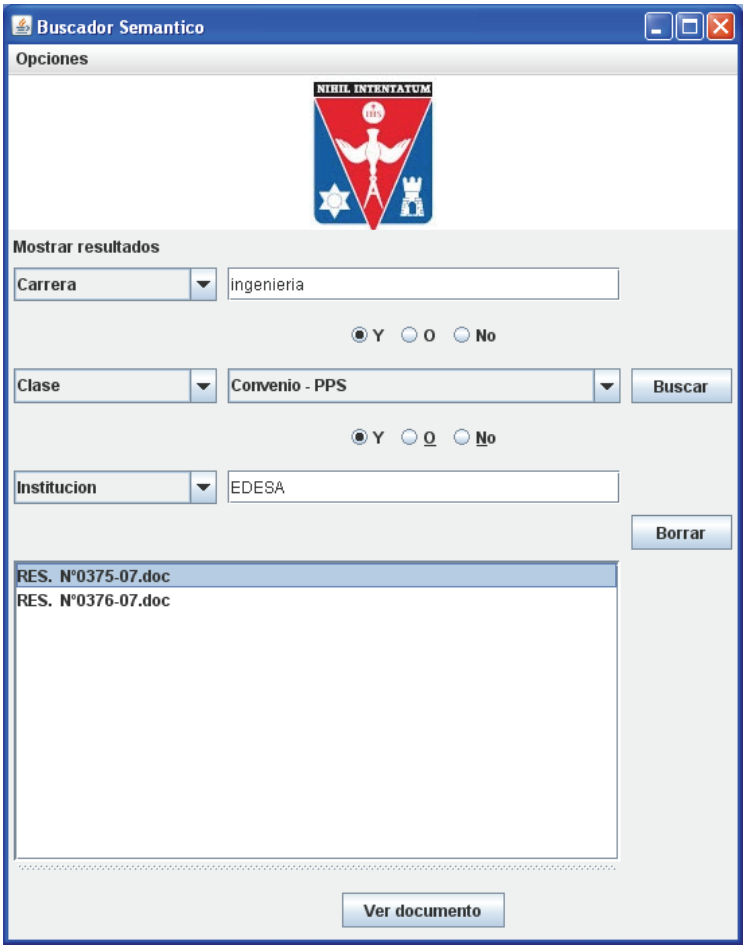
 *<Institucion>Arzobispado +de +salta</Institucion>*	Devuelve resoluciones que tengan como institución al Arzobispado de Salta
 *+<UA>facultad de ingenieria</UA>* *+<Clase valor="ConvPasant"></Clase>*	Devuelve resoluciones de la Unidad Académica Facultad de Ingeniería e Informática y que traten sobre convenios o pasantías
 *+<UA>facultad de ingenieria</UA>* *+<Clase valor="ConvPasant"></Clase>* *+<FechaResol anio="2007"></FechaResol>*	Devuelve resoluciones de la Facultad de Ingeniería que no traten sobre convenios ni pasantías pero que sean del año 2007
 *+"evaluacion en el nivel superior"* *+<Clase valor="Curso"></Clase>*	Devuelve resoluciones que contengan la frase exacta "evaluación en el nivel superior" y se refieran a un curso, jornada, seminario o taller
 *+<Clase valor="LicenRenunc"></Clase> +<UA>ingenieria</UA>*	Devuelve resoluciones sobre licencias o renuncias de docentes o autoridades que no sean de la Facultad de Ingeniería e Informática

Figura 10: Ejemplos de consultas en el prototipo.



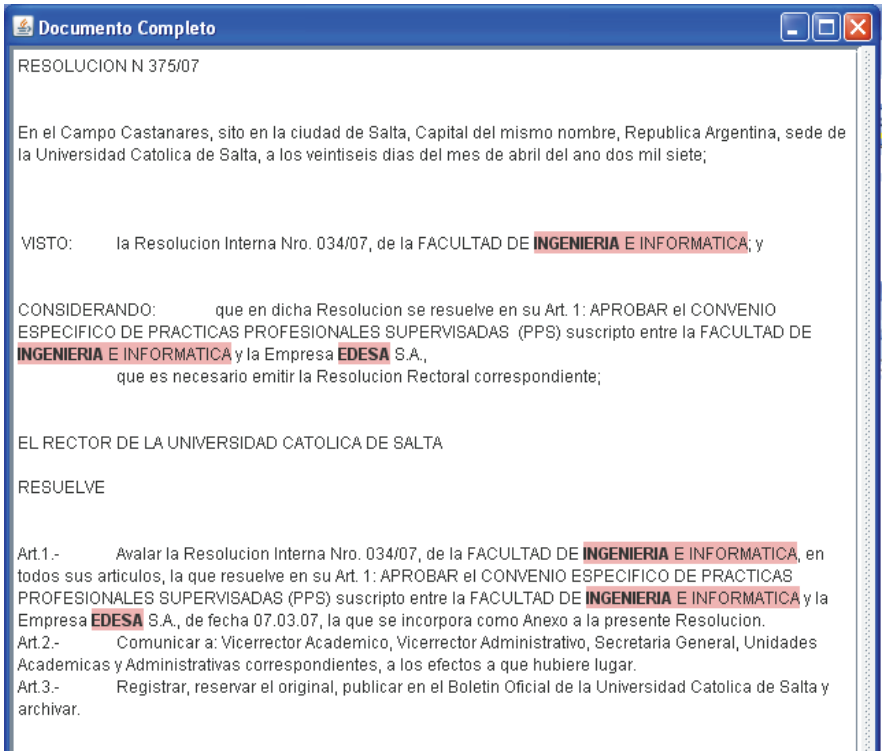
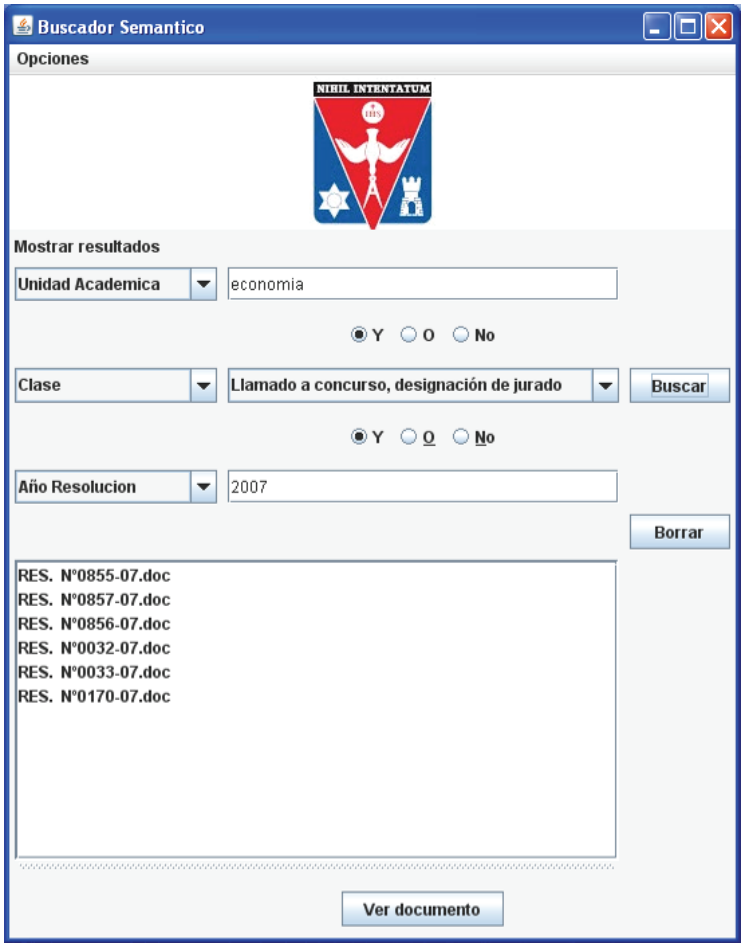


Figura 11. Ejemplo del resultado de la búsqueda de documentos.



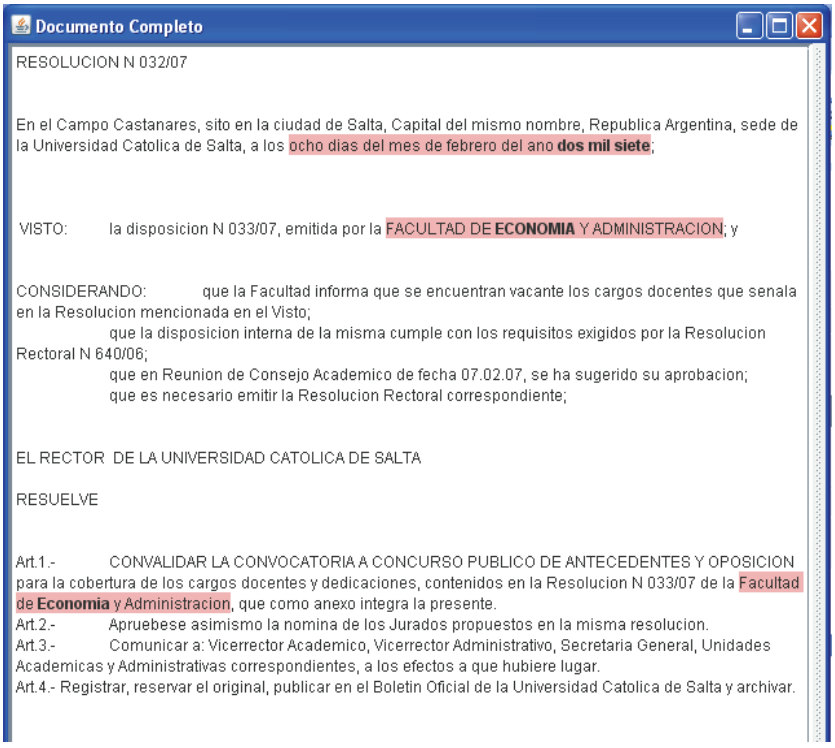


Figura 12. Ejemplo del resultado de la búsqueda de documentos.

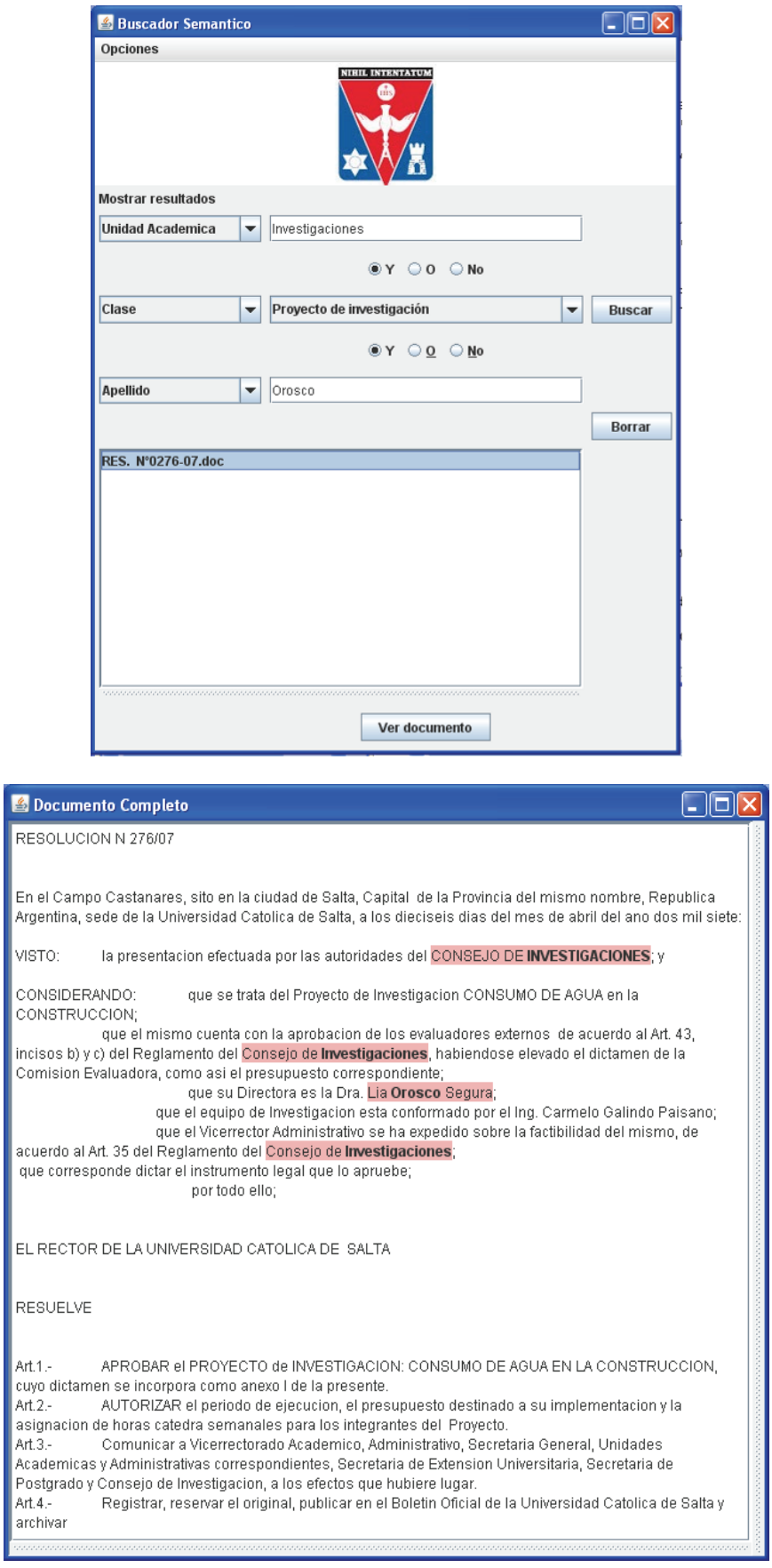


Figura 13. Ejemplo del resultado de la búsqueda de documentos.

5.7. Implementación utilizando Lucene

LuCas (Fäßler et al, 2009) es un componente para UIMA del tipo CAS Consumer que sirve de puente entre UIMA y la librería Lucene para construir motores de búsqueda. LuCas almacena los datos procedentes del CAS (anotaciones) en un índice de Lucene, para que después puedan ser recuperados de forma muy eficiente. LuCas ha sido recientemente incorporado a la sandbox de UIMA (http://incubator.apache.org/uima/ sandbox.html) y es software libre bajo la licencia Apache Software License.

LuCas va llenando automáticamente un índice de Lucene convirtiendo las anotaciones de UIMA en instancias de "documentos" de Lucene según un archivo de configuración que define el mapeo. Esto permite que Lucene recupere documentos no sólo en base a la información contenida en el texto sino también en base a los metadatos añadidos durante el procesamiento de UIMA en forma de anotaciones. La Figura 14 muestra una parte del archivo de configuración de LuCas. En este ejemplo, se crea una campo en el índice de Lucene llamado "institución" para cada anotación UIMA del tipo "org.apache.uima.mio.institucion". En el segundo ejemplo, la entrada en el índice de Lucene es a partir de uno de los campos (o *features*) "apellido" de la anotación de UIMA del tipo "org.apache.uima.mio.persona". la definición del tercer campo permite la indexación de todas las palabras del documento. Puede consultarse (Fäßler et al, 2009) para más información sobre LuCas.

La Figura 15 muestra una consulta al índice generado con Lucene. Utiliza una interfaz disponible como componente denominada Luke, pensada para la etapa de desarrollo con el fin de verificar el correcto funcionamiento del índice y las consultas. Estamos desarrollando una interfaz más amigable y robusta para el buscador semántico que esté basada en web. Esta interfaz está siendo construida sobre el motor construido con Lucene.

6. Conclusiones

La minería de textos, y la categorización de documentos en particular, son un campo de investigación y aplicación prometedor dado que más y más las organizaciones están interesadas en aprovechar el gran cuerpo de conocimiento no estructurado de que disponen. Las técnicas de recuperación de la información y de aprendizaje automático, combinadas con la potencia de la arquitectura UIMA y su reuso de

```
<field name="institucion" index="yes">
      <filter name="lowercase" />
      <annotations>
            <annotation sofa="_InitialView"
                        type="org.apache.uima.mio.Institucion"
                        tokenizer="standard" />
      </annotations>
</field>

<field name="apellido" index="yes">
      <filter name="lowercase" />
      <annotations>
            <annotation sofa="_InitialView"
                        type="org.apache.uima.mio.Persona"
                        tokenizer="standard" />
                  <features>
                        <feature name="apellido" />
                  </features>
      </annotations>
</field>

<field name="content" index="yes" stored="no">
      <filter name="lowercase" />
      <filter name="stopwords" filePath="stopwords.txt"
              ignoreCase="true" />
      <annotations>
        <annotation sofa="_InitialView"
                    type="org.apache.uima.TokenAnnotation"
                    tokenizer="standard" />
      </annotations>
</field>
```

Figura 14: Ejemplo parcial del archivo de configuración de LuCas.

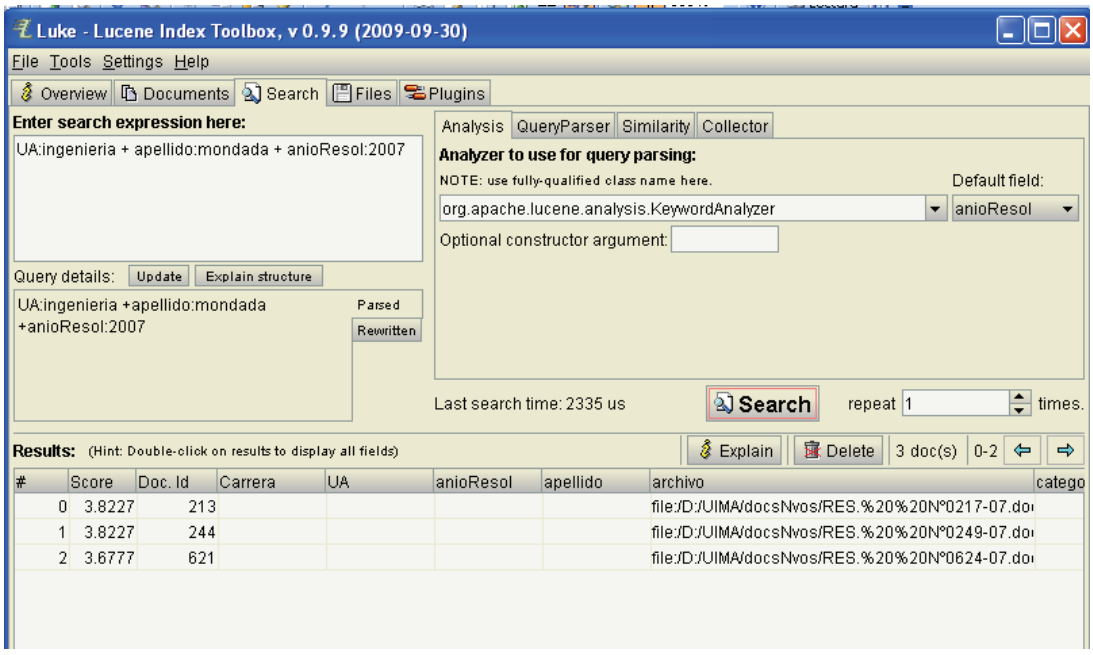


Figura 15. Resultado de una búsqueda con Lucene utilizando la interfaz Luke.

componentes, facilitan la tarea de extracción de conocimiento. En particular, hemos investigado e implementado anotadores que extraen del texto entidades, relaciones entre ellas, y la información necesaria para asignar automáticamente categorías a documentos de texto en base a un corpus relativamente pequeño de ejemplos.

El enfoque utilizado para la extracción de entidades, en particular instituciones y carreras, funciona relativamente bien, pero su calidad depende de la calidad del diccionario y de la precisión en la equiparación de las ocurrencias del texto con las del diccionario. A menudo una misma entidad, por ejemplo el nombre de una carrera, aparece de muy diversas formas en el texto. El diccionario debe incluirlas todas o variaciones suficientemente cercanas. Una clara posibilidad de trabajo futuro es flexibilizar este enfoque, posiblemente aplicando algoritmos de aprendizaje automático para detectar estas entidades.

La clasificación automática de documentos utilizando el modelo aprendido es de bastante calidad comparada con las etiquetas asignadas manualmente. En base a los experimentos realizados, el algoritmo semi-supervisado *cotraining* tiene un rendimiento bastante bueno en cuanto a los resultados de la clasificación y requiere menos ejemplos etiquetados para aprender al poder aprovechar los ejemplos no etiquetados. No obstante los modelos aprendidos por el algoritmo SMO (máquinas de vectores soporte) son muy buenos para este problema sin necesidad de tener muchos ejemplos etiquetados de entrenamiento. Por tanto lo hemos elegido para la implementación en este problema.

Hemos desarrollando un prototipo que integra todas las anotaciones poniéndolas a disposición de un buscador semántico y así en última instancia de un usuario que pueda efectuar consultas y visualizar los documentos resultantes de las mismas mediante una interfaz apropiada. El prototipo utiliza la API SemanticSearch y presenta una interfaz básica suficiente como prueba de concepto. Se está construyendo un sistema más robusto y una interfaz más amigable utilizando la API Lucene en un proyecto de grado asociado a este trabajo de investigación.

El conocimiento y la experiencia obtenida de este proyecto son base para futuras implementaciones que pueden hacer extensivas estas técnicas a otros problemas de clasificación e interpretación de textos, y en una perspectiva más amplia a otros problemas de gestión del conocimiento.

Agradecimientos

Este trabajo ha sido financiado en parte por la Convocatoria 2007 del Consejo de Investigaciones de la Universidad Católica de Salta (Resolución Rectoral 723/08). Las autoras agradecen la colaboración de la Secretaria General de la UCASAL, Prof Constanza Diedrich por su asesoramiento sobre la taxonomía de resoluciones rectorales. También agradecen la colaboración de los alumnos David Zamar e Iván Ramos en diversas etapas del trabajo

Bibliografía

Alias-I, 2008, LingPipe 3.8.2. http://alias-i.com/lingpipe (acceso 1 Septiembre 2009).

Apache UIMA Development Community, 2008, *UIMA Tutorial and Developers' Guides*, version 2.2, http://incubator.apache.org/uima/downloads/releaseDocs/2.2.2-incubating/docs/html/

Apache UIMA Development Community, 2008, *UIMA Tools Guide and Reference*, version 2.2, http://incubator.apache.org/uima/downloads/releaseDocs/2.2.2-incubating/docs/html/

Apache UIMA Development Community, 2008, *UIMA References*, version 2.2, http://incubator.apache.org/uima/downloads/releaseDocs/2.2.2-incubating/docs/html/

Blum, A. & Mitchell, T., 1998, Combining labeled and unlabeled data with co-training. En *Proceedings of the Eleventh Annual Conference on Computational Learning theory, COLT' 98*. ACM, New York, NY, 92-100.

Carmel, D., Maarek, Y. S., Mandelbrod, M., Mass, Y., & Soffer, A., 2003, Searching XML documents via XML fragments. En *Proceedings of the 26th Annual international ACM SIGIR Conference on Research and Development in informaion Retrieval* (Toronto, Canada, July 28 - August 01, 2003). SIGIR '03. ACM, New York, NY, 151-158. DOI= http://doi.acm.org/10.1145/860435.860464.

Chu-Carroll, J., Prager, J., Czuba, K., Ferrucci, D., y Duboue, P., 2006, Semantic search via XML fragments: a high-precision approach to IR. En *Proceedings of the 29th Annual international ACM SIGIR Conference on Research and Development in information Retrieval*, SIGIR '06. ACM, New York, NY, 445-452.

Fäßler, E., Landefeld, R., Tomanek, K. & Hahn, U., 2009, LuCas - A Lucene CAS Indexer. En Proceedings of the 2nd UIMA@GSCL Workshop (German Society for Computational Linguistics and Language Technology)

Feldman, R. & Sanger, J., 2007, *The Text Mining Handbook: advanced approaches in analyzing unstructured data*, Cambridge University Press, New York.

Ferrucci, D. & Lally, A. 2004. Building an example application with the unstructured information management architecture. *IBM Systems Journal* 43, 3, 455-475.

Gospodnetić, O. & Hatcher, E., 2004, *Lucene in Action*, Manning, Greenwich, CT

Hampp, T. & Lang, A., 2005, Semantic search in WebSphere Information Integrator OmniFind edition: The case for semantic search. *IBM Developer Works*

IBM, 2006, *UIMA SDK Semantic Search Engine: Search and Index API (SIAPI)*

Moore, C., 2002, Diving into data, *Infoworld*, 25 de octubre, http://www.infoworld.com/article/02/10/25/021028feundata_1.html

Nigam, K. & Ghani, R., 2000, Analyzing the effectiveness and applicability of co-training. En *Proceedings of the Ninth international Conference on information and Knowledge Management. CIKM '00.* ACM, New York, 86-93.

Nigam, K., McCallum, A. K., Thrun, S., & Mitchell, T., 2000, Text classification from labeled and unlabeled documents using EM. *Machine Learning* 39 (2-3) 103-134.

OMG, 2007, XML Metadata Interchange (XMI), v 2.1.1

Paul, P., 2004, The Lucene Search Engine - Adding search to your applications, *JavaRanch Journal* April 2004. http://www.javaranch.com/journal/index.jsp?p=2004

Pérez, M. A. & Cardoso, A. C., 2009, Comparación de algoritmos de aprendizaje semi-supervisado. En *V Jornadas de Ciencia y Tecnología de Facultades de Ingeniería del NOA*, Salta, Septiembre 2009.

Sebastiani, F., 2005, Text categorization. En A. Zanasi (ed.) *Text Mining and its Applications to Intelligence, CRM and Knowledge Management*, WIT Press, Southampton, UK, 109-129.

Tomanek, K & Wermter, J., 2008, JULIE Lab Lingpipe Gazetteer Annotator Version 2.1. Jena University Language & Information Engineering (JULIE) Lab. Disponible en www.julieblab.de

Yang, Y. & Joachims, T., 2008, Text categorization, *Scholarpedia*, 3(5):4242

Witten, I. & Frank, E., 2005, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed., Morgan Kaufmann.