Herramientas para la investigación en Recuperación de Información: KARPANTA, un motor de búsqueda experimental / Tools for research on Information Retrieval: KARPANTA, an experimental search engine

Carlos G. Figuerola, José Luis Alonso Berrocal, Ángel Francisco Zazo Rodríguez, Emilio Rodríguez Vázquez de Aldana

Universidad de Salamanca, Grupo REINA (http://reina.usal.es)
e-mail: figue | berrocal | alzazo | aldana@usal.es

Resumen
La investigación en Recuperación de la Información es un área que conoce en la actualidad un desarrollo sin precedentes. Uno de sus principales atractivos reside en su carácter esencialmente multidisciplinar, participando de muy diversos ámbitos del conocimiento: Ciencias de la Documentación, Informática, Matemáticas, Lingüística y otros. Sin embargo, la investigación experimental requiere disponer de utilidades y herramientas que no siempre están al alcance de los investigadores. Se presenta KARPANTA, un motor de recuperación extremadamente flexible, que implementa un gran número de algoritmos diferentes (más de 300), y que aísla el proceso de indización automática y resolución de consultas de las fases de análisis léxico y visualización. El código es extremadamente simple y fácilmente modificable, dado que resuelve la totalidad de las operaciones mediante sencillas sentencias SQL, almacenando los datos en tablas relacionales. KARPANTA es un paquete de código libre y abierto (licencia GPL) que puede ser utilizado, modificado y adaptado libremente por cualquier investigador. De otro lado, KARPANTA puede ser también usado con éxito operacionalmente, en entornos reales y para tareas reales como las que puedan darse en un Centro de Documentación.

Abstract
Research in Information Retrieval is field that knows a development without precedents. One of its main attractiveness is his multidisciplinary character participating in very diverse scopes of theknowledge: Information Science, Computer science,Mathematics, Linguistics and others. Nevertheless, the experimental research requires utilities and tools that not always are within reach of the researchers. We introduce KARPANTA, a search engine that implements a great number of different algorithms (more than 300), and that isolates the process of automatic indexing and resolving queries of the phases of lexical analysis and visualization.The code is very simple and easily modifiable, since it solves the totality of the operations by means of simple SQL sentences, storing the data in relational tables. KARPANTA is free and open code (GPL license) that can be used, freely modified and adapted by any researcher. Of another side, KARPANTA also can be successfully used operationally for real tasks like which they can occur in a Documentation Center.

1. Introducción

La Recuperación de la Información, aunque no es precisamente un área de investigación reciente, experimenta en los últimos tiempos un auge notable, debido a la disponibilidad cada vez mayor de documentos en formato electrónico. El desarrollo y generalización del uso de Internet ha puesto de manifiesto las carencias y los retos en este campo, de manera que son numerosos los grupos de investigadores que dirigen sus esfuerzos hacia estas materias. Uno de los campos de investigación en RI es la experimentación con diversos algoritmos, referentes a cualquiera de las fases o tareas que pueden darse en el proceso de recuperación. La investigación experimental en este campo, sin embargo, requiere, además de los conocimientos básicos necesarios, de una serie de herramientas o instrumentos que permitan la realización de experimentos. Entre tales instrumentos, podemos distinguir, a grandes rasgos los siguientes:

- Colecciones de documentos adecuadas, tanto por sus características documentales, como lingüísticas, e incluso de tamaño. Estas colecciones no sólo incluyen documentos, sino también baterías de preguntas o consultas, así como las correspondientes estimaciones de relevancia para las mismas.
- Programas que permitan indizar los documentos y resolver las consultas
- Medidas eficaces y aceptadas ampliamente por la comunidad científica, que permitan evaluar y comparar los resultados de los experimentos (Robertson, 1992; Su, 1992; Warner, 2000; Rijsbergen, 1979)

Este trabajo se centra en la producción de herramientas comprendidas en el segundo punto, esto es, de programas capaces de indizar documentos y resolver consultas. Más concretamente, en la producción de un motor experimental de recuperación, que permita utilizar alternativamente y con facilidad distintos algoritmos.

2.- Motores experimentales de recuperación

Básicamente, un motor de recuperación es un programa (o un conjunto de) que es capaz de indizar documentos y de resolver o ejecutar consultas sobre tales documentos. Sus componentes pueden esquematizarse de la siguiente manera (Prager, 2000; Baeza-Yates, 2000):

1. Análisis léxico, es decir, la extracción de términos clave que han de representar el contenido de cada documento. Este análisis léxico puede consistir en un simple *parsing* o en procesos más complejos, como la lematización, el etiquetado semántico, etc.
2. Indización, o construcción de índices que permitan acceder a los documentos; este proceso incluye la determinación del poder descriptivo de cada uno de los términos extraídos en la fase anterior
3. Resolución de consultas, o la estimación de la similitud entre una consulta y cada uno de los documentos de la colección
4. Interfaz de usuario, que debe permitir a éste formular sus necesidades informativas, es decir, interactuar con el sistema. Esta interacción puede incluir elementos más complejos, como la realimentación de consultas, la selección de nuevos términos de búsqueda, la visualización de documentos o resúmenes de éstos, etc.

No obstante, suele entenderse que el corazón o núcleo, lo que realmente constituye un motor de recuperación, son los componentes 2 y 3 mencionados antes. Existen, como es bien sabido numeroso motores de recuperación operacionales, diseñados para trabajar en entornos reales. Cada uno implementa un modelo teórico y utiliza un juego de algoritmos fijo; deben atender a las necesidades del mundo real, como, por ejemplo, la velocidad en la ejecución; y, debido a esto, además de razones comerciales en muchos casos, presentan una codificación específica destinada a resolver de la forma más eficiente posible sus tareas de una manera fija. Los motores experimentales, sin embargo, están destinados a la experimentación y no están coercionados por factores como la velocidad de ejecución. Su misión es admitir diversas vías de resolución de problemas, en distintos entornos y con distintos objetivos específicos. A grandes rasgos, las características deseables son las siguientes:

1. los componentes deben ser independientes entre sí, de manera que sea factible operar sobre parte de ellos, modificándolos, sin necesidad de tener que tocar el resto. Un motor experimental debería ser independiente de, por ejemplo, el analizador léxico, de firma que fuera posible alterar el comportamiento de éste o incluso sustituirlo por otro con diferentes capacidades.
2. el motor debe ser flexible como para incluir diversos algoritmos o aproximaciones a las tareas que debe resolver
3. debería permitir la observación de resultados intermedios, incluso su manipulación o modificación
4. el código debería ser lo más sencillo y modular posible, para facilitar su modificación.
5. en relación con el punto anterior, el código debería ser abierto y libremente disponible, así como estar escrito en versiones estándar de lenguajes estándar

Lamentablemente, no existen muchos motores experimentales, y que cumplan las condiciones mencionadas menos. Existen motores experimentales que no son abiertos y que sólo pueden operar los investigadores que los diseñaron, y existen motores no experimentales que son utilizados -con grandes dificultades- por algunos grupos de investigación. Uno de los paradigmas de motor experimental, utilizado durante años por diferentes grupos de investigación, es el conocido *SMART* (Salton, 1971). Sin embargo, *SMART*, que ha prestado una ayuda inestimable a muchos investigadores, y que es una excelente herramienta de experimentación, tiene algunos inconvenientes:

- está escasamente documentado, en lo que se refiere a operación y estructura interna. Sorprendentemente, además de la magra documentación ofrecida por sus autores junto con el programa, el recurso más conocido es un breve curso de utilización básica (Paimans, 1999); ni una ni otro cubren más que las capacidades más elementales del programa
- sus componentes están fuertemente integrados, de manera que, por ejemplo, no es posible aislar el parser del motor propiamente
- el código, es prolijo y complejo, lo que hace difícil su modificación. Presenta, además, algunos problemas de portabilidad.

3.- Objetivos

El objetivo de este trabajo es, pues, la realización de un motor de recuperación experimental. Así, las características de partida de nuestro motor son:

- consta tan sólo de dos componentes: un indizador y un estimador de similitud entre consulta y documentos.
- cada uno de estos dos componentes debe aceptar como entrada los resultados de componentes previos, y producir, en su caso, salidas que puedan ser usadas por componentes o procesos posteriores.
- el motor debe permitir la utilización, a elección por el usuario, de diferentes algoritmos, e incluir la mayor cantidad de éstos posibles.
- el motor debería permitir la inclusión fácil de nuevos algoritmos
- el motor permitirá la inspección y manipulación, de resultados intermedios
- el motor debe estar escrito en un código lo más breve y simple posible, fácil de modificar, e incluso fácil de utilizar como modelo para la realización de otros programas o implementaciones
- dado su carácter experimental, las características de flexibilidad, legibilidad y sencillez de código deberían primar sobre las de eficacia (especialmente velocidad)

Teniendo en cuenta todo esto, se ha considerado una opción razonable la utilización de tablas relacionales para almacenar la información extraída de los documentos, y de sentencias SQL para la realización de las operaciones necesarias. Ambas cosas están lo suficientemente estandarizadas como para poder ser modificadas fácilmente por cualquiera. La estructura relacional y el SQL son especialmente potentes e intuitivos y facilitan la comprensión y ejecución de las operaciones necesarias.

5.- El modelo vectorial

El modelo teórico más difundido en RI es el llamado modelo vectorial (Salton, 1983). Básicamente, según éste, cada documento es representado por un vector de *n* elementos donde *n* es el número de términos posibles en toda la colección de documentos, y cada elemento del vector, en consecuencia, corresponde a cada uno de tales términos. Los elementos del vector, por otra parte, consisten en un valor numérico que trata de expresar la importancia o peso del término en cuestión dentro del documento. Es obvio que un mismo término en documentos diferentes debe tener pesos diferentes.

Las consultas se tratan igual que los documentos, y se representan igualmente mediante un vector de pesos. Así, la resolución de una consulta consiste simplemente en la computación de alguna función de similitud entre el vector consulta y cada uno de los vectores de los documentos. Este tratamiento tiene dos ventajas importantes: una, permite que las consultas se hagan en lenguaje natural, y pueden ser del tamaño que se desee; y dos, dado que el resultado de la función de similitud no tiene porqué ser binario, es posible establecer una graduación o escala en las respuestas a las consultas.

La clave de todo el sistema reside en lo bien que documentos (y consultas) estén representados a través de los vectores; y esto depende de dos factores: la determinación de los términos que se extraen de cada documento, y la forma en que se estiman o calculan los pesos de cada término en cada documento. El primero de estos factores (análisis léxico) queda fuera de nuestro objetivo, pero debe indicarse la conveniencia de aislar esta parte, de forma que, a afectos de experimentación, pueda operarse sobre ella libremente. El segundo factor (el cálculo de los pesos) constituye uno de los elementos centrales de nuestro trabajo.

5.1.- El peso de los términos

La estimación del peso de cada término en cada documento puede hacerse de diversas formas, y de

hecho se han propuesto una buena cantidad de ellas. El cálculo de los pesos se efectúa a partir de dos factores (Harman, 1992): la frecuencia de cada término en cada documento, y un elemento conocido como IDF (*Inverse Document Frequency*). Adicionalmente, suele aplicarse algún factor de normalización que permita soslayar las diferencias en tamaño de los documentos. El IDF es una función inversamente proporcional a la frecuencia del término en toda la colección de documentos o base de datos. La idea base es que términos que aparezcan en muchos documentos tienen un poder discriminatorio pobre, y viceversa. El peso, en consecuencia, podría estimarse a partir de una ecuación genérica:

$$peso\,termino\,en\,documento = \frac{frecuencia\,termino\,en\,documento \times IDF}{factor\,de\,normalizacion}$$

Cada uno de los tres elementos que intervienen en la ecuación puede ser calculado de distintas formas, lo cual da lugar a un gran número de variantes o *esquemas* de pesado. Usualmente, un esquema se representa mediante tres letras, cada una de las cuales identifica la forma en que se han calculado, respectivamente, la frecuencia del término en el documento, el IDF del término y el factor de normalización.

5.2.- **Esquemas de peso**

Entre las muchas posibilidades, las formas más utilizadas de calcular estos tres elementos son:

- La frecuencia del término en el documento:
  - $\mathbf{ninguna\ (n)} := n_{td}d$
  - $\mathbf{binaria(b)} := 1$
  - $\mathbf{max\text{-}norm(m)} := \frac{n_{td}}{max_{nd}}$
  - $\mathbf{aug\text{-}norm(a)} := 0.5 + 0.5\frac{n_{td}}{max_{nd}}$
  - $\mathbf{square(s)} := n_{td}^2$
  - $\mathbf{log(l)} := ln(n_{td}) + 1$
  - $\mathbf{double\text{-}log(d)} := ln(ln(n_{td}) + 1) + 1$
  - $\mathbf{length\text{-}norm(t)} := \frac{ln(n_{td}+1}{ln(avg_{nd}))+1}$

  donde

  n_{td} es el número de veces que el término *t* aparece en el documento *d*

  max_{nd} es el número de veces que aparece el término más frecuente en el documento *d*

  avg_{nd} es la media de todas las frecuencias de términos en el documento *d*

- El IDF:

- $\mathrm{none}(\mathbf{n}) := 1$
- $\mathrm{tfidf}(\mathbf{t} := ln(\frac{N}{n_t})$
- $\mathrm{prob}(\mathbf{p}) := ln(\frac{N-n_t}{n_t})$
- $\mathrm{frec}(\mathbf{f}) := \frac{1}{n_t}$
- $\mathrm{square}(\mathbf{s}) := (ln\frac{N}{n_t})^2$

donde
N es el número de documentos en la colección
n_t es el número de documentos en que aparece el término *t*

- Factor de normalización:

- $\mathrm{none}(\mathbf{n}) := 1$
- $\mathrm{cosine}(\mathbf{c}) := \sqrt{\sum_{i=1}^{n} {p_{id}}^2}$
- $\mathrm{sum}(\mathbf{s}) := \sum_{i=1}^{n} p_{id}$
- $\mathrm{fourth}(\mathbf{f}) := \sum_{i=1}^{n} p_{id}^4$
- $\max(\mathbf{m} := max_{pd}$

donde
n es el número de términos únicos en el documento *d*
p_{id} es el peso (*frecuencia* x *IDF*) del término *i* en el documento *d*
max_{pd} es el valor más alto de peso sin normalizar en el documento *d*

6.- Implementación

Se puede almacenar un fichero invertido procedente de una colección de documentos en una tabla, de manera que, a partir de ahí, es posible calcular pesos de términos así como similitudes entre documentos y consultas. Un fichero invertido, en su forma más básica, no es más que una serie de entradas, una para cada término de la colección de documentos; para cada uno de estos términos se almacena una lista de los documentos en que aparece. Naturalmente, en esa lista pueden almacenarse más cosas (*offset* del documento en que aparece el término, etc,). Esta estructura puede mapearse simplemente a una tabla con dos campos: `termino` y `clave de documento`, pero nada impide añadir más columnas para información vinculada a cada una de las parejas término-documento, como frecuencia, *offset*, etc.

A partir de aquí es sencillo obtener información adicional para calcular pesos: el número de documentos en la colección (`select count(documento) from tabla;`), el número de documentos en que aparece cada término (`select termino, count(documento) from tabla group by`

`termino;`), etc. (Grossman, 1996; Grossman, 1997).

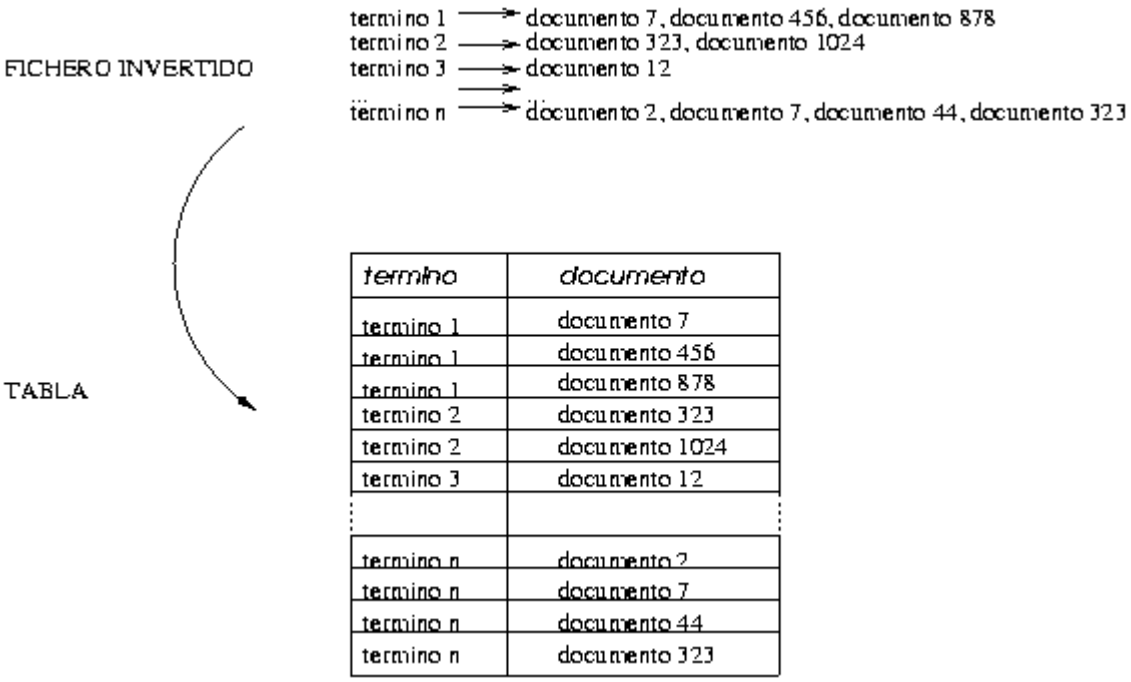

Fig. 1 Mapeo de fichero invertido a tabla simple

6.1- **La librería *sqlite***

Para la realización de los programas hemos optado como motor SQL por la librería *SQlite* (Hipp, 2003). No se trata de una librería interfaz con un servidor SQL, sino que ella misma contiene el servidor. Esto proporciona una mayor autonomía, al no depender de terceros programas. *SQlite* tiene algunas características generales notables, algunas de las cuales son:

- es una librería de código libre
- el código es completamente portable, de forma que se puede recompilar prácticamente en cualquier plataforma (en *Linux* hemos compilado con GNU C (v. 2.9.54), en MS-DOS/Windows con Mingw32 (Minimalist, 2003) sin necesidad de ningún reajuste)
- el conjunto de sentencias SQL que reconoce es reducido, pero completamente estándar, lo cual significa que esas mismas sentencias funcionarán con cualquier otro servidor SQL.
- el conjunto de funciones regulares y agregadas de que dispone es muy reducido, aunque implementa mecanismos para añadir funciones de usuario (por ejemplo, para obtener logaritmos o raíces cuadradas)
- el modo de *pasar* sentencias SQL es sencillo y claro, lo que facilita enormemente la legibilidad y modificabilidad del código del programa
- la velocidad de ejecución, aunque no es factor prioritario, es razonable
- el consumo de memoria es bajo, aunque el de disco, sin embargo, es alto

6.2 - Estructura básica de la base de datos

La base de datos consta de varias tablas. Algunas son temporales, y desaparecen una vez calculados

los pesos, y dependen del esquema de cálculo concreto adoptado en cada ocasión. Consta también de algunas vistas (para el procesamiento de consultas), aunque se ha procurado evitar el uso general de vistas, dada la poca eficacia de *SQlite* con las vistas. Las tablas básicas son:

- `terminos(termino char(35), documento char(35), veces double)`
- `pesos_def(termino char(35), documento char(35), peso double)`

En realidad, *SQlite* no distingue tipos de datos, almacenando todo como secuencias de caracteres con un byte 0 como terminador. Pero, por razones de portabilidad se han establecido estos tipos. De otro lado, el número de veces que un término aparece en un documento, que debería ser un entero, se establece como doble. La razón es que esto permite que el *parser* u otro proceso previo aplique, si se desea, algún tipo de coeficiente que prime de distinta forma los términos en función de distintos criterios (lugar del documento donde aparece, tipografía, función sintáctica, etc.) Para la resolución de consultas, en realidad, sólo es precisa la tabla `pesos_def`, pero la tabla `terminos` es preciso conservarla para posibles recalculados de pesos posteriores.

6.3 - Entrada de datos

El motor de recuperación espera recibir como entrada lo siguiente:

```
"término","documento",número de ocurrencias en documento
```

y, tal cual lo almacena en una tabla. Sobre la información almacenada en esta tabla se harán las operaciones posteriores. El programa no efectúa ningún chequeo ni ninguna otra operación previa sobre los datos de entrada. Esto significa que cosas como la normalización de caracteres, eliminación de palabras vacías, etc. es responsabilidad del *parser* o de otros cualesquiera procesos intermedios que se quieran añadir.

6.4 - Cálculo de pesos

El cálculo de pesos se efectúa en tres fases, una para cada componente del peso (Salton, 1988). Cada una de estas fases termina con una tabla temporal que recoge el componente calculado y que es usada en la fase siguiente; la tercera y última fase finaliza con la consecución de la tabla `pesos_def`, con lo que esas tablas temporales dejan de ser necesarias. No hay atajos, en aras de una mayor claridad del código.

6.4.1 La frecuencia del término en el documento

El mecanismo concreto depende del esquema de cálculo aplicado, pero, en general, se resuelve con un `select` sobre la tabla `terminos` que contiene los datos originales de entrada. Por ejemplo, para el esquema **l** (log)

```
create table frecuencia as
select termino as termino, documento as documento,
neperiano(frecuencia)+1 as frecuencia
from terminos;
```

Con algunos esquemas que utilizan cosas como la frecuencia máxima en el documento, es preciso algún paso intermedio que calcule tales elementos.

6.4.2 - El IDF y el peso sin normalizar

El IDF es el mismo para cada término, independientemente de en qué documento aparezca éste. De manera que el resultado del cálculo del IDF podría ser una tabla con los campos `termino` e `idf`. El peso sin normalizar, por otra parte, es el resultado de multiplicar frecuencia por IDF; así, obtenido el IDF, puede obtenerse en la misma fase el peso sin normalizar. El producto final de esta fase es, una tabla con los campos `termino`, `documento` y `peso`. Por ejemplo, para calcular el IDF según el esquema **f** (frec):

```
create table idf as
select termino as termino, 1 / count(documento) as idf
from terminos
group by termino;
```

Una vez obtenido el IDF, sólo nos queda calcular el peso sin normalizar y almacenarlo en una tabla:

```
create table pesos as
select frecuencia.termino as termino,
frecuencia.documento as documento,
frecuencia.frecuencia*idf.idf as peso
from frecuencia, idf
where frecuencia.termino=idf.termino;
```

6.4.3 - El factor de normalización y pesos definitivos

Esta fase, última por lo que se refiere a los documentos, requiere el cálculo de un factor de normalización, y la posterior división del peso sin normalizar que acabamos de almacenar en la tabla *pesos* por dicho factor. El factor de normalización, por otra parte, es único para cada documento. A modo de ejemplo, para normalizar mediante el esquema **c** (cos):

```
create table sumatorios as
select documento as documento,
raiz_cuadrada(sum(peso*peso)) as s
from pesos group
by documento
```

y luego

```
create table pesos_def as
select pesos.termino as termino,
pesos.documento as documento,
pesos.peso / sumatorios.s as peso
from pesos, sumatorios
where pesos.documento=sumatorios.documento;
```

Hasta aquí, hemos obtenido los pesos de los términos de los documentos, con lo que sólo necesitamos la tabla `pesos_def` y la que contiene los datos originales, `terminos`, de manera que podemos desacernos de las demás. La tabla `pesos_def`, por otra parte, requiere un índice de `termino`, para resolver más rápidamente las consultas.

6.5 - Pesos de las consultas

Los pesos de los términos de las consultas se calculan aplicando esquemas que no tienen porqué ser iguales. Sin embargo, dado que se resuelve una sola consulta de cada vez, el volumen de datos a

manejar es considerablemente menor; además, las consultas suelen ser mucho más cortas que cualquier documento, usualmente 2 ó 3 palabras. La solución adoptada aquí se basa en el uso de vistas. Se comentó más arriba que *SQlite* tiene un tratamiento poco eficaz de éstas. Sin embargo, cuando las vistas tienen que desenvolver una cantidad pequeña de datos, el rendimiento es aceptable; esto nos permite, al tiempo que se indiza la colección de documentos, dejar construidas las vistas necesarias para calcular los pesos de sus términos. En el momento de la consulta, estas vistas se ejecutan, obteniendo los pesos correspondientes.

6.6 - Resolución de consultas y salida

Para la resolución de consultas se ha habilitado un programa independiente que opera sobre la base de datos y sus correspondientes tablas y vistas. Al igual que se ha hecho con los documentos, el programa de búsqueda espera recibir el resultado del análisis léxico de la consulta a resolver, en el mismo formato. Esta entrada pasa a una tabla, sobre la que se ejecutarán las vistas definidas en el proceso de la indización de los documentos, de acuerdo con el esquema de pesos que se haya especificado. De hecho, dichas vistas son llamadas directamente con la sentencia SQL que resuelve la consulta, de manera que dichas vistas se ejecutan calculando los pesos de los términos de la consulta y, acto seguido, calculando la similitud entre la consulta cada uno de los documentos; en realidad, sólo se calcula la similitud con los documentos que tienen en común al menos un término con la consulta. Con la misma sentencia se ordenan dichas similitudes y sus documentos asociados en forma decreciente. A partir de aquí, sólo queda producir una salida con las claves de los documentos recuperados y sus coeficientes de similitud. Puesto que el uso de este motor es la experimentación, uno de los formatos de salida posibles es acorde con el conocido programa `trec-eval`, el cual, como es conocido, calcula las medidas más utilizadas para evaluar la eficiencia en la recuperación.

7.- Conclusiones

Se ha mostrado la estructura funcionamiento de un motor de recuperación de información diseñado para la investigación experimental. Este motor es de código abierto y uso libre, y puede ser descargado de la dirección de Internet `http://reina.usal.es/materiales.htm`

REFERENCIAS:

Baeza-Yates, R. y Ribeiro-Neto, B. (2000). Modern Information Retrieval. Harlow: Addisson-Wesley, 2000

Grossman, D. A.; Lundquist, C.; Reichart, J.; Holmes, D.; Chowdhury, A. Y Frieder, O. (1996).Using Relevance Feedback withim the Relational Model for TREC-5// en Vorhees, E. M. y Harman, D. (eds.): The Fifth Tex Retrieval Conference (TREC-5). Gaithersburg, Maryland:NIST, 1996

Grossman, D. ; Frieder, O.; Holmes, D. y Roberts, D. (1997). Integrating Structured Data and Text: A Relational Approach. // JASIS 48:2 122-132

Harman, D. (1982). Ranking Algorithms.// Frakes, W. B. Y Baeza-Yatez, R. (eds.). Information Retrieval. Data Structures and Algorithms. Upper Sadle River NJ: Prentice-Hall, 1982. 363-392

Hipp, R. (2003). SQLite. An Embeddable SQL Database Engine. URL <http://www.sqlite.org>. Consultado: 31-10-2003

Minimalist GNU For Windows. URL <http://www.mingw.org>. Consultado: 31-10-2003

Paimans, H. (1999). SMART. Tutorial for beginners.URL <http://pi0959.uvt.nl/Paai/Onderw/Smart/hands-on-tekst.html>.Consultado: 31-10-2003

Prager, J.; Brown, J. ; Radev, D. y Czuba, K. (2000). One Search Engine or Two for Question Answering.// TREC-9 (2000) 235-240

Rijsbergen, C. J. van (1979). Information Retrieval. London: Butterworths, 1979

Robertson, S. E. y Hancock-Beaulieu, M. H. (1992). On the Evaluation of the IR Systems. // Information processinf and Management. 28 (1992) 457-466

Salton, G.: The SMART Retrieval System. Experiments in Automatic Document Processing, Prentice Hall, Englewood Cliffs, NJ, 1971

Salton, G. y Buckley, C.: Term-weighting approaches in automatic text retrieval, Information processing & Management, 24(5), 513-523.

Salton, G. (1989). Automatic Text Processing: the Transformation,Analysis an Retrieval of Information by Computer. Reading MA:Addison-Wesley, 1989

Singhal, A., Buckley, C. y Mitra, M.: Pivoted document length normalization, SIGIR 96, ,21-29

Su, L. T. (1992). Evaluation Measures for Interactive Information Retrieval. // Information Processing and Management 28 (1992) 503-516

Warner, J. (2000) In the Catalalogue Ye Go for Men: Evaluation Criteria for Information Retrieval.// Aslib Procs. 52:2 (2000) 76-82