ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

TESIS DE GRADO

"MARCO DE TRABAJO PARA INDEXACIÓN, CLASIFICACIÓN Y RECOPILACIÓN AUTOMÁTICA DE DOCUMENTOS DIGITALES"

Previa a la obtención del titulo de:

INGENIERO EN COMPUTACIÓN ESPECIALIZACIÓN SISTEMAS DE INFORMACIÓN

PRESENTADA POR:

JAVIER ANDRES CAICEDO ESPINOZA
GONZALO ALBERTO PARRA CHICO

GUAYAQUIL - ECUADOR

2007

AGRADECIMIENTO

A todos quienes nos

ayudaron siendo

pilares importantes para la

presentación de este trabajo.

En especial a Xavier Ochoa y

Federico Raue quienes nos apoyaron

y alentaron en todo momento.

DEDICATORIA

A nuestros padres

TRIBUNAL DE GRADO

PRESIDENTE

Ing. Holger Cevallos Ulloa

DIRECTOR DE TESIS

Msc. Xavier Ochoa Chehab

MIEMBROS PRINCIPALES

Ph.D. Katherine Chiluiza

Ing. Verónica Uquillas

DECLARACIÓN EXPRESA

"La responsabilidad por los hechos, ideas y doctrinas expuestas en esta tesis, nos corresponden exclusivamente; y, el patrimonio intelectual de la misma, a la Escuela Superior Politécnica del Litoral"

(Reglamento de exámenes y títulos profesionales de la ESPOL)

Javier Andrés Caicedo Espinoza

Gonzalo Alberto Parra Chico

RESUMEN

En la actualidad el crecimiento de la Web tiene dos implicaciones: existe más información disponible, pero el obtener información que cumpla nuestros requerimientos se vuelve una tarea compleja. Los motores de búsqueda son buenas soluciones de manera general, pero existen enfoques alternativos de solución que pueden mejorar los resultados de búsquedas en la Web. El objetivo de este trabajo es analizar diferentes enfoques de solución a los problemas de exploración, clasificación e indexación de documentos digitales en la Web, e incorporarlos en una sola herramienta, un marco de trabajo. Este trabajo detalla el análisis, diseño e implementación de un marco de trabajo para la búsqueda de documentos digitales en la Web.

En el primer capítulo se describe cómo la búsqueda de información en línea se convierte en una parte primordial de cualquier actividad investigativa, debido a la necesidad de información relevante y actualizada por parte de los usuarios. De igual manera se establecen los objetivos de éste trabajo de investigación y se presentan los diferentes mecanismos que mejoran el proceso de exploración en línea. Para propósitos de este trabajo investigativo hemos decidido separar el proceso de búsqueda de información en tres componentes principales:

- Exploración automática de páginas web
- Clasificación de documentos digitales

- Indexación de documentos digitales

En la parte final de este capítulo, presentamos las alternativas disponibles y estado actual de investigación para cada uno de estos componentes.

En el segundo capítulo establecemos los requerimientos básicos necesarios para el funcionamiento adecuado de nuestro marco de trabajo. Luego, procederemos a realizar un análisis de las alternativas existentes para cada uno de los componentes, y a la selección de la opción más apropiada de acuerdo a los requerimientos.

En el tercer capítulo se explica todo lo referente al diseño del marco de trabajo, basándose en el conjunto de requerimientos definidos en los capítulos anteriores. Se definen de manera explícita detalles referentes a la arquitectura y patrones de diseño aplicados en el marco de trabajo. Finalmente, se presenta el diseño específico utilizado en cada uno de los componentes.

En el cuarto capítulo se describe la plataforma utilizada para el desarrollo del marco de trabajo descrito en el capítulo anterior, así como sus requerimientos de funcionamiento. Además se especifican los pasos requeridos para su instalación. Al final de este capítulo, se detalla la aplicación de ejemplo orientada a la búsqueda de documentos digitales en

Internet. Esta herramienta hará uso de todas las características principales del marco y mostrará la flexibilidad del mismo.

En el quinto capítulo introducimos las métricas más comúnmente utilizadas en la evaluación de mecanismos para búsqueda en la Web, y presentamos el esquema de pruebas a utilizarse; luego se procede a realizar una evaluación del rendimiento del marco de trabajo usando este esquema de pruebas.

Finalmente se detallan las conclusiones de nuestro trabajo de investigación, así como recomendaciones para su aplicación en proyectos futuros. Como resultado de la implementación se obtuvo un marco de trabajo flexible, que puede ser modificado o actualizado a nuevas tendencias o enfoques del área.

ÍNDICE GENERAL

ÍNDICE DE GRÁFICOS

ÍNDICE DE TABLAS

INTRODUCCIÓN

La creciente cantidad de textos disponibles en la Web y la imperiosa necesidad de información relevante por parte de los usuarios, hacen de la búsqueda de información en la Web un punto crítico en cualquier actividad investigativa. Para ayudar al usuario en ésta tarea se han desarrollado poderosos motores de búsqueda (por ejemplo, Google o Yahoo!) que organizan la información disponible en base a categorías, títulos o contenido. Estas herramientas basan su funcionamiento en crear grandes índices de documentos, usados para resolver las consultas de los usuarios. La mayor parte de los documentos obtenidos como resultados a dichas consultas suelen ser poco relevantes para el usuario a pesar de cumplir los criterios de búsqueda especificados.

En la actualidad existen nuevas herramientas que pueden brindar un enfoque diferente a la búsqueda de documentos en la Web, dando un mayor énfasis a las áreas de interés del usuario que requiere la información y obteniendo resultados más precisos. El campo de la inteligencia artificial provee estas nuevas herramientas utilizando la estadística y las matemáticas, permitiendo realizar análisis más profundos sobre el contenido de los documentos.

El presente documento describe el proyecto de implementación de un marco de trabajo que tenga la capacidad de clasificar automáticamente documentos

digitales, de acuerdo a áreas temáticas definidas; además de explorar en la Web y localizar publicaciones relacionadas con estos documentos. Adicionalmente, este marco tendrá la capacidad de indexar los resultados obtenidos para poder resolver búsquedas locales.

CAPÍTULO 1

1 PLANTEAMIENTO DEL PROBLEMA Y ESTADO ACTUAL DE INVESTIGACIÓN

En este capítulo se describe cómo la búsqueda de información en línea se convierte en una parte primordial de cualquier actividad investigativa, debido a la necesidad de información relevante y actualizada por parte de los usuarios. De igual manera se establecen los objetivos de éste trabajo de investigación y se presentan los diferentes mecanismos para la mejora del proceso de exploración en línea. Para propósitos de este trabajo hemos decidido separar el proceso de búsqueda de información en tres componentes principales:

- Exploración automática de páginas web
- Clasificación de documentos digitales
- Indexación de documentos digitales

En la parte final del capítulo, presentamos las alternativas disponibles y estado actual de investigación para cada uno de estos componentes.

1.1 Definición del Problema

La Escuela Superior Politécnica del Litoral (ESPOL) se ha caracterizado por promover la investigación entre sus docentes y estudiantes. En la actualidad, la Web se convierte en el recurso más valioso para el desarrollo de estas investigaciones, debido a la gran cantidad de información disponible [1] [2]. Sin embargo, al no poseer la Web una entidad central que la administre, y debido al continuo incremento de información, es cada vez más difícil la búsqueda de documentos relevantes para un investigador [2].

Los motores de búsqueda actuales intentan ayudar al investigador en su continua necesidad de información actualizada, pero, dado que basan su funcionamiento en palabras claves, y sin tomar en cuenta las áreas de interés del usuario, los resultados obtenidos no siempre serán representativos. Por ejemplo, al buscar el término “palma” obtendremos resultados con distintos significados de acuerdo al contexto en que se la considere: la palma de la mano, el árbol, lugares geográficos, etc. Pero si enfocamos la búsqueda dentro de un contexto definido, sea éste lugares geográficos, obtendremos rápidamente el resultado esperado.

El desarrollo actual de nuevas herramientas en el área de Inteligencia Artificial, permite dirigir el enfoque de una búsqueda no sólo a palabras claves, sino a las áreas de interés del investigador, proporcionando así resultados más precisos y de mayor relevancia.

En este documento presentamos la integración de varias herramientas desarrolladas en el área de inteligencia artificial en un marco de trabajo [3]. El cual permitirá el desarrollo de diferentes aplicaciones que pueden ser incorporadas al proceso de investigación que se ejecuta en ESPOL.

1.2 Objetivo del Trabajo de Investigación

El presente trabajo de investigación tiene por objetivo principal construir un marco de trabajo (framework) que tenga la capacidad de buscar, clasificar e indexar automáticamente documentos digitales disponibles en Internet.

De manera específica se esperan cumplir los siguientes objetivos:

- Analizar los diferentes enfoques de solución para la clasificación, indexación y recopilación automática de documentos digitales en línea.

- Experimentar y luego escoger qué tipo de técnicas basadas en Inteligencia Artificial pueden ser utilizadas en la clasificación, indexación y recopilación automática de documentos digitales en línea.
- Diseñar e implementar un marco de trabajo, cumpliendo los requerimientos definidos en el análisis. Para dicho efecto, se emplearán herramientas de código abierto, por ende, el producto será distribuido bajo una licencia de este tipo.
- Diseñar e implementar una aplicación que utilice el marco de trabajo desarrollado previamente, para indexar, clasificar y recopilar publicaciones científicas (papers). Esta aplicación también será desarrollada bajo una licencia de código abierto.
- Comprobar el grado de efectividad de la solución basada en el marco de trabajo, y medir su desempeño en relación a otras soluciones similares ya existentes.

1.3 Teoría e Investigación hasta la actualidad

1.3.1 Exploración automática de páginas Web

Un agente explorador (web crawler, también conocido como robot web o araña web) es un programa diseñado para explorar el Internet en una forma metódica y organizada. Este tipo de programas

aprovechan la estructura en forma de grafo del Internet para moverse de una página a otra [4].

Tradicionalmente, el propósito de los agentes exploradores ha sido el de recolectar páginas web y añadir éstas (o una representación de éstas) en un repositorio local para su uso posterior. La aplicación más común de un agente explorador es la de proveer datos actualizados para motores de búsqueda: el agente guarda una copia de las páginas que va visitando, y éstas son indexadas por el motor de búsqueda para proveer resultados rápidamente. Otras posibles aplicaciones incluyen automatizar tareas de mantenimiento para sitios web, como por ejemplo, validación del código HTML o de la estructura de navegación del sitio.

La Fig. 1.1 muestra el flujo de trabajo de un agente explorador básico. El agente mantiene una lista de hipervínculos a visitar, denominada la *frontera*. Al principio, esta lista contiene páginas denominadas *semillas*, a partir de las cuales se iniciará el proceso de exploración. Estas semillas pueden ser provistas por un usuario u obtenidas de algún proceso previo (por ejemplo, a partir de un directorio web). Cada iteración del agente involucra escoger el siguiente vínculo a explorar de la frontera, obtener la página asociada

al vínculo, analizar el contenido de la página para extraer los hipervínculos y la información relevante para la aplicación, y finalmente añadir los nuevos vínculos a la frontera. Se puede asignar una calificación al vínculo al momento de añadirlo a la frontera, que representa el beneficio relativo de explorar la página relacionada a ese vínculo. El proceso se repite hasta alcanzar un determinado número de páginas visitadas, o hasta que la frontera no contiene más vínculos por visitar.

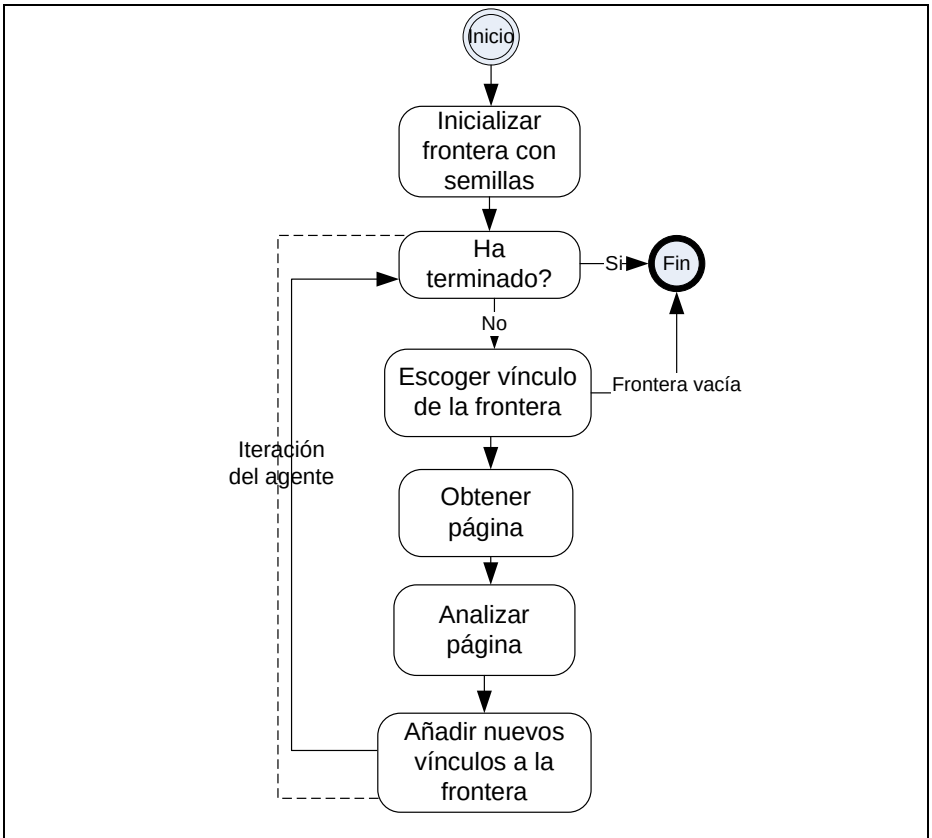


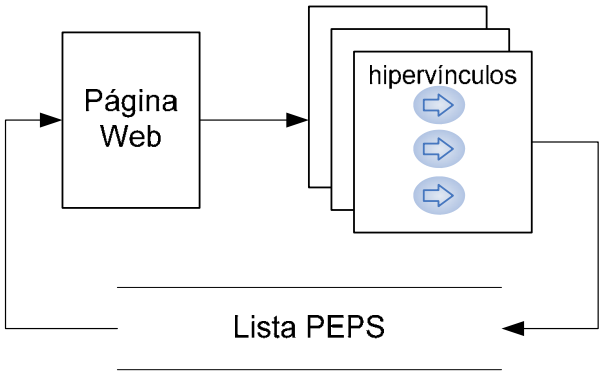
Figura 1.1. Flujo de trabajo de un agente explorador básico [F1].

Una iteración del agente puede contener muchos períodos de tiempo en los cuales no se aprovechan al máximo los recursos de red o del procesador. Por ello, la mayoría de agentes exploradores hacen uso de técnicas multihilo, que permiten un mejor aprovechamiento de los recursos de procesamiento y de ancho de banda. En esta versión del agente, cada hilo maneja un hipervínculo a la vez, manteniendo la frontera compartida. Se hace necesario incorporar mecanismos de bloqueo y liberación de la frontera, a fin de mantener la sincronización entre todos los hilos del agente. Adicionalmente, la tarea de determinar cuándo el agente ha terminado la exploración requiere de consideraciones especiales, ya que en un momento determinado la frontera puede encontrarse vacía, a pesar de que haya hilos que todavía estén procesando vínculos.

Al analizar el esquema de funcionamiento de un agente explorador, vemos que la estrategia para escoger el siguiente vínculo a visitar es una de las partes más determinantes para el desempeño del agente. A continuación revisaremos algunas de las estrategias o políticas de selección que se sugieren en la literatura.

1.3.1.1 Estrategia primero-en-anchura

Un agente primero-en-anchura es la forma más simple para explorar la Web. Esta estrategia aprovecha la estructura en forma de grafo del Internet para realizar la exploración. La Fig. 1.2 ilustra un agente explorador primero-en-anchura y el algoritmo utilizado para su implementación.




```
Primero_en_anchura(semillas) {
    Para_cada vínculo(semillas) {
      añadir (frontera,vínculo);
    }
    mientras (visitados < MAX_PAGINAS) {
      vínculo := extraer_vínculo (frontera);
      doc := obtener (vínculo);
      añadir(frontera, extraer_vínculos (doc));
      si (#frontera > MAX_FRONTERA) {
      remover_últimos_vínculos (frontera); }
    }
}
```

Figura 1.2. Agente explorador primero-en-anchura y pseudocódigo del algoritmo utilizado para su implementación. [F2]

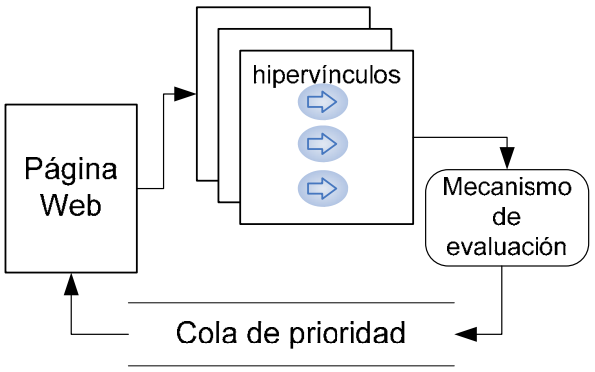
Esta estrategia usa la frontera como una lista PEPS (Primero en Entrar, Primero en Salir), explorando los vínculos en el orden en que

son encontrados. Podemos observar que cuando la frontera se encuentra llena, el agente sólo puede añadir un vínculo a la vez.

La estrategia primero-en-anchura generalmente es usada como una línea base para otras técnicas más sofisticadas, dado que no utiliza información alguna sobre el contenido de las páginas que va encontrando; y por lo tanto, se esperaría que su desempeño sea menor.

1.3.1.2 Estrategia primero-el-mejor

Los agentes exploradores de este tipo son unos de los más usados en la literatura. La idea básica consiste en que, dada una frontera de vínculos, se escoge el mejor de acuerdo a un mecanismo de evaluación especificado. Este mecanismo de evaluación es el que determinará la complejidad y efectividad del agente. En las implementaciones más sencillas, o “ingenuas”, se usa el contenido de las páginas para calcular un grado de relevancia relativo al tópico objetivo. En la Fig. 1.3 podemos observar la arquitectura de un agente explorador primero-el-mejor, así como el algoritmo utilizado para su implementación.




```
Primero_el_mejor (tópico, semillas) {
    Para_cada vínculo(semillas) {
    añadir (frontera,vínculo,1); }
    mientras (visitados < MAX_PAGINAS) {
      vínculo := extraer_primer_vínculo (frontera);
      doc := obtener (vínculo);
      calif := sim(tópico, doc);
      añadir(frontera, extraer_vínculos (doc), calif);
      si (#frontera > MAX_FRONTERA) {
      remover_últimos_vínculos (frontera); }
    }
}
```

Figura 1.3. Agente explorador primero-el-mejor y pseudocódigo del algoritmo utilizado para su implementación. [F2]

1.3.1.3 Estrategia “banco de peces”

Los agentes exploradores que utilizan esta estrategia se basan en la suposición que documentos relevantes generalmente tienen vecinos relevantes. Así, se concentra en áreas en donde ha encontrado documentos relevantes para la búsqueda, y se detiene en áreas “estériles”. El comportamiento de estos agentes exploradores se asemeja a un banco de peces: en áreas con abundante comida (resultados relevantes), los peces (agentes) se reproducen y continúan su búsqueda; mientras que en áreas escasas, mueren.

Esta estrategia puede ser vista como una variante de primero-el-mejor, con un mecanismo de evaluación de vínculos más sofisticado. En la Fig. 1.4 podemos apreciar el pseudocódigo para este algoritmo. Este algoritmo asigna dos tipos de calificaciones, una denominada "vecinos" que representa el contexto de la página donde se halló el hipervínculo, y la otra se la denomina "heredada", y es derivada de las calificaciones de los ancestros de la página actual. Los parámetros d y r representan la profundidad máxima de la búsqueda y la importancia relativa entre las valoraciones de vecinos y las heredadas.

```
Peces (tópico, semillas) {
      Para_cada vínculo(semillas) {
        Fijar_profundidad(vínculo,d);
        añadir (frontera,vínculo,1);
      }
      mientras (visitados < MAX_PAGINAS) {
        vínculo := extraer_primer_vínculo (frontera);
        doc := obtener (vínculo);
        calif := sim(tópico, doc);
        si (profundidad(vínculo) > 0) {
          Para_cada vínculo_externo(extraer_vínculos (doc)) {
          calif:= (1-r) * calif_vecinos (vínculo_externo)
                  + r * calif_heredados(vínculo_externo);
          }
          si (calif > 0) {
          Fijar_profundidad(vínculo,d);
          }
          si no {
          Fijar_profundidad(vínculo, profundidad(vínculo) -1);
          }
          añadir(frontera, vínculo_externo, calif);
          si (#frontera > MAX_FRONTERA) {
          remover_últimos_vínculos (frontera); }
        }
}
```

Figura 1.4. Pseudocódigo del algoritmo utilizado para la implementación de la estrategia "banco de peces". [F2]

1.3.2 Clasificación de documentos

La tarea de clasificar documentos, también conocida como categorización, es asignar o ubicar documentos en categorías previamente definidas, basándose en el contenido del mismo [5]. Los sistemas de clasificación han sido desarrollados desde el siglo XIX y mantenidos hasta la actualidad por su gran efectividad en el manejo de grandes cantidades de información [6]. Esta clasificación es realizada por un humano, haciendo de éste un proceso no escalable. Con la era de Internet y el enorme crecimiento de bases de datos, miles o tal vez millones de documentos están disponibles para los usuarios, lo cual imposibilita su clasificación manual; de aquí que esta tarea debe ser asumida por una máquina.

1.3.2.1 Modelos de representación

Una máquina no puede entender documentos escritos en lenguaje natural (no estructurado), por ello debe procesarlos y obtener una representación estructurada de dichos documentos. Esta representación se basa en tres modelos clásicos definidos: el modelo booleano, el modelo vectorial y el modelo probabilístico.

1.3.2.1.1 Modelo booleano

Es un modelo básico de extracción de información (information retrieval) basado en la teoría de conjuntos y en el álgebra booleana. El uso de conjuntos y expresiones booleanas hizo de éste un modelo fácil de usar y muy popular hace algunos años entre los sistemas bibliográficos comerciales [7].

Por ejemplo, consideremos los siguientes documentos (D) que contienen los términos (ti).

D1: {t1, t2}

D2: {t3, t2}

D3: {t4, t2}

Se define la siguiente consulta (Q), Q = t2 y t3

La respuesta a la consulta es la intersección entre el subconjunto formado por todos documentos que posean el término t2 y el subconjunto formado por todos los documentos que posean el término t3.

Q = { D1 , D2 , D3 } ^ { D2 } = { D2 }

La respuesta a la consulta formulada es el documento D2.

Al analizar el ejemplo, se puede determinar que el modelo booleano sufre de dos grandes problemas [8]:

- Debido a su naturaleza, sólo puede predecir si un documento es relevante o no relevante; no tiene capacidad de evaluar alguna escala intermedia.
- Es muy complicado traducir la necesidad de un usuario a una expresión booleana.

A pesar de éstos problemas, el modelo booleano es uno de los más dominantes en el mercado comercial de bases de datos.

1.3.2.1.2 Modelo vectorial

El modelo vectorial surgió a partir de las falencias del modelo booleano, permitiendo una asignación de pesos a términos índices de un documento o una consulta. Estos pesos son usados para medir el grado de similaridad de un documento con la consulta del usuario. Con ello obtenemos resultados más precisos y en mayor cantidad (precision y recall) [8].

Un documento d_n, y una consulta del usuario q, son representados con un vector n-dimensional, tal como se muestra en la Fig. 1.5.

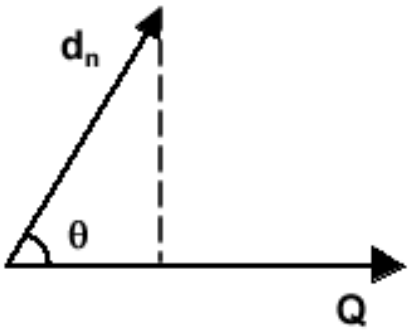


Figura 1.5. El coseno del ángulo es adoptado como la similaridad entre $\vec{d}_n$ y $\vec{Q}$ [F3].

En lugar de evaluar si un documento es relevante o no, el modelo propone evaluar el grado de similaridad entre el documento d_n, y la consulta q, con una correlación entre los vectores $\vec{d}_n$ y $\vec{q}$, tal como se muestra en la Fig. 1.5. Esta correlación puede ser cuantificada con el coseno del ángulo entre éstos dos vectores. De este modo, un documento puede ser obtenido como resultado, aunque solo una fracción de su contenido concuerde con la consulta.

Este modelo posee una desventaja: se asumen que los términos índice son mutuamente excluyentes, lo cual en la práctica puede resultar problemático. Este modelo es simple y rápido, por ello es muy popular en la actualidad. Los mecanismos para la asignación de pesos a cada uno de los términos índice determinarán en gran medida el desempeño que el modelo pueda alcanzar.

1.3.2.1.3 Modelo probabilístico

El modelo trata de estimar la probabilidad de que el usuario encontrará un documento d_n relevante, enmarcando el problema en un modelo probabilístico. Además asume que esta probabilidad de relevancia depende únicamente de la consulta y la representación del documento. Luego, asume que dada una consulta *q*, existe un subconjunto de documentos que el usuario considera relevantes para su consulta, este conjunto ideal de respuesta es etiquetado como R y debe maximizar la probabilidad de relevancia para el usuario [13].

El modelo hace un ranking de documentos en orden decreciente de la probabilidad de relevancia de acuerdo a la información requerida $P(R|q,d_i)$. Empieza con una estimación inicial y a medida que se reciben retroalimentaciones refina su estimación, la evaluación inicial se realiza en base a la ocurrencia de términos en la consulta y en el documento. El cálculo de las probabilidades es imposible, es por esto que se asumen y simplifican factores. La expresión clave (rsv – retrieval status value) para establecer un ranking de documentos dentro de una misma consulta es:

$$rsv(d_j, q) = \sum_t \log \frac{P(t_i \mid R)P(\bar{t}_i \mid \bar{R})}{P(t_i \mid \bar{R})P(\bar{t}_i \mid R)}$$

Por ejemplo, en la Tabla 1.1 se define un ejemplo del modelo probabilístico, en el cual se establecen 6 documentos y se define la ocurrencia de 10 términos dentro de los documentos, con lo cual se calcula el peso de cada uno de los términos [14].

d	Vectores de Documentos									
	col	**day**	**eat**	**hot**	**lot**	**nin**	**old**	**pea**	**por**	**pot**
1	1,00			1,00				1,00	1,00	
2								1,00	1,00	1,00
3		1,00				1,00	1,00			
4	1,00			1,00						1,00
5								1,00	1,00	
6			1,00		1,00					
w_t	0,26	0,56	0,56	0,26	0,56	0,56	0,56	0,00	0,00	0,26

Tabla 1.1 Ejemplo del modelo probabilístico.

Inicialmente no existen documentos relevantes; es por esto, que se simplifican factores de la fórmula inicial y se obtiene w_t.

$$w_t = \log \frac{N - n + 0.5}{n + 0.5}$$

Donde, N = # de total de documentos

n = # de documentos que poseen el término

Se ha definido un ranking inicial, luego el usuario ha elegido un documento (d_2) como relevante y con ésta retroalimentación calculamos nuevamente el ranking, ahora incluyendo un subconjunto de documentos relevantes definido por el usuario, como se observa en la Tabla 1.2.

d	Vectores de Documentos									
	col	**day**	**eat**	**hot**	**lot**	**nin**	**old**	**pea**	**por**	**pot**
1	1,00			1,00				1,00	1,00	
2								1,00	1,00	1,00
3		1,00				1,00	1,00			
4	1,00			1,00						1,00
5								1,00	1,00	
6			1,00		1,00					
wt	-0,33	0,00	0,00	-0,33	0,00	0,00	0,00	0,62	0,62	0,95

Tabla 1.2 Ejemplo del modelo probabilístico recalculada.

Para incluir la retroalimentación obtenida por el usuario, se modifican ciertas estimaciones, obteniendo una nueva fórmula w_t.

$$w_t = \log \frac{(r+0.5)(N-R-n+r+0.5)}{(n-r+0.5)(R-r+0.5)}$$

Donde, R = # de documentos relevantes

r = # de documentos relevantes y poseen el término.

Este modelo presenta la ventaja de, ordenar los documentos de acuerdo a la probabilidad de relevancia. Posee también algunas desventajas:

- Se necesita establecer una separación inicial de documentos relevantes y no relevantes.
- El método no considera la frecuencia con la cual un término se repite dentro de un documento.
- Se asume que los términos índice son independientes.

1.3.2.2 Algoritmos de clasificación

1.3.2.2.1 Naive Bayes

Es un clasificador probabilístico que utiliza las evidencias observables de un objeto para predecir su clasificación utilizando el teorema de Bayes, el cual se basa en las probabilidades condicionales. A pesar de su simpleza, es uno de los clasificadores que ofrece mayor eficacia en problemas del mundo real [10]. Se denomina ingenuo (naive) porque asume independencia entre las variables que describen a un objeto, algo que no siempre se cumple en la mayoría de problemas reales.

Para explicar el concepto del Clasificador Naive Bayes, consideremos el ejemplo que se muestra en la Fig. 1.6 [11].

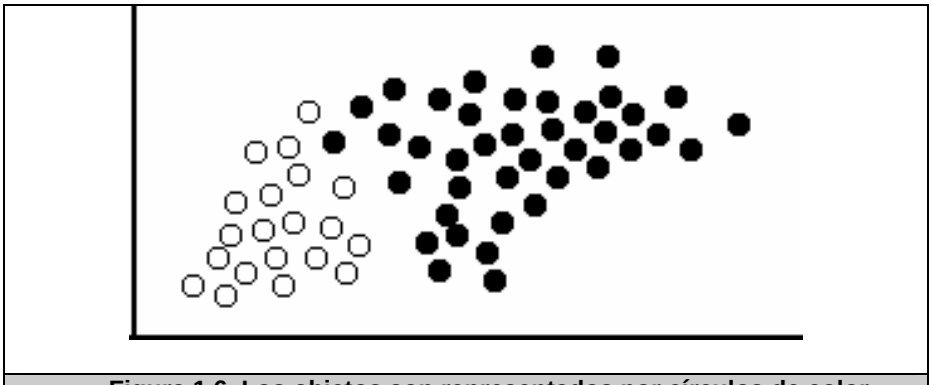


Figura 1.6. Los objetos son representados por círculos de color BLANCO o NEGRO [F4].

Como se observa, existen objetos que pueden ser clasificados ya sea como NEGROS o BLANCOS, y nuestra tarea es clasificar

nuevos objetos que son ingresados; es decir, determinar la etiqueta de la clase a la que pertenecen, basado en los objetos existentes actualmente.

Debido a que los objetos NEGROS doblan en cantidad a los objetos BLANCOS, es razonable creer que un nuevo objeto es dos veces más probable que sea catalogado NEGRO en lugar de BLANCO. En el análisis bayesiano, ésta creencia es conocida como la probabilidad previa. Las probabilidades se basan en experiencia previa, en este caso el porcentaje de objetos NEGROS y BLANCOS, y son comúnmente usadas para predecir respuestas antes de que sucedan.

Con esta explicación podemos escribir las probabilidades de las dos clases de objetos. Al existir un total de 60 objetos, de los cuales 40 son NEGROS y 20 son BLANCOS, la probabilidad previa para los objetos NEGROS es:

$$\text{Probabilidad Previa de NEGRO} \propto \frac{\text{Número de Objetos NEGROS}}{\text{Total de Objetos}}$$

$$\text{Probabilidad Previa de NEGRO} \propto \frac{40}{60}$$

Y para los objetos BLANCOS:

$$\text{Probabilidad Previa de BLANCO} \propto \frac{\text{Número de Objetos BLANCOS}}{\text{Total de Objetos}}$$

$$\text{Probabilidad Previa de BLANCO} \propto \frac{20}{60}$$

Habiendo formulado nuestras probabilidades previas, estamos listos para clasificar un nuevo objeto X, representado con un color gris; como se muestra en la Fig. 1.7.

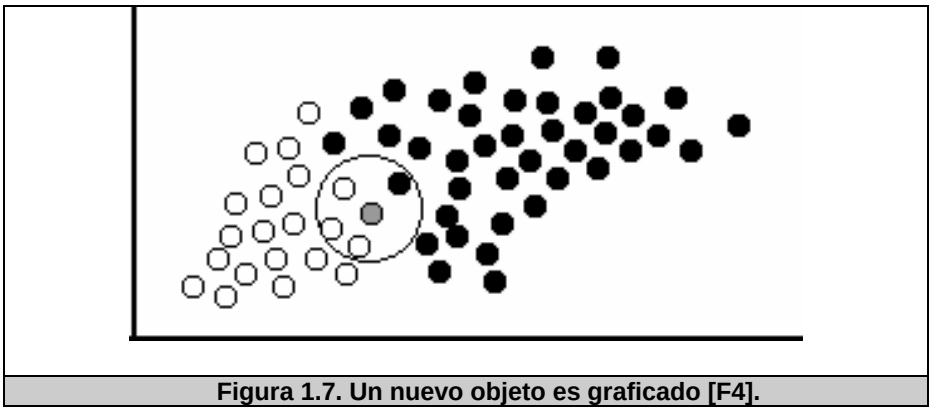


Figura 1.7. Un nuevo objeto es graficado [F4].

Ya que los objetos se encuentran bien agrupados, es razonable suponer que más objetos NEGROS (o BLANCOS) se encuentran en la vecindad de un objeto X y es más probable que los nuevos casos sean de uno de estos colores en particular. Para medir esta probabilidad, dibujamos un círculo alrededor del objeto X, que encierra un número (elegido previamente) de puntos sin tomar en cuenta su clase. Luego calculamos el número de puntos que se encuentran en el círculo por cada una de las clases; así, la probabilidad para la clase NEGRO:

$$\text{Probabilidad de X sea NEGRO} \propto \frac{\#\text{ de Obj. NEGROS en la vecindad de X}}{\text{Número Total de Objetos NEGROS}}$$

$$\text{Probabilidad de X sea NEGRO} \propto \frac{1}{40}$$

Y para la clase BLANCO:

$$\text{Probabilidad de X sea BLANCO} \propto \frac{\#\text{ de Obj. BLANCOS en la vecindad de X}}{\text{Número Total de Objetos BLANCOS}}$$

$$\text{Probabilidad de X sea BLANCO} \propto \frac{3}{20}$$

Comprobamos que la probabilidad de que un objeto X sea considerado de la clase BLANCO es mayor que la probabilidad de que sea considerado de la clase NEGRO, debido a que el círculo encierra a 1 objeto NEGRO y a 3 BLANCOS, como vimos en la Fig. 1.7.

A pesar de que las probabilidades previas indican que X puede ser perteneciente a la clase NEGRO (dado que hay más objetos NEGROS que BLANCOS) la probabilidad de vecindad indica lo contrario, que el objeto X es de la clase BLANCO. En los análisis bayesianos, la clasificación final es el resultado de la combinación de las dos fuentes de información; en este ejemplo, la probabilidad previa y la probabilidad por vecindad; forman una probabilidad posterior usando la regla de Bayes.

$$\text{Prob. posterior de X sea NEGRO} \propto$$
$$\text{Prob. Previa de NEGRO} \times \text{Prob. de X sea NEGRO}$$

$$\text{Prob. posterior de X sea NEGRO} = \frac{4}{6} \times \frac{1}{40} = \frac{1}{60}$$

$$\text{Prob. posterior de X sea BLANCO} \propto$$
$$\text{Prob. Previa de BLANCO} \times \text{Prob. de X sea BLANCO}$$

$$\text{Prob. posterior de X sea BLANCO} = \frac{2}{6} \times \frac{3}{20} = \frac{1}{20}$$

Finalmente, concluimos que el objeto X es parte de la clase BLANCO ya que obtiene una mayor probabilidad posterior.

El clasificador **Naive Bayes** puede ser usado con un número arbitrario de variables independientes. Dado un conjunto de variables, X = {x_1,x_2,x...,x_d}, se quiere construir la probabilidad posterior para el evento C_j, a partir de un conjunto de posibles respuestas C = {c_1,c_2,c...,c_d}. De una manera más sencilla, X es la variable predictora y C es el conjunto de categorías presentes en la variable dependiente. Usando la regla de Bayes:

$$p(C_j \mid x_1, x_2, ..., x_d) \propto p(x_1, x_2, ..., x_d \mid C_j) p(C_j)$$

Donde $p(C_j \mid c_1,c_2,c...,c_d)$ es la probabilidad posterior de una clase, por ejemplo la probabilidad de que X pertenece a la clase C_j. Naive Bayes asume que las probabilidades condicionales de las variables independientes son estadísticamente independientes, por tanto podemos descomponer la probabilidad de vecindad (likelihood) a un producto de términos:

$$p(X \mid C_j) \propto \prod_{k=1}^{d} p(x_k \mid C_j)$$

Luego procedemos a reescribir la probabilidad posterior como:

$$p(c_j \mid X) \propto p(C_j) \prod_{k=1}^{d} p(x_k \mid C_j)$$

Usando esta regla de Bayes, etiquetaremos un nuevo caso X con una clase C_j que obtenga la más alta probabilidad posterior.

A pesar de que el asumir que las variables predictoras son independientes no es siempre lo correcto, simplifica la clasificación de manera significativa, ya que permite que las densidades condicionales de las clases $p(X_k \mid C_j)$ puedan ser calculadas de manera separada para cada variable, reduciendo una tarea multidimensional a una tarea en una sola dimensión.

En efecto, Naive Bayes reduce una tarea de alta estimación de densidad dimensional a una de una dimensión. Además, las premisas parecen afectar a las probabilidades posteriores, en especial en regiones cercanas a los bordes de la decisión, dejando la tarea de clasificación intacta.

1.3.2.2.2 Support Vector Machines (SVM)

Es un clasificador basado en el principio de minimización del riesgo estructural de la teoría de aprendizaje computacional. La idea de este principio es encontrar una hipótesis de la cual podamos garantizar el menor error real. El error real es la probabilidad de que *h* cometa un error en un ejemplo no previsto y seleccionado de manera aleatoria. La meta del SVM es producir un modelo que prediga la clase de un objeto del cual sólo se tienen atributos que lo caracterizan [12].

Para que la técnica SVM pueda clasificar objetos, los objetos deben ser representados por puntos en un espacio multidimensional (mayor o igual a dos dimensiones); la intención del SVM es buscar la posibilidad de separar dichos puntos por un hiperplano (este hiperplano es conocido como una forma de clasificación lineal), generando dos regiones bien diferenciadas de puntos. El

requerimiento clave de SVM es separar los puntos de una manera limpia, esto es, buscando la mayor distancia de separación entre los puntos más cercanos de las dos regiones definidas por el hiperplano, ésta distancia es conocida como margen, como se muestra en la Fig.1.8.

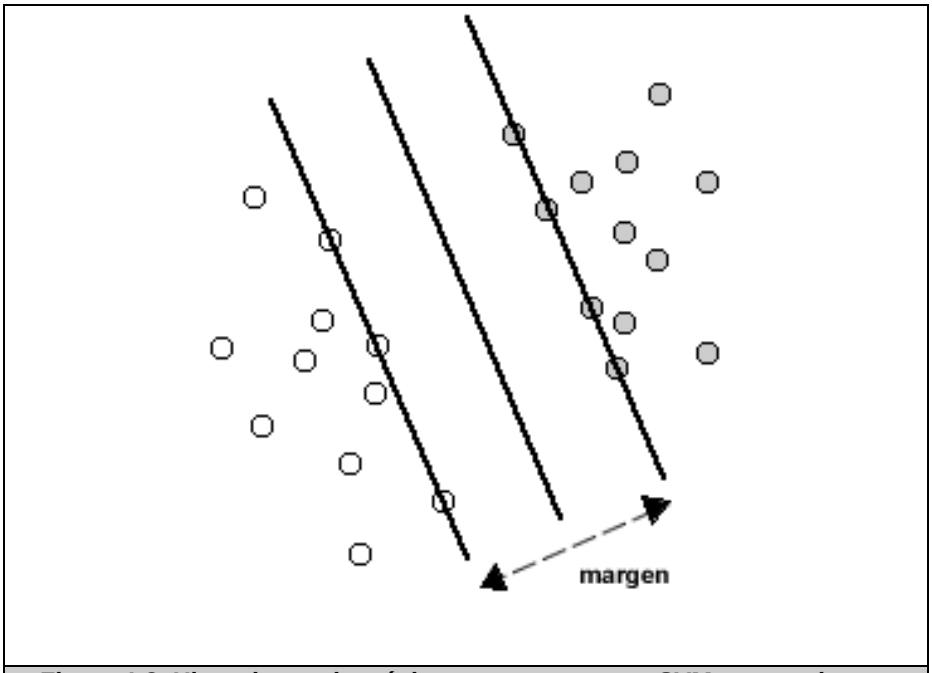


Figura 1.8. Hiperplanos de máximo margen para un SVM entrenado con ejemplos de dos clases [F5].

Este margen es el que permite ubicar un nuevo punto en el espacio, dado que la separación entre las regiones es grande. El hiperplano, si es que existe, es conocido como el hiperplano de máximo margen o el hiperplano óptimo, y los vectores más cercanos a este hiperplano son conocidos como vectores de soporte (support vectors).

Esta solución se aplica sólo a clasificadores lineales; posteriormente, se buscó extender este método de clasificación y generar nuevos clasificadores, con la principal característica de ser no lineales. Éstos se basan en el mismo principio, siendo la diferencia el tipo de funciones (llamadas funciones “núcleo”) usadas para separar las regiones, permitiendo así maximizar el margen impuesto por el hiperplano. Existen varios tipos de funciones núcleo que pueden utilizarse, entre ellos se encuentran: polinomial, basadas en radio y sigmoidal.

1.3.3 Indexación de documentos

Un índice es cualquier estructura de datos que mejora el proceso de búsqueda de un documento. Este índice puede ser generado a partir de: texto completo del documento, campos descriptivos (metadatos – datos sobre datos), o una combinación de las dos.

1.3.3.1 Extracción de metadatos

Existen una variedad de campos descriptivos (metadatos) que pueden ayudar a la descripción e identificación de un documento, como palabras claves, resumen, lenguaje, categoría, etc. Cada uno de estos campos permite dar mayor información sobre un documento

a un usuario final, evitando así la tarea de abrir y revisar un documento para determinar si es relevante o no a sus necesidades. Para obtener estos campos existen técnicas que los extraen a partir del texto del documento o los relacionan a valores definidos, sin requerir la intervención de algún experto humano.

Entre los campos descriptivos que son más frecuentemente usados se encuentran las palabras claves para describir un documento (muy usado por motores de búsqueda) y el resumen de documentos. A continuación describiremos ciertas técnicas para obtener estos campos.

1.3.3.1.1 Extracción de palabras claves

Las palabras claves son un conjunto de términos descriptivos de un documento, que dan una idea global de su contenido sin necesidad de revisar el texto completo. Son frecuentemente usadas en artículos académicos, donde además permiten categorizar los documentos o agruparlos fácilmente.

A pesar de su utilidad, son pocos los documentos que incluyen palabras claves que los describan. Muchos autores no las definen en sus artículos, a menos que sea obligatorio hacerlo. Extraer palabras claves manualmente es extremadamente difícil y consume mucho

tiempo, si la tarea no es realizada por el autor del texto. Por ello, es un proceso que necesita automatización.

Existen dos formas conocidas para automatizar el proceso de obtener palabras claves de un documento, las cuales son:

- Asignación

  Relaciona el texto del documento con algún conjunto de palabras claves ya definido en un vocabulario dado, este conjunto debe describir de la mejor manera al documento.

  Inicialmente se posee un gran conjunto de documentos de entrenamiento, cada frase del vocabulario es relacionado con un subconjunto de documentos y con ello se construye un clasificador de documentos. Cuando un documento nuevo es ingresado, se lo procesa con cada uno de los clasificadores de las frases definidas en el vocabulario, encontrando así una o varias frases a las cuales pertenezca.

  Como una gran desventaja es que sólo se pueden asignar palabras claves que han sido definidas por los documentos de entrenamiento; además se requiere de grandes cantidades de documentos de entrenamiento, lo cual hace que sea un método poco práctico.

- Extracción

  Obtiene las palabras claves de acuerdo a la relevancia dentro del texto, sin ningún conocimiento previo de un vocabulario. Se utilizan herramientas de análisis léxico y de extracción de información para extraer palabras que se encuentren dentro del documento y que puedan representarlo. También existen documentos de entrenamiento, pero los mismos son sólo utilizados para afinar los parámetros de los algoritmos de extracción. Esta técnica es la utilizada comúnmente.

  A continuación presentaremos dos métodos de extracción de palabras claves comúnmente utilizados:

  - Naive Bayes para entrenamiento y extracción de palabras claves [15]

    Este método recibe inicialmente documentos de ejemplo con sus palabras claves, definidas por un experto humano. Luego utiliza esos documentos para extraer sus palabras claves automáticas en base a su frecuencia y ubicación y las compara con las definidas por el autor, generando así un modelo de entrenamiento utilizado con los documentos nuevos a ser procesados.

    Cuando un documento nuevo es procesado, se extraen sus palabras claves tal como se lo realizó en el

entrenamiento y se clasifican utilizando el modelo generado.

- Información estadística de Co-ocurrencia de términos [16]

  Este método no necesita tener un conjunto de documentos de ejemplo inicial, sino que obtiene los términos del texto y analiza su co-ocurrencia entre ellos y los términos más frecuentes. La distribución de la co-ocurrencia determina la importancia de cierto término dentro del documento. Si un término aparece frecuentemente en un subconjunto particular de términos es muy probable que dicho término sea muy representativo para el documento.

1.3.3.1.2 Resumen automatizado de documentos

Se define a la generación de resúmenes (*summarization)* como el proceso de extraer la información más importante de una fuente (o fuentes), para producir una versión condensada, orientada a un usuario o tarea específica [17]. A medida que ha aumentado la información disponible, ha crecido el interés en el resumen automatizado de documentos realizado por un computador. El Procesamiento de Lenguaje Natural (NLP) es la rama de la

Inteligencia Artificial que trata el problema de la generación automática de resúmenes.

En general, podemos distinguir dos aproximaciones al problema, la *abstracción* y la *extracción.* La abstracción genera resúmenes más robustos, pero requiere efectuar un análisis semántico del texto, y en particular, un conocimiento profundo del contexto del mismo. Debido a esto, la gran mayoría de avances en el área se enfocan en la extracción, que únicamente requiere identificar los temas principales del texto, y retornar estos al usuario. A pesar de que el resultado no es necesariamente coherente, permite al lector formarse una opinión sobre el contenido del texto original.

Entre las aplicaciones para la generación automática de resúmenes podemos contar:

- Resúmenes de noticias en formato SMS o WAP para móviles o PDA.
- En motores de búsqueda, para presentar descripciones cortas de los resultados (por ejemplo, Google).
- En subscripciones a grupos de noticias basadas en palabras claves, cuyos boletines son resumidos y enviados al usuario [19].

- Para búsquedas en idioma extranjero, obteniendo una traducción automática de un resumen generado automáticamente.

1.3.3.2 Técnicas de indexación

1.3.3.2.1 Índices invertidos (inverted index)

Un archivo o índice invertido es una de las técnicas utilizadas para indexar una colección de documentos y acelerar la tarea de búsqueda. Es llamado de esta forma debido a que invierte la estructura tradicional de un documento: en un índice invertido, las palabras apuntan a los documentos que las contienen. Un índice invertido basa su funcionamiento en dos componentes: el *vocabulario* y las *ocurrencias*. El vocabulario es el conjunto de todas las palabras presentes en los documentos. Para cada palabra se crea una lista con las posiciones en el texto en las que aparecen. El conjunto de todas estas listas de posiciones se denomina “ocurrencias”. Podemos ver un ejemplo en la Fig. 1.9.

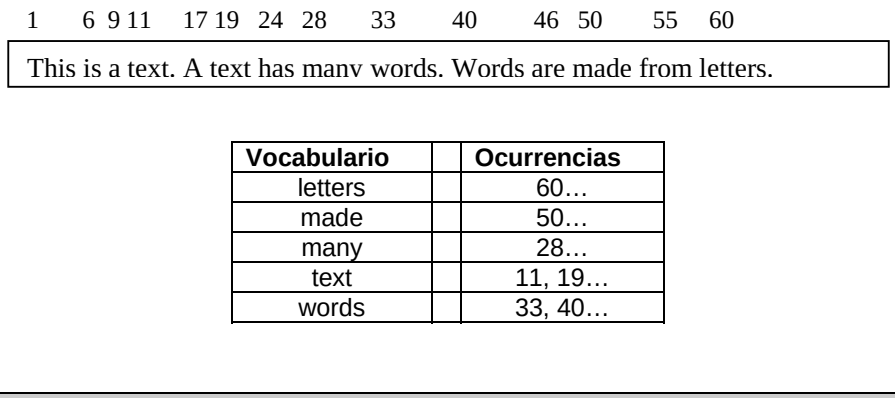


Figura 1.9. Texto de ejemplo y el índice invertido resultante. Vemos que no se indexan todas las palabras [F6].

Para optimizar el tamaño de los índices, se suele utilizar técnicas como el *direccionamiento por bloque*. En éste, el texto se divide en bloques, y las ocurrencias apuntan a éstos, en vez de apuntar a la posición exacta. Esta técnica permite reducir considerablemente el tamaño final de la estructura, a expensas de tener que realizar búsquedas lineales dentro del bloque para ciertas consultas (por ejemplo, de proximidad). Un índice invertido que incluye las posiciones exactas de las palabras en el texto se denomina *completo.*

La búsqueda en un índice invertido consta de tres pasos generales:

- *Búsqueda en vocabulario*. Las palabras y patrones presentes en la consulta se separan en palabras simples y son buscadas

en el vocabulario. Las frases o consultas de proximidad también son separadas.

- *Recolección de ocurrencias*. Se recolecta la lista de ocurrencias de todas las palabras que se ajustan a la consulta.
- *Manipulación de ocurrencias*. Las ocurrencias se procesan para resolver búsquedas de frases, de proximidad u operaciones booleanas. Puede ser necesaria una búsqueda directamente en el texto en caso de estar usando direccionamiento por bloque.

En general, la construcción y mantenimiento de un índice invertido es una tarea de bajo costo. En la práctica, se ha demostrado que tanto la búsqueda como el espacio de almacenamiento necesario pueden aproximarse a $O(n^{0.85)})$, donde n es el tamaño del texto. Así, los índices invertidos nos permiten tener tiempos de búsqueda y requerimientos de almacenamiento sublineales. Esto no es posible de alcanzar con otras técnicas de indexación.

1.3.3.2.2 Arreglos de sufijos (suffix arrays)

Dado que los índices invertidos asumen que el texto puede ser visto como una secuencia de palabras, restringen un tanto el tipo de consultas que pueden ser realizadas. Las búsquedas por frases, por

ejemplo, son costosas de realizar. En esta sección presentamos otra técnica de indexación, conocida como arreglo de sufijos. Este tipo de índices nos permite responder eficientemente a consultas más complejas.

El arreglo de sufijo toma el texto como una larga cadena de caracteres. Cada posición en el texto se toma como un sufijo (una cadena de caracteres que va desde esa posición en el texto hasta el fin del mismo). Cada sufijo se identifica de manera única por su posición en el texto. Un arreglo de sufijos no es más que un arreglo con los punteros a los sufijos, ordenados lexicográficamente. Podemos observar un ejemplo en la Fig. 1.10.

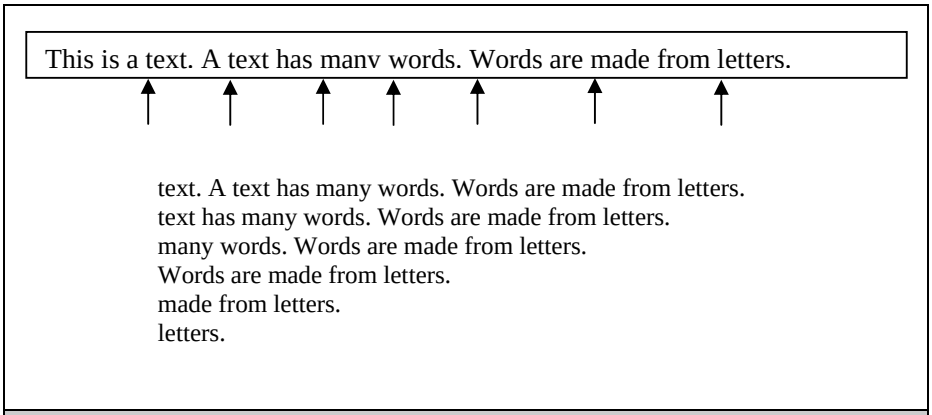


Figura 1.10. Texto de ejemplo con los puntos de interés marcados. Abajo, los sufijos correspondientes a los puntos de interés [F7].

Al igual que en la técnica anterior, vemos que algunas palabras no se toman en cuenta al establecer los puntos de interés, debido a que son muy comunes en el idioma del texto y no servirían de mucho al momento de realizar una búsqueda. Una vez obtenidos los sufijos,

solo restaría ordenarlos para obtener el arreglo sobre el que se realizarán las búsquedas.

La búsqueda en un arreglo de sufijos se realiza de la siguiente manera: la consulta origina dos “patrones de límite”, P1 y P2, y nos interesa cualquier sufijo S que cumpla la condición P1 <= S < P2. Realizamos una búsqueda binaria sobre el arreglo de sufijos para P1 y P2, y los elementos que se ubiquen entre las dos posiciones apuntarán a todos los sufijos que comiencen con el patrón que estamos buscando. Luego es cuestión de identificar el patrón en concreto.

Vemos que esta técnica de indexación se presta especialmente para búsquedas de frases, ya que estas se tratan como un patrón simple. En cambio, búsquedas booleanas o de proximidad requieren intersecciones o acceso directo al texto, igual que con los índices invertidos.

1.3.3.2.3 Archivos de firmas (signature files)

Los archivos de firmas son estructuras orientadas a palabras que basan su funcionamiento en la difusión (hashing). Ocupan poco espacio en disco, a costa de forzar una búsqueda secuencial sobre el

índice. A pesar de ser eficiente para textos de tamaño moderado, esta técnica se ve superada por los índices invertidos para la mayoría de aplicaciones.

Un archivo de firmas hace uso de una función de *hash* (la "firma"), que transforma palabras en máscaras de bits de tamaño B. Las palabras se agrupan en bloques lógicos de tamaño b. Estos bloques tienen asignada a su vez una máscara, que se calcula por medio de operaciones OR entre las palabras que la componen. Así, un archivo de firmas consiste en una secuencia de máscaras para cada uno de los bloques, junto con su posición en el texto. La idea principal consiste en que, si el bloque contiene el patrón de la búsqueda, los bits correspondientes al patrón estarán presentes en la máscara del bloque. Se muestra un ejemplo en la Fig. 1.11.

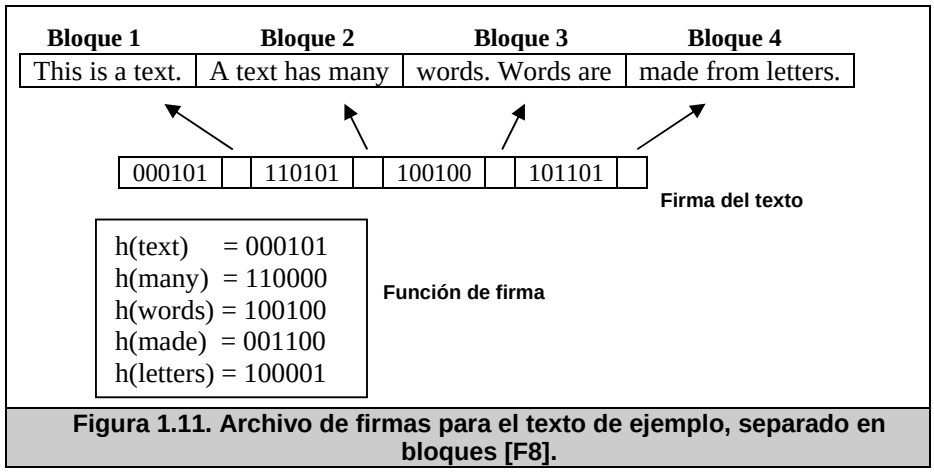


Figura 1.11. Archivo de firmas para el texto de ejemplo, separado en bloques [F8].

Es posible que los bits de la firma que estamos buscando estén presentes aún cuando el patrón no lo esté. En este caso tenemos lo que se conoce como un *false drop.* El porcentaje de *false drops* es inversamente proporcional al tamaño del índice. Debido a esto, una de las decisiones importantes al implementar esta técnica es mantener la probabilidad de errores lo suficientemente baja, sin descuidar el tamaño del índice.

La búsqueda en un archivo de firmas consiste en comparar la máscara de bits del patrón que queremos encontrar con la de los bloques de texto. Para una sola palabra, se realiza una operación AND entre la máscara de esta y la de los bloques, y así obtenemos una lista de candidatos posibles. Se requiere una búsqueda lineal en el texto para verificar que el patrón se encuentre realmente allí (a menos que se acepte el riesgo de un *false drop*). Para búsqueda de frases y de proximidad, el desempeño se ve mejorado, debido a que todos los bits de cada una de las palabras deben estar presentes, reduciendo la posibilidad de resultados falsos. La técnica de archivos de firmas es la única que mejora su rendimiento para el caso de búsqueda de frases.

Resumen

En este capítulo se presentó la razón que impulsó la realización de este trabajo, la gran abundancia de información que dificulta la obtención de contenido de calidad para el usuario. De igual manera se presentaron los objetivos de este trabajo y se realiza el planteamiento de la solución para el problema.

Posteriormente, se identificaron tres áreas que serán analizadas con sus métodos y herramientas e incorporadas a la solución. Estas son: exploración automática de páginas web, clasificación de documentos e indexación de documentos.

El siguiente capítulo realiza un análisis de cada una de las herramientas existentes y se determina la opción adecuada según nuestro problema.

CAPÍTULO 2

2 ANÁLISIS CONTEXTUAL

En el presente capítulo se establecen los requerimientos básicos necesarios para el funcionamiento adecuado de nuestro marco de trabajo. Se definen claramente los tres componentes que formarán la estructura del marco de trabajo y los requerimientos funcionales de cada uno de ellos.

Luego, procederemos a realizar un análisis de las alternativas existentes para cada uno de los componentes, y a la selección de la opción más apropiada de acuerdo a los requerimientos.

2.1 Requerimientos del Marco de Trabajo

Un marco de trabajo es una estructura de soporte, en la cual otro proyecto de software puede basarse. Los marcos de trabajo son diseñados para facilitar el desarrollo de software [3]; por tanto, la clave de un buen marco de trabajo es que sea escalable y fácil de usar.

En general, un buen marco de trabajo consta de los siguientes componentes [3]:

- **Arropadores**. Un arropador (wrapper) es una forma de empaquetar una función o conjunto de funciones con el fin de lograr los siguientes objetivos:
    - Simplificar la interfaz hacia una o varias tecnologías
    - Reducir o eliminar tareas repetitivas
    - Incrementar la flexibilidad de la aplicación por medio de la abstracción
    - Ser reutilizable independientemente de las consideraciones de diseño
- **Una arquitectura**. Podemos definir una arquitectura como un estilo que incorpora elementos de diseño específicos. En esencia, una arquitectura define relaciones entre objetos.

- **Una metodología**. En palabras sencillas, es una manera científica de hacer las cosas; esto es, consistente, repetible y respaldada en pruebas. Mientras la arquitectura maneja las asociaciones entre objetos, la metodología se ocupa de la *interacción* entre objetos. En general, una metodología:
    - Hace cumplir la adherencia a un enfoque consistente de diseño
    - Separa las dependencias entre objetos
    - Son frecuentemente reutilizadas sin importar los requerimientos de la aplicación

Dado que el marco de trabajo está orientado a resolver problemas sobre la exploración, clasificación e indexación de documentos digitales, hemos decidido separarlo en tres componentes, uno para cada tarea. De esta forma, cada componente tiene asignada una funcionalidad muy concreta, facilitando así la reutilización. Los componentes que forman el marco de trabajo son:

- Exploración automática de páginas web
- Clasificación de documentos
- Indexación de documentos

Estos componentes serán diseñados para funcionar de forma independiente. Con ello, un desarrollador podrá seleccionar, basado en sus requerimientos, la forma de usarlos y además podrá extender cada uno de los módulos a su gusto incorporando nuevas herramientas que se desarrollen.

A continuación, definiremos los requerimientos funcionales y no funcionales de nuestro marco de trabajo, especificando también el de cada uno de los componentes involucrados.

2.1.1 Requerimientos Funcionales

2.1.1.1 Exploración automática de páginas web

El agente explorador que será el primer componente descrito del marco de trabajo debe cumplir con las siguientes características:

- **Tópicos definidos**

  El agente tendrá acceso, como parámetro de inicio, a tópicos previamente definidos por el usuario. La definición de un tópico se lo realizará a través de la asignación de palabras claves. Con ello, el agente podrá realizar búsquedas iniciales para obtener así el listado de páginas que servirán de semillas para la búsqueda.

- **Conexión a buscadores comerciales**

  El agente necesitará inicialmente de un grupo de semillas a partir de las cuales empezará la exploración a través de la Web. Estas semillas serán obtenidas de manera dinámica cada vez que se inicie una exploración, por medio de una conexión con motores comerciales de búsqueda (por ejemplo, Google).

  Como parámetros de la búsqueda, se usarán los términos que identifican a cada uno de los tópicos previamente estructurados.

- **Conexión al evaluador**

  Al hacer una exploración, se irán obteniendo nuevas páginas, las cuales deberán ser evaluadas a fin de determinar su relevancia respecto al tópico bajo el cual se está realizando la búsqueda. Con este objetivo, el agente debe hacer uso de una herramienta evaluadora para obtener una calificación numérica que represente el grado de relevancia de una página con respecto al tópico en cuestión.

  Esta herramienta evaluadora es un factor crítico en el desempeño del agente explorador; por ello, debe mantenerse separada del mismo. Esta separación permite, de ser

necesario, modificar el mecanismo de evaluación sin alterar la estructura primaria del agente.

- **Adaptable**

  El agente puede ser adaptado a diferentes problemas o entornos gracias al establecimiento de diferentes configuraciones listadas a continuación:

  - Número máximo de niveles de exploración: es la máxima distancia que se permite al explorador alejarse de una semilla en búsqueda de un resultado relevante.
  - Número máximo de resultados: es la cantidad máxima de resultados a ser devueltos por el explorador al usuario.
  - Número máximo de páginas a visitar: es la cantidad máxima de páginas a ser exploradas por el agente en búsqueda de resultados relevantes.

2.1.1.2 Clasificación de documentos

El componente de clasificación de documentos debe cumplir con las siguientes características:

- **Tópicos definidos**

  La principal característica de un clasificador es tener definida una estructura a la cual puedan asociarse los diferentes

documentos ingresados al componente. Por ello, se debe definir tópicos representados por modelos previamente generados a partir de ejemplos positivos (documentos que son relevantes para el tópico) y negativos (documentos que no guardan relación con el tópico).

- **Clasificador binomial**

  El clasificador a usarse deberá ser de tipo binomial; es decir, distinguirá únicamente si un documento puede pertenecer a un tópico específico, sin importar si dicho documento pueda o no ser relevante para otro tópico presente en la estructura. Al utilizar un clasificador binomial, nos da la ventaja adicional de dar cabida a que un documento pertenezca a varios tópicos.

- **Asignaciones parciales**

  El método de clasificación usado debe soportar asignaciones parciales; es decir, el clasificador debe ser capaz de asignar valoraciones intermedias entre la relevancia total y la no relevancia; así, podemos establecer comparaciones entre documentos (cuán relevante es un documento en comparación a otros) usando la calificación arrojada por el clasificador.

- **Aprendizaje activo**

  El clasificador deberá soportar el aprendizaje activo [19]; es decir, deberá poder entrenarse a medida que va recibiendo retroalimentación del usuario. Con ello, al finalizar una búsqueda, el usuario será capaz de decirle al clasificador si los resultados obtenidos son o no relevantes para él, y así, ir ajustando el modelo a las preferencias o perfil del usuario en particular.

- **Preparación de un documento**

  Un documento antes de ser clasificado deberá pasar por ciertos filtros o herramientas de preparación del texto, a fin de que pueda ser procesado por el clasificador; estos filtros son:

  - Conversión a texto plano, eliminando el formato y otras características propias de cada tipo de documento.
  - Eliminación de las palabras más utilizadas en el idioma que no aportan contenido a una búsqueda (*stopwords*).
  - Segmentación de palabras a su raíz, también conocida como un *stem* o tema (stemming).

2.1.1.3 Indexación de documentos

El componente de indexación de documentos deberá tener las siguientes características:

- **Obtención de metadatos que describan al documento**

  Los metadatos son campos que delinean de una mejor manera el documento, y que pueden ser utilizados para realizar búsquedas locales. Entre los campos requeridos como metadatos se encuentran:

  - Palabras claves extraídas del texto del documento
  - Resumen del documento
  - Texto completo del documento
  - URL de donde fue obtenido el documento

- **Actualización**

  La estructura de índices generada con los documentos que el usuario desea conservar será actualizada a medida que el usuario vaya agregando nuevos documentos, creando un repositorio local.

- **Búsqueda local**

  Una vez construido el repositorio local, debe poder realizarse búsquedas sobre el mismo que estarán basadas en términos claves o campos de los metadatos.

- **Ordenación de acuerdo a relevancia**

  En los resultados enviados luego de una búsqueda, se deberá asignar una calificación a los documentos de acuerdo a la

relevancia que el documento tenga con respecto al término o términos utilizados en la búsqueda.

2.1.2 Requerimientos No Funcionales

El marco de trabajo además de funcionalidad, debe brindar escalabilidad y adaptabilidad, características fundamentales a la hora de determinar si es útil o no.

- **Escalabilidad**

  En cada uno de los tres módulos principales del marco de trabajo, el desarrollador deberá ser capaz de extender su funcionalidad, integrando nuevas herramientas o técnicas que en un futuro surjan para resolver el mismo problema, ya sea la búsqueda, clasificación o indexación. Estas nuevas herramientas serán desarrolladas con el mismo lenguaje de programación con el cual se desarrollará el marco de trabajo y que será definido posteriormente.

- **Adaptabilidad**

  Cada uno de los componentes podrá ser configurado según necesidades o requerimientos del desarrollador que los use, manteniéndose ajustable a los diferentes contextos en que los componentes pueden ser empleados.

Además, el desarrollador tendrá la libertad de usar el componente que le sea útil para su tarea, o integrar los componentes (y sus herramientas) de la manera que se requiera.

- **Portabilidad**

  El marco de trabajo deberá poder ser instalado en cualquier máquina que cumpla requerimientos mínimos de hardware y de software, definidos posteriormente durante la implementación de éste trabajo, logrando así una herramienta que pueda ser usada sin trabas o inconvenientes para el desarrollador. Cabe recalcar que el marco de trabajo será desarrollado con un lenguaje específico, haciendo sencilla su integración con proyectos desarrollados bajo el mismo lenguaje. En su defecto, la integración del marco de trabajo con otros lenguajes, se complica un poco; debiendo desarrollar puentes o servicios que sirvan de enlace entre las diferentes arquitecturas.

2.1.3 Casos de Uso

Los posibles casos de uso que tiene el marco de trabajo dependen de los componentes que sean incluidos en la aplicación final que se desarrolla sobre el marco de trabajo. A continuación, detallaremos

los casos de uso más generales contemplados por este marco de trabajo, no significa que sean los únicos.

2.1.3.1 Creación de repositorio local personalizado

En este caso de uso, se utilizarán los tres componentes del marco de trabajo, como se muestra en la Fig. 2.1. La aplicación cliente a desarrollarse sobre el marco de trabajo proveerá de un conjunto de direcciones de páginas web relacionadas a un tópico específico (las *semillas*), y usará las diferentes herramientas listadas en cada uno de los componentes para obtener como resultado un conjunto de documentos referentes a dicho tópico. Estos documentos serán almacenados localmente e indexados para búsquedas posteriores.

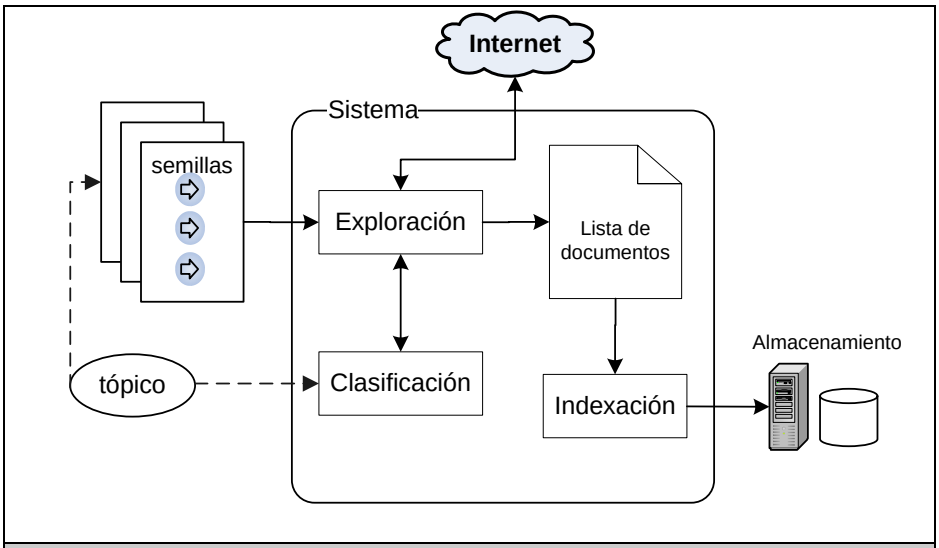


Figura 2.1. Esquema de caso de uso: Creación de repositorio local personalizado.

2.1.3.2 Clasificación de documentos

En este caso de uso, la aplicación cliente utilizará únicamente el componente de clasificación, como se muestra en la Fig. 2.2. Deberá proveer un conjunto de documentos a ser etiquetados para saber su relevancia en una categoría o no. Como resultado se obtendrán dos listas de documentos, una relevante a la categoría (ejemplos positivos), y una irrelevante a la categoría (ejemplos negativos).

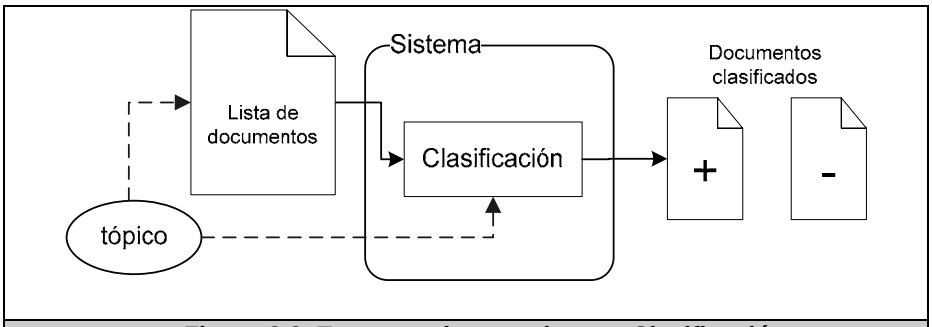


Figura 2.2. Esquema de caso de uso: Clasificación.

2.1.3.3 Exploración web personalizada

En este caso de uso, serán usados dos de los tres componentes del marco de trabajo, los componentes de exploración y clasificación, como se muestra en la Fig. 2.3. La aplicación cliente será responsable de proveer un listado de direcciones de páginas web relacionadas a un tópico específico, y usará las herramientas de los dos componentes para obtener como resultado documentos

relevantes al tópico ingresado. El marco de trabajo no será responsable de generar índices o mantener una copia local de los documentos resultantes.

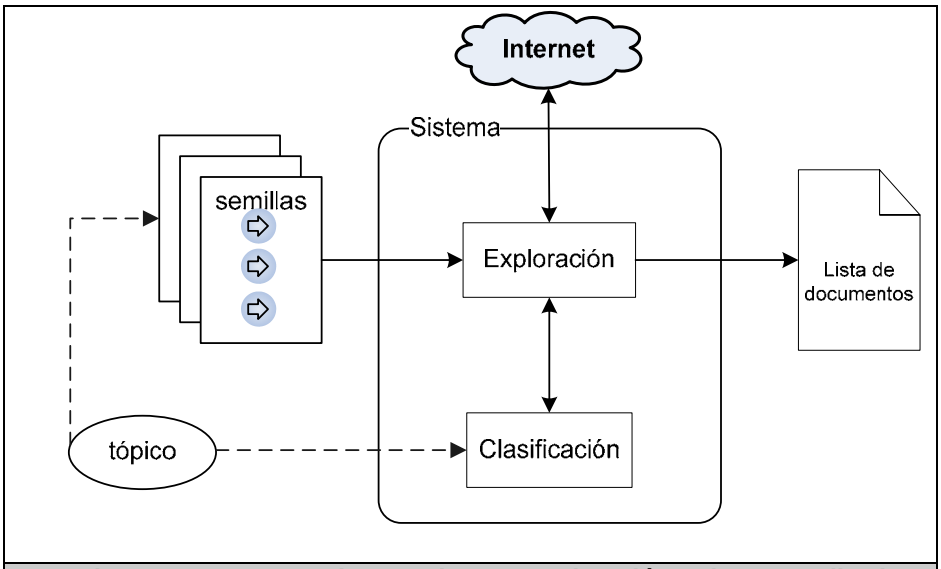


Figura 2.3. Esquema de caso de uso: Exploración web personalizada.

2.2 Análisis de alternativas y selección de la solución más apropiada

2.2.1 Exploración automática de páginas Web

Tomando en cuenta los requerimientos establecidos para este componente, podemos proceder a evaluar las diferentes opciones disponibles para la exploración automática de páginas web.

Una primera opción sería el algoritmo de **primero-en-anchura**. Éste tiene la ventaja de ser fácil de implementar y estar ampliamente

probado dado que proviene de la teoría de grafos. Este algoritmo es adecuado para tareas que requieren explorar todo un espacio de páginas web (los motores de búsqueda, que indexan todo el contenido disponible), pero tiene que ser descartado dado que no utiliza información para guiar la exploración. En este trabajo será referenciado como una guía para establecer una comparación frente a estrategias más sofisticadas.

Otra opción que tenemos es el algoritmo **primero-el-mejor**. Este ofrece amplias ventajas en relación al algoritmo **primero-en-anchura**, manteniendo una relativa facilidad de implementación. Además, permite variar considerablemente su funcionamiento simplemente alterando el mecanismo de evaluación de vínculos. Estas características, junto a la popularidad de la que goza en trabajos científicos [20] [21], avalan nuestra elección del algoritmo **primero-el-mejor** como el más adecuado para el marco de trabajo. Hemos establecido anteriormente que los algoritmos tipo “banco de peces” pueden ser vistos como una variante de primero-el-mejor; sin embargo en este trabajo no tomaremos en cuenta estas variantes, sino la versión general; conservando así una mayor flexibilidad.

2.2.2 Clasificación de documentos

Con los requerimientos ya definidos, procedemos a evaluar las opciones presentadas para resolver el problema del componente de clasificación de documentos.

La primera opción presentada es la clasificación por medio de Naive Bayes, conocido como “ingenuo”. Su gran ventaja es que es muy fácil de implementar, además su tiempo de ejecución es mucho menor comparado a otros algoritmos de clasificación, como el SVM. Trabaja muy bien sobre datos de texto o numéricos y a pesar de su “ingenuidad” al asumir independencia entre sus condiciones, se desempeña muy bien en la mayoría de situaciones reales [22]. Pero su gran desventaja es la disminución de su rendimiento cuando las condiciones están fuertemente relacionadas entre sí. [23].

Como segunda opción se presentó a la nueva generación de algoritmos de clasificación, representado por el SVM. La gran ventaja de éste algoritmo es la capacidad de resolver problemas lineales o no lineales, encontrando una solución en diferentes espacios dimensionales; y logrando un buen desempeño en la mayoría de problemas reales. Pero posee dos grandes desventajas, primero, el tiempo computacional requerido para resolver un problema es muy alto, por la cantidad y complejidad de los cálculos a

realizarse [24]; y segundo, su rendimiento se ve afectado por la elección de la función núcleo a ser utilizada de acuerdo al problema, conjuntamente con una estimación de los parámetros de la función para obtener su rendimiento óptimo [25]. Haciendo de éste, un algoritmo que requiere mucha más atención por parte del usuario para poder ser utilizado.

Por ello, hemos elegido a **Naive Bayes**, debido a su simplicidad, facilidad de implementación, velocidad y capacidad de trabajar independientemente de la retroalimentación del usuario, y que se ajusta más a la capacidad de enseñanza activa, al poder modificar el modelo fácil y rápidamente, cada vez que un usuario ingrese nuevos documentos positivos o negativos para un tópico específico.

2.2.3 Indexación de documentos

Para la generación de un índice de documentos se requiere tener una representación de los mismos. Como se definió en el análisis, se extraerán metadatos que describan el documento, los cuales serán: palabras claves, resumen, texto completo y dirección web de ubicación del documento original. Luego de generados los metadatos, se procederá a la construcción del índice.

2.2.3.1 Extracción de Metadatos

La extracción de metadatos no representa una parte clave en la solución propuesta al problema descrito en el presente trabajo; más bien, es una herramienta complementaria para el usuario. Por tal motivo, seleccionaremos el mejor algoritmo disponible en cada rama de acuerdo a investigaciones previas, que nos permita generar los metadatos requeridos.

Para generar las palabras claves de un documento, hemos decidido incluir en el marco de trabajo el método TF/IDF [26]. El método de **Naive Bayes** adaptado a la extracción de palabras claves necesita generar un modelo, esto es, se necesita un entrenamiento previo para cada uno de los tópicos definidos, convirtiéndose ésta en su mayor desventaja. Mientras que el método TF/IDF extrae las palabras claves sin requerir la generación de un modelo previo o entrenamiento con documentos similares, simplemente calculando la relevancia de cada término dentro de su documento, y así estableciendo su gran ventaja.

En cuanto al resumen automatizado de documentos, el método de la abstracción a pesar de su gran potencial para obtener resúmenes relevantes, no presenta al momento el suficiente desarrollo como

para poder ser incluido en el marco de trabajo. Por lo tanto, el método de la extracción se presenta como la opción más viable debido a que ha sido probada y la calidad de los resultados que arroja es adecuada para su uso dentro del marco de trabajo.

2.2.3.2 Representación de datos

Probablemente la mejor opción para la gran mayoría de aplicaciones es la de índices invertidos. Como mencionamos en el capítulo anterior (1.3.3.2), las otras técnicas de indexación presentan detalles ocultos y consideraciones especiales al momento de llevarlas a la práctica. Esto que ocasiona que sean más difíciles de usar y menos eficientes, a la vez que son menos flexibles en el momento de lidiar con nuevos tipos de búsqueda. Por lo tanto, consideramos que la técnica de índices invertidos es la que más se ajusta a los requerimientos establecidos para el marco de trabajo, ya que ofrece un rendimiento adecuado en un gran número de situaciones, sin sacrificar flexibilidad. Sin embargo, las otras técnicas de indexación mantienen su validez para aplicaciones específicas (por ejemplo, en bases de datos genéticas, para el caso de arreglos de sufijos, etc.).

Resumen

En este capítulo se presentaron los requerimientos funcionales y no funcionales del marco de trabajo a ser desarrollado. Además, se

presentaron los casos de uso comunes para nuestro marco de trabajo, los cuales varían dependiendo de las componentes a utilizarse. Los casos de uso presentados son: creación de repositorio local personalizado, clasificación y exploración.

Se definieron los tres componentes que forman su estructura: exploración, clasificación e indexación. Se analizaron las alternativas de solución que pueden aplicarse a cada uno de los componentes del marco de trabajo y se procedió a elegir la mejor según el problema identificado y los requerimientos especificados.

En el siguiente capítulo se explica todo lo referente al diseño del marco de trabajo, con las respectivas herramientas integradas para cumplir los requerimientos especificados en este capítulo.

CAPÍTULO 3

3 DISEÑO DEL MARCO DE TRABAJO

En el presente capítulo se explica todo lo referente al diseño del marco de trabajo, basándose en el conjunto de requerimientos definidos en los capítulos anteriores. Se definen de manera explícita detalles referentes a la arquitectura y patrones de diseño aplicados en el marco de trabajo.

A continuación se detalla de forma gráfica los tres componentes principales que forman el marco de trabajo, describiendo sus características, configuración e interfaces de comunicación.

3.1 Arquitectura

El marco de trabajo estará diseñado de manera modular, como se aprecia en la Fig. 3.1, lo cual permitirá identificar claramente los tres principales componentes y usarlos según los requerimientos de la aplicación cliente. Además, se facilitará la tarea de agregar o desarrollar nuevas herramientas o enfoques de solución a uno o varios de los componentes principales, independiente uno del otro; haciendo de éste un proceso transparente para los demás componentes del marco de trabajo.

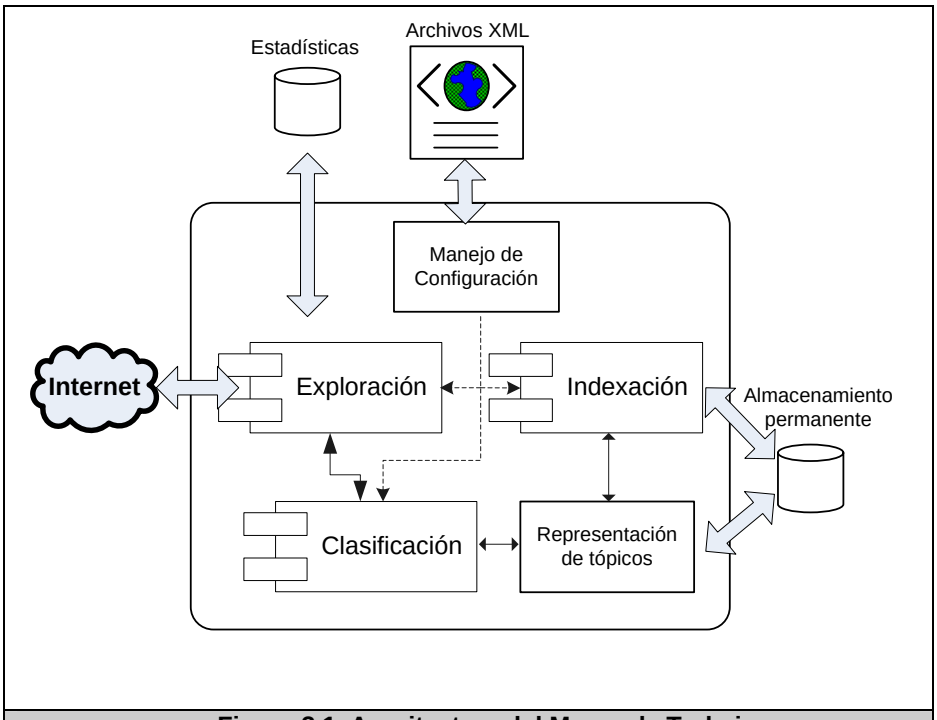


Figura 3.1. Arquitectura del Marco de Trabajo.

A continuación explicaremos de manera general los tres componentes principales:

- **Componente de Exploración**

  El componente de exploración es el encargado de buscar páginas o documentos relacionados con un tópico específico a través de la Web. El componente tendrá conexión a un buscador comercial el cual, a través de palabras claves, retornará las páginas web iniciales referentes al tópico (semillas). Las páginas iniciales serán analizadas a fin de obtener vínculos a nuevos recursos similares; este proceso de análisis y extracción de nuevos vínculos se repite iterativamente hasta tener un máximo definido de recursos referentes a un mismo tópico. Los recursos obtenidos serán calificados de acuerdo a su relevancia para el tópico, por tal motivo, se deberá establecer una conexión con el componente de clasificación.

- **Componente de Clasificación**

  El componente de clasificación es el encargado de determinar si un documento es relevante para un tópico específico. Para ello se utilizarán tópicos definidos con documentos positivos y negativos, los cuales servirán para el entrenamiento del algoritmo. El componente recibirá el contenido de un recurso

a ser clasificado, el cual será preparado previo a su análisis. El algoritmo de clasificación puede ser adaptado al perfil del usuario, y así mejorar progresivamente su tasa de aciertos.

- **Componente de Indexación**

  El componente de indexación es el encargado de obtener metadatos descriptivos de un documento y luego agregarlos al índice creado para futuras búsquedas.

- **Componentes Auxiliares**

  Entre lo componentes auxiliares se encuentran: Componente de Configuración y Componente de Representación de Tópicos.

  El Componente de Configuración es el encargado de administrar de forma centralizada la configuración de los tres componentes principales, mencionados anteriormente. Esta configuración será realizada a través de archivos XML.

  El Componente de Representación de Tópicos es el encargado de crear la estructura de los tópicos, definiendo sus palabras claves y sus documentos de entrenamiento (positivos y negativos).

3.2 Patrones de Diseño

Los patrones de diseño son soluciones existentes a problemas comunes encontrados al momento de crear sistemas complejos. Estos patrones fueron propuestos anteriormente como soluciones a un problema similar y luego fueron probados en sistemas diferentes donde se presenta el mismo problema; al ser la mejor solución posible se convirtieron en un estándar de diseño.

Los patrones de diseño que van a emplearse dentro del marco de trabajo propuesto, son los siguientes:

- **Método de Fabricación (Factory)**

  Devuelve diferentes implementaciones de una interfaz de acuerdo a parámetros especificados en tiempo de ejecución. Este patrón es usado en los componentes de Exploración y de Clasificación, ya que se usan diferentes implementaciones de algoritmos de solución.

- **Adaptador (Adapter)**

  Adapta clases ya existentes a una nueva interfaz. Este patrón es usado en el componente de Clasificación, donde se usan librerías de clasificación y de preparación de texto existentes en la actualidad.

- **Arropador - (Wrapper)**

  Añade funcionalidad a instancias particulares de una clase. Este patrón es implementado en el componente de Clasificación, donde la clase utilizada para la preparación del texto puede integrar funciones de segmentación de palabras (stemming) y eliminación de palabras más utilizadas que no aportan contenido a una búsqueda (stopwords).

- **Observador (Observer)**

  Permite a un objeto mantener informados a otros objetos de cambios en su estado. Este patrón de diseño es usado dentro del componente de Exploración de páginas web, el cual debe informar sobre el estado de la exploración (cuando ésta haya finalizado).

- **Método Plantilla (Template Method)**

  Provee una definición abstracta de un método o una clase y luego modifica su comportamiento sin cambiar su estructura. El patrón es implementado dentro del componente de Exploración de páginas web, como una forma de delinear los pasos del proceso de exploración.

3.3 Componentes

Como se mencionó anteriormente, el marco de trabajo estará formado por tres componentes principales: Exploración de páginas web, Clasificación de documentos e Indexación de documentos. A fin de seguir estándares internacionales, se ha decidido utilizar el idioma inglés para el nombre de los componentes, subcomponentes y programación en general. A continuación detallaremos el diseño de cada uno de los componentes.

3.3.1 Exploración automática de páginas Web

3.3.1.1 Diagramas lógicos

El componente de Exploración de páginas web estará formado por seis subcomponentes que procesarán los requerimientos de la aplicación cliente, como observamos en la Fig. 3.2. A continuación describiremos cada uno de los subcomponentes:

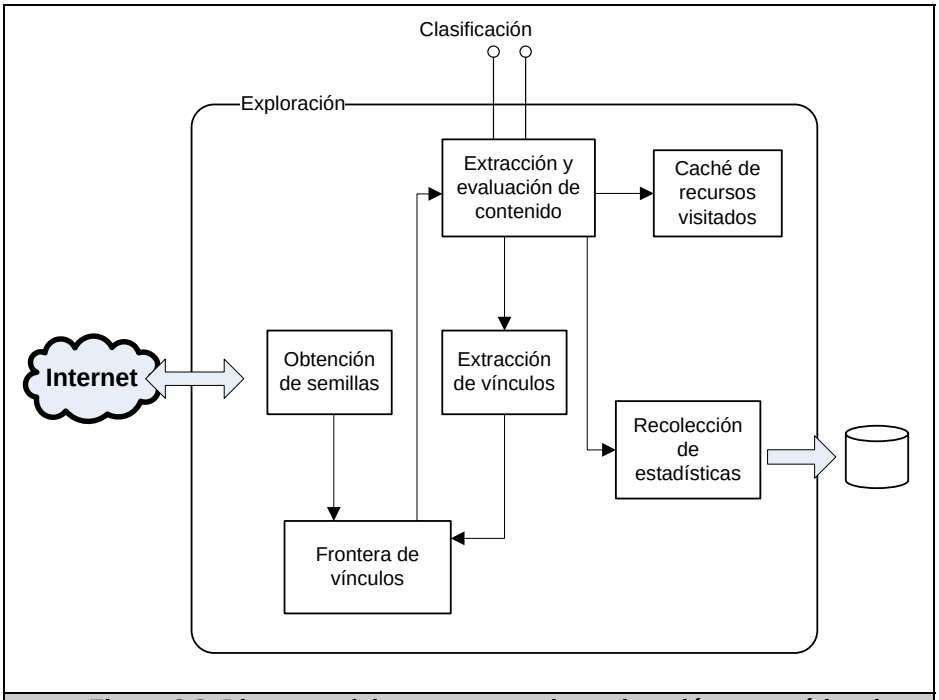


Figura 3.2. Diagrama del componente de exploración automática de páginas web.

- **Obtención de semillas**

  Este subcomponente toma las palabras claves de un tópico que proporciona la aplicación cliente, y las utiliza para obtener los vínculos desde los cuales se iniciará el proceso de exploración. Para ello utiliza conexiones con buscadores comerciales (por ejemplo, Google).

- **Extracción y evaluación de contenido**

  Es el encargado de extraer el contenido del recurso encontrado en Internet, de su conversión a texto plano y de la estimación de su importancia con respecto al tópico actual;

esta evaluación es realizada por medio de una conexión con el componente de clasificación.

- **Extracción de Vínculos**

  Una vez que se ha determinado la relevancia de un recurso, se procede a obtener vínculos (direcciones web) que apunten a recursos todavía no visitados por el agente explorador.

- **Caché de recursos visitados**

  Utilizado para almacenar temporalmente los recursos que ya han sido analizados por el agente explorador, a fin de asegurarse de no volver a procesarlos. Los resultados que se retornan a la aplicación cliente al final del proceso son obtenidos de este almacenamiento temporal, para así evitar tener que descargarlos nuevamente del Internet. Los recursos en este caché se mantienen ordenados de acuerdo a su relevancia.

- **Frontera de Vínculos**

  Es el encargado de almacenar los vínculos a los recursos aún no procesados por el componente. Estos vínculos se mantienen ordenados de acuerdo a su relevancia con respecto al tópico seleccionado por la aplicación cliente.

- **Recolección de estadísticas**

  Es el encargado de almacenar estadísticas de la exploración, como páginas visitadas, documentos obtenidos, etc.

Para un mayor entendimiento del funcionamiento del componente de Exploración Automática de Páginas Web, explicaremos su flujo de trabajo, que se presenta en la Fig. 3.3.

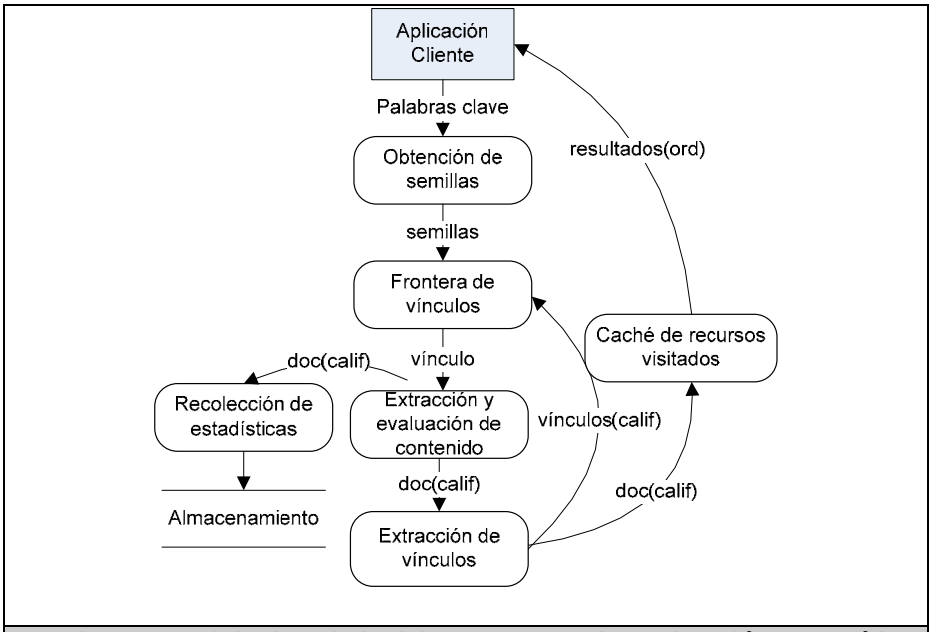


Figura 3.3. Flujo de trabajo del componente de Exploración Automática de Páginas Web.

El proceso se inicia con las palabras claves que provee la aplicación cliente; con ellas el subcomponente de obtención de semillas realiza una consulta a través de un motor comercial de búsqueda y obtiene los recursos web iniciales para la exploración (semillas). Las semillas son insertadas en la frontera de vínculos, y cada una será

procesada por el subcomponente de Extracción y Evaluación de Contenido, encargado de visitar cada una de los vínculos de la frontera y obtener su contenido.

Este subcomponente también analiza el texto completo de cada uno de los recursos web que se encuentran en la frontera y le asigna una calificación de acuerdo a su relevancia para el tópico escogido; en caso de ser necesario, el componente de recolección de estadísticas puede almacenar datos sobre la exploración. A continuación, el recurso web que está siendo procesado pasa al subcomponente de Extracción de Vínculos, el cual obtiene referencias a otras páginas relacionadas, las cuales serán exploradas posteriormente. Estos nuevos vínculos son insertados en la frontera, asegurándose que ninguno de estos se encuentre presente en el caché de recursos visitados. Luego, el recurso con su respectiva calificación pasa al caché de recursos visitados, el cual va generando una lista de resultados, ordenados de acuerdo a su relevancia. Al finalizar el proceso de exploración, ésta es la lista devuelta como resultado a la aplicación cliente.

Los subcomponentes estarán representados dentro del sistema mediante el diagrama de clases mostrado a continuación, Fig. 3.4.

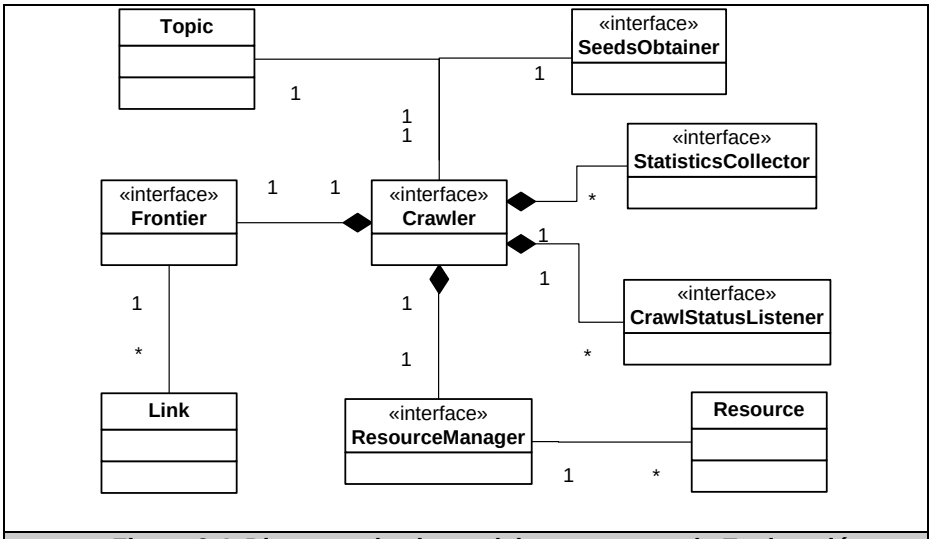


Figura 3.4. Diagrama de clases del componente de Exploración Automática de Páginas Web.

3.3.1.2 Interfaz y Configuración

El componente de Exploración Automática de Páginas Web tiene sus propias características que definen su interfaz y configuraciones internas. A continuación, detallaremos las entradas del componente, sus salidas, y los parámetros de configuración que requiere:

- **Entradas**
  - *Tópico*. Sobre el cual se realizará la exploración. Cada tópico tiene asociados palabras clave, usadas para la obtención de semillas, y un clasificador usado para estimar la relevancia de los resultados.

- *Configuración del Clasificador*. Archivo de configuración a partir del cual se construirá el clasificador a usarse durante la exploración.

- **Salidas**
  - *Lista de resultados*. Nuevos recursos encontrados durante la exploración, ordenados de acuerdo a su relevancia.

- **Parámetros de configuración**
  - *Número de semillas (entero)*. El número máximo de vínculos que se insertarán en la frontera al momento de iniciar la exploración. Utilizado por el componente de obtención de semillas.
  - *Distancia máxima desde la raíz (entero)*. Distancia máxima permitida desde una semilla hasta un recurso cualquiera (medida en número de vínculos). Usado en el componente de extracción de vínculos, para no extender infinitamente la exploración. *Valor recomendado*: valores mayores a 3 incrementan la complejidad sin mejorar necesariamente la calidad de los resultados [27].
  - *Tamaño máximo de la frontera (entero)*. Número máximo de vínculos pendientes que pueden ser

almacenados en la frontera. *Valor recomendado*: sujeto a los recursos de sistema disponibles.

- *Número máximo de recursos a visitar (entero)*. Máximo de recursos web que serán analizados. El parámetro más adecuado a modificar cuando se tienen restricciones de tiempo. *Valor recomendado*: sujeto a los recursos de sistema disponibles y a restricciones de tiempo.
- *Número de resultados (entero)*. Máximo de resultados que serán retornados a la aplicación cliente. *Valor recomendado*: a discreción de la aplicación.
- *Número de hilos (entero)*. Número de hilos de ejecución utilizados por el agente explorador. *Valor recomendado*: a discreción de la aplicación.

El componente de Exploración Automática de Páginas Web define cuatro interfaces básicas: Crawler, SeedsObtainer, Frontier y ResourceManager. A continuación realizaremos una descripción de estas interfaces y sus métodos principales.

- **Crawler**

  La interfaz principal del componente, encargado de coordinar la interacción entre los subcomponentes y la

aplicación cliente. Ejerce las funciones de controlador para las demás interfaces. Sus funciones principales son:

- *String getStatus()*. Devuelve el estado actual de la exploración; uno de los siguientes: “NOT_READY, INACTIVE, ACTIVE, PAUSING, PAUSED, RESUMING, FINISHING, FINISHED”.
- *List getResults()*. Devuelve la lista de resultados (recursos) obtenidos luego de que el proceso de exploración ha finalizado. La lista devuelta es formada por instancias de la clase Resource.
- *void startCrawling(Topic t, String classifierConfig)*. Inicia el proceso de exploración en base al tópico definido, usando el clasificador definido por el archivo de configuración.
- *void pauseCrawling()*. Detiene temporalmente el proceso de exploración.
- *void resumeCrawling()*. Reanuda un proceso de exploración que ha entrado en estado de pausa.
- *void stopCrawling()*. Detiene el proceso de exploración.

- **SeedsObtainer**

  Encargado de la obtención de semillas que servirán para iniciar la exploración. Tiene un método principal:

- *List obtainSeeds(List keywords)*. Devuelve la lista de vínculos que serán las semillas (requisito inicial de cualquier técnica de exploración), a partir de una lista de palabras claves de un tópico.

- **Frontier**

  Administra la lista de vínculos pendientes de visitar por el agente explorador. Dependiendo de la implementación, los vínculos pueden estar ordenados de acuerdo a la calificación asignada por el clasificador. Sus funciones principales son:

  - *Link getNextLink()*. Retorna el siguiente vínculo a analizar por el explorador.
  - *boolean add(Link l)*. Añade un vínculo a la frontera. Retorna verdadero si el vínculo pudo ser añadido a la frontera, y falso en caso contrario.
  - *boolean contains(Link l)*. Retorna verdadero si un vínculo ya se encuentra contenido en la frontera.
  - *boolean isEmpty()*. Retorna verdadero si la frontera no contiene vínculos.

- **ResourceManager**

  Administra la lista de recursos ya visitados por el agente explorador. Los vínculos se mantienen ordenados de

acuerdo a la calificación asignada por el clasificador. De este componente se obtiene la lista de resultados que devolverá el agente al final de su ejecución. Define los siguientes métodos principales:

- *boolean contains(String url)*. Retorna verdadero si el recurso identificado con ese URL ya se encuentra en el administrador.
- *boolean add(Resource resource)*. Añade un recurso descubierto por el agente. Retorna verdadero si el recurso pudo ser insertado correctamente.
- *Resource getFirstResource()*. Devuelve el recurso ubicado al tope de la lista, sin removerlo del administrador.
- *boolean isEmpty()*. Retorna verdadero si el administrador no contiene recursos.

3.3.2 Clasificación de documentos

3.3.2.1 Diagramas lógicos

El componente de Clasificación de documentos estará formado por cuatro subcomponentes que procesarán los requerimientos de la aplicación cliente, como observamos en la Fig. 3.5.

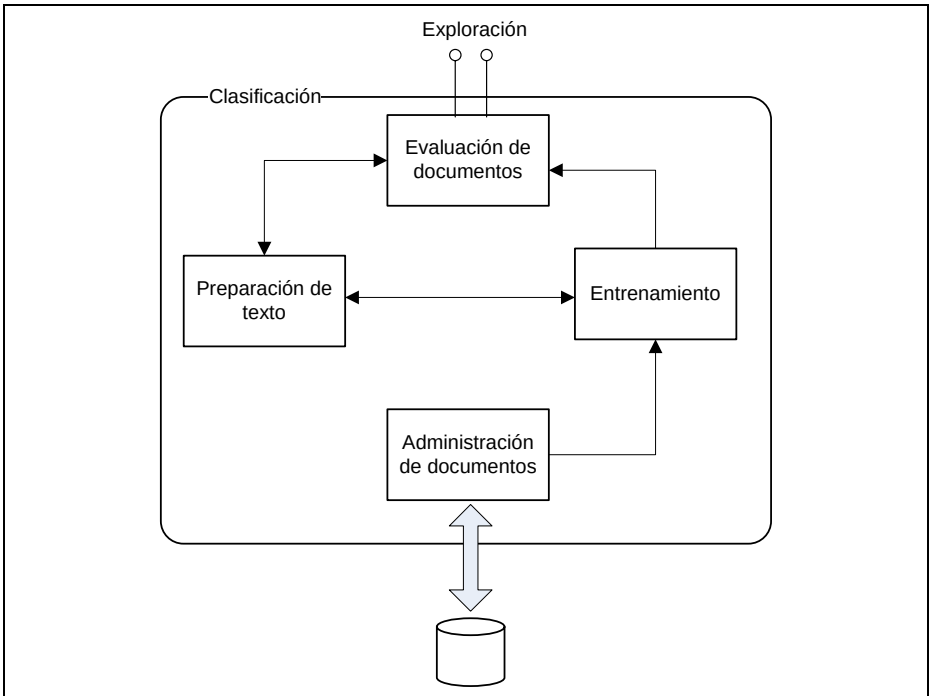


Figura 3.5. Diagrama del componente de clasificación de documentos.

A continuación describiremos cada uno de los subcomponentes:

- **Preparación de texto**

  Es el subcomponente encargado de procesar el texto original de un documento previo a su análisis. Inicialmente, se eliminan las palabras más utilizadas en el idioma que no aportan contenido a una búsqueda (stopwords), y luego, se segmentan palabras reduciendo sus variantes a la forma léxica canónica (stemming). A continuación, en la Fig. 3.6 mostramos un ejemplo de la aplicación de éste subcomponente sobre un texto prueba.

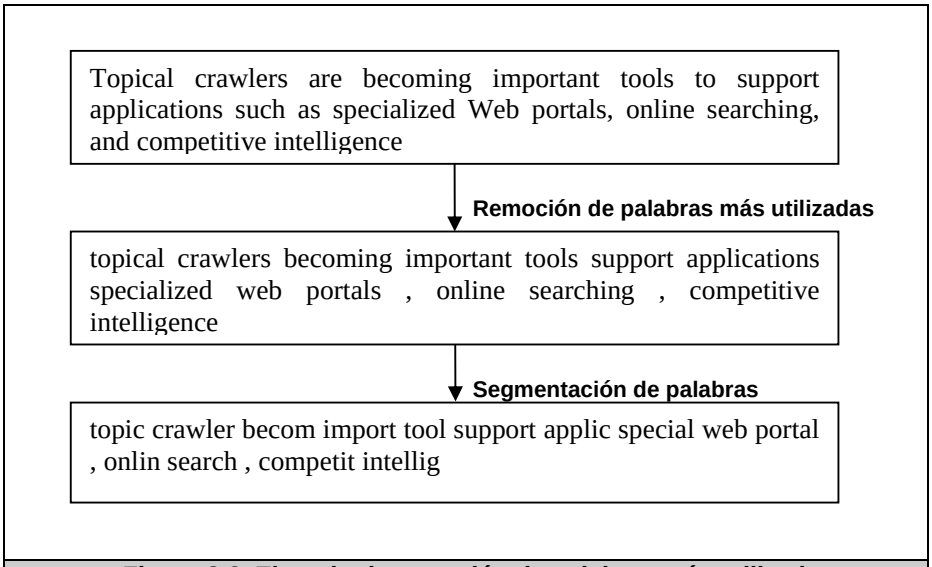


Figura 3.6. Ejemplo de remoción de palabras más utilizadas y segmentación de palabras.

- **Entrenamiento**

  Es el encargado de instruir al algoritmo de clasificación para su correcto funcionamiento, basándose en ejemplos negativos y positivos con relación al tópico.

- **Evaluación de documentos**

  Subcomponente que se encarga de la aplicación del algoritmo de clasificación sobre el documento; producto de este proceso se obtiene una calificación proporcional a la relevancia del recurso para el tópico escogido.

- **Administración de documentos**

  Es el encargado de procesar el texto previamente preparado y representarlo de una manera entendible para una máquina, utilizando un modelo estructurado de representación.

Para un mayor entendimiento del funcionamiento del componente de Clasificación de documentos, explicaremos su flujo de trabajo, que se presenta en la Fig. 3.7.

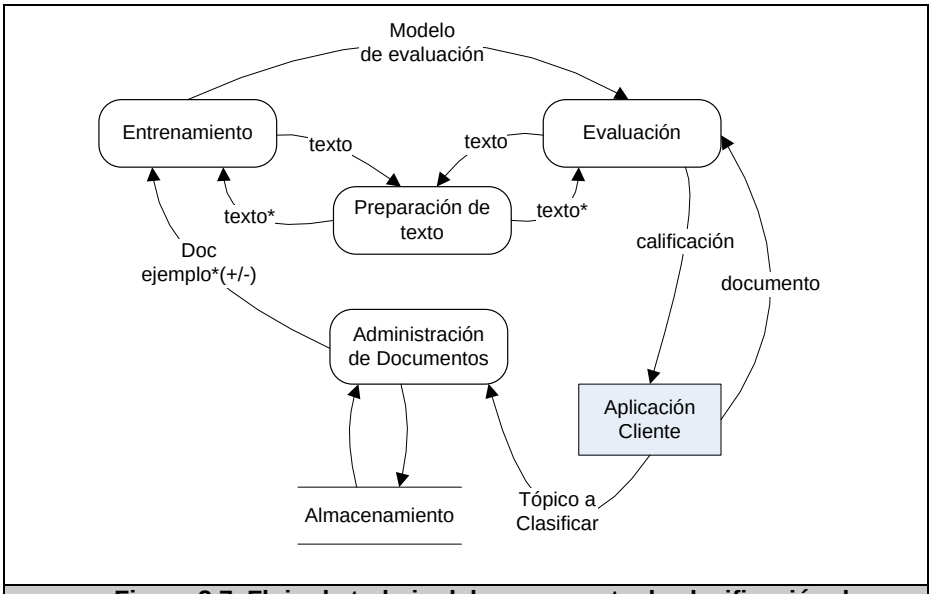


Figura 3.7. Flujo de trabajo del componente de clasificación de documentos.

El proceso se inicia con el envío de documentos positivos y negativos de un tópico por parte del componente de Administración de documentos, hacia el subcomponente de Entrenamiento, el cual,

analiza el texto del documento utilizando las herramientas de Preparación de Texto para posteriormente representarlo de una manera estructurada. Finalmente, el subcomponente de Entrenamiento genera el modelo representativo del tópico a clasificar.

Luego de generado el modelo de evaluación, el proceso continúa con la recepción del documento a clasificar; éste pasa al subcomponente de Evaluación, el cual hace uso de las herramientas de Preparación de texto para procesarlo, eliminando palabras no representativas y segmentando las palabras restantes.

A continuación, se evalúa el documento estructurado con ayuda del modelo de evaluación obtenido del subcomponente de Entrenamiento; como resultado de este proceso obtenemos una valoración numérica para el documento, proporcional a su relevancia con respecto al tópico escogido.

Los subcomponentes son representados dentro del sistema mediante el diagrama de clases mostrado a continuación, Fig. 3.8.

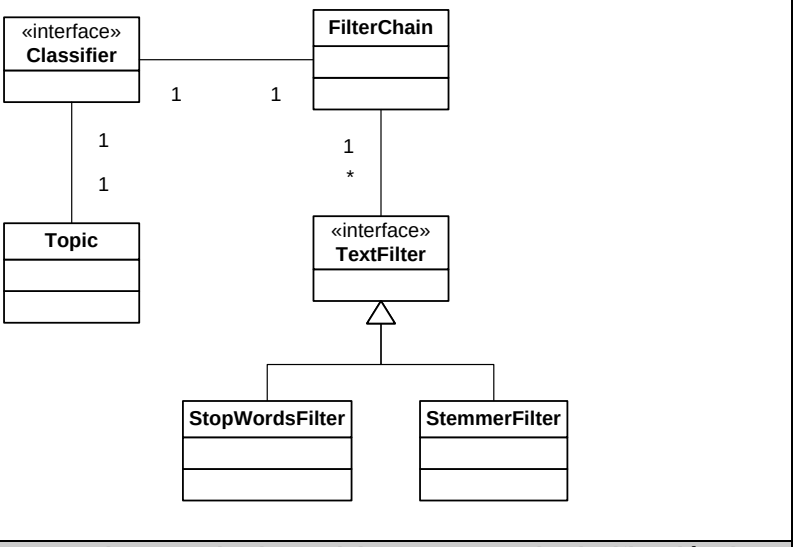


Figura 3.8. Diagrama de clases del componente de clasificación de documentos.

3.3.2.2 Interfaz y Configuración

El componente de clasificación de documentos tiene sus propias características que definen su interfaz y configuraciones internas. A continuación, detallaremos las entradas del componente, sus salidas, y los parámetros de configuración que requiere:

- **Entradas**
  - *Tópico con ejemplos*. Ejemplos positivos y negativos asociados con el tópico escogido para la clasificación.
  - *Configuración del Clasificador*. Archivo de configuración a partir del cual se construirá el clasificador a usarse durante la exploración.

- **Salidas**
    - *Calificación*. Valoración numérica que representa la relevancia del documento con respecto al tópico. Permite establecer comparaciones entre documentos.
    - *Categoría.* Alternativa a la calificación, para los casos en que únicamente se necesite determinar si un documento pertenece o no al tópico.
- **Parámetros de configuración**
    - *Lenguaje y esquema de preparación de texto.* Se indica la lista de filtros a ser usados en la preparación del texto, además del lenguaje de los documentos a ser procesados. Este parámetro requiere ser ingresado al momento de creación, a fin de asegurar que el proceso de preparación del texto sea el mismo tanto para el entrenamiento como para la evaluación de documentos.
    - *Directorio de almacenamiento.* Lugar en el que serán almacenados los datos necesarios para el funcionamiento del clasificador.

El componente de clasificación de documentos define dos interfaces, Classifier y TextFilter. A continuación realizaremos una descripción de estas interfaces y sus métodos principales.

- **Classifier**

  Es la interfaz principal, debe implementar un algoritmo de clasificación; para ello, se han definido las siguientes funciones:

  - *void startLearner.* Inicializa el clasificador y empieza en entrenamiento con los ejemplos positivos y negativos del tópico recibido.
  - *boolean addPositiveExample(String content, Topic topic).* Añade el documento como un ejemplo positivo dentro de un tópico definido.
  - *boolean addNegativeExample(String content, Topic topic).* Añade el documento como un ejemplo negativo dentro de un tópico definido.
  - *double score(String content, Topic topic).* Devuelve la calificación que valora la pertenencia de un documento a un tópico definido.
  - *Topic classify(String content).* Devuelve el tópico al cual pertenece un documento.

- **TextFilter**

  Interfaz encargada de la preparación del texto para su posterior análisis y clasificación, la cual está formada por la siguiente función:

  - *String doFilter(String text, FilterChain filterChain).* Devuelve el texto procesado por el método implementado por el filtro.

El orden de los filtros y su aplicación será llevado a cabo por la clase FilterChain, que se conecta directamente con el clasificador, así el clasificador se preocupa de aplicar la cadena de filtros que se le indique, haciendo más flexible su uso.

3.3.3 Indexación de documentos

3.3.3.1 Diagramas lógicos

El componente de Indexación de documentos estará formado por tres subcomponentes que procesarán los requerimientos de la aplicación cliente, como observamos en la Fig. 3.9.

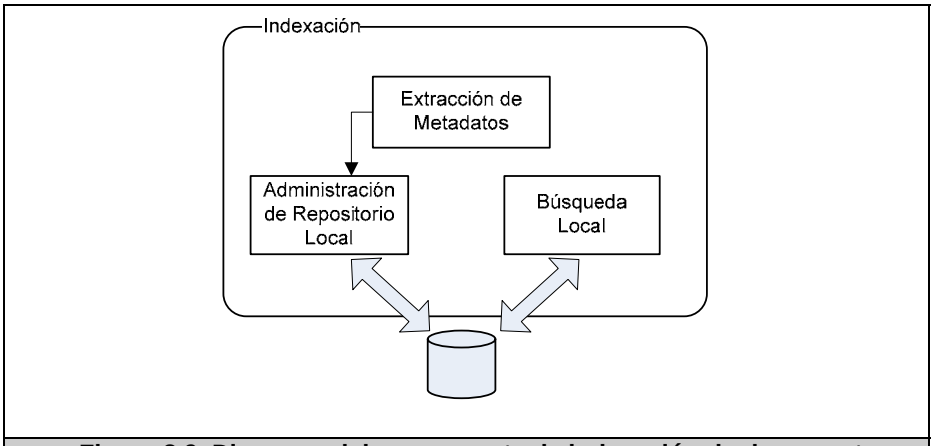


Figura 3.9. Diagrama del componente de indexación de documentos.

A continuación describiremos cada uno de los subcomponentes:

- **Extracción de Metadatos**

  Es el subcomponente encargado de extraer datos descriptivos sobre un documento a ser almacenado. Los datos a ser extraídos son: palabras claves y resumen del texto.

- **Administración de Repositorio Local**

  Es el encargado de manejar el repositorio local de documentos; esto es, la inserción y modificación de documentos dentro del índice generado.

- **Búsqueda Local**

  Encargado de manejar las búsquedas locales requeridas por una aplicación cliente, a través del índice creado para el repositorio local.

Para un mayor entendimiento del funcionamiento del componente de Indexación de documentos, explicaremos su flujo de trabajo, que se presenta en la Fig. 3.10.

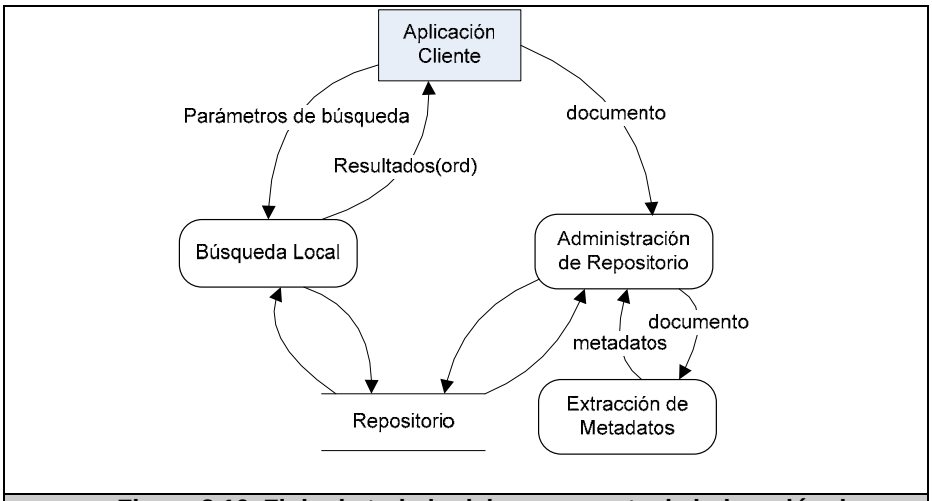


Figura 3.10. Flujo de trabajo del componente de indexación de documentos.

El proceso se divide en inserción o modificación y búsqueda. El subproceso de inserción o modificación inicia con la recepción de un documento que requiere ser almacenado localmente. El subcomponente de Administración del Repositorio utiliza las herramientas de Extracción de Metadatos para obtener la información requerida para llenar los campos descriptivos del documento, y luego realiza las modificaciones necesarias en el índice del repositorio local.

El subproceso de búsqueda inicia con la recepción de los parámetros de búsqueda enviados por la aplicación cliente; el subcomponente de Búsqueda Local se conecta con el repositorio, resuelve la consulta y devuelve los resultados encontrados localmente a la aplicación cliente, ordenados de acuerdo a su relevancia. A continuación se presenta el diagrama de clases propuesto para este componente, Fig. 3.11.

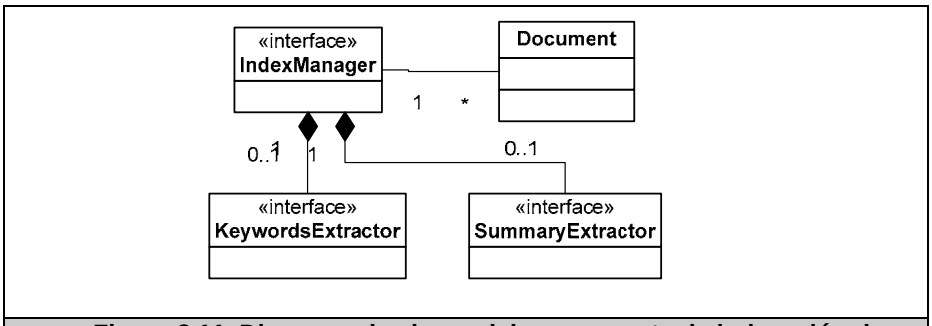


Figura 3.11. Diagrama de clases del componente de indexación de documentos.

3.3.3.2 Interfaz y Configuración

El componente de indexación de documentos tiene sus propias características que definen su interfaz y configuraciones internas. A continuación, detallaremos las entradas del componente, sus salidas, y los parámetros de configuración que requiere:

- **Entradas**
  - *Documento*. El documento a ser procesado para obtener sus palabras claves y generar su resumen.

  - *Parámetros de búsqueda*. Palabras claves que forman la consulta de un usuario.

- **Salidas**
  - *Resultados.* Documentos encontrados en el repositorio local, ordenados de acuerdo a su relevancia.
- **Parámetros de configuración**
  - *Idioma.* De manera opcional, se puede definir el idioma en el que se encuentra el texto a ser almacenado; esto permitirá a las implementaciones del componente introducir mejoras al proceso de indexación, como la eliminación de *stopwords*. Se requiere ingresarlo al momento de la creación del índice.
  - *Extractores de metadatos y opciones de extracción*. Se pueden definir implementaciones de los extractores de resúmenes y de palabras claves, junto con las opciones que estos requieran para su funcionamiento.
  - *Directorio de almacenamiento.* Lugar en el que se almacenará el índice creado por el componente.

El componente de Indexación contiene una interfaz básica:

- **IndexManager**

  Esta interfaz define los siguientes métodos principales:

- *Document addDocument(Resource r)*. Convierte el recurso en un documento y lo añade al índice. Útil cuando se tiene una lista de recursos a añadir, como la que devuelve el componente de Exploración. Utiliza los extractores definidos en el constructor.
- *void addDocument(Document d).* Añade el documento al índice tal y como fue proporcionado (sin utilizar los extractores de metadatos).
- *void removeDocument(String uri).* Quita del índice el documento asociado con el identificador proporcionado.
- *List Search(String query)*. Realiza una búsqueda sobre los documentos del índice, retornando los documentos que se ajustan a los criterios definidos por la aplicación cliente, y ordenados de acuerdo a su relevancia.
- *void flushChanges()*. Se asegura que los cambios realizados en el índice (adiciones y eliminaciones de documentos) se reflejen en el almacenamiento permanente, y, consecuentemente, en las búsquedas posteriores.
- *void close()*. Cierra el índice, indicando que se ha terminado de trabajar con él. No se pueden realizar más operaciones sobre un índice cerrado.

Resumen

En este capítulo se explicaron los detalles referentes a la arquitectura del marco de trabajo y los patrones de diseño a seguir en el mismo. Además, se detalla de manera extensa los tres componentes principales del marco de trabajo, su arquitectura, características, interfaces y configuración de los mismos.

El siguiente capítulo se explica todo lo referente a la implementación de los componentes y del marco de trabajo en sí, además se detalla la aplicación que será desarrollada como prueba de concepto para el marco de trabajo.

CAPÍTULO 4

4 IMPLEMENTACIÓN

En el presente capítulo se describe la plataforma utilizada para el desarrollo del marco de trabajo descrito en el capítulo anterior, junto con la definición de los requerimientos de funcionamiento de hardware y de software del mismo. Luego, se presentan los detalles de implementación de cada uno de los tres componentes del marco de trabajo y se especifican los pasos requeridos para el uso e instalación del marco.

Al final de este capítulo, se introduce la aplicación que servirá como prueba de concepto para el marco de trabajo propuesto en este proyecto. Se detallan sus requerimientos de funcionamiento y se presenta el uso de la aplicación por medio de capturas de pantalla.

4.1 Marco de trabajo

4.1.1 Plataforma de implementación

La plataforma seleccionada como base para el desarrollo y ejecución de nuestro marco de trabajo es Java Platform. Esta conocida plataforma agrupa un conjunto de librerías estandarizadas que permiten el fácil desarrollo de aplicaciones utilizando el lenguaje Java. Además, la plataforma tiene la principal característica de no depender de un procesador o sistema operativo para su ejecución, ya que posee un motor de ejecución (máquina virtual) en cada una de las diferentes variantes.

El marco de trabajo será desarrollado en el lenguaje Java, implicando así que las aplicaciones que usen su funcionalidad también deberán ser implementadas usando este lenguaje de programación. Se ha escogido este lenguaje debido a su característica de ser portátil, uno de los requerimientos principales del marco de trabajo; además por su gran favoritismo dentro de la comunidad de código abierto, de la cual obtenemos variadas librerías que permiten un fácil desarrollo del marco de trabajo.

4.1.2 Requerimientos de funcionamiento

4.1.2.1 Software

Para un correcto funcionamiento del marco de trabajo implementado en este proyecto, se requieren los siguientes componentes de software:

- Java Development Kit (JDK)

  Esta es la plataforma sobre la cual se desarrolló el marco de trabajo, permite a los desarrolladores extender su funcionalidad, así como modificar su comportamiento adaptándolo a diferentes necesidades.

  En caso de no requerir modificaciones, se podrá utilizar en su defecto el Java Runtime Enviroment (JRE), el cual es el conjunto de librerías necesarias para ejecutar un programa implementado en Java.

  Para el desarrollo de nuestro marco de trabajo, hemos utilizado la versión 1.5.0_09 del kit de desarrollo de Java. Se recomienda como una versión mínima la 1.4 del JDK o JRE, para poder ejecutar sin problemas el marco de trabajo.

- Sistema Operativo

  Como se dijo previamente, la plataforma Java no está ligada a un procesador o sistema operativo en particular, para nuestra implementación lo hemos realizado bajo el Sistema Operativo

Windows XP (Home y Professional) corriendo en un procesador Intel.

Además, hemos realizado pruebas de funcionamiento en el sistema operativo Fedora Core 2, una de las distribuciones existentes en el mercado del sistema operativo GNU-Linux.

4.1.2.2 Hardware

Para un correcto funcionamiento del marco de trabajo implementado en este proyecto, se requieren los siguientes componentes de hardware:

- Conexión a Internet
- Computador con las siguientes características mínimas:
  - Procesador Intel Pentium de 166 Mhz
  - Memoria RAM de 32 Megabytes
  - Espacio libre en disco duro principal de 125 Megabytes

Estas características son las mínimas necesarias para que corra una máquina virtual de Java (requerida por el marco de trabajo).

4.1.3 Programación

En la programación de nuestro marco de trabajo, se decidió utilizar el paradigma de la Programación Orientada a Objetos, dado que este es el utilizado por el lenguaje Java. La plataforma de desarrollo de Java provee una serie de librerías estándar, con las funciones más comunes a todos los proyectos de programación. Adicionalmente, y de acuerdo al concepto de reutilización de código, hemos utilizado una serie de librerías disponibles en el Lenguaje Java, bajo licencias de código abierto. Entre las principales podemos mencionar:

- **Jakarta Commons Library**

  Desarrollada como parte del proyecto Apache Jakarta[1], el conjunto de librerías Jakarta Commons se enfoca en proveer componentes reutilizables desarrollados bajo el lenguaje Java. Las clases utilizadas dentro del proyecto están organizadas en una serie de componentes, cada uno orientado a una tarea específica, por ejemplo: manejo de archivos de configuración o extensiones a las librerías nativas de Java.

---

[1] Apache Jakarta: es un proyecto que ofrece un conjunto de soluciones en Java bajo código abierto (http://jakarta.apache.org/).

- **Log4j**

  Librería que facilita el registro de sucesos (logging) y la depuración de errores dentro de programas. Esta librería permite habilitar o deshabilitar el registro de sucesos sin necesidad de alterar el código de la aplicación, a través de un archivo de configuración.

- **HtmlParser**

  Librería que implementa la interpretación y análisis de código HTML para el Lenguaje Java. En nuestro proyecto es utilizada por el agente explorador para extraer tanto el texto plano como los vínculos contenidos en un documento HTML.

A continuación se presentarán los detalles de implementación de nuestro marco de trabajo, al cual hemos decidido asignar el nombre de Framework para Indexación, Clasificación y Recopilación de Documentos (FICR). Por tanto, los paquetes desarrollados estarán bajo estas siglas.

4.1.3.1 Exploración automática de páginas Web

Las clases que implementan el componente de exploración están ubicadas dentro del paquete ficr.crawling, que a su vez está

compuesto por varios subpaquetes. A continuación describiremos las clases que conforman el componente de exploración:

- **Paquete ficr.crawling**

  Define las clases básicas que implementan el componente de exploración: AbstractCrawler, CrawlerFactory, así como la interfaz Classifier. Esta interfaz define los métodos básicos que una clase debe implementar a fin de ser compatible con el marco de trabajo, y la clase abstracta AbstractCrawler provee una implementación básica de esta interfaz, y cuya funcionalidad puede ser extendida por medio de la herencia.

  La implementación por defecto del crawler está diseñada para hacer uso de las técnicas multihilo durante su funcionamiento, y para ello se ha definido una clase denominada CrawlerThread. Ésta implementa los pasos en el proceso de exploración, y recae en la implementación de Crawler la responsabilidad de coordinación entre las distintas instancias de CrawlerThread, así como la interacción con la aplicación cliente.

  A fin de asegurarse de no visitar un determinado recurso más de una vez a lo largo de una búsqueda, la implementación por defecto del agente aplica un proceso llamado *canonicalización* a las direcciones web que va descubriendo; esto permite

determinar con una mayor precisión si dos direcciones web son equivalentes [28]. El proceso de canonicalización que aplicamos es el siguiente:

- Eliminación de espacios en blanco extras.
- Conversión de la dirección a letras minúsculas.
- Eliminación de anclas de la dirección (“#”).
- Eliminación de caracteres “?” y “&” extra.
- Eliminación de referencias a las páginas por defecto (“index.html”, “index.htm”).
- Eliminación del caracter “/” si se encuentra al final de la dirección.

La clase CrawlerFactory permite instanciar objetos de tipo Crawler, a partir de un archivo de configuración en formato XML. La Fig. 4.1 presenta un ejemplo de un archivo de configuración.

Adicionalmente, dentro de este paquete se definen las interfaces de Frontier y ResourceManager, detalladas la sección 3.3.1.2; cada una con una implementación por defecto: BestFirstFrontier y JDBMResourceManager. Esta

última hace uso de la librería JDBM para manejo simple de persistencias[1].

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<crawler>
<crawler-class>ficr.crawling.BestFirstCrawler</crawler-
class>
<max-depth>3</max-depth>
<num-threads>8</num-threads>
<num-results>10</num-results>
<max-pages-to-visit>20</max-pages-to-visit>
<max-frontier-size>200</max-frontier-size>
<header-params>
       <param>
              <param-name>From</param-name>
              <param-value>ficr@cti.espol.edu.ec</param-
value>
       </param>
       <param>
              <param-name>User-Agent</param-name>
              <param-value>FICR/FICR</param-value>
       </param>
</header-params>
<seeds-obtainer>
       <obtainer-
class>ficr.crawling.seeds.GoogleSeedsObtainer</obtainer-
class>
       <obtainer-params>
       <param>
              <param-name>client-key</param-name>
              <param-value>XxcqCvpQFHJt5</param-value>
       </param>
       <param>
              <param-name>max-seeds</param-name>
              <param-value>10</param-value>
       </param>
       </obtainer-params>
</seeds-obtainer>
</crawler>
```

Figura 4.1. Ejemplo de archivo XML para configuración del componente de Exploración.

[1] Proyecto JDBM: es un motor de persistencia de transacciones para Java (http://jdbm.sourceforge.net/).

- **Paquete ficr.crawling.event**

Define las clases necesarias para la generación y captura de eventos generados durante el proceso de exploración. Las aplicaciones que necesiten mantenerse informadas del estado de la exploración deben implementar la interfaz CrawlStatusListener, y registrarse con la instancia de su interés a través de los métodos definidos en la interfaz Crawler.

- **Paquete ficr.crawling.extractor**

Define clases encargadas de procesar el contenido de los distintos tipos de archivos encontrados en la red. Nuestro marco de trabajo incluye por defecto implementaciones para los siguientes tipos de archivo: HTML, PDF, DOC, RTF, y TXT.

- **Paquete ficr.crawling.seeds**

Este paquete contiene las clases utilizadas durante el proceso de obtención de semillas para la exploración. Define la interfaz básica, SeedsObtainer, así como las clases AbstractSeedsObtainer y SeedsObtainerFactory. Adicionalmente el marco de trabajo incluye una

implementación de la interfaz, utilizando el API de Búsqueda de Google[1].

- **Paquete ficr.crawling.statistics**

Las aplicaciones que requieran recolectar estadísticas de la exploración, necesitan implementar la interfaz StatisticsCollector, definida en este paquete, y registrarse a través de los métodos incluidos en la interfaz Crawler. La interfaz StatisticsCollector es una subinterfaz de CrawlStatusListener, por lo tanto las clases que implementen esta interfaz también podrán registrar cambios de estado durante la exploración. El marco de trabajo incluye una implementación por defecto de StatisticsCollector, que recoge estadísticas básicas sobre el funcionamiento del agente y las almacena en un archivo de texto.

[1] API de Búsqueda de Google: creado para desarrolladores e investigadores interesados en usar la búsqueda de Google como un recurso en sus aplicaciones (http://code.google.com/apis/soapsearch/).

4.1.3.2 Clasificación de documentos

El componente de clasificación de documentos está conformado por un paquete principal de clases denominado ficr.classify. Este paquete contiene las 2 clases principales: **ClassifierFactory** y **AbstractClassifier**; además de la interfaz del clasificador denominada: **Classifier**.

La interfaz **Classifier**, es la que especifica los métodos comunes que las implementaciones de clasificadores deben poseer dentro del marco de trabajo; permitiendo así la implementación de las variadas estrategias de clasificación existentes en la actualidad.

La clase **ClassifierFactory**, es la encargada de crear una instancia de clasificador a partir de un archivo XML de configuración. Este archivo de configuración es creado a partir del esquema mostrado en la Fig. 4.2.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<classifier>
<classifier-
class>ficr.classify.NaiveBayesClassifier</classifier-class>
<filters>
<filter-class>ficr.util.text.StopWordsFilter</filter-class>
<filter-class>ficr.util.text.StemmerFilter</filter-class>
</filters>
<lang>en</lang>
<topic-dir>/training</topic-dir>
</classifier>
```

Figura 4.2. Ejemplo de archivo XML para configuración del componente de Clasificación.

Este esquema especifica el clasificador a usar, los filtros requeridos para la preparación del texto, el lenguaje de los documentos a procesar dentro del clasificador y el directorio relativo donde se encuentran los documentos iniciales de entrenamiento del clasificador.

La segunda clase de importancia es la clase **AbstractClassifier**, la cual implementa operaciones comunes a todos los clasificadores, como el manejo de los documentos de ejemplo a través de un tópico y el manejo de filtros a usarse para preparar el texto de éstos documentos. Esta clase abstracta luego será extendida por una o varias clases que implementen un clasificador en particular.

Para prueba de nuestro marco de trabajo se ha implementado un clasificador, el **NaiveBayesClassifier**, el cual extiende la clase **AbstractClassifier** y además implementa los métodos descritos en la interfaz **Classifier**. Esta clase fue desarrollada utilizando una colección de herramientas denominada Minorthird[1], la cual provee

[1] Minorthird: librería para almacenamiento de texto, generación de etiquetas, aprendizaje y extracción de entidades, y categorización de texto desarrollada por William Cohen. (http://minorthird.sourceforge.net).

métodos para identificar nombres y relaciones ontológicas[1] en un texto utilizando heurísticas.

El componente de clasificación requiere del manejo de tópicos definidos para su correcto funcionamiento; dichos tópicos incluyen documentos válidos (pertenecientes al tópico) y no válidos, con los cuales se entrenará al clasificador. Cada tópico está representado por la clase **Topic**, incluida en el paquete ficr.basic.

Para el manejo de los tópicos, el marco de trabajo posee una clase denominada **TopicManager**, la cual interactúa con archivos de texto de ejemplos y un motor de base de datos portátil en Java. Esta clase se encuentra dentro del paquete ficr.basic. Los archivos de texto de ejemplos se encuentran organizados en una estructura jerárquica de carpetas, de acuerdo al tópico perteneciente; esta estructura es especificada en un archivo XML, tal como se presenta en las Fig. 4.3 y 4.4.

[1] Ontología: esquema conceptual dentro de un dominio, para facilitar la compartición de información y comunicación entre diferentes sistemas.

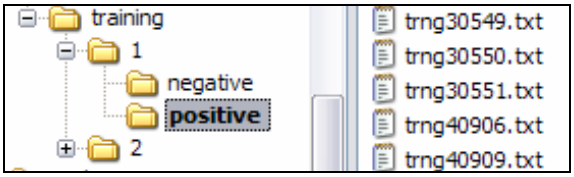


Figura 4.3. Estructura jerárquica de carpetas.

El archivo XML que define a los tópicos presentes permite a la clase **TopicManager** generar la base de datos e insertar los tópicos y ejemplos iniciales.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<topics>
	<topic>
	<id>1</id>
	<full-name>Environmental Health and Toxicology
	</full-name>
	<label>health</label>
	<keywords>
		<keyword>health</keyword>
		<keyword>toxicology</keyword>
	</keywords>
	</topic>
	<topic>
	<id>2</id>
	<full-name>Christian Religion</full-name>
	<label>christian</label>
	<keywords>
		<keyword>social</keyword>
		<keyword>christian</keyword>
		<keyword>religion</keyword>
	</keywords>
	</topic>
</topics>
```

Figura 4.4. Ejemplo de archivo XML de definición de estructura jerárquica de tópicos.

El modelo relacional de la base de datos se presenta en la Fig. 4.5. Se definieron 2 entidades, cada una con un rol distinto: la tabla

Example, almacena los documentos de ejemplo de un tópico, los cuales pueden ser válidos para el tópico (positivos) o no válidos (negativos). La segunda tabla es Document, la cual almacena los documentos que el usuario almacena referente a un tópico específico.

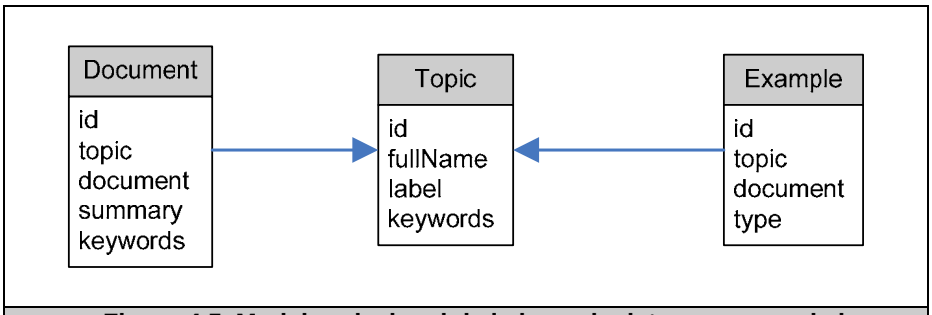


Figura 4.5. Modelo relacional de la base de datos que maneja los tópicos del clasificador.

La base de datos usada para la implementación es HSQLDB[1], una base de datos relacional implementada en Java la cual provee un pequeño y rápido motor de base de datos con tablas almacenadas en memoria y en disco. Esta base de datos permite tener los ejemplos de entrenamiento iniciales, nuevos ejemplos agregados por un usuario y documentos almacenados de búsquedas previas.

[1] HSQLDB: sistema gestor de bases de datos libre, escrito en Java (http://hsqldb.org/).

4.1.3.3 Indexación de documentos

Las clases que implementan el componente de indexación están agrupadas dentro del paquete ficr.indexing. Este paquete incluye tanto la interfaz **IndexManager** como las clases **AbstractIndexManager** e **IndexManagerFactory**.

Siguiendo el estándar implantado en los componentes anteriores, la clase **IndexManagerFactory** permite crear instancias de clases **AbstractIndexManager**, a partir de un archivo de configuración en formato XML. En la Fig. 4.6 se incluye un ejemplo de un archivo de configuración.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<index-manager>
<index-manager-class>
ficr.indexing.LuceneIndexManager</index-manager-class>
<index-path>c:/index</index-path>
<language>en</language>
<keyword-extractor>
ficr.indexing.metadata.YahooKeywordExtractor
</keyword-extractor>
<summary-extractor>
ficr.indexing.metadata.SimpleSummaryExtractor
</summary-extractor>
<metadata-options>
<option>
<option-name>appid</option-name>
<option-value>jcaicedo</option-value></option>
<option>
<option-name>max-keywords</option-name>
<option-value>10</option-value></option>
<option>
<option-name>num-sentences</option-name>
<option-value>3</option-value></option>
</metadata-options>
</index-manager>
```

Figura 4.6. Ejemplo de archivo XML para configuración del componente de Indexación.

La clase **AbstractIndexManager** incorpora elementos para extracción de metadatos, además de proveer una implementación de los métodos básicos de la interfaz **IndexManager**. Para ello hace uso de las clases definidas en el paquete ficr.indexing.metadata, que proveen interfaces para la extracción de resúmenes y palabras clave a partir de un documento de texto; **SummaryExtractor** y **KeywordExtractor**, respectivamente. El marco de trabajo incluye además dos implementaciones de estas interfaces: **SimpleSummaryExtractor**, que utiliza la librería Classifier4j[1], y **YahooKeywordExtractor**, que emplea el servicio web para extracción de palabras claves proveído por Yahoo[2].

Para propósitos de prueba, el marco de trabajo incluye una implementación de **IndexManager**, utilizando la librería para

---

[1] Classifier4J: librería de Java diseñada para realizar clasificación de texto, además posee otras características como generación automática de resúmenes. (http://classifier4j.sourceforge.net/)

[2] Yahoo! Search Developer Kit: librerías para acceder a los servicios web de Yahoo!, como un kit de desarrollo de software (SDK) (http://developer.yahoo.com/)

búsqueda de texto Apache Lucene[1]. Ésta provee tanto mecanismos de indexación como de búsqueda de texto, y está disponible bajo una licencia de código abierto.

El marco de trabajo incluye implementaciones de las interfaces de Lucene en el paquete ficr.indexing.lucene. Éstas son integradas con las capacidades de stemming y eliminación de stop words del marco de trabajo, a fin de mejorar la calidad de los resultados obtenidos en las búsquedas locales.

4.1.4 Instalación

El marco de trabajo esta diseñado para funcionar como una librería extra a una aplicación cliente que lo requiera. Es por ello que las clases necesarias para su funcionamiento van a ser empaquetadas en un archivo Java (Java Archive - JAR). Al agregar esta librería a una aplicación cualquiera, se puede usar los diferentes componentes del marco de trabajo, así como extender su funcionamiento si es requerido.

---

[1] Apache Lucene: librería de motor de búsqueda de texto escrita completamente en Java (http://lucene.apache.org/java/docs/index.html).

Además del archivo Java (JAR) con las clases que forman el marco de trabajo, se deberá incluir dentro del directorio del proyecto los archivos XML de configuración de los componentes de exploración, clasificación e indexación, los cuales van a definir el comportamiento y detalles de ejecución de los componentes antes mencionados. Además, se deberá incluir el directorio que posee los archivos de entrenamiento de los diferentes tópicos definidos para su utilización, así como el archivo XML donde se especifican los diferentes tópicos existentes, los cuales serán utilizados para generar la base de almacenamiento de tópicos. Otra alternativa es proporcionar los archivos de la base de datos generados previamente.

4.1.5 Interfaz para programadores de aplicaciones (API)

La meta de la interfaz para programadores de aplicaciones del marco de trabajo desarrollado es facilitar la implementación de una aplicación cliente que utilice las herramientas desarrolladas, o en su defecto las extienda. Cada uno de los tres componentes posee una interfaz principal, la cual es el vínculo de interacción entre una aplicación y el marco de trabajo en sí.

Para más información acerca de las interfaces de cada uno de los componentes consulte el Apéndice A.

4.2 Aplicación de ejemplo

Para probar el funcionamiento de nuestro marco de trabajo, se ha propuesto la implementación de una pequeña y simple aplicación que utilizará todos los recursos de los tres componentes principales desarrollados. Esta aplicación de ejemplo de ejemplo también estará programada en el lenguaje Java. Es importante indicar que la aplicación no está diseñada para ser utiliza por un usuario final, simplemente nos permite mostrar todos los beneficios de nuestro marco de trabajo.

4.2.1 Requerimientos de la aplicación

La función principal de la aplicación es la búsqueda de documentos digitales en la Web. La aplicación diseñada debe cumplir con los siguientes requisitos:

- **Espacio de Búsquedas**
  - Se mostrarán búsquedas finalizadas o activas.
  - Las búsquedas arrojarán resultados no inmediatos.
  - Cada búsqueda estará relacionada a un tópico específico.
  - Se deberá soportar la modificación de los parámetros de configuración de las búsquedas.

  - Se presentará el estado de las búsquedas, estadísticas y resultados almacenados (si existen documentos almacenados de búsquedas finalizadas).
  - Los resultados obtenidos por las búsquedas pueden usarse para formar parte del entrenamiento del agente en búsquedas a futuro.

- **Manejo de Tópicos**
  - Cada usuario tiene la facultad de crear, modificar, o eliminar sus tópicos.
  - Los tópicos pueden ser creados a partir de categorías existentes, o dando ejemplos de documentos relevantes al tópico (positivos) así como irrelevantes al mismo (negativos).
  - Además de ejemplos positivos y negativos, un tópico tendrá asociadas palabras claves; el usuario tiene opción a agregar, modificar o eliminar palabras claves de un tópico según su conveniencia.
- **Repositorio Personal**
  - La aplicación manejará un repositorio local con los documentos obtenidos de búsquedas realizadas, que el usuario haya almacenado.

- Un documento no necesita estar relacionado con un tópico específico.
- Deberá soportar búsquedas locales.

4.2.2 Requerimientos de funcionamiento

4.2.2.1 Software

Para un correcto funcionamiento de la aplicación de prueba implementada para este proyecto, se requieren los siguientes componentes de software:

- **Java Runtime Enviroment o Entorno de Ejecución Java (JRE)**

  Es un software que permite ejecutar una aplicación que se encuentre desarrollada en lenguaje Java. Nos permitirá utilizar las herramientas desarrolladas en nuestro marco de trabajo, además de ejecutar nuestra aplicación de ejemplo.

- **FICR - Marco de Trabajo**

  Es el conjunto de herramientas propuestas y desarrollado según éste trabajo de investigación, el cual proveerá la funcionalidad requerida por la aplicación de ejemplo.

4.2.2.2 Hardware

Para un correcto funcionamiento de la aplicación de ejemplo, se requieren los siguientes componentes de hardware:

- Conexión a Internet
- Computador con las siguientes características mínimas:
  - Procesador Intel Pentium de 166 Mhz
  - Memoria RAM de 32 Megabytes
  - Espacio libre en disco duro principal de 125 Megabytes

Estas características son las mínimas necesarias para que corra una máquina virtual de Java (requerida por el marco de trabajo).

4.2.3 Instalación

La aplicación cliente será empaquetada en un archivo Java (Java Archive - JAR). Además, se deberá incluir dentro del directorio de la aplicación los archivos XML de configuración (bajo una carpeta llamada XML) de los tres componentes del marco de trabajo, estos archivos definirán el comportamiento de los mismos. Además, se deberá incluir el archivo XML donde se especifican los diferentes tópicos existentes junto con el directorio que posee los archivos de

entrenamiento. Otra alternativa es proporcionar los archivos de la base de almacenamiento de tópicos.

4.2.4 Interfaz con el usuario

4.2.4.1 Configuración

Como se mencionó anteriormente, cada uno de los componentes posee un archivo XML de configuración de funcionamiento. La aplicación cliente permite (vía modo gráfico) manipular cada uno de los archivos de configuración, como podemos apreciar en la Fig. 4.7.

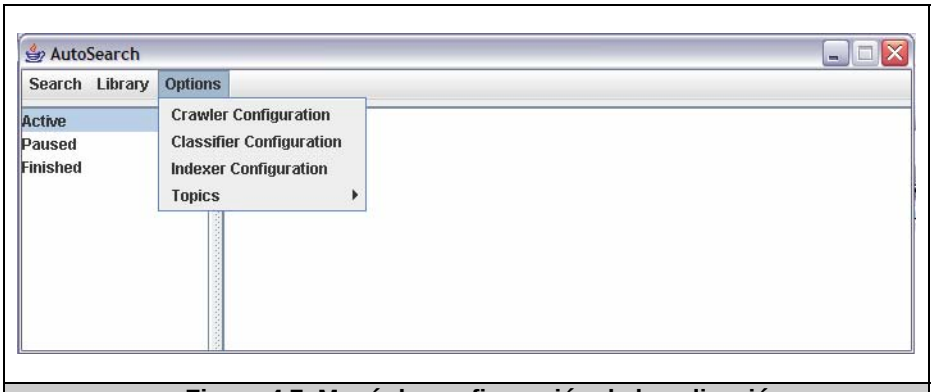


Figura 4.7. Menú de configuración de la aplicación.

Como se observa en la Fig. 4.8, se presentan los parámetros por defecto del componente de exploración, dando la posibilidad a modificarlos de ser el caso.

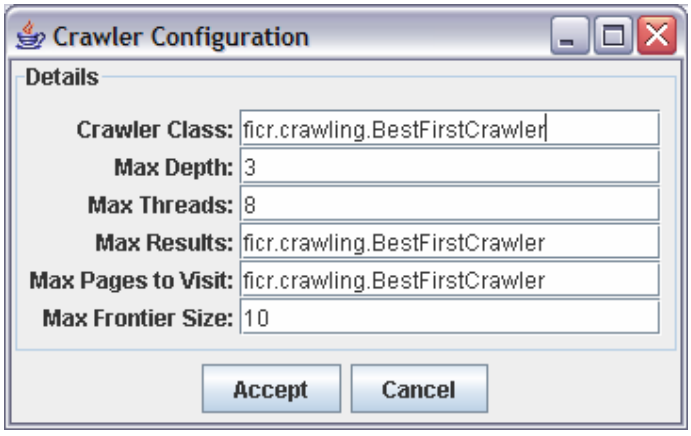


Figura 4.8. Pantalla de configuración del componente de exploración.

Existen ciertos parámetros que son particulares a la herramienta utilizada para obtener los vínculos con los cuales se inicia la exploración (semillas), sabiendo que esta herramienta puede variar; estos parámetros no se encuentran contemplados en la ventana de configuración y esperamos que sean definidos modificando directamente el archivo XML.

En la Fig. 4.9 se observa la ventana de configuración del componente de clasificación. Al igual que con el componente de exploración, ciertos parámetros no son incluidos en la interfaz gráfica. Se presenta la ventana de configuración para el componente de indexación, en la Fig. 4.10.

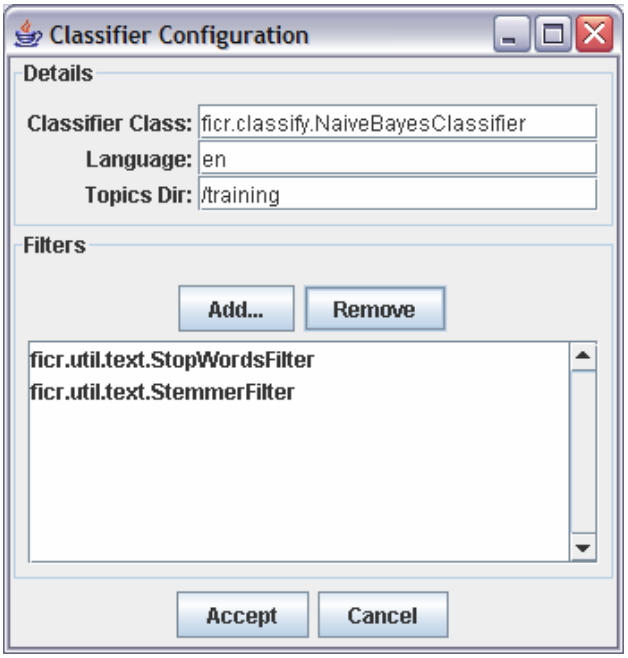


Figura 4.9. Pantalla de configuración del componente de clasificación.

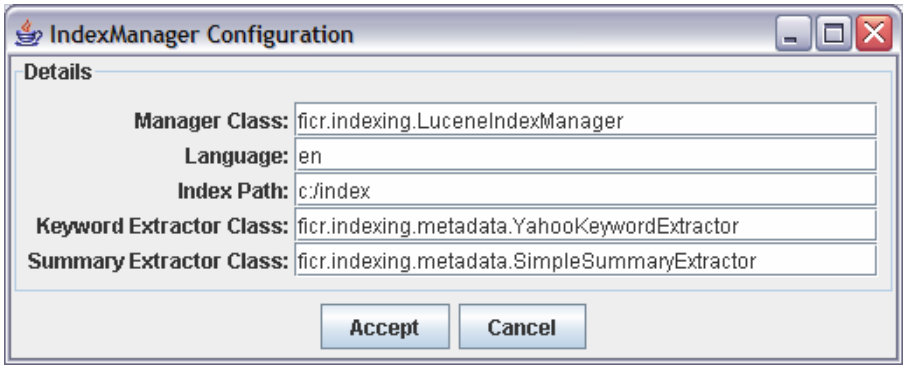


Figura 4.10. Pantalla de configuración del componente de indexación.

Además de la configuración de los diferentes componentes, la aplicación permite el manejo de los tópicos que se encuentran definidos para realizar búsquedas, o en su defecto, insertar nuevos

tópicos, como se observa en la Fig. 4.11. Los parámetros que son definidos para un tópico son: nombre, etiqueta y palabras claves. Además, se pueden agregar ejemplos positivos o negativos que se copien al área de texto de la pantalla.

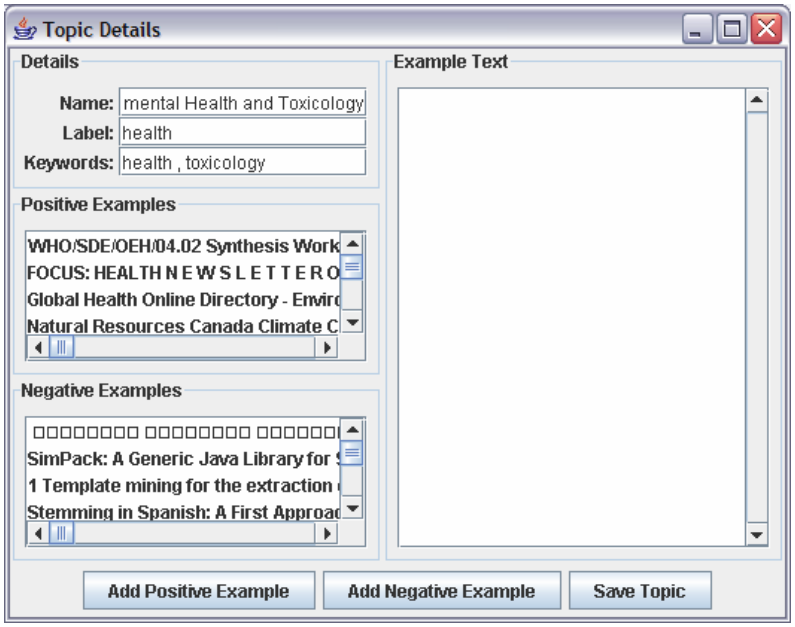


Figura 4.11. Pantalla de modificación o inserción de tópicos.

4.2.4.2 Uso de la aplicación de ejemplo

La aplicación cliente tiene dos usos principales: realizar búsquedas automáticas de documentos en Internet y buscar documentos de un repositorio local.

Al iniciar una búsqueda, se debe elegir el tópico sobre el cual se basarán los documentos a ser encontrados, como se observa en la Fig. 4.12.

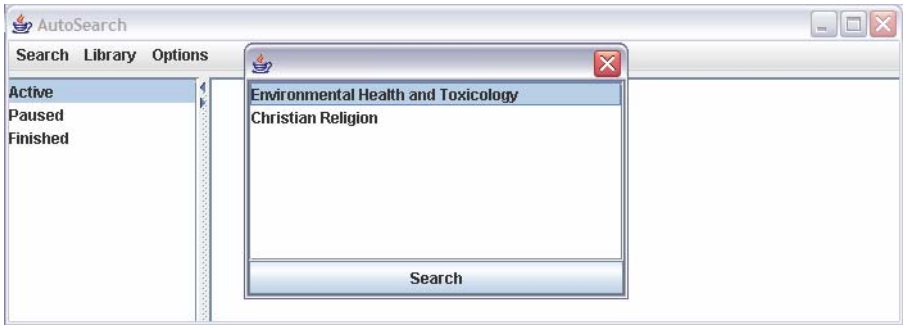


Figura 4.12. Pantalla de elección de tópico a ser utilizado para la búsqueda de documentos.

Luego de definir el tópico, automáticamente se iniciará el proceso de búsqueda, los resultados no son inmediatos y se mostrarán al finalizar una búsqueda. En la Fig. 4.13 observamos un proceso de búsqueda iniciado.

Figura 4.13. Pantalla principal de la aplicación.

Los documentos encontrados son almacenados en un repositorio local y luego indexados para su posterior búsqueda, la cual es realizada en la aplicación por medio de la ventana que observamos en la Fig. 4.14. En la parte derecha de la ventana se presenta los detalles del documento obtenido en una búsqueda previa.

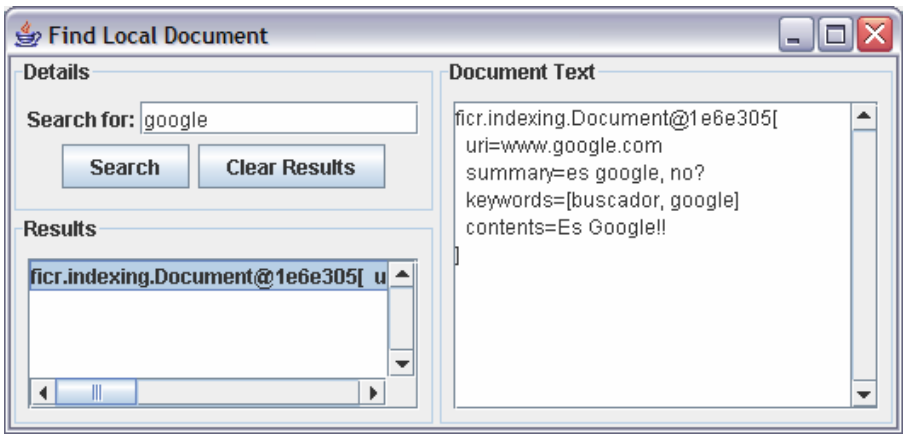


Figura 4.14. Pantalla de búsqueda en el repositorio local de documentos.

Resumen

En este capítulo se explicaron los detalles referentes a la implementación del marco de trabajo y los requerimientos existentes para su uso e instalación. Además, se presenta la aplicación que servirá como prueba de concepto para el marco de trabajo propuesto en éste proyecto.

El siguiente capítulo se procede a realizar las pruebas de funcionamiento y evaluación del rendimiento del marco de trabajo.

CAPÍTULO 5

5 PRUEBAS

El presente capítulo se inicia con la introducción de las métricas[1] comúnmente utilizadas en la evaluación de mecanismos para búsqueda en Internet. Luego, se procede a definir un esquema de pruebas de funcionamiento integrado por dos componentes: una evaluación automática en base a las métricas presentadas y otra por parte de usuarios reales

Finalmente se procede a evaluar el rendimiento del marco de trabajo usando el esquema de pruebas definido y se presentan y explican los resultados obtenidos.

[1] Medida de una alguna propiedad de un software o de sus especificaciones.

5.1 Métricas de Evaluación

Las métricas son medidas de alguna propiedad de un software o de sus especificaciones que permiten su evaluación. Las aplicaciones que se especializan en Recuperación de Información basan sus métricas en la relevancia.

Luego de una búsqueda, el usuario puede llegar a preguntarse s*i ha obtenido el material con mayor relevancia, o, por el contrario, si ha obviado documentos importantes*; además espera que los resultados que reciba *no contengan "basura"*. Es posible medir el rendimiento de una búsqueda tomando en consideración estas dos incógnitas. A continuación explicaremos ambos en más detalle.

5.1.1 Relevancia

En general, los agentes exploradores son evaluados por su capacidad para obtener resultados "buenos" o "relevantes". La relevancia es una calificación numérica asignada a cada uno de los resultados obtenidos durante un proceso de extracción de información, ésta representa el grado de cumplimiento de un resultado en base requerimientos establecidos dentro de una consulta hecha por un usuario. Existen varios algoritmos que pueden

utilizarse para calcular la relevancia de un resultado, como el uso de la frecuencia de términos (tf) o también combinarlo con la discriminación de un término dentro de los documentos buscados (tf-idf) [29].

En las siguientes secciones revisaremos dos de las técnicas para la evaluación de agentes exploradores y la relevancia de sus resultados: la Precisión (Precision) y la Retentiva (Recall), basadas en los conceptos de Recuperación de la Información (Information Retrieval - IR). Ambas técnicas utilizan la noción de documentos relevantes para la exploración, y por lo tanto la mayoría de mecanismos empleados para su implementación requieren de algún nivel de participación del usuario, generalmente en forma de documentos de ejemplo.

5.1.1.1 Precisión (Precision)

En el campo de la IR, se define a la precisión como el número de documentos relevantes obtenidos durante una búsqueda, dividido para el total de documentos obtenidos. Si definimos S como el conjunto de documentos relevantes para una búsqueda, y D como el total de documentos obtenidos, podemos expresar la precisión con la siguiente fórmula:

$$\text{Precisión} = \frac{|S \cap D|}{|D|}$$

5.1.1.2 Retentiva (Recall)

La retentiva se define como el número de documentos relevantes obtenidos durante una búsqueda, dividido para el total de documentos relevantes. Su fórmula es:

$$\text{R}etentiva = \frac{|S \cap D|}{|S|}$$

Donde S es el conjunto de documentos relevantes para una búsqueda, y D es el total de documentos obtenidos.

Podemos observar que generalmente la retentiva y la precisión mantienen una relación inversa entre ellas: el intento de maximizar la precisión conlleva a una disminución de la retentiva, y viceversa. Conocer el objetivo de la búsqueda, así como escoger el mecanismo adecuado de exploración, permitirá establecer un balance adecuado entre las dos métricas.

5.2 Esquema de Pruebas

Se ha decidido medir el rendimiento del marco de trabajo en base a su capacidad para obtener resultados relevantes para una búsqueda;

para ello hemos definido un esquema de pruebas integrado por dos componentes: una evaluación por parte de usuarios reales y una evaluación automática en base a métricas (precisión y retentiva). Para propósitos de esta prueba se ha escogido el idioma inglés, ya que este idioma es el que presenta mayor cantidad de documentos en Internet.

El primer componente de pruebas se basa en la exploración de tres tópicos definidos y la obtención de resultados, los cuales serán calificados en base a su relevancia para cada tópico por tres usuarios escogidos al azar que tengan conocimiento en el área. Los tópicos escogidos están relacionados con el área de las Ciencias Informáticas y son: *Computer Graphics, Machine Learning, Distributed Computing*; estos tópicos serán construidos a partir del Open Directory[1]. El proceso de exploración será realizado por el algoritmo **BestFirst**, complementado con un clasificador **Naive Bayes**.

[1] El Proyecto Open Directory (http://dmoz.org/) es el más grande y comprensible directorio en la Web editado por personas. Es construido y mantenido por una gran comunidad de editores voluntarios alrededor del mundo.

En el segundo componente del esquema de pruebas, la exploración se la realizará en base a un tópico definido, y nos permitirá obtener la cuantificación de las métricas de precisión y retentiva de los resultados de la exploración. Al ser el Internet una gran fuente de recursos, es imposible conocer exactamente el conjunto de documentos relevantes para una búsqueda; por tal motivo definiremos un conjunto de documentos objetivo (T), el cual nos permitirá aproximar el conjunto desconocido de todos los documentos relevantes (R), como se observa en la Fig. 5.1. De esta manera, se pueden aproximar las métricas de precisión y retentiva, al momento de ejecutar una exploración.

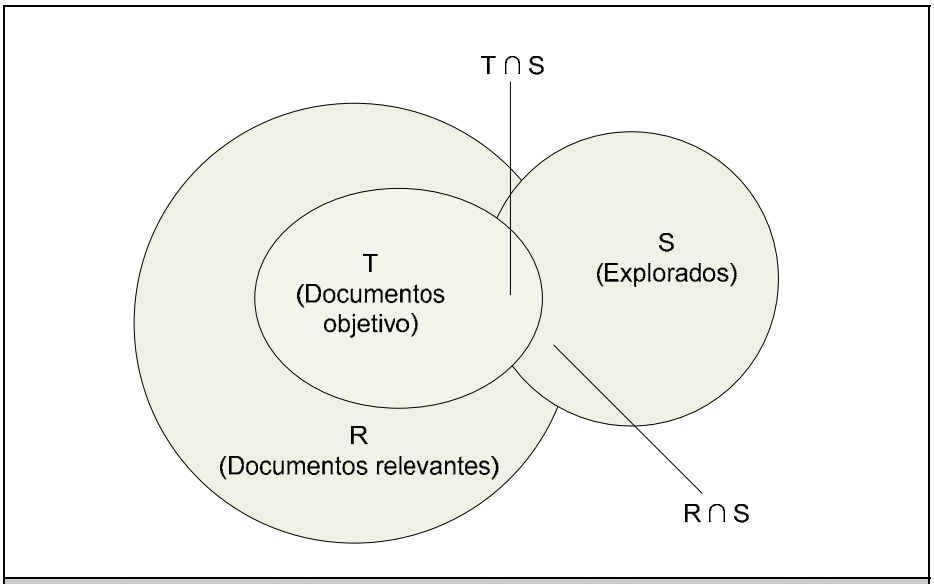


Figura 5.1. Relaciones entre conjuntos de documentos objetivo, relevantes y explorados [F9].

El proceso de exploración será realizado por dos algoritmos. El primero es el **BestFirst**, algoritmo elegido para la implementación dentro del marco de trabajo; el cual será complementado con un clasificador Naive Bayes entrenado con los ejemplos del tópico a ser explorado. Como segundo algoritmo, se utilizará el **BreadthFirst**, el cual servirá como punto de comparación de la eficiencia del algoritmo propuesto como solución.

A fin de que los resultados tengan validez, es necesario asegurarnos que el conjunto de documentos objetivos sean alcanzables, partiendo de las direcciones iniciales (semillas) escogidas para la exploración. Para ello, hemos definido un proceso de generación de semillas, que se muestra en la Fig. 5.2.

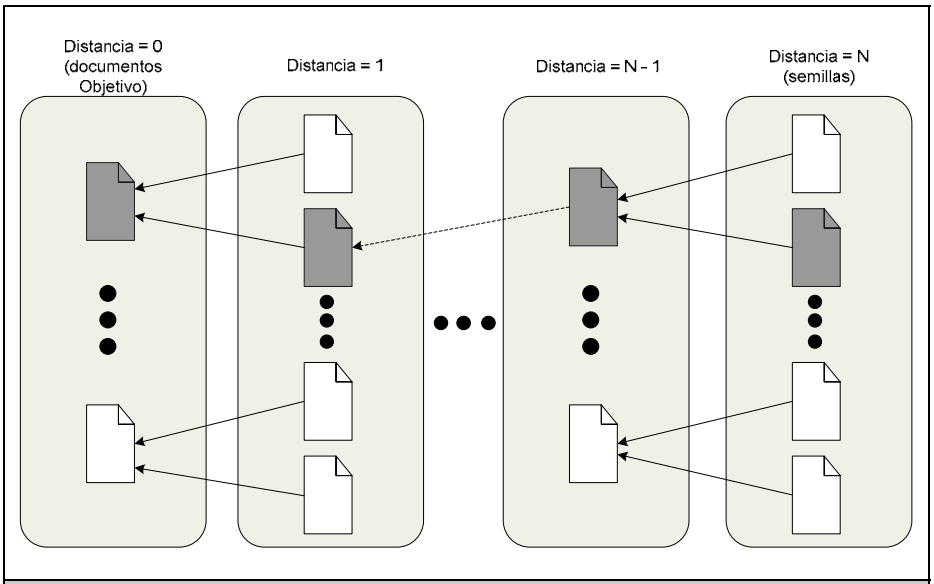


Figura 5.2. Proceso para extracción de semillas en el segundo esquema de pruebas. A mayor N, mayor dificultad en la tarea [F9].

El proceso consiste en obtener las direcciones que apuntan a un documento objetivo y tomar una muestra aleatoria de este nuevo conjunto. Este proceso se repite un número N de veces definido previamente, el cual representa el número de saltos desde una semilla a un documento objetivo. A medida que aumenta N la dificultad del proceso se incrementa considerablemente.

Para este proceso de pruebas hemos escogido un valor para N de 2. Además, las pruebas se correrán para cinco conjuntos de semillas diferentes obtenidos a partir de un grupo de documentos objetivo del tópico *Computer Graphics*. Estos conjuntos de semillas serán obtenidos con ayuda de la interfaz para desarrolladores de Yahoo.

5.3 Medición del Rendimiento

El esquema de pruebas descrito en la sección anterior será utilizado para medir el rendimiento del marco de trabajo.

Primer componente – Usuarios Reales

Para el primer componente se obtuvieron 30 resultados de la exploración por cada uno de los tópicos que se encuentran en el Anexo B.

Tres usuarios con conocimientos en el área calificaron los 10 primeros resultados como relevantes o no relevantes. En las Fig. 5.3, 54 y 5.5, podemos apreciar la cantidad de resultados relevantes para los usuarios por cada uno de los tópicos explorados.

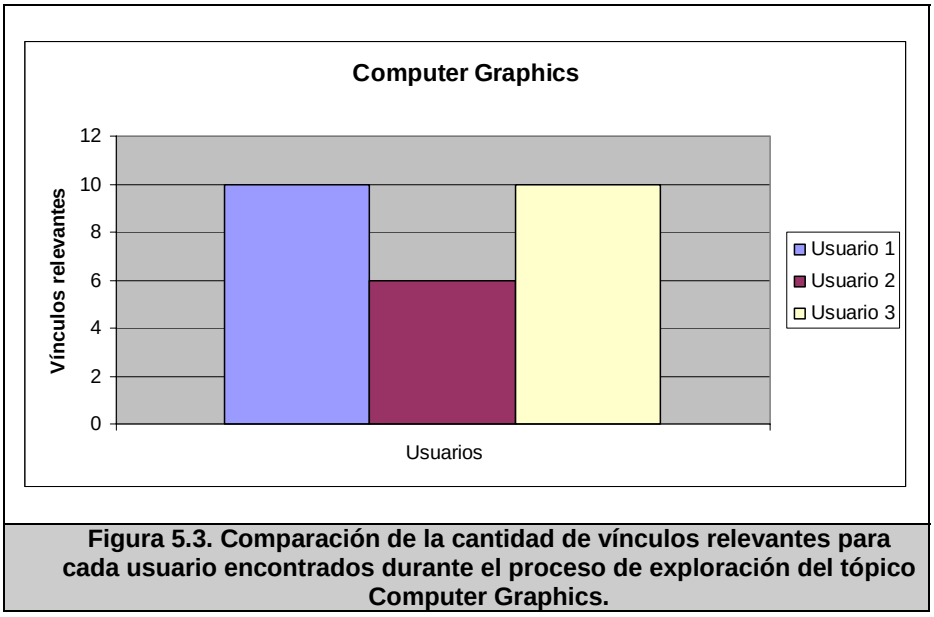


Figura 5.3. Comparación de la cantidad de vínculos relevantes para cada usuario encontrados durante el proceso de exploración del tópico Computer Graphics.

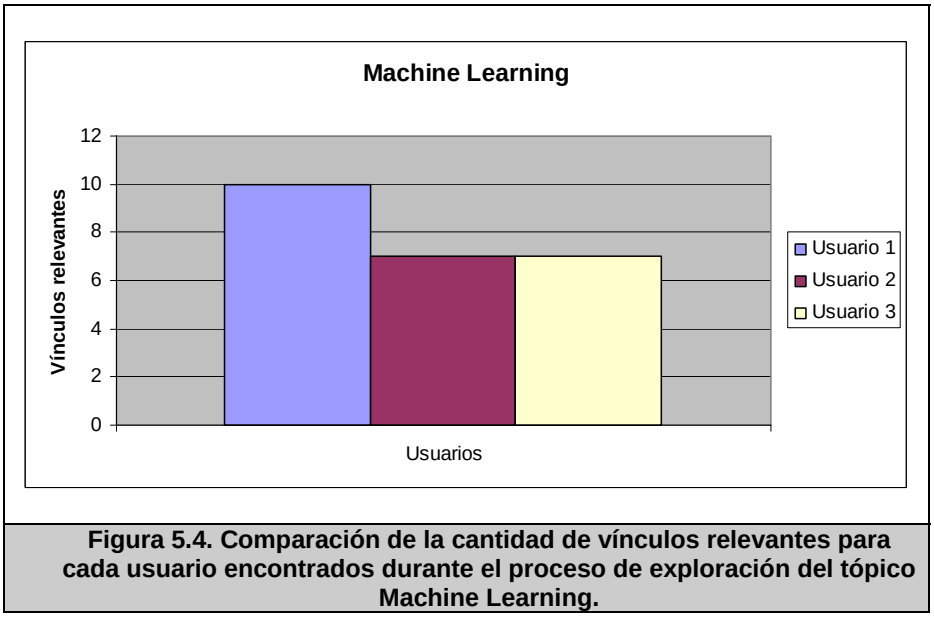


Figura 5.4. Comparación de la cantidad de vínculos relevantes para cada usuario encontrados durante el proceso de exploración del tópico Machine Learning.

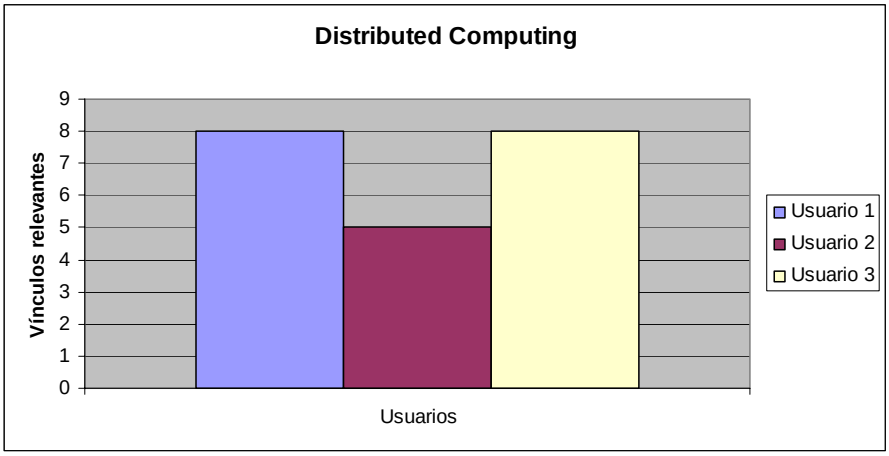


Figura 5.5. Comparación de la cantidad de vínculos relevantes para cada usuario encontrados durante el proceso de exploración del tópico Distributed Computing.

Como se puede apreciar en las figuras, el promedio de vínculos relevantes para los usuarios es de 8, esto quiere decir que de los primeros 10 documentos, 8 son relevantes para el usuario y a medida que el usuario retroalimente al marco de trabajo con su elección de documentos relevantes, la efectividad puede incrementarse. En general, se obtuvieron mejores resultados con el tópico *Computer Graphics,* lo cual puede ser resultado de una mejor calidad de los documentos de ejemplo utilizados para entrenar al clasificador. Así mismo podemos observar que los resultados de la evaluación del Usuario 2 son inferiores a los de los demás; una razón de esta diferencia puede ser un menor dominio del lenguaje en el que se encuentran los resultados, en este caso el inglés.

Segundo Componente – Pruebas Automatizadas

Para el segundo componente se realizó una exploración de un máximo de 1000 páginas a partir de cinco grupos de semillas del tópico *Computer Graphics*, y como se describió en la sección previa, se requiere encontrar la mayor cantidad de documentos objetivo durante la exploración. Para esta exploración se definieron 46 documentos objetivos y fue realizada con los algoritmos **BestFirst** y **BreadthFirst**. A medida que el explorador avanza, se calculan las métricas de precisión y retentiva, lo cual nos permite efectuar un análisis en el tiempo del proceso de exploración. En las Fig. 5.6 y 5.7 se observan las mediciones de precisión y retentiva obtenidas por cada uno de los algoritmos, promediadas para los 5 grupos de semillas.

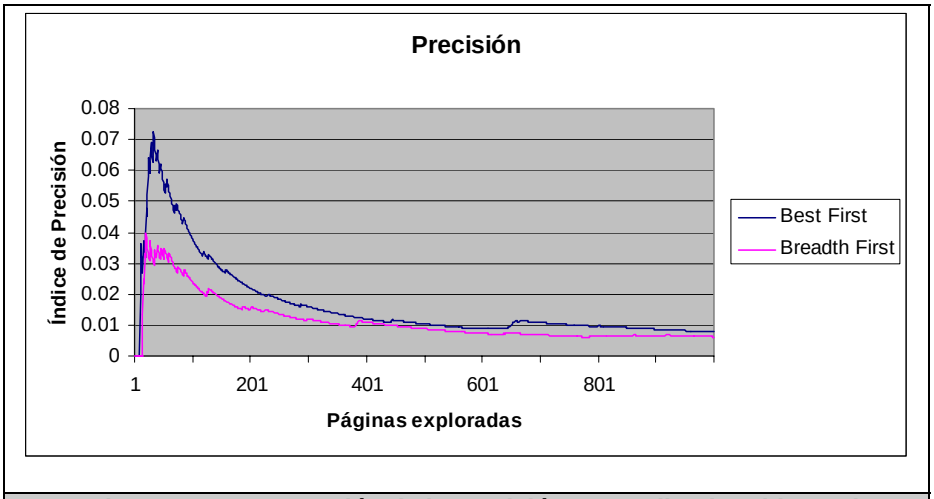


Figura 5.6. Comparación de la precisión promedio obtenida por los algoritmos BestFirst y BreadthFirst en cinco exploraciones.

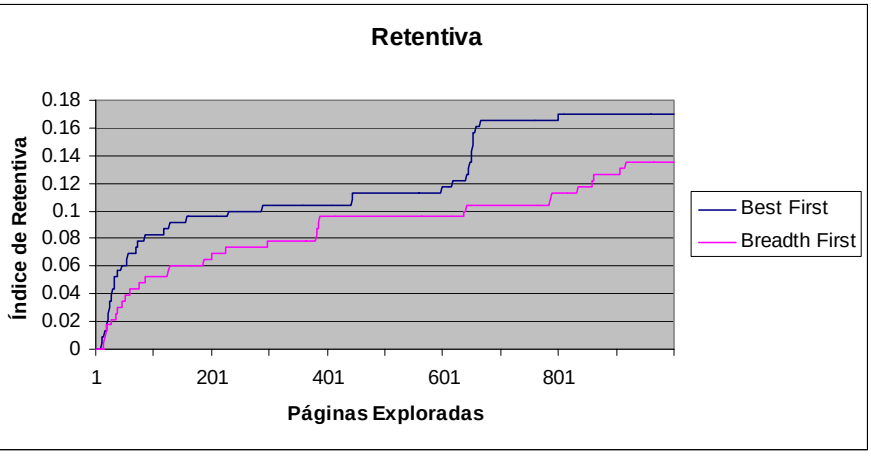


Figura 5.7. Comparación de la retentiva promedio obtenida por los algoritmos BestFirst y BreadthFirst en cinco exploraciones.

En la Fig. 5.6 podemos apreciar que en ambos algoritmos la precisión tiende a disminuir, algo que podía esperarse, dada la dificultad de la tarea. Además, el algoritmo **BestFirst** guiado con un clasificador (solución propuesta por este trabajo) se desempeña de mejor manera que el algoritmo a ciegas **BreadthFirst** según el cálculo de las dos métricas; es decir, encuentra más documentos objetivos que su contraparte. Vemos, que al acercarse a las 1000 páginas exploradas, el algoritmo **BestFirst** presenta una precisión ligeramente inferior a 0.01, y una relevancia cercana a 0.18. Estos valores son cercanos a los obtenidos en investigaciones anteriores [23], donde se usó un esquema de pruebas similar.

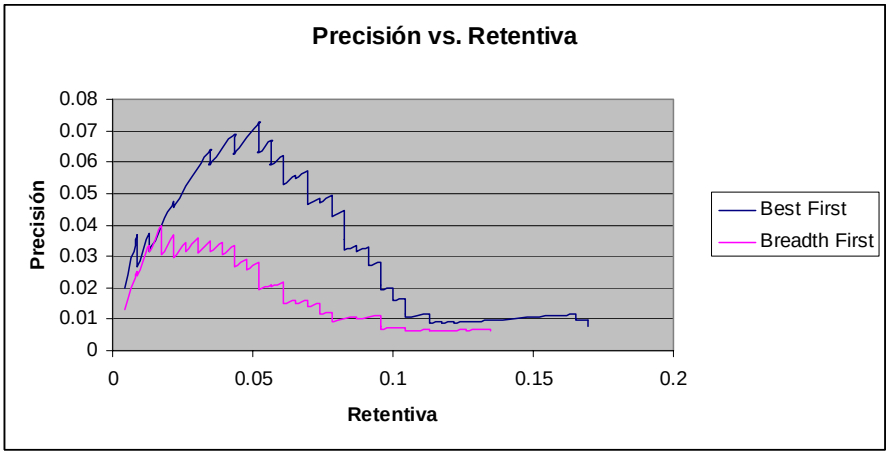


Figura 5.8. Comparación entre la retentiva y la precisión por el algoritmo BestFirst.

En la Fig. 5.8 observamos la comparación entre los algoritmos **BestFirst** y **BreadthFirst**. Como se puede observar dado un valor de retentiva, se obtiene una mejor precisión con el algoritmo Best First. A medida que la retentiva es mayor a 0.05 la precisión va disminuyendo progresivamente, explicando así la relación inversa entre estas dos métricas.

Resumen

En este capítulo se explicaron las métricas a utilizar para la evaluación del marco de trabajo, además se presentó el esquema de pruebas diseñado para los experimentos y se presentaron los resultados de los mismos.

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

1. La existencia de una considerable cantidad de trabajos previos en las áreas de la inteligencia artificial relacionadas con este trabajo, facilitó el análisis de las soluciones a incluir en el marco de trabajo.

2. Luego de realizar un análisis de las técnicas disponibles para la búsqueda de documentos digitales en línea, se decidió escoger la estrategia de “primero el mejor”, por ser la más general, y la que mejor se adapta a los requerimientos del marco de trabajo.

3. Para la tarea de clasificación de documentos digitales, se escogió el algoritmo “Naive Bayes”, ya que el tiempo requerido para su utilización es mucho menor comparado a las otras opciones existentes; y, adicionalmente, es el que mejor soporta el aprendizaje activo, permitiendo adaptar el modelo de clasificación a los requerimientos específicos del usuario.

4. Se decidió incorporar la técnica de índices invertidos como solución a la indexación de documentos, ya que es la que más se ajusta a los requerimientos establecidos en el marco de trabajo, al tener un desempeño adecuado en un gran número de situaciones.

5. La capacidad que tiene el marco de trabajo de poder añadir nuevas funcionalidades nos permitió poder hacer modificaciones e inclusive desarrollar diferentes enfoques de solución para realizar pruebas de rendimiento.

6. Una herramienta orientada a la exploración del Internet, como este marco de trabajo, no tiene sentido si se lo aplica en búsquedas generales, donde un buscador comercial es mucho más eficiente. En cambio, el utilizarlo en un campo específico puede dar resultados mucho más alentadores, como los obtenidos al realizar las pruebas.

7. Es importante verificar la calidad de los documentos de ejemplo utilizados para entrenar al clasificador bayesiano, ya que la eficiencia del clasificador afecta directamente al proceso de exploración.

8. Los parámetros de la exploración pueden afectar el rendimiento del algoritmo a utilizar; éste fue el caso de Best First, donde si se define un tamaño de frontera muy pequeño podemos perder recursos valiosos que no serán explorados.

RECOMENDACIONES

1. El marco de trabajo es flexible, por tanto es recomendable estar atento a nuevos enfoques y proyectos en el área, a fin de implementar nuevas características que permitan actualizar y mejorar el desempeño del marco de trabajo.

2. Dada la naturaleza del Internet, existe mucha redundancia de contenido, esto es, existen páginas web con contenidos iguales o muy similares, pero con diferente dirección. Esta característica es perjudicial para la exploración, porque tiende a encontrar resultados muy relevantes pero repetidos. Por ello, se debería considerar a futuro la búsqueda de otro mecanismo independiente del URL o en su defecto una variante para tener un identificador único por recurso evitando así resultados repetidos.

3. Soluciones implementadas utilizando este marco de trabajo, pueden ser integradas en sistemas de manejo de contenidos, donde un agente explorador puede obtener nuevos recursos automáticamente, para que sean revisados por usuarios reales y agregados a una biblioteca interna.

4. En este trabajo se escogió el inglés como el idioma para los procesos de exploración y clasificación, al ser éste el más común en Internet. Se podría considerar la incorporación del soporte para otros idiomas al marco de trabajo, entre ellos el español. Esto incrementaría la utilidad del marco de trabajo para los investigadores dentro de ESPOL.

5. Promover entre la comunidad científica de ESPOL el uso de herramientas alternativas para la búsqueda de documentos digitales en Internet, similares a las presentadas en este trabajo.

APÉNDICES

A APÉNDICE A: API DEL MARCO DE TRABAJO

A.1 Explorador (Crawler)

Interface Crawler

Clases que implementan esta interfaz:

AbstractCrawler, BestFirstCrawler, BreadthFirstCrawler

---

public interface **Crawler**

Define la interfaz principal del componente de exploración.

Detalle de métodos

getResults

java.util.List **getResults**()
Una vez que el proceso de exploración ha terminado, este método retornará una lista de objetos de tipo Resource, ordenados por su calificación.
Retorna:
resultados del proceso de exploración, ordenados por su calificación
Ver también:
getStatus()

---

getStatus

java.lang.String **getStatus**()
Retorna una cadena de caracteres representando el estado del explorador. Las constantes que definen los estados están especificadas en la interfaz ficr.basic.Constants.
Retorna:
cadena de caracteres que representa el estado del explorador

---

addCrawlStatusListener

void
addCrawlStatusListener(ficr.crawling.event.CrawlStatusListener listener)
Añade un objeto de tipo CrawlStatusListener para recibir eventos de cambios en el estado del explorador.

Paramétros:
listener – el objeto de tipo CrawlStatusListener
Ver también:
CrawlStatusListener

---

removeCrawlStatusListener

void
removeCrawlStatusListener(ficr.crawling.event.CrawlStatusListener listener)
Remueve un objeto de tipo CrawlStatusListener de la lista de observadores registrados.
Parámetros:
listener – el objeto de tipo CrawlStatusListener a remover

---

addStatisticsCollector

void
addStatisticsCollector(ficr.crawling.statistics.StatisticsCollector collector)
Registra un objeto de tipo StatisticsCollector, para recibir los eventos relacionados con el descubrimiento y procesado de los vínculos.
Parámetros:
collector – el objeto de tipo StatisticsCollector
Ver también:
StatisticsCollector

---

removeStatisticsCollector

void
removeStatisticsCollector(ficr.crawling.statistics.StatisticsCollector collector)
Elimina un StatisticsCollector de la lista de observadores.
Parámetros:
collector – el objeto de tipo StatisticsCollector a eliminar

---

startCrawling

void **startCrawling**(ficr.basic.Topic topic,
java.lang.String xmlClassifierConfiguration)
Inicia un proceso de exploración, usando los parámetros especificados.
Parámetros:
topic – el tópico usado para construir el clasificador

xmlClassifierConfiguration – dirección al archivo que contiene la configuración para el clasificador a utilizarse
Ver también:
ClassifierFactory

---

pauseCrawling

void **pauseCrawling**()
Suspende un proceso de exploración.

---

resumeCrawling

void **resumeCrawling**()
Reanuda un proceso de exploración que ha sido suspendido previamente.

---

stopCrawling

void **stopCrawling**()
Detiene un proceso de exploración.

---

getTopic

ficr.basic.Topic **getTopic**()
Retorna:
tópico usado en el proceso de exploración.

A.2 Clasificador (Classifier)

Interface Classifier
Clases que implementan esta interfaz:
AbstractClassifier, LpNaiveBayesClassifier, NaiveBayesClassifier, NullClassifier

---

public interface **Classifier**

Define la interfaz principal del componente de clasificación.

Detalle de métodos

startLearner

void **startLearner**()
Inicializa el clasificador para comenzar el proceso de aprendizaje.

---

addPositiveExample

boolean **addPositiveExample**(java.lang.String content)
Añade el contenido proveído como un ejemplo positivo para el clasificador.
Parámetros:
content – contenido a ser usado como ejemplo
Retorna:
verdadero si el ejemplo fue correctamente añadido

---

addNegativeExample

boolean **addNegativeExample**(java.lang.String content)
Añade el contenido proveído como un ejemplo negativo para el clasificador.
Parámetros:
content – contenido a ser usado como ejemplo
Retorna:
verdadero si el ejemplo fue correctamente añadido

---

score

double **score**(java.lang.String content)
Evalúa el contenido y retorna un valor indicativo del grado de relevancia. Un valor mayor indica una mayor relevancia para el tópico al que se refiere esta instancia.
Parámetros:
content – contenido a ser clasificado
Retorna:
la calificación calculada por el clasificador

---

classify

java.lang.String **classify**(java.lang.String content)
Evalúa el contenido y retorna cadena de caracteres que determina si éste puede o no ser considerado parte del tópico.
Parámetros:
content – contenido a clasificar

Retorna:
cadena de caracteres representando la pertenencia al tópico

A.3 Administrador de Indices (IndexManager)

Interface IndexManager

Superinterfaces:
ficr.basic.Constants

Clases que implementan esta interfaz:
AbstractIndexManager, LuceneIndexManager

---

public interface **IndexManager**
extends ficr.basic.Constants

Define la interfaz principal para el componente de indexación.

Method Detail

addDocument

ficr.basic.Document **addDocument**(Resource doc)
Construye un objeto de tipo Document a partir de un objeto de tipo Resource, y procede a llamar al método addDocument(Document). Utiliza las clases para extracción de metadatos.
Parámetros:
doc – el objeto de tipo Resource a añadir
Retorna:
el documento añadido al índice
Ver también:
SummaryExtractor, KeywordExtractor

---

addDocument

void **addDocument**(ficr.basic.Document doc)
Añade un documento al índice.
Parámetros:
doc – el documento a añadir

---

removeDocument

void **removeDocument**(java.lang.String uri)
Remueve el documento asociado con el URI (Uniform Resource Identifier) especificado

Parámetros:
uri – identificador del documento

---

search

java.util.List **search**(java.lang.String query)
Realiza una búsqueda en el índice utilizando los parámetros especificados.
Parámetros:
query – parámetros de búsqueda
Retorna:
lista de documentos, ordenados por relevancia

---

search

java.util.List **search**(java.lang.String query,
int startIndex,
int numResults)
Realiza una búsqueda en el índice utilizando los parámetros especificados.
Parámetros:
query – parámetros de búsqueda
startIndex – índice del primer documento a retornar
numResults – número de resultados a retornar
Retorna:
lista de documentos, ordenados por relevancia

---

flushChanges

void **flushChanges**()
Hace permanentes todos los cambios en el índice.

---

close

void **close**()
Cierra el administrador. Llame a este método cuando haya terminado de trabajar con el administrador.

B APÉNDICE B: LISTA DE RESULTADOS DE PROCESOS DE EXPLORACION

B.1 Computer Graphics

http://www.cs.ubc.ca/nest/imager/imager.html
http://www.graphics.cornell.edu
http://www9.informatik.uni-erlangen.de/research
http://www.gg.caltech.edu
http://www.arl.hpc.mil/scivis
http://herakles.zcu.cz/index.php?lang=en&print=
http://www.cs.brown.edu:80/stc
http://dmoz.org/computers/computer_science/computer_graphics
http://www.multires.caltech.edu
http://multires.caltech.edu
http://www.cs.utah.edu/research/areas/graphics
http://www.bpvizcenter.com
http://www.grapp.org
http://www.cg.inf.ethz.ch/main.php?menu=6&submenu=1
http://graphics.ethz.ch/main.php?menu=7
http://ddg.cs.columbia.edu
http://www-imagis.imag.fr/index.gb.html
http://www.visapp.org
http://www.cs.rit.edu/~ncs/graphics.html
http://www-graphics.stanford.edu
http://www-static.cc.gatech.edu/gvu
http://www.cs.cornell.edu/courses/cs665
http://www.nch.med.uni-erlangen.de/e1846/e117/index_ger.html
http://www-evasion.imag.fr
http://graphics.usc.edu/cgit
http://graphics.cs.brown.edu/home.html
http://www.cs.ucy.ac.cy/egsr2006
http://www.magix.ucla.edu
http://www.evl.uic.edu
http://konwihr.in.tum.de

B.2 Distributed Computing

http://boinc.bakerlab.org/rosetta/forum_thread.php?id=1501
http://www.bacchae.co.uk/docs/dist-v3.html
http://www.answers.com/topic/x86-64
http://www.thocp.net/hardware/supercomputers.htm
http://developer.apple.com/releasenotes/developertools/compilertools.html
http://www.hyper.net/dc-howto.html
http://en.wikipedia.org/wiki/list_of_distributed_computing_projects
http://www.extremetech.com/article2/0,1697,11769,00.asp
http://boinc.berkeley.edu/chart_list.php
http://www.linuxhpc.org
http://boinc.truxoft.com/security.htm
http://techreport.com/etc/folding
http://www.start64.com

http://www.boincstats.com
http://www.internetnews.com/ent-news/article.php/3518781
http://www.wired.com/wired/archive/9.04/proteomics.html
http://www.winhpc.org
http://philip.greenspun.com/seia/distributed-computing
http://team-scifi.com
http://dmoz.org/computers/computer_science/distributed_computing
http://www.newsfactor.com/news/japan-bests-ibm-in-supercomputer-stakes/story.xhtml?story_id=1220059r0ady
http://billpg.me.uk
http://www.pcworld.com/news/article/0,aid,46143,00.asp
http://www.theregister.co.uk/2005/08/23/intel_fixes_em64t
http://www.xtremepccentral.com
http://www-03.ibm.com/servers/deepcomputing/bluegene.html
http://forum.folding-community.org/viewtopic.php?t=14500
http://arstechnica.com/etc/dcteams/dc_teams.html
http://www.hq.nasa.gov/hpcc/insights/vol6/supercom.htm
http://www.hardfolding.com

B.3 Machine Learning

http://www.aaai.org/aitopics/html/interview.html
http://home.earthlink.net/~dwaha/research/machine-learning.html
http://www.dl.ac.uk/ccp/ccp14/ccp/web-mirrors/alife/machinelearning/~aha/research/machine-learning.html
http://sampletalk.8m.com
http://www.csse.monash.edu.au/~dld/mixturemodel.html
http://www.cs.monash.edu.au/~dld/mixture.modelling.page.html
http://www.cs.iit.edu/~circsim
http://www.cs.unm.edu/%7eluger/ai-final/chapter1.html
http://www.cs.bham.ac.uk/%7eaxs/misc/talks
http://www.medg.lcs.mit.edu/ftp/doyle/sdai96.html
http://www.cs.colostate.edu/~anderson/interesting-pages.html
http://www.nap.edu/readingroom/books/far/ch9.html
http://www.cs.indiana.edu/%7eleake/papers/p-01-07/p-01-07.html
http://hunch.net
http://www-lsi.upc.es/~talavera/conceptual-clustering.html
http://www.cse.unsw.edu.au/%7ebillw/mldict.html
http://en.wikipedia.org/wiki/machine_learning
http://www.rr.cs.cmu.edu/turing.htm
http://www.cs.wisc.edu/%7edyer/cs540/notes/learning.html
http://www.cs.iastate.edu/~cs573x/homepage.html
http://www-2.cs.cmu.edu/~learning
http://www.machinelearning.net
http://lis.epfl.ch/resources/podcast
http://www.amazon.com/machine-learning-tom-m-mitchell/dp/0070428077
http://www.stat.washington.edu/fraley
http://research.microsoft.com/mlas
http://www.acm.org/ubiquity/interviews/v4i43_russell.html
http://www.edge.org/documents/archive/edge29.html
http://dmoz.org/computers/artificial_intelligence/machine_learning
http://www.pbs.org/wgbh/nova/sciencenow/3204/03-talk.html

C APÉNDICE C: LISTA DE DOCUMENTOS OBJETIVO UTILIZADA PARA PROCESO DE EXPLORACIÓN

C.1 Computer Graphics

http://public.kitware.com/VTK
http://graphics.lcs.mit.edu
http://www-imagis.imag.fr/index.gb.html
http://www.cg.inf.ethz.ch
http://www.sve.man.ac.uk/mvc
http://www.cs.brown.edu/stc/home.html
http://www.soe.ucsc.edu/research/slvg
http://www9.informatik.uni-erlangen.de
http://www.cs.utah.edu/research/areas/graphics
http://www.bpvizcenter.com
http://www-static.cc.gatech.edu/gvu
http://www.dgp.toronto.edu
http://isgwww.cs.uni-magdeburg.de/graphik
http://graphics.cs.uni-sb.de
http://www.magix.ucla.edu
http://www.cs.ubc.ca/nest/imager/imager.html
http://www.cis.upenn.edu/~hms
http://www.grapp.org
http://miralabwww.unige.ch
http://multires.caltech.edu
http://www.graphics.cornell.edu
http://www.opengl.org
http://graphics.cs.brown.edu/home.html
http://www.cs.princeton.edu/gfx
http://www.arl.hpc.mil/SciVis
http://www.cs.unc.edu/Research/stc
http://www.cs.sunysb.edu/~vislab
http://www.sci.utah.edu
http://www.msi.umn.edu/user_support/scivis
http://www.cs.utah.edu/stc
http://people.bath.ac.uk/ensab/G_mod/g_mod.html
http://www.cs.utah.edu/gdc
http://www.cs.utep.edu/interval-comp
http://www.evl.uic.edu
http://www.research.microsoft.com/research/graphics
http://graphics.eng.uci.edu
http://graphics.usc.edu/cgit
http://graphics.cs.ucdavis.edu
http://www.navo.hpc.mil/Visualization.html
http://herakles.zcu.cz
http://vrlab.epfl.ch
http://www.mesa3d.org
http://www.hpcmo.hpc.mil/Htdocs/SCIVIZ/index.html
http://www.eecs.berkeley.edu/Research/Areas/CS/GR
http://www-graphics.stanford.edu
http://www.gg.caltech.edu

D APÉNDICE D: LISTAS DE SEMILLAS UTILIZADAS PARA PROCESO DE EXPLORACIÓN

D.1 Lista de Semillas para proceso 1

http://www.eecs.umich.edu/~guskov/acg/projects.html
http://www.evl.uic.edu/EVL/RESEARCH/art_science2.shtml
http://www.cse.ucsc.edu/~guozheng/research.htm
http://www.rock3.co.uk/cgi-bin/ode/index.cgi?base=%2FComputers%2FComputer_Science%2F
http://www.ranlog.com/ramsey/bookmarks.html
http://www.cs.utah.edu/~tthompso/animation.html
http://graphics.cs.brown.edu/research/sciviz/hand/hand.html
http://www.shoecake.com/gamedev.html
http://dir.yahoo.com/Computers_and_Internet/Graphics/Data_Formats/
http://www.navo.hpc.mil/imglib/IMAGES/GIFs_small/Boats/Ships2.html
http://www.cs.utah.edu/~reinhard/
http://www.inonesearch.com/directory.php?cid=124430
http://www.desktoppublishing.com/fonts.html
http://en.wikipedia.org/wiki/User_interface
http://www.virtualstockquote.com/
http://www.cs.utah.edu/npr/
http://www.gfdl.noaa.gov/products/vis/data/netcdf/GFDL_VG_NetCDF_Utils.html
http://www.cs.sunysb.edu/~lujin/VS04.htm
http://www.actwon.com/more/art.htm
http://physicsingraphics.endofinternet.org/

D.2 Lista de Semillas para proceso 2

http://software.sci.utah.edu/map3d-6.3_docs/manual/index.html
http://en.wikipedia.org/wiki/Multimedia_framework
http://people.csail.mit.edu/fredo/Book/lab.html
http://isg.cs.tcd.ie/eg_ireland/egirl02.html
http://www.cs.utah.edu/~reinhard/
http://www.docguide.com/news/content.nsf/MedicalResourcesWeb?OpenForm&id=ef0b09e5a776dcb785256cc6004e5d60&cond=Research
http://www.gg.caltech.edu/hbp
http://ls7-www.cs.uni-dortmund.de/cgotn
http://www.inonesearch.com/directory.php?cid=43037
http://www.rahul.net/kenton/xsites.framed.html
http://www.masterin.it/site/master_in_romania.asp?page=11
http://www.red3d.com/cwr/games/
http://physicsingraphics.endofinternet.org/
http://imsc.usc.edu/research/
http://www.icebergsystems.co.uk/195/2

http://www.red3d.com/cwr/npr/
http://userwww.sfsu.edu/~infoarts/links/wilson.artlinks2.html
http://www.nanomedix.com/
http://www.cs.sunysb.edu/~lujin/VS04.htm
http://graphics.stanford.edu/~henrik/mylinks.html

D.3 Lista de Semillas para proceso 3

http://www.kub.it/dir/86145/
http://www.red3d.com/cwr/links.html
http://www.virtinet.com/
http://lofi.forum.physorg.com/lofi5746.html
http://www.sdsc.edu/~johnson/projects/resource/library.html
http://www.red3d.com/cwr/behave.html
http://www.nanomedix.com/
http://www.agirlsworld.com/rachel/sock-hop/computerpets.html
http://www.cc.gatech.edu/gvu/stats/NSF/merit.html
http://www.gg.caltech.edu/publications.html
http://www.inonesearch.com/directory.php?cid=43037
http://www.gg.caltech.edu/~jeff/outpost/gamekit.html
http://es.wikipedia.org/wiki/Ilusi%C3%B3n_%C3%B3ptica
http://www.cs.brown.edu/research/graphics/out.html
http://www.z4.cn/bbs/dmoz.php?/Reference/Knowledge_Management/Knowledge_Discovery/Information_Visualization/
http://w4.lns.cornell.edu/~pvhp/ptk/ptkFAQ.html
http://www.jlab.org/~semenov/rlinks/soft.html
http://www.ryanchurch.com/06Q&A.htm

D.4 Lista de Semillas para proceso 4

http://www.cs.brown.edu/research/graphics/home.html
http://www-graphics.stanford.edu/~munzner/
http://www.oscer.ou.edu/news.html
http://www.inonesearch.com/directory.php?cid=43037
http://portal.opera.com/web/?cat=43037
http://www.cs.yale.edu/homes/hamid/bookmarks.html
http://marg.www.media.mit.edu/people/marg/haptics-pages.html
http://www.virtualengland.com/
http://www.virtualtopography.com/
http://www.realtimerendering.com/
http://people.csail.mit.edu/fredo/Book/lab.html
http://graphics.stanford.edu/people.html
http://www.refdesk.com/
http://profs.logti.etsmtl.ca/paquette/Research/conferences.html
http://www.cs.uic.edu/
http://en.wikipedia.org/wiki/Doom
http://www.cse.ust.hk/~wuyc/Vis06.htm
http://dmoz.org/Computers/desc.html
http://www.ottumwa.k12.ia.us/quests/anatomy/Organelle%20X-file%20WebQuest/

D.5 Lista de Semillas para proceso 5

```
http://www.cs.sunysb.edu/~vislab/list_projects.html
http://search.ask.com/web?q="human%2Bfactors%2Band%2Bergonomics"
http://stommel.tamu.edu/~baum/linuxlist/linuxlist/linuxlist.html
http://www.avidpets.com/fish-tanks.htm
http://www.math.psu.edu/MathLists/Contents.html
http://dmoz.org/Computers/desc.html
http://info-s.com/vrml.html
http://blog.kairaven.de/archives/2005/07/C1.html
http://www.lloydwatts.com/
http://www.cs.ndsu.nodak.edu/~kalle/links.html
http://haptic.mech.northwestern.edu/
http://www.cs.toronto.edu/DCS/People/Faculty/
http://www.cs.brown.edu/~tor/sig2000.html
http://www.airbergyarchives.com/adi074/regina/
http://www.intelligentedu.com/cat3-4.html
http://www.vrarchitect.net/links/src/Bookmarks.html
http://www.cs.brown.edu/stc/
http://en.wikipedia.org/wiki/Computer-generated_imagery
http://www.eng.utah.edu/~hsci/
```

REFERENCIAS DE GRÁFICOS

[F1]. **PANT GAUTAM, SRINIVASAN PADMINI, MENCZER FILIPPO,** Crawling the Web. Web Dynamics: Adapting to Change in Content, Size, Topology and Use, edited by M. Levene and A. Poulovassilis,<http://dollar.biz.uiowa.edu/~pant/Papers/crawling.pdf>, 2004, 153-178 p.

[F2]. **MENCZER FILIPPO, PANT GAUTAM, SRINIVASAN PADMINI,** Topical web crawlers: Evaluating adaptive algorithms. ACM Transactions on Internet Technology, Vol. V, No. N, February 2003, <http://dollar.biz.uiowa.edu/~pant/Papers/TOIT.pdf>, 2003, 1-38 p.

[F3]. **BAEZA-YATES RICARDO, RIBEIRO-NETO BERTHIER,** Modern Information Retrieval, PEARSON, 1999, 28 p.

[F4]. **“NAIVE BAYES CLASSIFIER”,** StatSoft Inc., <http://www.statsoft.com/textbook/stnaiveb.html>, 2006.

[F5]. **“SUPPORT VECTOR MACHINES”,** WIKIPEDIA LA ENCICLOPEDIA LIBRE, <http://en.wikipedia.org/wiki/Support_vector_machines>, Diciembre 2006.

[F6]. **BAEZA-YATES RICARDO, RIBEIRO-NETO BERTHIER,** Modern Information Retrieval, PEARSON, 1999, 193 p.

[F7]. **BAEZA-YATES RICARDO, RIBEIRO-NETO BERTHIER,** Modern Information Retrieval, PEARSON, 1999, 200 p.

[F8]. **BAEZA-YATES RICARDO, RIBEIRO-NETO BERTHIER,** Modern Information Retrieval, PEARSON, 1999, 206 p.

[F9]. **MENCZER FILIPPO, PANT GAUTAM, SRINIVASAN PADMINI**, A General Evaluation Framework for Topical Crawlers, Information Retrieval 8(3), 2005, 417-447p.

REFERENCIAS BIBLIOGRÁFICAS

[1] **“INTERNET, PRIMERA FUENTE DE INFORMACIÓN EUROPEA ANTES QUE EL PAPEL”**, IBLNEWS, <http://iblnews.com/story.php?id=18637>, Octubre 2006.

[2] **LAWRENCE STEVE, GILES C. LEE**, “Accessibility of information on the web”, Nature, VOL400, <http://nicomedia.math.upatras.gr/courses/mnets/mat/Lawrence&Giles_AccessibilityOfInformationOnTheWeb.pdf>, Julio 1999, 107-109 p.

[3] **CLIFTON MARC**, “What Is a Framework?”, The Code Project, <http://www.codeproject.com/gen/design/WhatIsAFramework.asp>, Noviembre 2003.

[4] **PANT GAUTAM, SRINIVASAN PADMINI, MENCZER FILIPPO,** “Crawling the Web”, Web Dynamics: Adapting to Change in Content, Size, Topology and Use, edited by M. Levene and A. Poulovassilis,<http://dollar.biz.uiowa.edu/~pant/Papers/crawling.pdf>, 2004, 153-178 p.

[5] **DOCUMENT CLASSIFICATION**, WIKIPEDIA LA ENCICLOPEDIA LIBRE, <http://en.wikipedia.org/wiki/Document_classification>, Diciembre 2006.

[6] **CAI LIJUAN, HOFMANN THOMAS**, “Hierarchical Document Categorization with Support Vector Machines”, Universidad

Brown,<www.int.tu-darmstadt.de/publications/CaiHof-CIKM2004.pdf >, 2004.

[7] **BAEZA-YATES RICARDO, RIBEIRO-NETO BERTHIER,** Modern Information Retrieval, PEARSON, 1999, 25 p.

[8] **BAEZA-YATES RICARDO, RIBEIRO-NETO BERTHIER,** Modern Information Retrieval, PEARSON, 1999, 27 p.

[9] **BAEZA-YATES RICARDO, RIBEIRO-NETO BERTHIER,** Modern Information Retrieval, PEARSON, 1999, 28 p.

[10] **MITCHELL TOM**, “Machine Learning”, McGraw Hill,1997, 177 p.

[11] **“NAIVE BAYES CLASSIFIER”,** StatSoft Inc., <http://www.statsoft.com/textbook/stnaiveb.html>, 2006.

[12] **“SUPPORT VECTOR MACHINES”,** WIKIPEDIA LA ENCICLOPEDIA LIBRE, <http://en.wikipedia.org/wiki/Support_vector_machines>, Diciembre 2006.

[13] **BAEZA-YATES RICARDO, RIBEIRO-NETO BERTHIER,** Modern Information Retrieval, PEARSON, 1999, 31 p.

[14] **SOBOROFF IAN**, IR Models: The Probabilistic Model, Departamento de Ciencias de la Computación e Ingeniería Eléctrica - Universidad de Maryland, <http://www.csee.umbc.edu/~ian/irF02/lectures/08Models-Prob.pdf>, 2002.

[15] **WITTEN IAN, PAYNTER GORDON, FRANK EIBE, GUTWIN CARL, NEVILL-MANNING CRAIG**, “KEA: Practical Automatic Keyphrase

Extraction", Procedings DL '99, <http://www.nzdl.org/Kea/Nevill-et-al-1999-DL99-poster.pdf>, 1999, 254-256 p.

[16] **MATSUO YUTAKA, ISHIZUKA MITSURU**, "Keyword extraction from a single document using word Co-occurrence Statistical Information", International Journal on Articial Intelligence Tools, <http://citeseer.ist.psu.edu/matsuo04keyword.html>, 2004, 157-169 p.

[17] **MANI, INDERJEET.** Automatic Summarization**.** John Benjamins Publishing Company, 2001, 1-25 p.

[18] **RADEV, D. R., BLAIR-GOLDENSOHN, S., ZHANG, Z., RAGHAVAN, R**. NewsInEssence: a system for domain-independent, real-time news clustering and multi-document summarization. En *Proceedings of the First international Conference on Human Language Technology Research* (San Diego, March 18 - 21, 2001). <http://lada.si.umich.edu:8080/clair/nie1/nie.cgi>. 2001.

[19] **COHN DAVID, GHAHRAMANI ZOUBIN, AND JORDAN MICHAEL**, Active Learning with Statistical Models, Massachusetts Institute Of Technology - Artificial Intelligence Laboratory and Center For Biological And Computational Learning - Department Of Brain And Cognitive Sciences, <http://citeseer.ist.psu.edu/17446.html >, 1995.

[20] **MENCZER FILIPPO, PANT GAUTAM, RUIZ MIGUEL, SRINIVASAN PADMINI**, Evaluating Topic-Driven Web Crawlers, Proceedings of the 2001 Annual Conference of the Association of Computing Machinery,

Special Interest Group in Information Retrieval, <http://dollar.biz.uiowa.edu/~pant/Papers/sigir-01.pdf>, 2001, 241 - 249 p.

[21] **MENCZER FILIPPO, PANT GAUTAM, SRINIVASAN PADMINI**, Topical Web Crawlers: Evaluating Adaptive Algorithms, ACM Transactions on Internet Technology, <http://dollar.biz.uiowa.edu/~pant/Papers/TOIT.pdf>, Noviembre 2004.

[22] **DOMINGOS PEDROS, PAZZANI MICHAEL**, Beyond Independence: Conditions for the Optimality of the Simple Bayesian Classifier, Proceedings of the Thirteenth International Conference on Machine Learning, <http://www.cs.washington.edu/homes/pedrod/papers/mlc96.pdf>, 1996, 105-112 p.

[23] **MENCZER FILIPPO, PANT GAUTAM, SRINIVASAN PADMINI**, A General Evaluation Framework for Topical Crawlers, Information Retrieval 8(3), 2005, 417-447p.

[24] **EITRICH TATJANA, LANG BRUNO**, Parallel Cost-Sensitive Support Vector Machine Software for Classification, NIC Workshop 2006, From Computational Biophysics to Systems Biology, <www.kfa-juelich.de/nic-series/volume34/eitrich.pdf>, 2006, 141-144 p.

[25] **HSU CHIH-WEI, CHANG CHIH-CHUNG, AND LIN CHIH-JEN**, A Practical Guide to Support Vector Classification, Department of

Computer Science andInformation Engineering - National Taiwan University,<http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf>, Julio 2003.

[26] **SALTON GERARD, BUCKLEY CHRISTOPHER**, Term-weighting approaches in automatic text retrieval, Information Processing and Management: an International Journal 24 - 5, Agosto 1988, 513-523 p.

[27] **CASTILLO CARLOS,** “Effective Web Crawling”, Universidad de Chile - Departamento de Ciencias de la Computación, <http://www.dcc.uchile.cl/~ccastill/crawling_thesis/effective_web_crawling.pdf>, Noviembre 2004.

[28] **HO LEE SANG, JIN KIM SUNG, HOO HONG SEOK**. "On URL normalization". Proceedings of the International Conference on Computational Science and its Applications (ICCSA 2005), 2005, 1076-1085 p.

[29] **RELEVANCE**, WIKIPEDIA LA ENCICLOPEDIA LIBRE, <http://en.wikipedia.org/wiki/Relevance_(information_retrieval)>, Diciembre 2006.