TRABAJO DE FIN DE GRADO

Grado en Matemáticas

Título: Implementación y evaluación de un método para la desambiguación de documentos relativos a personas

Autora: Irene Baena Sánchez

Directora: Alícia María Ageno Pulido
Codirector: Jordi Turmo Borrás

Departamento: Ciencias de la Computación

Convocatoria: 2014-2015

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

Índice general

Capítulo 1

Introducción

1.1. Contextualización y objetivos

Es habitual utilizar motores de búsqueda para encontrar información relativa a una persona introduciendo su nombre en dicho buscador, y es frecuente que estos motores de búsqueda devuelvan una lista ordenada de URL's que normalmente se refieren a varias personas que comparten el mismo nombre. Idealmente el usuario querría obtener grupos de los documentos que se refieren a un mismo individuo. Desde un punto de vista práctico, la tarea de la desambiguación de documentos relativos a personas es de una gran relevancia puesto que entre un 11 y un 17 % de las consultas que se realizan incluyen un nombre de persona y el 4 % son sólo un nombre de persona. Además, los nombres de persona son muy ambiguos: según la Oficina del Censo de Estados Unidos tan sólo 90.000 nombres diferentes son compartidos por 100 millones de personas (datos extraidos de [1]). La tercera edición de la campaña de evaluación WePS *(Web People Search)* [1] propone una tarea de *clustering* que consiste en agrupar las páginas relativas a una misma persona. La organización de WePS-3 aporta datos de prueba y ejecutables para la evaluación de los resultados. Esta campaña tuvo lugar en 2009-2010 y participaron en ella 13 grupos de investigación de Europa, Asia y Norte América, entre los cuales se encuentran los autores del artículo *'Using Web Graph Structire for Pearson Name Desambiguation'* [2]. Este es el artículo que tomaremos como referencia durante la realización del proyecto.

Tal y como se describe en dicho artículo, procederemos a implementar un método para el *clustering* de documentos web relativos a personas en dos fases: el *clustering* basado en la estructura web y el *clustering* basado en el contenido. Para el *clustering* basado en la estructura de las webs se em-

plearan los *links* de cada una de ellas para crear los *clusters*, para ello se determinará por cada web cuales son los enlaces que resultarán útiles en el proceso. En el *clustering* por contenido se extraen los términos considerados relevantes para la tarea y se utilizan para crear vectores n-dimensionales que caracterizarán a cada documento web y permitirán el agrupamiento de aquellos documentos con un contenido más similar.

El objetivo de este proyecto es obtener un método eficiente para la desambiguación de documentos web relativos a personas además de familiarizarse con el lenguaje de programación Java y desarrollar un proyecto de programación por completo, desde el análisis del problema y la búsqueda de posibles soluciones, hasta el diseño y la implementación del código, o el análisis y la evaluación de los resultados proporcionados por el método empleado.

A continuación se presenta el trabajo realizado en el diseño, implementación y ejecución de un método para la desambiguación de documentos relativos a personas. En el capítulo 2 se presenta el diseño del algoritmo general del desambiguador, en 3 se describe el pretrataimento de los documentos para su posterior *clustering* que se describe detalladamente en el capítulo 4. En la sección 5 se evaluan los resultados obtenidos y se comparan con los del artículo. Por último en 6 se muestra cual ha sido la planificación llevada a cabo y en el capítulo 7 se recogen las conclusiones y el trabajo futuro.

1.2. Estado del arte

A continuación se comentan, sin ánimo de ser exhaustivos, otras propuestas y enfoques para la tarea de desambiguación de documentos relativos a personas, que se presentaron en la tercera edición de *Web People Search Evaluation Campaign (WePS-3)*. Junto con la escoguida como referente para este proyecto, estas tres propuestas fueron las que obtuvieron mejores resultados en la campaña WePS-3.

- En el artículo titulado *'PolyUHK: A Robust Information Extraction System for Web Personal Names'* [3] sus autores proponen un sistema distinto para la tarea de desambiguación de documentos cuya principal innovación es que incorpora información externa adicional. En concreto extraen bigramas útiles mediante el uso del corpus *Web 1T 5-gram* y de un método de relación de probabilidad logarítmica, y les asignan pesos según la ponderación de *TFIDF*. Tras esto, recuperan más información acerca del objeto a desambiguar mediante un motor de búsqueda. En

concreto concatenan el bigrama con mayor peso de cada documento web y el nombre de persona ambiguo y hacen la consulta en el motor de búsqueda (en este caso utilizaron Yahoo). Con los 100 primeros fragmentos devueltos por el motor, eliminan los que no contengan la consulta ordenada y con el resto crean un nuevo documento que puede contener información adicional sobre el objeto ambiguo. Tanto la información del documento web como la del nuevo documento asociado son utilizadas para calcular las características que se utilizarán como criterios del proceso de *clustering*.

- Los autores del artículo *'Person Name Disambiguation by Bootstrapping'* [4] distinguen entre dos tipos de características, las fuertes y las débiles. Toman como carcaterísticas fuertes los nombres de entidades, las palabras claves compuestas y las URL's de cada documento, y como características débiles el resto de palabras sueltas. Con esta información ejecutan un algoritmo de *clustering* en dos fases por el cual la primera fase utiliza solamente las características fuertes de cada documento y las revisa en la seguna fase haciendo uso de las débiles. El algoritmo otorga pesos a las características débiles en la segunda fase haciendo uso de técnicas de *bootstrapping*, que es habitual en el procesamiento del lenguaje natural. Esta segunda fase permite disminuir la tendencia de la primera de concluir con valores elevados de precisión en las clasificaciones realizadas y las dificultades para clasificar un gran volumen de documentos. En este caso las similitudes entre documentos las calculan mediante el coeficiente de solapamiento (coeficiente Szymkiewicz-Simpson).

- Por último, en *'Web Person Name Disambiguation by Relevance Weighting of Extended Feature Sets'* [5] sus autores describen el proceso en varias fases. Primeramente extraen los conceptos de Wikipedia como características del documento y esto, junto con otras características más convencionales como son las *named entities*, se usa para configurar un vector de características del documento. El peso de cada característica en el vector se estima en función de la relevacia respecto al nombre de la consulta y la relevancia respecto al contenido del documento, además aplican *TFIDF* para el cáclulo de los pesos. Calculan las similitudes entre documentos haciendo uso de los correspondientes vectores de características y utilizando dos medidas de similitud: la similitud del coseno y la similitud de superposición o solapamiento. Tras localizar los dos documentos más próximos los unen y continuan con el proceso de nuevo.

Capítulo 2

Diseño del algoritmo general del desambiguador de documentos

En este apartado vamos a describir a grandes rasgos el proceso completo que se ha realizado para la desambiguación de documentos relativos a personas y que se detalla en los capítulos siguientes.

En [fig. 2.1] podemos ver un esquema de la arquitectura del sistema que recoge las tareas realizadas a lo largo del proceso, y en el diagrama de [fig. 2.2] se muestran los datos con los que se trabaja y como evolucionan durante el proceso.

Partimos de un conjunto de documentos que contienen el código $HTML$ de varias webs y que están organizados en carpetas cuyo nombre es un nombre de persona ambiguo. Por tanto, cada carpeta contiene un conjunto de documentos (webs) que se refieren a individuos distintos que comparten el mismo nombre (se entiende que cada documento sólo se refiere a un individuo). El objetivo del sistema es, para cada nombre ambiguo, agrupar entre sí los documentos que se refieren a una misma persona.

Primeramente se limpian los ficheros originales (el proceso se detalla en el capítulo 3) para obtener el contenido íntegro que un usuario visualizaría al abrir las webs ya que para realizar el *clustering* de los documentos necesitaremos localizar los *links* a otras páginas y disponer del conjunto de palabras que aparecen en cada uno de los documentos.

Una vez se ha obtenido y guardado la información de los enlaces de cada

documento a otras webs se procede a ejecutar un algoritmo para agrupar los documentos tomando en cuenta las coincidencias de estos enlaces. De esta parte del proceso se obtienen los primeros resultados que luego se compararán con los obtenidos por los autores del artículo que hemos tomado como referencia y con los obtenidos por los otros algoritmos que se van a implementar.

Por otro lado, el contenido íntegro y explícito que se visualiza en las webs es lanzado a un procesador lingüístico que se va a encargar de lematizar cada término y de clasificarlo morfológicamente. Esta información se va a utilizar para hacer una nueva agrupación de los documentos originales valorando la semejanza de su contenido semántico. En este punto se van a distinguir dos casos; el uso del total de términos que se considera que representan semánticamente al documento, y el uso de tan sólo los treinta términos que aparecen más veces en él, es decir, los más frecuentes de entre los que se considera que representan al documento. De esta manera obtendremos dos nuevos conjuntos de resultados a analizar, dos nuevas formas de agrupar los ficheros originales. Existe también la posibilidad de iniciar este *clustering* por contenido tomando cada uno de los documentos como documento independiente o partiendo de los resulados del *clustering* por *links* en el cual ya se han agrupado algunos de los documentos y por tanto existe relación entre ellos y no son todos independientes entre sí.

De esta manera contamos con un total de 5 conjuntos de resultados finales que analizaremos y compararemos con los resultados obtenidos en el artículo [2].

A continuación de describen de forma más minuciosa y específica las diferentes fases del proceso llevado a cabo para la desambiguación de documentos.

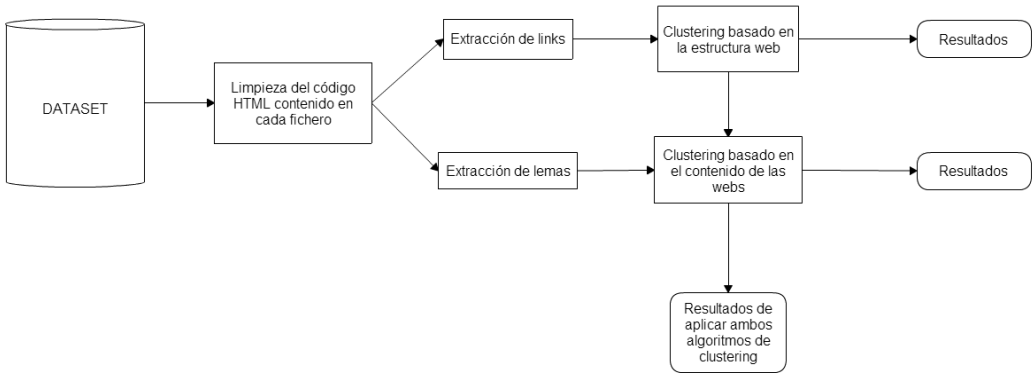

Figura 2.1: Arquitectura del sistema.

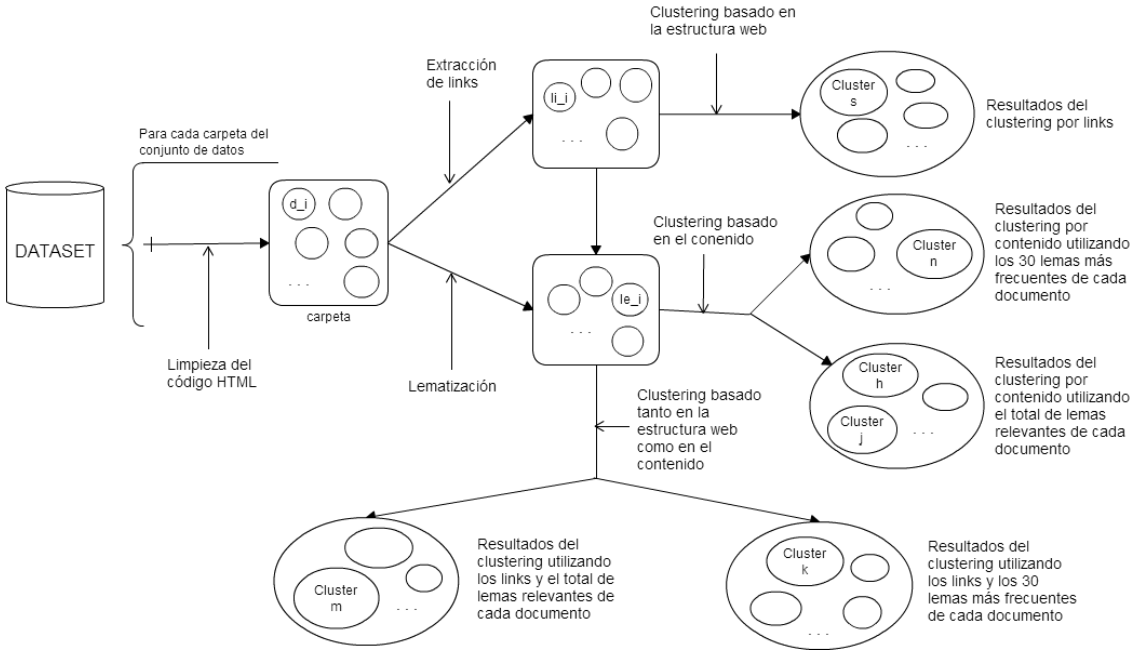

Figura 2.2: Esquema de los datos a lo largo del proceso.

Capítulo 3

Procesador del corpus

3.1. Pretratamiento de los documentos originales

Inicialmente contamos con un conjunto de ficheros *html* organizados en diferentes carpetas cuyo nombre corresponde a un nombre de persona. Cada uno de dichos ficheros *html* contienen el código fuente de una página web, la cual se refiere a una persona con el mismo nombre que el de la carpeta en la que está ubicado. Para llevar a cabo el *clustering* de los ficheros (webs) tan sólo podremos utilizar las información contenida en estos mismos.

Mediante un sencillo programa en Java [fig. 3.1] procesaremos los documentos para eliminar las etiquetas propias de *HTML* puesto que nos interesa tan sólo el contenido propio de la web. De esta manera no daremos más importancia a ciertas partes que a otras, es decir, no vamos a diferenciar entre el valor de la información que nos puede aportar un título o un pie de fotografía y el que nos puede aportar un párrafo cualquiera de la web, por ejemplo. Para llevar a cabo esta limpieza de los archivos *html* nos hemos ayudado de la librería JSoup [6] que entre otras utilidades nos permite encontrar y extraer datos, manipular los elementos, atributos y texto, y analizar código *HTML*.

Aprovechamos este primer recorrido por todos los ficheros originales para eliminar de cada uno de ellos una serie de caracteres extraños que aparecen por utilizar estos archivos distintos juegos de caracteres y distintas codificaciones para ellos. Cada uno de los documentos originales utiliza un juego de caracteres *(charset)* dependiendo, por ejemplo, de la lengua en que estan escritas sus palabras o simplemente dependiendo de la necesidad que se tuviera, al constuir la web que representa, de utilizar unos caracteres u otros,

y utilizan también distintas codificaciones *(encoding)* para estos juegos de caracteres. Un *charset* o juego de caracteres es un conjunto de caracteres representables, con un código numérico asociado a cada uno, y un *encoding* o codificación es la forma de representar en bytes el código numérico del carácter. De esta manera surgen los problemas y apariciones de caracteres extraños ya que puede ocurrir que el texto esté almacenado con una codificación determinada pero la aplicación que lo muestra lo está interpretando con otra codifiación.
La tarea de eliminación de caracteres extraños se realiza por dos motivos principalmente; los ficheros posteriormente serán tratados por un procesador lingüístico y así disminuimos las posibilidades de fallo del procesador al encontrarse algún carácter que no contemple, y además vamos eliminando contenido exento de valor semántico y reduciendo la carga de datos, la cual cosa nos combiene particularmente en nuestro caso por estar trabajando con un gran volumen de datos. Dichos caracteres se localizan como resultado del estudio de un conjunto de documentos de entre el total. No se tiene por tanto control completo sobre los caracteres extraños que puedan aparecer en los documentos pero en cualquier caso se reducen enórmemente las probabilidades de que un documento contenga algún carácter que provoque problemas de codificación. El apéndice B recoge una tabla con los nombres de las carpetas (nombres de persona) y los ficheros contenidos en ellas, que, tras la limpieza, han presentado problemas de este tipo sobre el conjunto total de los datos con los que trabajamos.

El resultado obtenido será entonces un conjunto de ficheros de texto plano organizados de igual manera a los originales cuyo contenido será el contenido propio de cada web.

```
for each folder c in dataset do
    for each document d in c do
        remove html tags
        clean the content of strange characters
        put personalized tags
    end for
end for
```

Figura 3.1: Pseudocódigo limpieza ficheros *html*.

3.2. Preparación previa al procesador lingüístico

Será necesario posteriormente que todo el conjunto de datos sea tratado por un procesador lingüístico por lo que tomamos una serie de medidas para luego tener más control sobre la salida de este procesador. Aprovechamos el programa descrito en el apartado anterior para colocar en cada archivo de texto plano, con el contenido explícito de cada web, unos *tags* de inicio y final de fichero [fig. 3.2]. Estos tags nos permitirán tener el control sobre qué documentos han empezado a ser tratados por el procesador y cuales de ellos han logrado finalizar este tratamiento con éxito.

Los procesadores lingüísticos se encargan del procesamiento de lenguajes naturales (PLN ó NLP - *Natural Language Processing*), traducen el lenguaje natural (LN ó NL - *Natural Language*) a una representación formal equivalente. Se entiende por lenguaje natural a la lengua o idoma, hablado o escrito, que utiliza el ser humano con el fin de comunicarse y que surge de un modo espontáneo entre la gente. Así, se consideraría lenguaje natural el que utilizamos todos los días los seres humanos para comunicarnos y no serían un lenguaje natural los lenguajes de programación por ser estos diseñados e impuestos.
Existen diversos procesadores lingüísticos, en nuestro caso hemos utilizado FreeLing [7] [8], una herramienta *open source* de análisis del lenguaje. FreeLing fue creado como resultado de la investigación llevada a cabo por el grupo de investigación del lenguaje natural de la UPC y, entre otros servicios, ofrece el análisis morfológico, la detección y clasificación de entidades, el análisis sintáctico y la desambiguación.
Unicode es un *charset* que incluye todos los posibles caracteres de todos los idiomas del mundo y UTF-8 es un *encoding* de Unicode. FreeLing precisa recibir ficheros con codificación UTF-8 por lo cual hemos tenido que realizar, previo al uso del procesador lingüístico, una conversión masiva de todos los archivos a esta codificación.

Realizados ya todos estos pasos previos nos encontramos en disposición de iniciar el proceso de *clustering* que se dividirá en dos fases que se detallan en el siguiente capítulo y que pueden tratarse tanto de forma independiente como pueden complementarse la una a la otra.

```
< DOCUMENT >
        < TEXT >
                ...
        <\TEXT >
<\DOCUMENT >
```

Figura 3.2: *Tags* de control de ficheros.

Capítulo 4

Clasificador de documentos

4.1. *Clustering* por *links*

Para realizar el *clustering* por *links* será necesario contar con un listado de los enlaces que aparecen en cada documento, es decir, en cada web. Estos *links* son los *forward-links*, las webs a las que enlaza directamente la página. Existen también los *backward-links*, páginas que tienen a ésta como enlace directo, pero en nuestro caso ésta es una información a la que no tendremos acceso y que por tanto no utilizaremos en el algoritmo de *clustering*, pero que puede resultar muy útil en procesos similares por el valor de la información que aportarían.
Conseguimos estos listados de *forward-links* para cada web mediante un programa en Java [fig. 4.1] que utiliza los documentos originales para localizar los *links* mediante el *tag* <a> *(anchor)* para enlaces de *HTML* y el atributo *href*, requerido para definir un hipervínculo que indica el destino del vínculo. Un ejemplo de extracto de código $HTML$ de donde obtendriamos una URL sería:

<a href="http://www.answers.com/main/science.jsp"name=
"&lid=left_Science&lpos=left_Explore">Science

Y la URL extraida sería:

http://www.answers.com/main/science.jsp

Además, una vez obtenidos todos los enlaces de cada web, se obtiene el *host name* para cada uno de ellos, es decir, guardamos tan sólo el primer nivel de la URL (para el ejemplo anterior sería: *http://www.answers.com*).

Por último, antes de generar la salida del programa, se limpian los listados de posibles repeticiones, ya existentes desde el principio o surgidas por el hecho de quedarnos con la parte principal de cada URL, para disminuir el volumen de datos y los tiempos de ejecución de procesos posteriores.

Una vez ejecutado este programa que se acaba de describir, contamos ya con una estructura de carpetas y ficheros igual a la de los documentos originales pero con cada uno de los listados de *links*, hasta el primer nivel de las URLs y sin repeticiones, para cada web. Se puede ver un ejemplo de estas listas en [tab. 4.1].

En el artículo que tratamos de seguir lo más fielmente posible [2], nos exponen el uso que ellos hacen tanto del servicio *Google Search* como de un *PageRank* personalizado *(PPR - Personalized PageRank)* que utiliza el método de Monte Carlo, para llevar a cabo el *clustering* por *links*.
Google Search es el motor de búsqueda propiedad de Google y *PageRank* es el término con el que se conoce al algoritmo patentado que utiliza Google para determinar el rango de prioridad de una web en la red y el cual se nutre, en parte, de la información proporcionada por los *backward-links* de cada página. También se conoce como *PageRank* al valor numérico (entre 0 y 10) que representa la importancia de una página web, es decir, el valor numérico que devuelve la propia tecnología *PageRank*. Google entiende que cuando un sitio web enlaza a otro, es como si se emitiera un voto. Cuanto más votos (enlaces) son emitidos para una web, el buscador entiende que debe ser más importante. Asimismo, la importancia de la página que haya emitido el voto, por tanto su *PageRank*, determina la importancia de la votación en sí. Así es como Google formaliza la relevancia de una página y éste es uno de los factores que determina el ranking en los resultados de una búsqueda en Google. Además de *PageRank*, Google utiliza otros criterios, que mantiene en secreto (alrededor de 250), para determinar el ranking de páginas en las listas de resultados.
Los autores del artículo utilizan alternativamente *Personalized PageRank* [9] y consideran encontrar los top-k links vecinos más importantes de un nodo según su *PageRank* y determinar el orden exacto de estos elementos (los valores particulares del *PageRank* no son cruciales para el proceso que realizan). Proponen el uso del método de Monte Carlo [10] para la detección rápida de los top-k links con mayor valor de *Personalized PageRank* utilizando tan sólo *forward-links*. Defienden que la naturaleza perezosa de las iteraciones del método de Monte Carlo puede verse como una considerable ventaja en términos de tiempo y almacenamiento de recursos frente a métodos que requieren conocer el grafo de la estructura Web, y además es paralelizable, reduciendo

así el tiempo computacional.
Por tanto, juegan con la posibilidad de utilizar una u otra opción además de la posibilidad de fijar distintos valores de k a la hora de obtener los top-k links para cada web y la posibilidad de variar el número de iteraciones y el factor de amortiguación en el método de Monte Carlo.

En nuestro caso no hemos utilizado el *PageRank* personalizado con el método de Monte Carlo y tampoco hemos podido utilizar el servicio de *Google Search* puesto que Google nos bloqueaba al intentar hacer consultas masivas del *PageRank* de las webs. Google controla el número de peticiones en un periodo de tiempo y, llegado al límite establecido por ellos, bloquea las consultas tanto a nivel de IP como a nivel de red desde donde fueron emitidas esas consultas. Consideramos entonces que una buena manera de obtener listados de *links* relevantes que nos sirvieran para realizar el *clustering* era eliminar una serie de *links* que consideramos demasiado comunes o populares para aportar valor al proceso. Se ha realizado un estudio, que se especifica en el siguiente punto, sobre el conjunto total de *links* que se tratarán, para determinar qué *links* son demasiado comunes.

El algoritmo de *clustering* por *links*, que vamos a reproducir de lo descrito en el artículo, consistirá básicamente en materializar la definición de *cluster* que encontramos en el propio artículo: dos páginas webs pertenecen al mismo *cluster* si tienen al menos un *forward-link* en común.

A continuación se trata con más detenimiento el proceso llevado a cabo para el *clustering* por *links* y el propio algoritmo utilizado.

```
for each folder c in dataset do
    for each document d in c do
        locates all links looking for the href attribute
        get the host name (first level in the URL) for all of them
        eliminates repeated URLs of the list
    end for
end for
```

Figura 4.1: Pseudocódigo extracción de *links*.

4.1.1. Pasos previos

Contamos con un listado de todos los *links*, hasta el primer nivel de la URL (*host name*) y sin repeticiones, para cada uno de los documentos origi-

Links
http://storytellers.vh1.com
http://fanjam.vh1.com
http://movies.thefablife.com
http://adready.com
http://jessicasimpson.com
http://mp3.rhapsody.com
http://www.lyricsmode.com
http://forums.vh1.com
http://www.mcarecords.com
http://blog.vh1.com
http://www.vh1classic.com
http://www.facebook.com
http://twitter.com
http://www.myspace.com
http://exodus.vh1.com
http://www.mtvncareers.com
http://adspecs.mtvn.com
http://learn.rhapsody.com
http://www.socialproject.com

Tabla 4.1: Ejemplo de listado de *links* para una de las webs.

nales. Antes de iniciar el *clustering* debemos caer en la cuenta de que existen *links* a páginas muy populares que podrían influir de forma negativa en el proceso puesto que harían que se uniesen en un mismo *cluster* webs que no tienen porqué tener nada en común. En la actualidad es habitual encontrar enlaces a webs como por ejemplo *'www.facebook.com'*, *'www.twitter.com'*, *'www.linkedin.com'*, *'www.wikipedia.org'* o *'www.myspace.com'* en cualquier web que consultemos. Esto se debe a que son portales que contienen un gran volumen de información de muy diversos ámbitos que no tienen porqué estar relacionados entre ellos y que hacen por tanto que sea muy poco probable que, las webs consultadas que enlazan a estas, tengan algún tipo de relación entre ellas. Resulta claro entonces que no nos conviene unir en un mismo *cluster* dos de los documentos (webs) por compartir uno de estos *links* que consideramos más populares.

Llegados a este punto nos encontramos ante el problema de dar una definición particular y rigurosa del término '*link* demasiado común'. Para resolverlo hemos considerado oportuno obtener, mediante programación en Java [fig. 4.2], un listado completo de todos los *links* que aparecen en los ficheros

de todas las carpetas del conjunto de datos con el que trabajamos, y guardar junto a cada enlace el número de veces que ha sido encontrado. Este listado de pares <*link*, número de apariciones> ha sido ordenado de manera descendiente por el parámetro numérico para poder interpretar de forma más ágil la información que nos proporciona. En [tab. 4.2] podemos ver las 25 primeras posiciones de esta lista resultado, que contiene un total de 144,517 *links* con valores de apariciones que van desde un máximo de 7,042 hasta un mínimo de tan sólo una aparición (cabe destacar que aproximadamente tres quintas partes de las lista la forman *links* que aparecen solamente una vez en el conjunto total de datos).

Llevamos a cabo un estudio de los datos del listado; para ello hemos considerado representarlos gráficamente. Primeramente identificamos cada *link* con un número en [1,|*links*|] y representamos los *links* en el eje de abcisas usando el orden de sus identificadores. En el eje de ordenadas representamos la frecuencia de cada *link*. Normalizamos ambos conjuntos de datos, tanto los valores simbólicos que representan los *links* como la frecuencia de estos, en intervalos de la forma $(0, 1]$. Contamos entonces con un total de 144,517 puntos de la forma (x, y) sobre el plano.
El método que hemos utilizado para determinar el umbral de apariciones para considerar un *link* demasiado común consiste en encontrar el codo de la gráfica que representan estos puntos, considerando que los *links* con apariciones iguales o superiores a este codo son '*links* demasiado comunes'. En [fig. 4.3] podemos ver un fragmento ampliado de la representación gráfica. Para encontrar dicho codo hemos calculado la distancia euclidea de los puntos al origen de coordenadas (podéis ver esta información para los primeros 25 puestos de la lista en [tab. 4.3]). Observamos que la mínima distancia se consigue, con 58 apariciones en el conjunto total de datos, en:

Links	x	y	Distance
http://studio-5.financialcontent.com	0,005445726	0,008094291	0,00975569

Por tanto consideraremos que los *links* del listado que aparecen por debajo de éste son valiosos para el proceso de *clustering* de las webs por *links*, por no considerarlos demasiado comunes.

Con toda esta información nos disponemos a diseñar, implementar y ejecutar el *clustering* por *links*.

```
for each folder c in links dataset do
    for each list l in c do
        for each link e in l do
            if e is in the global link list then
                number of appearances of the link e +1
            else
                add e to the global link list
            end if
        end for
    end for
end for
```

Figura 4.2: Pseudocódigo extracción del listado completo de *links*.

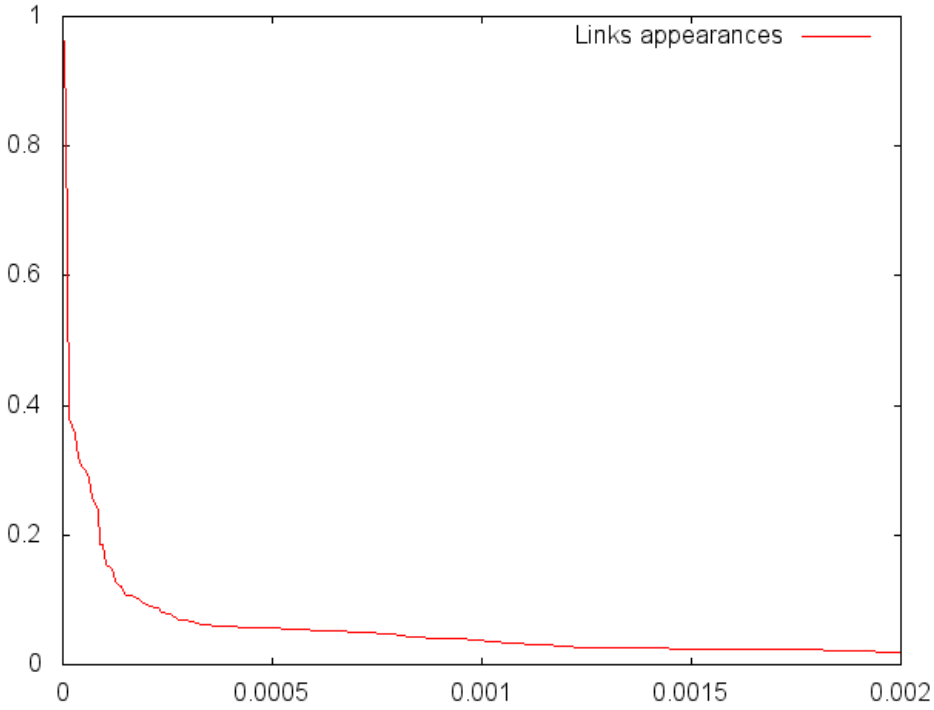

Figura 4.3: Porción ampliada de la representación gráfica de las apariciones de los *links* en el conjunto total de datos.

Links	Appearances
http://www.facebook.com	7.042
http://twitter.com	5.999
http://www.quantcast.com	2.670
http://www.google.com	2.587
http://digg.com	2.510
http://www.addthis.com	2.357
http://del.icio.us	2.195
http://www.amazon.com	2.144
http://ad.doubleclick.net	2.101
http://www.myspace.com	2.033
http://www.twitter.com	1.812
http://www.stumbleupon.com	1.764
http://www.youtube.com	1.669
http://reddit.com	1.310
http://en.wikipedia.org	1.304
http://www.linkedin.com	1.094
http://creativecommons.org	1.075
http://www.imdb.com	1.030
http://www.omniture.com	910
http://feeds.feedburner.com	863
http://www.amazon.co.uk	847
http://www.amazon.de	763
http://www.amazon.co.jp	755
http://www.amazon.fr	755
http://www.flickr.com	749

Tabla 4.2: *Links* más repetidos en el conjunto de datos.

Links	x	y	Distance
http://www.facebook.com	0,00000000	0,999857995	0,999857995
http://twitter.com	0,0000069196	0,851746663	0,851746663
http://www.quantcast.com	0,0000138392	0,379011644	0,379011645
http://www.google.com	0,0000207588	0,36722522	0,367225221
http://digg.com	0,0000276784	0,356290826	0,356290828
http://www.addthis.com	0,000034598	0,334564044	0,334564046
http://del.icio.us	0,0000415176	0,311559216	0,311559219
http://www.amazon.com	0,0000484372	0,304316955	0,304316959
http://ad.doubleclick.net	0,0000553568	0,298210736	0,298210741
http://www.myspace.com	0,0000622764	0,288554388	0,288554395
http://www.twitter.com	0,000069196	0,257171258	0,257171267
http://www.stumbleupon.com	0,0000761156	0,250355013	0,250355024
http://www.youtube.com	0,0000830352	0,236864527	0,236864542
http://reddit.com	0,0000899548	0,185884692	0,185884714
http://en.wikipedia.org	0.0000968744	0,185032661	0,185032687
http://www.linkedin.com	0,000103794	0,155211588	0,155211622
http://creativecommons.org	0,000110714	0,15251349	0,152513531
http://www.imdb.com	0,000117633	0,14612326	0,146123308
http://www.omniture.com	0,000124553	0,129082647	0,129082707
http://feeds.feedburner.com	0,000131472	0,122408407	0,122408477
http://www.amazon.co.uk	0,000138392	0,120136325	0,120136405
http://www.amazon.de	0,000145312	0,108207895	0,108207993
http://www.amazon.co.jp	0,000152231	0,107071855	0,107071963
http://www.amazon.fr	0,000159151	0,107071855	0,107071973
http://www.flickr.com	0,00016607	0,106219824	0,106219954

Tabla 4.3: Datos de los *links* más recurrentes.

4.1.2. Algoritmo de *clustering*

Procedemos a realizar el *cclustering* basado en la estructura de las webs, en nuestro caso, en los *links* de cada web. Queremos agrupar los ficheros originales según la definición: dos webs pertenecen al mismo grupo si tienen por lo menos un *link* en común.

Recodamos que contamos con un listado de los *links* de cada web y otro listado con los *link*s que no serán utilizados para esta tarea por considerar que son demasiado comunes y que no aportan valor al proceso. Estas listas serán los datos utilizados (datos de entrada) en el algoritmo de *clustering*, que ha sido implementado en lenguaje Java y del cual puede verse el pseudocódigo en [fig. 4.4].

Se trata de un algoritmo aglomerativo jerárquico *(HAC - Hierarchical Agglomerative Clustering)* modificado. En minería de datos (*datamining*), un algoritmo jerárquico es aquél que busca crear una descomposición jerárquica de grupos del conjunto de datos, y si dicha descomposición se realiza de abajo-arriba (*bottom-up*, *merging*) entcones se dice que es aglomerativo, si se realizan de arriba-abajo (*top-down*, *splitting*) son llamados algoritmos divisivos. En un algoritmo HAC inicialmente cada documento representa un grupo independiente y en cada iteración se unen los dos grupos con mayor similitud. La elección de una distancia como métrica para determinar la similitud entre dos grupos infuencia a la forma final de estos. Como ventajas de éste tipo de algoritmos encontramos el determinismo de sus resultados y como desventajas que el coste computacional es elevado. Decimos que hemos utilizado HAC modificado puesto que no nos hemos ceñido a la definición que se acaba de dar. A continuación se describe el algoritmo al completo y se matizan las modificaciones.

Descripción del algoritmo:

Recorremos la estructura original de datos y procedemos a tratar cada una de las carpetas. Para cada uno de los ficheros de cada carpeta contamos con el listado de *links* que aparecen en ese fichero (web). El primer documento tratado formará el primer *cluster* y a partir de ahí, utilizando la información de *links*, determinaremos si el fichero que estamos tratando pertenece a algún *cluster* ya creado o él formará uno nuevo. Un fichero pertenecerá a un *cluster* si comparte algún *link* con alguno de los ficheros del *cluster*, es decir:

$$distance(a,b) = \begin{cases} 1 \text{ if } a \text{ and } b \text{ have some common link} \\ 0 \text{ otherwise it} \end{cases} \quad (4.1)$$

Utilizar esta distancia comporta que un documento puediera tener que asignarse a más de un *cluster*, en cuyo caso lo que haríamos es añadir dicho documento a uno de ellos y unir el resto de *clusters* con éste. Por tanto, en una iteración se podrían estar uniendo más de dos grupos. En este hecho, y en el modo de recorrer los ficheros para construir los *clusters*, es donde radican las principales modificaciones respecto al algoritmo aglomerativo jerárquico convencional. Además puede ocurrir que uno de los *links* del listado del fichero que estamos tratando implique la addición del fichero a un determinado *cluster* y que otro *link* del mismo fichero implique la addición a otro *cluster*, en cuyo caso estaríamos en un situación similar a la descrita en las líneas anteriores y estos dos *clusters* a los que se debe asignar el fichero deben fusionarse. Esto provoca que nos encontremos con que webs que no tienen ningún *link* en común acaban perteneciendo al mismo *cluster* por haberse fusionado en algún momento los *clusters* a los que pertenecían.

Tras haber tratado todos los documentos utilizando estos criterios de unión y unificación habremos conseguido una partición del conjunto total de *links* y por tanto del conjunto de ficheros de cada carpeta.

El coste computacional del algoritmo aglomerativo jerárquico modificado que hemos implementado es del orden de $O(n^2)$.

La salida del programa consiste en un fichero *.xml* por cada carpeta del conjunto de datos original, es decir, por cada nombre de persona. Este *.xml* tiene la estructura que podemos observar en [fig. 4.5] y que recoge la información del conjunto de *clusters* resultantes con los correspondientes documentos que pertenecen a cada *cluster*. Seguimos este patrón para expresar el resultado ya que ésta es la estructura que se solicita para luego poder evaluar los resultados y la eficiencia del algoritmo mediante el programa que nos facilita la organización [12].

```
for each folder c in dataset do
    for each document d in c do
        if d is the first file then
            the first cluster is created
            d is added like the first document of the created cluster
            d links are added like links of the created cluster
        else
            for each link treated_link in the links list of d do
                if treated_link is not considered a common web site then
                    for each created clusters k do
                        if d does not belong to k and d is not related to k then
                            for each link consulted_link in the links list of k do
                                if consulted_link is equals to treated_link then
                                    if d already belongs to another cluster k then
                                        Documents of cluster k are added
                                         like documents of cluster q
                                        Links of cluster k are added like links
                                         of cluster q
                                        Relates d with cluster k
                                        Delete cluster k
                                    else
                                        d is added like document of k
                                        d links are added like links of k
                                        d belongs to cluster k
                                    end if
                                end if
                            end for
                        end if
                    end for
                end if
            end for
            if d does not belong to a cluster yet then
                New cluster is created
                d is added like the first document of the new cluster
                d links are added like links of the new cluster
            end if
        end if
    end for
end for
```

Figura 4.4: Pseudocódigo del clustering basado en la estructura web.

```
< clustering  searchString = "person_name" >
        < entity  id = "id_cluster" >
                < documents >
                        < doc  rank = "id_doc" / >
                        < doc  rank = "id_doc" / >
                        ...
                < /documents >
        < /entity >
        < entity  id = "id_cluster" >
                < documents >
                    ...
                < /documents >
        < /entity >
           ...
< /clustering >
```

Figura 4.5: Estructura de los ficheros resultado.

4.2. *Clustering* por contenido

Para llevar a cabo el *cluseting* basado en el contenido de las webs precisamos, como se anunció en el apartado 3.2, del uso de un procesador lingüístico que nos va a facilitar la tarea.
Recordamos que contábamos ya con los ficheros con el contenido $HTML$ de las webs libre de etiquetas y de la mayor parte de caracteres extraños, y que en nuestro caso el procesador lingüístico utilizado ha sido FreeLing [7]. De esta herramienta aprovechamos la asignación de lemas y la clasificación morfológica que hace de cada una de las palabras de los documentos. La idea es poder unir aquellos documentos web en los que el contenido tenga mayor relación semántica.

Los autores del artículos hablan del uso que ellos hacen del contendio también de los sitios webs a los que enlaza cada una de los documentos (ficheros web) tratados, y éste es uno de los pasos que hemos necesitado modificar por no tener acceso inmediato a esa información. Existen otras variaciones realizadas respecto a lo mostrado en el artículo, que se describen más detalladamente en los dos próximos subapartados donde exponemos de forma minuciosa el proceso llevado a cabo para el *clustering* por contenido.

4.2.1. Ejecución del preprocesador lingüístico

En el artículo, sus autores nos describen como utilizan *Apache HTML parser* para tratar el contenido de los documentos y quedarse tan solo con el contenido de las etiquetas $'meta'$ propias de $HTML$, tanto de los originales como de los que corresponderían a las webs a las que tienen enlace directo las que estamos tratando.
Luego aplican un proceso de limpieza distinguiendo el contenido principal del texto de navegación y para ello utilizan una simple heurística por la cual se considera que el contenido significativo consta al menos de 10 términos consecutivos [13] (considerándose un término una secuencia de números o letras acotada por separadores de texto (espacios, signos de puntuación) y restringida a una longitud mayor que uno). Con este trabajo realizado, han obtenido el lema (forma que por convenio se acepta como representante de todas las formas flexionadas (plurales, femeninos, conjugados,...) de una misma palabra) de los términos y han eliminado aquellos que aparecen en la lista estándard de palabras de parada o palabras vacías de la lengua inglesa *(English stopword list)* (algunos ejemplos son: *about*, *above*, *after*, *again*, *against*, *all*, *some*, *such*, *than*, *few*, *each*, *for*, *but*, *by...*).
Con todo ello, han procedido a ejecutar un algoritmo de *clustering* para el cual han creado un vector de términos para cada documento y los han utilizado para ver la similitud entre ellos. El peso de cada término, en el vector del documento, está influenciado por el peso que tenga dicho término en las webs a las que la web tratada enlaza directamente.

En nuestro caso, para realizar el *clustering* basado en el contenido de los documentos no hemos considerado utilizar el contenido de las webs enlazadas, como ya se ha comentado, y tampoco utilizar la heurística mencionada unas líneas más arriba puesto que queremos aprovechar al máximo la información que nos proporciona el documento y consideramos que emplear la heurística podría comportar perder información relevante. Partíamos, como bien hemos dicho antes, de los documentos originales con el contenido parseado, que hemos tratado con el procesador lingüístico FreeLing [7]. La salida devuelta por este procesador de textos nos proporciona un gran número de datos para cada palabra, tales como la posición que ocupa la palabra en la oración a la que pertenece o su correspondiente lema.
Para nosotros sólo será relevante la información de la lematización y el análisis morfológico de las palabras por lo cual hemos realizado una limpieza, mediante un programa Java [fig. 4.6], de todo el conjunto de datos obtenido del procesador y además hemos eliminado aquellas palabras vacías de valor semántico. Para realizar esta tarea nos hemos basado en la información

morfológica que FreeLing nos proporciona al asignarle a cada palabra una de las etiquetas que se pueden ver en Apéndice D. Consideramos con una carga semántica implícita, y por tanto relevantes para el proceso, los siguientes tipos de palabras: nombres *(NN, NNS, NNP, NNPS)*, verbos *(VB, VBD, VBG, VBN, VBP, VBZ)*, palabras extranjeras *(FW)*, adjetivos *(JJ, JJR, JJS)* y adverbios *(RB, RBR, RBS)*. Guardamos este conjunto de lemas (con repeticiones) para cada uno de los documentos con el objetivo de obtener una representación del contenido semántico de cada webs.

Cabe mencionar que se ha realizado una validación para cerciorarse de la correcta ejecución del procesador sobre el gran volumen de documentos con el que trabajamos y en Apéndice C se recoge un listado de los documentos que han sufrido algún tipo de problema en este punto del proceso y de los que por tanto no hemos obtenido una salida del procesador de textos.

Tras realizar estas tareas, contamos con una estructura de carpetas y ficheros, igual a la de los datos originales, dónde hemos almacenado la información obtenida del análisis del contenido de las webs, que nos resultará útil para el proceso de *clustering* que se describe en los siguientes apartados.

```
for each folder do
    for each file f in the dataset returned by linguistic processor
do
        for each line of f do
            if the word has semantic value then
                save lemma of the word for f
            end if
        end for
    end for
end for
```

Figura 4.6: Pseudocódigo del tratamiento de los ficheros obtenidos del procesador de textos.

4.2.2. Representación de los documentos

Contamos con una serie de lemas atesorados de valor semántico para cada documento y con esta información procedemos a representar los documentos como vectores en el espacio n-dimensional, en los cuales cada término representativo del documento web se considera una dimensión.

Los autores del artículo describen como, para confeccionar dichos vectores, utilizan información del contenido de las webs con las que enlazan las webs originales. Nosotros construiremos los vectores de manera que v_i representa el peso del término i en el documento, el cual se calcula utilizando la fórmula de *tf-idf (TF - Term Frecuency, IDF - Inverse Document Frecuency)*, que nos proporciona una medida numérica de cuán relevante es un término de un documento en una colección. El valor del *tf-idf* de una palabra se obtiene como la multiplicación del valor de la frecuencia relativa en el documento (tf) y la inversa del valor de la frecuencia de la palabra en la colección (idf), de manera que aumenta proporcionalmente al número de veces que una palabra aparece en el documento pero se compensa con la frecuencia del término en la colección, lo cual permite manejar el hecho de que algunas palabra son generalmente más comunes que otras.

Sean:

t = término
d = documento
D = colección de documentos
$f(t, d)$ = apariciones del término t en el documento d

Entonces:

$$tf(t, d) = \frac{f(t, d)}{max\{f(\omega, d) : \omega \in d\}} \tag{4.2}$$

$$idf(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|} \tag{4.3}$$

Se ha dividido el pseudocódigo del *clustering* por contenido en dos partes diferenciadas (construcción de los vectores característicos de los documentos de cada carpea y algoritmo de agrupación de los documentos) para poder describir con más detalle y hacer más clara su comprensión. En [fig.4.7] podemos ver el algoritmo de creación de los vectores como representación de los documentos:

Para cada carpeta se crea un objeto HashMap de Java al que llamaremos

map_folder (hemos utilizado este objeto de Java por la facilidad que ofrece para consultar la información que contiene). Hacemos una primera vuelta a todos los documentos de la carpeta para guardar la información de cuántas veces aparece cada lema en el conjunto total de ficheros de dicha carpeta (esta información nos es necesaria para poder aplicar luego la fórmula del *idf*).
Volvemos a recorrer la carpeta y para cada uno de los documento que contiene creamos otro HashMap, al que nos referiremos como *map_document*, y recorremos el documento guardando los lemas que aparecen en él y el número de veces que lo hacen (*map_document: <lemma, #appearances>*), y lo ordenamos por el valor numérico en orden descendiente (este es el punto en el cual obtamos o bien por quedarnos con los treinta lemas más frecuentes en el documento o bien con el total de lemas que aparecen en él). Estos HashMap representan los vectores que mencionan en el artículo y para asignar un peso a cada dimensión, es decir, a cada lema, aplicamos la fórmula del *tf-idf* (*map_document: <lemma,* $tfidf\_value$*>*). Aprovechamos este recorrido por los lemas de cada documento para calcular la norma de cada vector, que utilizaremos posteriormente para el cálculo de las similitudes del coseno. Añadimos los *map_document* al mapa de la carpeta, *map_folder*, de manera que éste contenga el nombre de todos los ficheros de la carpeta y los lemas de cada documento con el número de repeticiones (*map_folder: <document_name, map_document>*).

En la siguiente sección se describe explicitamente el algoritmo de *clustering* que emplea los vectores construidos.

```
for each folder c in dataset do
    create map_folder
    for each document d in c do
        create map_document
        for each lemma l in d do
            save number of documents where l appear (for apply idf)
            add lemma to map_document: <lemma, # appearances>
        end for
    end for
    for each document d in c do
        create d_norm
        for each lemma l in d do
            create tfidf_value for l
            calculate tfidf_value like the multiplication of (eq. 4.2) and (eq. 4.3)
            change lemma value in map_document: <lemma, tfidf_value>
            add tfidf_value to calculate d_norm
        end for
        add map_document to map_folder: <d, map_document>
        calculate d_norm
    end for
    ...
end for
```

Figura 4.7: Pseudocódigo representación de los documentos como vectores.

4.2.3. Algoritmo de *clustering*

Nos disponemos a realizar el *clustering* de los ficheros originales basándonos en su contenido. Llegados a este punto disponemos de los vectores representativos de cada documento cuya contrucción se ha descrito en el apartado anterior.

Contamos con dos versiones del mismo algoritmo, en las cuales tan sólo varia el número de términos de cada documento que utilizamos para llevar a cabo el proceso. En el artículos, sus autores toman como representación de un documento sus treinta términos más repetidos y nosotros, además de reproducir esto, hemos hecho otra versión en la que utilizamos todos los lemas que habíamos guardado de cada documento. Posteriormente compararemos resultados y valoraremos el impacto de utilizar una u otra opción.

A diferencia del *clustering* por *links*, en este caso el algoritmo utilizado se

ajusta perfectamente a la definición de *HAC (Hierarchical Agglomerative Clustering)*, y hemos tomado la similitud del coseno entre los vectores (eq. 4.4) como medida para determinar la distancia (eq. 4.5) entre *clusters*, de modo que mayor valor de similitud del coseno implica mayor semejanza entre el contenido de los *clusters*. Se ha fijando el umbral de similitud en 0.1 lo cual significa que llegados al punto en que el mayor valor de similitud del coseno entre *clusters* se encuentra por debajo de ese umbral, se detiene el proceso. A continuación vemos con más detalle la parte de pseudocódigo que corresponde al algoritmo de agrupación de los documentos que hemos aplicado.

$$cosine\_similarity(a, b) = \frac{\sum_{i,j} a_i b_j}{\sqrt{\sum_i a_i^2}\sqrt{\sum_j b_j^2}} \tag{4.4}$$

$$distance(a, b) = 1 - (cosine\_similarity(a, b)) \tag{4.5}$$

Descripción del algoritmo [fig. 4.8]:

Seguimos un procedimiento similar al de la creación de *map_document* y *map_folder*, que se ha descrito en la sección anterior, para guardar ahora la información de los *clusters* desde los que partimos, ya sea considerando que cada documento forma por si sólo un *cluster* o considerando una fase de agrupamiento previa al *clustering* por contenido (como podría ser el caso de partir de los recultados obtenidos del *clustering* por *links*). De esta manera, contaremos con un nuevo HashMap, *map_clusters*, con la información de los *clusters* con los que contamos previamente a la aplicación del algoritmo *HAC* (*map_clusters: <cluster_name, list of documents belonging to the cluster>*).
Con toda esta información almacenada y accesible desde *map_clusters* y *map_folder*, iniciamos el algoritmo de *clustering* por contenido.
Creamos la variable *joining_cosine_similarity* inicializada a 0, que va a almacenar el valor de la máxima similitud entre *culsters* obtenida en cada iteración del algoritmo. Siempre que *joining_cosine_similarity* sea superior a 0.1, que es el umbral fijado para detener el algoritmo, se inicia una nueva iteración en la que recorremos cada uno de los *clusters* formados en ese momento y para cada uno de ellos recorremos los que son diferentes a él (se controla que no haya repeticiones mediante una variable booleana). Para cada par de *clusters* creamos una variable que guardará la similitud del coseno entre ellos y a la cual hemos llamado *calculated_cosine_similarity*. Ahora recorremos todos los

documentos de cada uno de los dos *clusters* y calculamos para cada combinación de dos documentos (uno de cada *cluster*) la similitud del coseno entre sus vectores. Almacenamos esta información para seguidamente calcular el valor de la similitud entre *clusters*, *calculated_cosine_similarity*, que se obtiene como la media de los valores de las similitudes entre los documentos de los dos *clusters*.
Para cada *cluster* tenemos la infomación de cuál es el *cluster* más próximo a él *cluster_cosine_similarity*, y nos quedamos, en cada iteración del algoritmo, con el par de *clusters* cuyo contenido es más similar, es decir, que sus tienen mayor similitud del coseno.
Tras realizarse el número oportuno de iteraciones del algoritmo obtenemos una partición de los documentos agrupados según su contenido, de manera que la similitud entre los grupos resultantes será inferior a 0.1.

El coste computacional del algoritmo algomerativo jerárquico implementado para el *clustering* por contenido es del orden de $O(n^2)$.

Cabe destacar que los tiempos de ejecución varian mucho entre las dos implementaciones realizadas. Así pues, el hecho de utilizar los 30 términos más frecuentes de cada documento reduce el tiempo de ejecución entorno a un 70 % respecto a utilizarlos todos ya que trabajamos con un gran volumen de datos.

Guardamos la información de los *clusters* resultantes del proceso en ficheros que mantiene la misma estructura que en el *clustering* por *links* y de la cual podemos ver el esquema en [fig. 4.5]. En este caso tendremos diferentes carpetas con estos ficheros, en concreto una por cada variación realizada en el algoritmo. De esta manera vamos a poder valorar el grado en el que influyen estas variaciones en los resultados finales, comparándolos entre ellos y con los obtenidos por los autores del artículo que estamos tomando como referencia.

```
for each folder c in dataset do
    ...
    create map_clusters
    Input: initial_clusters
    if initial_clusters is not empty then
        for each created cluster k do
            add k to map_clusters: <k, documents of k list>
        end for
    else
        for each document d in D do
            create a cluster k with d
            add k to map_clusters
        end for
    end if
    create joining_cosine_similarity = 0
    while (joining_cosine_similarity > threshold) or (it's first time) do
        for each cluster A in map_clusters do
            create clusters_cosine_similarity = 0
            for each cluster B in map_clusters different to A do
                create calculated_cosine_similarity = 0
                for each document d1 in A do
                    for each document d2 in B do
                        calculate the similarity between documents (eq. 4.4)
                        add it to calculated_cosine_similarity
                    end for
                end for
                calculated_cosine_similarity = calculated_cosine_similarity / (|A|·|B|)
                if calculated_cosine_similarity > clusters_cosine_similarity then
                    clusters_cosine_similarity = calculated_cosine_similarity
                end if
            end for
            if clusters_cosine_similarity > joining_cosine_similarity then
                joining_cosine_similarity = clusters_cosine_similarity
            end if
        end for
        join clusters with cosine similarity joining_cosine_similarity
        update map_clusters
    end while
end for
```

Figura 4.8: Pseudocódigo del *clustering* basado en el contenido web.

Capítulo 5

Evaluación

5.1. Marco de evaluación

El artículo que se ha tomado como referencia se enmarca dentro de una competición internacional, WePS-3 [1] (*Web People Serch Task*, concretamente a la tarea de desambiguación de nombres de persona consistente en agrupar documentos referentes a un mismo individuo) cuya organización ha facilitado datos de prueba sobre los cuales aplicar los algoritmos oportunos y luego poder medir su eficiencia. El corpus que hemos empleado han sido estos mismos datos que consisten en 300 carpetas, es decir, 300 nombres de persona, y 200 documentos web para cada nombre, por tanto, hemos trabajado con un volumen de unos 60.000 documentos en total. Cabe destacar que se ha detectado la ausencia de algunos de los documentos desde el origen, que corresponden a páginas web vacías, en concreto en el apéndice A se recoge un listado de los documentos no encontrados (alrededor de un $4,5\,\%$ sobre el total de datos). Los documetnos webs corresponden a ficheros *.html* con un nombre de la forma *index*000 y estan organizados por carpetas según el nombre de persona al que se refieren.

La lista de nombres esta compuesta por:

- 50 nombres del Censo de Estados Unidos.
- 50 nombres de Wikipedia.
- 50 nombres de alguna lista del Comité del Programa de Ciencias de la Computación.
- 50 nombres donde hay al menos un abogado para cada nombre.

- 50 nombres donde hay al menos un ejecutivo de una empresa para cada nombre.
- 50 nombres donde hay al menos un agente inmobiliario para cada nombre.

Y los documentos webs para cada nombre corresponden a los primeros 200 resultados de búsqueda en *Yahoo!* para ese nombre.

Para la evaluación de los resultados, la organización ha seleccionado manualmente a dos personas para cada nombre (que denotaremos A y B) que tengan una canitidad razonable de infomación disponible en el conjunto de datos. De esta manera se considerarán tres casos para cada documento de cada carpeta con nombre de persona: que corresponda a la persona A, que corresponda a la persona B o que corresponda a una persona distinta de A y B.

Las métricas de evaluación usadas (que se describen con detalle en el siguiente apartado) serán: *BCubed precision* y *BCubed recall* combinado con *Van Rijsbergen's F measure*.

5.2. Resultados

5.2.1. Descripción

Tras haber obtenido los distintos clusterings de los documentos originales se ha generado para cada uno de ellos una carpeta dentro de la cual hay un fichero por cada carpeta del conjunto de datos inicial (es decir, por cada nombre de persona). Estas carpetas con estos ficheros han sido creadas por cada uno de los códigos Java ejecutados, por tanto contamos con 5 carpetas distintas que corresponden a:

- **CL**: Resultado de la ejecución del algoritmo de *clustering* basado en la estructura web, en este caso, en los *forward-links* (no se incluye el algoritmo de *clustering* basado en el contenido).
- **CC**: Resultado de la ejecución del algoritmo de *clustering* basado en el contenido de las webs (no incluye el clustering basado en los links). En este caso se utilizan el total de términos que se consideran descriptivos o representativos para cada documento (web).
- **CC30**: Resultado de aplicar el algoritmo de *clustering* basado en el contenido (no incluye el clustering por links). En este caso, del total

de palabras que se considera que representan semánticamente al documento, se han utilizado solamente los treinta términos más frecuentes.

- **CL-CC**: Resultado obtenido de aplicar el algoritmo basado en el contenido, con el total de términos relevantes de cada documento, sobre los *clusters* resultantes de la ejecución del algoritmo basado en los *links*.

- **CL-CC30**: Resultado de aplicar el algoritmo basado en el contendio sobre los *clusters* resultantes de la agrupación por *links*, pero en este caso empleando tan solo los treinta términos relevantes más frecuentes en el documento.

Como se ha dicho en el capítulo anterior la estructura de los ficheros es la que se muestra en [fig. 4.5]. Esta es la estructura fijada por la organización de *WePS-3* [2] para poder ejecutar el programa de validación cuyo código facilitan ellos mismos. La salida de dicho programa consta de un fichero por cada clustering realizado en el cual aparece información detallada sobre cada una de las carpetas originales. Además se genera un fichero (*global_averages.tsv*) donde aparece información sobre los distintos clusterings en general y que nos permite analizar y comparar facilmente los resultados de los diferentes algoritmos programados.
Los parámetros de los cuales se informa, tanto en los ficheros individuales de cada clustering como en global de todos, son: *precision*, *recall* y *F-measure*.

Sean:

True positives: documentos clasificados correctamente dentro de un *cluster*.

False positives: documentos que se han clasificado en un determinado *cluster* pero que realmente no deberian pertenecer a él.

False negatives: documentos que no han sido clasificados como pertenecientes al *cluster* cuando realmente sí deberían pertenecer a él.

True negatives: documentos que no han sido clasificados como pertenecientes al *cluster* y que ciertamente no deben pertenecer.

Entonces:

- **Precision**: Indica que proporción de elementos clasificados está correctamente clasificado, es decir, es una medida de la exactitud con la que se clasifica un elemento.

$$precision = \frac{true\_positives}{true\_positives + false\_positives} \tag{5.1}$$

- **Recall**: Indica la proporción de elementos clasificados correctamente del total de elementos que deberian haber sido clasificados, es decir, es una medida de la capacidad del modelo de predicción para seleccionar elementos de un determinado grupo dentro de un conjunto de datos.

$$recall = \frac{true\_positives}{true\_positives + false\_negatives} \tag{5.2}$$

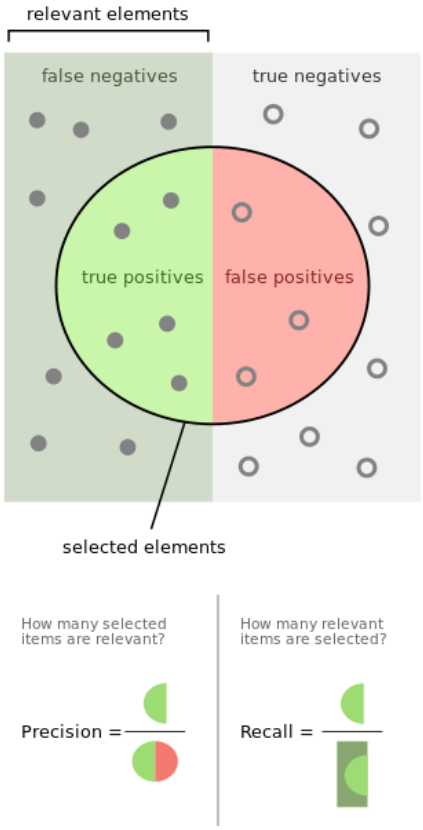

Figura 5.1: Esquema obtención *precision* y *recall*.

- **F-measure**: medida de la exactitud de una prueba que utiliza tanto la *precision* como el *recall*. Toma valor en [0,1] y se obtiene de:

$$F - measure = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{5.3}$$

Las medidas **Bcubed**, que son las empleadas en esta ocasión, son un caso particular de *precision* y *recall* en que ambas medidas se calculan para cada elemento individual (en nuestro caso para cada documento), y luego se devuelve la media de los valores obtenidos para cada documento [14].

Contar con un mayor valor de *precision* o un mayor valor de *recall* es preferible dependiendo de los intereses particulares del uso que se vaya a hacer del clustering, es por ello que es util contar con *F-measure* para el análisis de los resultados, ya que involucra ambas medidas.

En [tab. 5.1] podemos ver los resultados que hemos obtenido y que se procede a comparar y analizar en el siguiente apartado.

System	avg. precision	avg. recall	avg. F-measure
CL-CC	0,76	0,39	0,48
CL-CC30	0,76	0,39	0,47
CL	0,85	0,34	0,46
CC	0,81	0,34	0,43
CC30	0,83	0,32	0,42

Tabla 5.1: Resultados de los *clusterings*.

5.2.2. Análisis comparativo

Los autores del artículo que hemos seguido para la realización de este proyecto nos exponen los resultados de los distintos *clusterings* que han realizado, de los cuales nos interesan los siguientes:

- **PPR**: Resultados del *clusterings* basado en los top-8 links de cada web y utilizando *Personalized PageRank* (posteriormente no se realiza el clustering basado en el contenido).

- **HAC**: Resultados del *clusterings* utilizando un algoritmo HAC *(Hierarchical Agglomerative Clustering)* sobre el contenido de las webs (no interviene el clustering utilizando links).

- **PPR-HAC**: Resultados obtenidos del *clusterings* utilizando los top-8 links de cada web y *Personalized PageRank*, y aplicando HAC sobre el contenido de las webs.

En [tab. 5.2] hemos recogido la información referente a sus resultados junto con los obtenidos por nosotros, y se han agrupado adecuadamente para hacer más sencilla la tarea de comparación. Se puede ver como aparece también en la tabla los valores pertinentes a considerar que cada documento dentro de una carpeta es un *cluster* por si sólo (*One-in-one*), y considerar que existe un único *cluster* que contiene a todos los documentos de la carpeta (*All-in-one*).

En el caso del algoritmo de *clusterings* basado en la estructura de *links* (CL) debemos tomar como referencia su PPR, aunque finalmente no se ha utilizado el mismo criterio que en el artículo para seleccionar los *links* que se han utilizado de cada web ni siquiera se ha empleado el mismo número de *links* (recordamos que ellos emplean los top-8 *links* en base a su criterio de clasificación). Los resultados que hemos obtenido en nuestro caso están 0,07 puntos por encima en lo que se refiere a *F-measure* porque, aunque disminuye la *precision*, aumenta el valor de *recall* respecto a sus resultados.

El algoritmo de *clusterings* basado en el contenido (CC y CC30) debe ser comparado con su HAC. En este caso vemos que los valores de *recall* y de *precision* son muy distintos y no siguen la tendencia que se aprecia en los recultados del HAC, en la que los valores son muy similares entre ellos. Recordamos que ellos hacen uso también de los contenidos de las webs con las que enlaza cada documento original para calcular las métricas de los contenidos y nosotros tan sólo hemos utilizado el contenido explícito de cada documento. Pese a las diferencias en los resultados, que son fruto de las continuas variaciones que han sido necesarias realizar respecto lo descrito en el artículo, se han obtenido unos resultados de nuevo un poco superiores a los suyos.

Por último comparamos los resultados de los algoritmos que involucran ambas fases, (CL-CC30) y PPR-HAC. Vemos que en este caso no logramos alcanzar unas cifras como las suyas y nos quedamos 0,02 puntos por debajo en *F-measure*, aunque con un valor de *precision* algo mayor.

En general se observan resultados satisfactorios, sobre todo en la ejecución de los algoritmos de forma independiente (CL, CC30 y CC), pese a las licencias tomadas a lo largo del proceso y pese a la perdida de documentos que se ha producido en algunos puntos del desarrollo (por caracteres extraños, en el procesador lingüístico,..).

Analizando los resultados que hemos obtenido [tab. 5.1], independientemente de los suyos, podemos ver como los mejores resultados se consiguen al utilizar conjuntamente el *clusterings* basado en la estructura web y el *clusterings* basado en el contenido. El hecho de utilizar el total de términos relevantes para un fichero, respecto a solamente utilizar los treinta más frecuentes, no parece muy indicado ya que los resultados finales son muy similares pero sin embargo el coste temporal de ejecución del algoritmo es mucho mayor.
Vemos también como se obtienen resultados muy positivos del *clusterings* solo basado en la estructura de *links* de las webs y como, igual que antes, el hecho de utilizar los treinta términos más frecuentes o utilizarlos todos en el *clusterings* basado sólo en el contenido, no provoca grandes variaciones en los resultados pero sí existen grandes diferencias en los tiempos de ejecución.
En todos los casos vemos que la tendencia es un valor alto de *precision* y un valor de *recall* inferior, lo cual significa que no se logran clasificar muchos documentos pero los que sí se clasifican lo hacen con una probabilidad alta de estar clasificados correctamente.

System	avg. precision	avg. recall	avg. F-measure
PPR-HAC	0,7	0,45	0.5
CL-CC	0,76	0,39	0,48
CL-CC30	0,76	0,39	0,47
HAC	0,43	0,4	0,41
CC	0,81	0,34	0,43
CC30	0,83	0,32	0,42
PPR	0,93	0,27	0,39
CL	0,85	0,34	0,46
One-in-one	1	0,23	0,35
All-in-one	0,22	1	0,32

Tabla 5.2: Comparativa de resultados.

Capítulo 6

Planificación

Al inicio del proyecto se elaboró un diagrama de Gantt con la planificación que se esperaba llevar a cabo durante los meseses comprendidos entre la elección del proyecto y la entrega y exposición del mismo. Puede verse dicho diagrama en [fig. 6.1], con el título de 'Diagrama de Gantt inicial'. Como se puede observar, el número de tareas que se consideraron en un principio no es muy elevado y se distribuyen básicamente en una fase inicial de familiarización con la temática del proyecto y los datos a tratar, una fase más práctica de implementación y ejecución de los programas necesarios para la desambiguación de los ficheros, y una fase teórica de documentación del proceso y redacción de la memoria.

Paralelamente al desarrollo del proyecto se ha ido elaborando y actualizando otro diagrama que refleja las tareas realizadas y los tiempos reales que se han necesitado para cada una de ellas. Puede verse dicho diagrama en [fig. 6.2] bajo el tílulo 'Diagrama de Gantt real'.

Las diferencias entre el primer y el segundo diagrama son bastante claras. Primeramente cabe desacar que las tareas del primer diagrama se han fraccionado en el segundo, y han aparecido otras nuevas que no se contemplaron inicialmente.

Las variaciones en la planificación han sido debidas a diversos motivos, uno de ellos es la gran cantidad de datos con el que se ha trabajado y que ha dificultado enormemente cualquier tarea, desde la obtención de los links o la asignación de lemas hasta los propios algoritmos de *clutering* implementados. El volumen de datos, que como hemos matizado en el apartado 5.1 alcanza entorno a los 60.000 ficheros, ha afectado tanto a los tiempos de ejecución de cada uno de los distintos programas implementados en Java, como

a la memoria necesaria para almacenar y procesar todos esos datos y las correspondientes salidas de cada programa. Además, nos hemos encontrados con impedimentos como la denegación de permisos por parte de Google para consultar el *PageRank* de webs de forma masiva y que nos obligó a buscar otras alternativas eficientes y eficaces para conseguir resultados similares a los obtenidos en el artículo. Cada variación del proceso respecto a lo descrito en el artículo ha implicado unos tiempos de reflexión, análisis y pruebas para finalmente encontrar las alternativas que hemos utilizado en cada caso.

En el diagrama real vemos como el tiempo dedicado a las tareas previas a la programación se dilata más de lo que se pensó en un principio. Ha llevado tiempo familiarizarse con los datos, entender el objetivo a conseguir y madurar unos algoritmos eficientes para llevar a cabo el *clustering* de los ficheros (webs). Cabe destacar también que el pretratamiento de los documentos, tanto para extraer los enlaces de cada fichero como para obtener los lemas dotados de valor semántico mediante el procesador lingüístico, ha sido una de las partes más arduas del proceso y que no se tuvo en cuenta inicialmente.

Otra de las cosa que no se valoró inicialmente es que para cada programa ejecutado se tenía que hacer un análisis de la calidad de los resultados obtenidos, así como repetir las ejecuciones en ciertos casos tras aplicar pequeñas modificaciones en el código para finalmente conseguir los resultados definitivos.
En este segundo diagrama hemos separado en dos fases independientes la implementación y ejecución de los dos algoritmos de *clustering* la cual cosa nos ha permitido paralelizar temporalmente estas tareas.

Vemos también como el tiempo de aprendizaje del lenguaje de programación Java (el cual era uno de los objetivos del proyecto) se dilata en el tiempo coincidiendo prácticamente con el tiempo dedicado a implementar los diferentes algoritmos puesto que el aprendizaje ha sido continuo y creciente a lo largo del proceso. Por último comentar la compresión del tiempo dedicado a la redacción de la memoria escrita, que ha tenido que adaptarse a las circunstancias del proyecto.

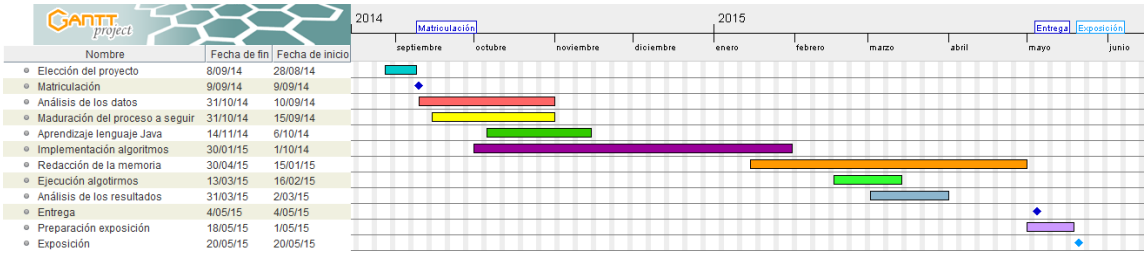

Figura 6.1: Diagrama de Gantt inicial.

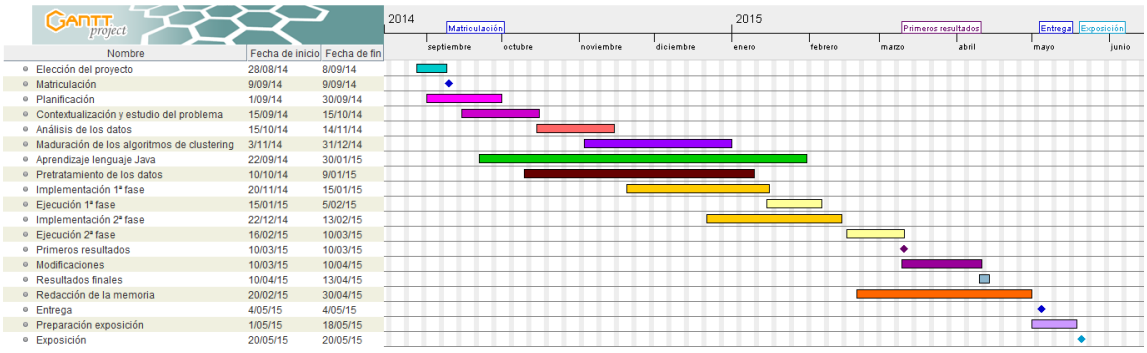

Figura 6.2: Diagrama de Gantt real.

Capítulo 7

Conclusiones y Trabajo futuro

7.1. Posibles extensiones

A lo largo del desarrollo de este proyecto ha sido necesario tomar ciertas decisiones para encontrar alternativas a algunos de los pasos realizados por los autores del artículo.

El hecho de no poder hacer uso del *PageRank* por parte de Google debido a la falta de permisos para hacer consultas sobre un gran número de webs supuso la necesidad de buscar alternativas con el fin de obtener un conjunto de *links* por cada web que nos fuese útil en el propósito de agrupar webs referidas a una misma persona. Para ello se recurrió a ordenar el total de *links* encontrados en el conjunto de datos según su frecuencia, determinar cuales de ellos eran demasiado comunes para resultar útiles en el proceso y encontrar así los *links* que caracterizaban a cada documento, sin hacer uso del *Personalized PageRank* y del método de Monte Carlo (en [10] encontramos la descripción y el análisis del uso del método de Monte Carlo en el cálculo del *PageRank*).

Otra limitación la encontramos al no poder contar con el contenido de las webs a las que enlazan las webs tratadas durante el *clustering* por contenido. Como alternativa al vacío de información en este sentido se intentó aprovechar al máximo la información que nos proporciona el documento directamente, por ejemplo, pasando por alto el uso de la heurística que ellos consideran para reducir el volumen de datos, y que determina que el contenido que es relevante en un documento consta al menos de 10 términos consecutivos (en [13] podemos encontrar la información y el análisis pertinente al uso de dicha heurística).

Resulta evidente que todas estas licencias tomadas a lo largo del desarrollo comportan diferencias en los resultados obtenidos.
En esta misma línea podrian realizarse otras modificaciones en el proceso para estudiar el impacto que supondría en los resultados.

- Tomar otro criterio a la hora de determinar si un *link* es considerado común o no.
- Fijar un máximo de *links* a utilizar de cada uno de los documentos.
- Priorizar el valor de ciertas partes del documento, haciendo uso de las correspondientes etiquetas de $HTML$, como podría ser el título del documento web o las descripciones de las fotografías.
- Tomar sólo los sustantivos y adjetivos como términos relevantes en el resultado del análisis morfológico.
- Si se tubiese acceso a los *backward-links*, realizar con ellos el mismo algoritmo de *clustering* de los documentos basado en la estructura web que en nuestro caso se ha realizado con los *forward-links*.

Estos son sólo algunos ejemplos de las numerosas posibles variaciones o modificaciones del proceso descrito en este proyecto pero podrían haberse citado muchas otras. Dependiendo de los resultados obtenidos y del interés con el que se aplique la agrupación de documentos será más apropiado utilizar uno u otro modelo de *clustering*; puede priorizarse una *precision* alta a un *recall* alto o viceversa, o pueden priorizarse los resultados al tiempo de ejecución,...

7.2. Retrospectiva

Realizar un proyecto con un alto componente práctico comporta algunos riesgos e inevitables alteraciones en la planificación inicial.

Los tiempos destinados a la programación han sido mayores a los estimados en un principio, y la problemática común en el mundo de la programación así como el volumen de datos con el que se ha trabajado, han sido algunas de las causas de que el proceso se haya relentizado. En general el modo de proceder ha sido bastante regular, para cada nuevo paso ha sido necesaria una fase de interiorización de los objetivos a conseguir y de valoración de posibles vías para conseguir dichos objetivos. Tras cada fase ha sido necesario

un análisis de los resultados y las pertinentes correcciones.

El punto débil lo encontramos en los diversos problemas surguidos durante el desarrollo que se han comentado a lo largo de la memoria, y sin duda el punto fuerte han sido las maneras halladas para resolverlos, que como se ha podido comprobar en el capítulo 5, nos han permitido obtener resultados muy similares a los obtenidos por los autores del artículo y en algunos casos incluso mejores, como son las ejecuciones independientes del algoritmo de *clustering* por *links* y del algoritmo de *clustering* por contenido.

Uno de los objetivos personales logrado con la realización de este proyecto ha sido el aprendizaje del lenguaje de programación Java. Se ha hecho una inmersión total en un proyecto de programación, con la correspondiente fase de diseño previa, la codificación en sí, y una fase de testeo posterior, todo ello con un volumen de datos real y considerable. A esto se le une el uso y la familiarización con conceptos propios de la minería de datos (*datamining*) y del procesamiento del lenguaje natural. Además, se han aplicado numerosos conceptos matemáticos a lo largo del desarrollo: similitud del coseno, tf-idf, cálculo del codo de la gráfica de *links*,...

7.3. Trabajo futuro

El objetivo práctico de este proyecto sería poder utilizar los algoritmos realizados para implementar un clasificador de documentos.

Como ya se ha dicho, con este proyecto se logra desambiguar documentos relativos a personas, es decir, dado un conjunto de webs que se refieren a personas que comparten el mismo nombre pero que son personas distinas, se consigue (con mayor o menor probabilidad de acierto) agrupar los documentos dependiendo de la persona a la que se refieran en concreto. El objetivo final sería utilizar todo esto para que, si por ejemplo un usuario tiene abierta en el navegador una web que se refiere a una persona, el buscador supiera filtrar los datos de búsqueda para que al usuario sólo le aparecieran los resultados referentes a esa persona concreta. Por tanto, se trataría de lograr clasificar un nuevo documento en alguno de los *clusters* obtenidos.

En [fig. 7.1] podemos ver el pseudocódigo simplificado de lo que sería el clasificador de documentos. El primer paso sería realizar el *clustering* de los datos, excluyendo el que se quiere clasificar, basándose en el modelo que ha dado mejores resultados, es decir, empleando primero el *clustering* por *links*

y posteriormente el *clustering* por contenido sobre los resultados del primero. Con los *clusters* resultantes y añadiendo el documento a clasificar como un nuevo *cluster*, se repetiría el proceso. Concretamente se fusionarían en el mismo *cluster* todos aquellos que compartan algún *link* con el documento a clasificar en la fase de *clustering* basado en la estructura web. Para el *clustering* por contenido se obtendrían los correspondientes vectores para cada documento y se utilizarían de igual modo que en lo descrito en 4.2.2. El motor de búsqueda debería filtrar los resultados de modo que al usuario le aparecieran los documentos que pertenecen al mismo *cluster* que él al finalizar el proceso.

```
for each document D for be classified do
    apply structure based clustering to all document d ≠ D → link_clusters
    apply content based clustering to link_clusters → content_clusters
    consider D like a cluster → clusters = content_clusters + D
    apply structure based clustering to clusters → final_link_clusters
    apply content based clustering to final_link_clusters → final_clusters
    if D ∈ k cluster ∈ final_clusters then
        return all documents in k different to D
    end if
end for
```

Figura 7.1: Pseudocódigo simplificado del clasificador de documentos.

Bibliografía

[1] Javier Artiles, Andrew Borthwick, Julio Gonzalo, Satoshi Sekine and Enrique Amigó. *WePS-3 Evaluation Campaign: Overview of the Web People Search Clustering and Attribute Extraction Tasks.* WePS-3 Person Name Disambiguation Task (2009-2010).

[2] Elena Smirnova, Konstantin Avrachenkov and Brigitte Trousse. *Using Web Graph Structure for Person Name Disambiguation.* WePS-3 Person Name Disambiguation Task (2010).

[3] Ying Chen, Sophia Yat Mei Lee and Chu-Ren Huang. *PolyUHK: A Robust Information Extraction System for Web Personal Names.* WePS-3 Person Name Disambiguation Task (2010).

[4] Minoru Yoshida, Masaki Ikeda, Shingo Ono, Issei Sato and Hiroshi Nakagawa. *Person Name Disambiguation by Bootstrapping.* WePS-3 Person Name Disambiguation Task (2010).

[5] Chong Long and Lei Shi *Web Person Name Disambiguation by Relevance Weighting of Extended Feature Sets.* WePS-3 Person Name Disambiguation Task (2010).

[6] Jonathan Hedley. *Jsoup: Java HTML Parser* [en línea] <http://jsoup.org>

[7] Lluís Padró y Evgeny Stanilovsky. *FreeLing Home Page* [en línea] <http://nlp.lsi.upc.edu/freeling/>

[8] Xavier Carreras and Isaac Chao and Lluís Padró and Muntsa Padró. *FreeLing: An Open-Source Suite of Language Analyzers.* Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC'04) (2004).

[9] T. Haveliwala. *Topic-sensitive pagerank.* Proceedings of the 11th WWW Conference pp. 517-526 (2002).

[10] K.Avrachenkov, N.Litvak, D.Nemirovsky and N.Osipova *Monte Carlo methods in PageRank computation: When one iteration is sufficient.* SIAM Journal on Numerical Analysis (2007).

[11] *Gnuplot homepage* [en línea] <http://www.gnuplot.info>

[12] Natural Language Processing and Information Retriveral Group at UNED. *WePS 3: searching information about entities in the Web* [en línea] <http://nlp.uned.es/weps/weps-3>

[13] Christian Kohlschütter, Peter Fankhauser and Wolfgang Nejdl *Boilerplate Detection using Shallow Text Features.* WSDM '10 Proceedings of the third ACM international conference on Web search and data mining (2010).

[14] Amit Bagga and Breck Baldwin. *Algorithms for Scoring Coreference Chains.* First International Conference on Language Resources and Evaluation Workshop on Linguistics Coreference. (1998)

Apéndice A

Listado de documentos no encontrados desde origen

Folder name	Files not found
aaron_hall	4, 5, 41, 43, 49, 50, 79, 83, 135, 184
alan_clark	34, 50, 57, 85
alan_cox	10, 25,3 1, 123, 135
alan_morrison	58, 68, 84, 88, 92, 152, 155, 180
alan_smeaton	3, 71, 95
albert_kramer	5, 74, 99, 101, 105, 149, 150, 179, 186
alexis_charles	25, 28, 33, 43, 50, 51, 61, 64, 124, 161, 188
alice_morris	4, 6, 60, 79, 94, 157, 170, 182, 184, 186, 191
alice_perez	10, 22, 31, 45, 71, 146, 174, 186
alice_robinson	3, 7, 19, 55, 76, 147, 172, 180, 193
al_feldstein	82, 97, 118, 129, 138
al_miller	2, 19, 26, 66, 81, 130, 133, 144, 153
amanda_grace	2, 3, 71, 93, 99, 100, 130, 167, 183
andrea_mcdaniel	1, 10, 15, 24, 25, 27, 42, 48, 60, 63, 74, 93,147
andy_fitzpatrick	7, 10, 35, 55, 65, 92, 95, 117, 137, 139, 152, 164, 170, 174, 188, 196
andy_meyers	3, 4, 92, 101
anna_anderson	16, 27, 30, 31, 34, 46, 56, 73, 89, 109, 117, 133, 135, 146, 162, 168, 172, 176, 178
annetta_scott	7, 19, 49, 51, 56, 95, 101, 106, 145, 156, 195
anne_de_roeck	36, 96, 106, 123, 149, 151, 174, 192, 193

joyce_thomas	5, 13, 17, 77, 86, 119, 129, 134, 135, 162, 166, 193
juan_dominguez	7, 61, 123, 198
judy_morris	32, 49, 56, 61, 64, 79, 126, 136, 176
julie_stephens	3, 10, 16, 17, 37, 95, 117, 156
jun_yan	14, 18, 19, 77, 97, 113, 122, 192
kalervo_jarvelin	4, 12, 46, 80, 120, 121, 138, 143, 148, 153, 163, 164, 173, 198
karen_edwards	2, 4, 8, 51, 61, 64, 110, 174
karen_shaver	4, 15, 58, 70, 160
katherine_perez	15, 17, 77, 85, 105, 115
kathleen_horton	4, 19, 27, 55, 57, 67, 108, 157, 166, 195
kathy_wood	2, 3, 25, 51, 111, 145, 163, 197
keith_mcallister	2, 3, 15, 22, 48, 191
kelly_warren	16, 17, 57, 64, 90, 92, 94, 96, 102, 110, 127, 129, 139, 179, 185
kelvin_phillips	5, 8, 80, 184
kenneth_mackenzie	32, 42, 74, 75, 121, 172, 194, 195
ken_brown	8, 9, 46, 48, 57, 67, 74
ken_hale	35, 40, 66, 89, 140, 160
ken_olsen	8, 10, 31, 49, 50, 85, 121, 164, 192
ken_smith	16, 27, 49, 50, 96, 103, 108, 118, 123, 173
kimberly_allen	6, 9, 48, 84, 98, 109, 117, 143, 163, 166
kris_jensen	8, 49, 98, 101, 112, 120, 131, 132, 155, 159
laura_clark	16, 19, 109, 125, 128, 146
laura_poe	5, 14, 67, 80, 162, 169, 190
laura_williams	8, 11, 102, 173
laurel_clark	20, 22, 32, 42, 52, 115, 120, 130, 174, 176
louise_roberts	4, 5, 18, 25, 69, 103, 114, 118, 150, 152, 161, 173
luigi_sartor	43, 92, 129, 154, 167, 180
luke_barrington	1, 10, 12, 19, 25, 45, 74, 82, 124, 131, 137, 143, 161, 163, 164, 194
lynn_robinson	3, 13, 46, 107, 109, 119, 126, 161, 192
mae_carter	2, 29, 76, 93, 110, 115, 117, 140
margaret_norman	28, 58, 72, 93, 106, 117, 161
marie_howard	14, 23, 34, 71, 99, 100, 154, 177, 178
marie_miller	12, 13
marion_king	1, 2, 5, 8, 10, 12, 22, 110, 127, 159, 172, 175, 184
mark_hutton	26, 69, 78, 79, 117, 118, 120, 126, 135
mark_oakley	21, 28, 29, 64, 74, 110, 130, 134, 147, 148, 149, 151, 154, 155, 174, 176, 181, 189
mark_spencer	4, 30, 40, 156
mark_towery	56, 88, 90, 115, 116, 140, 144, 166, 168, 175, 187
martin_jansche	13, 42, 79, 165, 188

richard_schroeder	6, 8, 28, 65, 67, 71, 90
richard_thomas	11, 23, 106, 116, 162, 167, 175
rick_woods	2, 5, 29, 88, 91, 128, 134, 161, 164, 189, 192
roberta_berry	32, 47, 55, 114, 124, 153, 172
roberto_alanis	16, 19, 33, 40, 52, 53, 64, 86, 94, 103, 161, 162, 174, 180
robert_cohn	2, 3, 37, 60, 77, 132, 149
robert_moran	6, 23, 64, 155, 157, 166, 198
robert_rosen	15, 19, 31, 58, 118, 152, 159, 190, 193
robert_scott	51, 65, 67, 73, 97, 160
robert_worthington	2, 5, 34, 47, 67, 80, 90, 91, 100, 135, 163, 188
roderick_murphy	18, 40, 84, 88, 129, 142, 143, 170
ronald_rice	13, 15, 21, 23, 77, 90, 102, 119, 143, 147, 165, 186, 190
ron_ellis	5, 10, 12, 28, 99, 140, 168, 182
russell_webb	8, 10, 35, 59, 74, 76, 106, 133, 152, 172, 181
ryan_maurer	2, 31, 80, 126, 167, 183, 190, 197
sandeep_pandey	95, 137, 140, 146
sandra_jones	10, 12, 102, 106, 145, 154, 185
sandy_steele	1, 15, 17, 48, 89, 95, 129, 191
sarah_ragland	5, 36, 37, 39, 43, 53, 120, 196, 198
scott_hedrick	11, 35, 54, 74, 76, 124, 139, 160
sean_mcnally	1, 2, 8, 38, 47, 89, 91, 109, 114, 116, 122, 138, 139, 140, 150, 153, 166, 168, 169, 170, 171, 172, 174, 183, 184, 186, 189, 190, 191, 192
seong-bae_park	20, 23, 36, 41, 66, 70, 120, 136, 179, 194
shelley_kramer	14, 48, 80, 109, 122, 169, 177, 179, 188
sheridan_glen	12, 27, 48, 57, 105, 114, 178
srihari_reddy	34, 49, 50, 62, 103, 106, 112, 129, 137, 164, 187, 198
stan_wilmoth	13, 24, 42, 50, 61, 65, 66, 76, 85, 86, 98, 116, 121, 138, 154, 155, 173
steven_nissen	12, 147, 151, 158, 162, 172, 174
tao_qin	5, 28, 30, 32, 45, 150
ted_mitchell	9, 12, 24, 36, 44
terry_hall	2, 55, 57, 71, 72, 123, 128, 129, 154, 158, 179, 184
thomas_ellsworth	14, 34, 39, 83, 102, 106, 109, 152, 172, 179, 188
thomas_werner	33, 34, 35, 37, 55, 66, 82, 92, 105, 112, 150, 152, 155, 156, 162, 176
tiffany_harris	5, 8, 52, 55, 57, 68, 75, 84, 118, 174
timothy_adams	13, 26, 33, 91, 102, 123, 146, 194
tony_rogers	20, 26, 106, 139, 174, 182, 189
tracy_raymond	2, 6, 16, 21, 89, 90, 115
troy_edwards	5, 13, 17, 26, 47, 50, 67, 100, 120, 157
vanessa_stewart	2, 5, 34, 119, 145, 184
virginia_smith	67, 99, 100, 125, 149, 152, 167, 195
wade_spencer	4, 13, 21, 22, 26, 90, 95, 115, 122, 133, 142, 187

wen-tau_yih	6, 11, 31, 37, 42, 105, 158, 169, 180, 182
wes_williams	6, 10, 32, 71, 77, 100, 129, 158, 169, 173, 174
william_donovan	54, 60, 65, 128, 131, 132, 144, 145, 181
william_rocha	74, 83, 102, 117, 127, 135, 164, 195
wolfgang_nejdl	87, 161, 182, 193
yolanda_jackson	3, 12, 15, 99, 102, 112, 158, 159, 180

Apéndice B

Listado de documentos perdidos por caracteres extraños

Folder name	Lost files by coding problems
alan_cox	90
alan_cox	171
alice_morris	159
alice_robinson	179
ashley_king	42
bin_wang	15
christopher_langford	142
dave_boggs	194
david_huffman	47
david_huffman	190
david_pettyjohn	20
dik_lun_lee	81
douglas_turnbull	89
erik_smith	155
helen_lockwood	193
ido_guy	71
janet_graham	98
john_breaux	182
joshua_keen	199
katherine_perez	26
katherine_perez	71
kathleen_horton	174
kenneth_mackenzie	146
luke_barrington	150
marie_howard	195
martin_jansche	164
maureen_johnson	39
megan_jackson	39
megan_jackson	40
nuria_oliver	157
philip_davey	101
qiang_wu	19
roberta_berry	89
sandy_steele	123
tiffany_harris	154
wen-tau_yih	123
yolanda_jackson	7
yolanda_jackson	54

Apéndice C

Listado de documentos perdidos en el procesador lingüístico

Folder name	Lost files in the linguistic process
charles_candy	168
charles_holton	178
charles_holton	180
charles_shaw	155
christopher_langford	120
christopher_langford	122
christopher_swan	121
chris_drew	32
chuck_peterson	187
clay_miller	39
hang_cui	134
harlan_price	78
harlan_price	79
harlan_price	81
harold_mills	77
harold_mills	80
harriet_brown	119
harriet_brown	120
helen_lockwood	143
hugh_herbert	85
hugh_herbert	86
ian_glover	107
martin_jansche	173
sarah_ragland	134
srihari_reddy	171
srihari_reddy	172
srihari_reddy	174
stan_wilmoth	144
thomas_werner	185
timothy_adams	40
timothy_adams	42

Apéndice D

Listado alfabético de etiquetas utilizadas por el procesador lingüístico

Number	Tag	Description
1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential there
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PRP	Personal pronoun

19.	PRP$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	to
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP$	Possessive wh-pronoun
36.	WRB	Wh-adverb