



TRABAJO FIN DE MÁSTER
MÁSTER UNIVERSITARIO EN INGENIERÍA INFORMÁTICA

Desarrollo de un prototipo de motor de búsqueda que incorpore técnicas bibliométricas para mejorar la recuperación

Autor

Aythami Estévez Olivas

Director

Juan Manuel Fernández Luna



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

Granada, 13 de septiembre de 2018

Desarrollo de un prototipo de motor de búsqueda que incorpore técnicas bibliométricas para mejorar la recuperación

Aythami Estévez Olivas

Palabras clave: Recuperación de información, Bibliometría, *Elasticsearch*

Resumen

Todos utilizamos motores de búsqueda en nuestra vida diaria y cada vez somos más dependientes de los mismos debido al exponencial incremento de información que se genera diariamente en los últimos tiempos. Esto hace cada vez más importante la mejora de los sistemas de búsqueda, para que sean capaces de priorizar y ayudarnos a encontrar la información que necesitamos.

En el mundo científico también ocurre esto, un investigador necesita la ayuda de algún sistema de recuperación de información para poder mantenerse al día en su rama de investigación. Por ello este proyecto propone un modelo alternativo de sistema de búsqueda, que tenga en cuenta medidas bibliométricas características de los artículos científicos, como el número de citas o el índice h, para mejorar la recuperación.

En concreto, el sistema planteado combinará la información bibliométrica con los resultados de búsquedas por contenido tradicionales de distintos modos. Este sistema se ha desarrollado como una aplicación web distribuida, que se servirá de una interfaz de usuario para realizar búsquedas por autores o artículos y comparar los resultados obtenidos con las distintas combinaciones.

Para desarrollar dicho sistema se ha empleado una metodología de desarrollo ágil que permite desarrollar de forma rápida e iterativa.

Project Title: Project Subtitle

First name, Family name (student)

Keywords: Keyword1, Keyword2, Keyword3,

Abstract

Write here the abstract in English.

Yo, **Aythami Estévez Olivas**, alumno de la titulación Master Universitario en Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 70918176E, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Máster en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Aythami Estévez Olivas

Granada a 13 de septiembre de 2018.

D. **Juan Manuel Fernández Luna**, Profesor del Área de Ciencias de la Computación e Inteligencia Artificial del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado *Desarrollo de un prototipo de motor de búsqueda que incorpore técnicas bibliométricas para mejorar la recuperación*, ha sido realizado bajo su supervisión por **Aythami Estévez Olivas**, y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 13 de septiembre de 2018.

El director:

Juan Manuel Fernández Luna

Índice general

1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	2
1.3. Organización de la memoria	2
2. Planificación	5
2.1. Planificación temporal	5
2.2. Gestión de recursos	6
2.2.1. Personal	6
2.2.2. Hardware	6
2.2.3. Software	6
2.3. Presupuesto	6
2.4. Metodología de desarrollo	7
3. Contexto	11
3.1. Recuperación de información	11
3.1.1. Relevancia y similitud	11
3.1.2. Las tres dimensiones de la RI	12
3.1.3. Componentes de un sistema de RI	13
3.1.4. Modelos	14
3.1.5. Contexto histórico	15
3.2. Bibliometría	16
3.2.1. Definición	16
3.2.2. Medidas	16
3.2.3. Limitaciones	18
3.3. Combinando ambas disciplinas	19
3.3.1. Sistemas de RI actuales con medidas bibliométricas .	19
3.3.2. Trabajos relacionados	20
4. Análisis	23
4.1. Enfoque planteado	23
4.2. Historias de usuario	24

5. Diseño	27
5.1. Fuentes de datos	27
5.2. Modelo de datos	28
5.2.1. <i>Author</i>	28
5.2.2. <i>Abstract</i>	29
5.3. Arquitectura inicial del sistema	30
6. Desarrollo	31
6.1. Herramientas software empleadas	31
6.2. <i>Sprint</i> 1	34
6.3. <i>Sprint</i> 2	35
6.4. <i>Sprint</i> 3	36
6.5. <i>Sprint</i> 4	37
6.6. <i>Sprint</i> 5	39
6.6.1. Obtención de datos	40
6.6.2. Preprocesado de los datos obtenidos	44
6.7. <i>Sprint</i> 6	45
6.7.1. Indexación de los datos	46
6.7.2. Desarrollo del buscador por contenido	47
6.8. <i>Sprint</i> 7	52
6.8.1. Aplicación de medidas bibliométricas y ordenación de resultados	53
6.9. <i>Sprint</i> 8	55
6.10. Arquitectura final del sistema	56
7. Conclusiones y Trabajos Futuros	59
7.1. Relación del TFM con lo aprendido en el Máster	59
7.2. Conclusiones	60
7.3. Trabajos futuros	61
Bibliografía	67
Anexos	68
A. Glosario	71
B. Lista de Acrónimos	73
C. Manual técnico de uso	75
C.1. Descripción de la arquitectura	75
C.2. <code>docker-compose.yml</code>	75
C.3. Instalación	77
C.3.1. Docker y Docker Compose	77
C.3.2. Descripción del repositorio	78
C.4. Despliegue	78

Índice de figuras

2.1. Ejemplo de tablero de un <i>sprint</i>	9
3.1. Componentes de un sistema RI [1]	13
3.2. Donut de Altmetric	18
5.1. Clase <i>Author</i>	28
5.2. Clase <i>Abstract</i>	29
5.3. Diagrama de arquitectura inicial del sistema	30
6.1. Fragmento de la lista priorizada creada	35
6.2. Ejemplo de resumen de uno de los papers	36
6.3. APIs de Scopus junto con sus límites	39
6.4. Aspecto tabla ranking UGRinvestiga	40
6.5. Distribución de nacionalidades entre los autores recuperados .	42
6.6. Fragmento de la distribución de ciudades entre los autores españoles recuperados	43
6.7. Representación del clustering de autores llevado a cabo en la limpieza de datos	43
6.8. Aspecto demo <i>Searchkit</i>	48
6.9. Aspecto buscador de autores	49
6.10. Aspecto vista de un autor	50
6.11. Aspecto buscador de artículos	51
6.12. Aspecto vista de un artículo	52
6.13. Diagrama de arquitectura final con las principales tecnologías usadas en cada nodo	57

Índice de cuadros

2.1. Planificación inicial de tareas	5
2.2. Especificaciones del equipo utilizado	6
2.3. Estimación de costes del proyecto	8
6.1. Campos del índice <i>author</i>	46
6.2. Campos del índice <i>abstract</i>	47
6.3. Resumen de las ordenaciones disponibles en cada módulo de búsqueda	55

Capítulo 1

Introducción

1.1. Motivación

A día de hoy usamos constantemente buscadores: web como Google o Bing, de ficheros como los integrados en todos los sistemas de ficheros modernos o de contenido como puede ser una búsqueda de algún término en un PDF como este. Todo buscador es conocido desde un punto de vista más técnico como un **Sistemas de Recuperación de Información (RI)**.

Tradicionalmente estos sistemas realizan una búsqueda por contenido, si buscamos una palabra o término de búsqueda concreto en Google, este retorna páginas relevantes que contentan dicha palabra.

Esto también funciona así para la recuperación de artículos científicos, pero en este caso los artículos científicos disponen de algunas medidas asociadas como el número de citas que pueden ser interesantes para determinar la relevancia de un artículo. Se puede entender que si un par de artículos contienen un término de búsqueda, él que sea más citado parece, a *priori*, más relevante, ya que la propia comunidad científica lo menciona con mayor frecuencia. Dichas métricas asociadas a la literatura científica se conocen como **medidas bibliométricas**.

Este proyecto pretende explorar las posibilidades de estas medidas para mejorar los procesos de recuperación de información.

Durante la asignatura del máster Gestión de Información en la Web (GIW), se vieron algunas pinceladas de las herramientas empleadas para llevar a cabo sistemas de Recuperación de Información, así como su base teórica. Esto me llamó realmente la atención, ya que todos utilizamos diariamente sistemas de búsqueda, pero no tenía ni idea de como se podía implementar uno. A pesar de haber desarrollado un pequeño sistema como parte de sus prácticas me quedé con la ganas de ver un sistema "más real",

desarrollado con herramientas más potentes. Este es el principal motivo por el que me decanté a la realización de este proyecto, a nivel personal me gustaría poder llenar esta curiosidad con el desarrollo del presente trabajo.

1.2. Objetivos

En este apartado recogeré de manera sintetizada los objetivos del proyecto, lo cual ayudará a comprender la funcionalidad del sistema a desarrollar así como definir su alcance. El objetivo principal es **desarrollar un Sistema de Recuperación de Información que incorpore medidas bibliométricas como mejora a la recuperación clásica**. Junto a este objetivo también tenemos

- **Descomponer el sistema de RI:** Observando los diversos sistemas reales de recuperación de información científica que he analizado, como Google Scholar o Scopus, la gran mayoría dividen la búsqueda en búsqueda de autores y de artículos en sí. Ya que estos son dos tipos de datos diferenciados (aunque relacionados profundamente) e intuitivamente, se comprende que un sistema de RI funcionará mejor si los datos del mismo son homogéneos. Por ello el sistema a desarrollar dispondrá de dos partes una búsqueda de autores y otra de artículos.
- **Desarrollar un sistema que sea usable:** Muchos de los enfoques que he visto durante mi proceso de documentación no pasan de modelos teóricos, prototipos o sistemas en los que la usabilidad y la orientación al usuario brillan por su ausencia. Aunque este no sea el punto objetivo principal del sistema, me parece muy importante que se tenga en cuenta al usuario durante todo el proceso de desarrollo, ya que un sistema puede ser increíble pero si los usuarios no lo entienden o no le saben sacar partido no sirve de nada. Por ello, pretendo diseñar una interfaz de usuario que sea simple de usar, empleando metáforas y componentes ampliamente conocidos por los usuarios.

1.3. Organización de la memoria

Esta memoria detallará el desarrollo del proyecto desde su comienzo hasta su conclusión y cuenta con los siguientes apartados:

- El siguiente capítulo versa sobre los aspectos de la **planificación** del proyecto, desde una perspectiva temporal, económica, de recursos y metodológica.

- Tras esto se definirá el **contexto** del trabajo, dando antecedentes, definiendo los principales conceptos teóricos y analizando el estado del arte.
- En el capítulo **análisis** se planteará el proyecto a desarrollar, ayudándose de historias de usuario para definir la funcionalidad del sistema creado.
- Este planteamiento se refinará y detallará en el **diseño**, centrándose en los datos y arquitectura del sistema.
- El grueso de la memoria está constituido por el **desarrollo**, donde se han apuntado los principales aspectos del mismo, siguiendo una estructura de *sprints*, así como la aclaración de la arquitectura del sistema final.
- Para finalizar el contenido principal, el apartado de **conclusiones y trabajos futuros** recoge algunas reflexiones personales y sobre los resultados del proyecto, incluyendo algunos posibles caminos futuros de ampliación del mismo.
- Tras esto se pueden encontrar la **bibliografía**, así como los anexos **glosario**, **lista de acrónimos** y un **manual técnico de uso** donde se detalla la arquitectura final con instrucciones de instalación y despliegue.

Capítulo 2

Planificación

2.1. Planificación temporal

Desde la asignación del TFM, en diciembre de 2017, me percaté que iba a ser realmente complicado alcanzar la primera convocatoria con la carga de trabajo que suponía el máster. Por lo que decidí tomarlo con calma y llegar a septiembre, pero tras finalizar el curso comencé la realización de mis prácticas, lo cual unido a lo extenuado que había acabado el año me impidió alcanzar el objetivo.

En Octubre de 2017 comencé a trabajar, lo que supuso muchas horas menos al día libres y me llevó un tiempo adaptarme, por lo que no fue hasta Febrero de 2018 cuando me lo empecé a tomar en serio. Teniendo en cuenta mi limitada disponibilidad horaria, que apenas me permitía dedicarle 1-2 horas entre diario, esboqué una planificación con el objeto de entregar el proyecto en Julio de 2018, dicha planificación inicial se recoge en la siguiente tabla.

Tarea	Inicio	Fin	Duración
Investigación	19/02/2018	23/04/2018	8 semanas
Obtención de datos	23/04/2018	07/05/2018	2 semanas
Procesado de datos	07/05/2018	21/05/2018	2 semanas
Búsqueda básica	21/05/2018	04/06/2018	2 semanas
Búsqueda con bibliometría	04/06/2018	02/07/2018	4 semanas
Refinamiento	02/07/2018	09/07/2018	1 semana

Cuadro 2.1: Planificación inicial de tareas

Desgraciadamente fui incapaz de alcanzar el objetivo de Julio, algunas vacaciones y asuntos personales me hicieron replanificar para llegar a esta convocatoria de Septiembre.

2.2. Gestión de recursos

En este apartado describiré brevemente los recursos utilizados para llevar a cabo este proyecto teniendo en cuenta personal, hardware y software.

2.2.1. Personal

El único recurso humano que ha contribuido a la realización del proyecto soy yo mismo actuando como cada uno de los diversos roles que llevan acabo el proceso de desarrollo de un proyecto de estas características.

2.2.2. Hardware

Respecto al hardware utilizado para este TFM solo he requerido mi ordenador portátil personal, cuyas características se destacan en la siguiente tabla:

CPU	Intel ®Core™i7-4700MQ CPU @ 2.40GHz x 8
RAM	8 GB RAM DDR3
Almacenamiento	HDD 750 GB (5400 RPM)

Cuadro 2.2: Especificaciones del equipo utilizado

Esto ha bastado para el desarrollo, pero sería necesario conseguir un servidor para llegar a poner en producción el sistema. Tampoco sería necesario nada muy potente ya que incluso en mi propio ordenador los tiempos empleados son bastante aceptables.

2.2.3. Software

Se han empleado utilidades de software libre en la totalidad del proyecto. Para una enumeración de las mismas y su función primordial ver el apartado Herramientas software empleadas.

2.3. Presupuesto

En este apartado haré una estimación a *grosso* modo de los costes de llevar a cabo el proyecto si el objetivo fuera desarrollar un producto listo para explotación (no olvidemos que esto es un prototipo).

A pesar de la dilatada planificación temporal, si se fuera a implementar este sistema la duración se reduciría drásticamente, al contar con una persona trabajando a tiempo completo. Estimo que en 2 meses a tiempo

completo un desarrollador de mis características podría finalizar el proyecto holgadamente. Para ponerle un costo a esto estimaré un salario de 15€/hora brutos, lo que a jornada completa serían 2400 € brutos al mes, a mi juicio, un salario digno para un ingeniero con máster, desgraciadamente nada real.

Respecto a los costes de hardware sería necesario un equipo para el desarrollo algo más potente que el utilizado, ya que en momentos puntuales la memoria de mi equipo se ha visto desbordada. Por ello lo indicado sería un portátil con 16 GB de RAM y almacenamiento SSD de 256GB que puede rondar los 1000€ ahora mismo. Teniendo en cuenta que el periodo de amortización de un equipo de estas características ronda los 3 años y que la duración del proyecto sería 2 meses eso dejaría un coste de 55,55 €

Además sería necesario un servidor donde desplegar el sistema para su explotación. Hoy en día son pocos los proyectos que puedan requerir la compra de un servidor físico teniendo en cuenta las posibilidades que ofrece la computación en la nube y la variada oferta de la que se dispone. Por ello se contrataría una instancia en *Amazon Web Services* de tipo *i3.xlarge* con 4 núcleos, 30 GB de RAM y 1 TB de SSD cuyos costes estimados mensuales son de 220 €.

Respecto al software a utilizar se apostaría por el software libre con licencias que permitan la comercialización como Apache2 o MIT, por lo que el coste estimado en Software es de 0€.

Si el proyecto se desarrollara en el ámbito académico, para ayudar a la labor investigadora de una universidad, seguramente no habría que considerar costes para usar datos de artículos ya que alta probabilidad la universidad en cuestión contará con licencias o acuerdos con editoriales científicas como Elsevier. En caso de querer desarrollar este sistema para su explotación económica los gastos editoriales podrían suponer un importante sobre coste que hiciera peligrar la viabilidad del proyecto.

Para esta estimación supondré que se desarrolla el proyecto en un ambiente académico obviando los gastos mencionados previamente. En la siguiente tabla se recogen los gastos requeridos para el desarrollo de este sistema (2 meses de proyecto), a esto habría que sumar 220€ por cada mes que se mantenga el sistema desplegado en concepto de costes de infraestructura.

2.4. Metodología de desarrollo

Para desarrollar este proyecto se ha empleado una metodología ágil similar a *SCRUM* 1 algo más relajada. Es una modelo particular ya que yo mismo soy el desarrollador, el coordinador (rol del *Scrum máster*) y la persona encargada de definir las tareas y evaluar el cumplimiento de los mismas (el *Product owner*).

Elemento	Coste
Recursos humanos	4800€
Hardware para el desarrollo	55,55€
Servidor	440 €
Software	0 €
Gastos editoriales	0 €
TOTAL	5295,55 €

Cuadro 2.3: Estimación de costes del proyecto

Me he decantado por este modelo ya que permite más flexibilidad al enfrentarse a problemas en entornos desconocidos, como es este proyecto. También ha influido mi experiencia personal ya que con esta metodología he llevado a cabo tanto mi TFG como mis proyectos en el ámbito laboral.

Se basa en la descomposición del proyecto completo en pequeños subproyectos o *sprints*, en los que define de manera acotada las tareas y objetivo del mismo. Permitiendo el refinamiento iterativo del producto final así como el aprovechamiento del conocimiento obtenido a lo largo de los *sprint* para mejorar los venideros, al contrario que otros modelos de desarrollo más clásicos que resultan más estáticos y rígidos.

Con el objetivo de almacenar y versionar todo el material producido en este proyecto he utilizado la plataforma *GitHub* y el sistema de control de versiones *git*. En concreto he creado el repositorio <https://github.com/AythaE/TFM>.

Para seguir el progreso de cada uno de los *sprints* del proyecto he utilizado los tableros que ofrece el propio *GitHub projects* donde cada una de sus tareas o tarjetas corresponden con *issues* como se puede ver en la siguiente imagen.

Además de esto he utilizado un archivo *Markdown* a modo de diario donde ir apuntando cosas interesantes según iban surgiendo, el estado actual de desarrollo o algunas tareas para completar próximamente, dicho fichero se llama *Diario.md*.

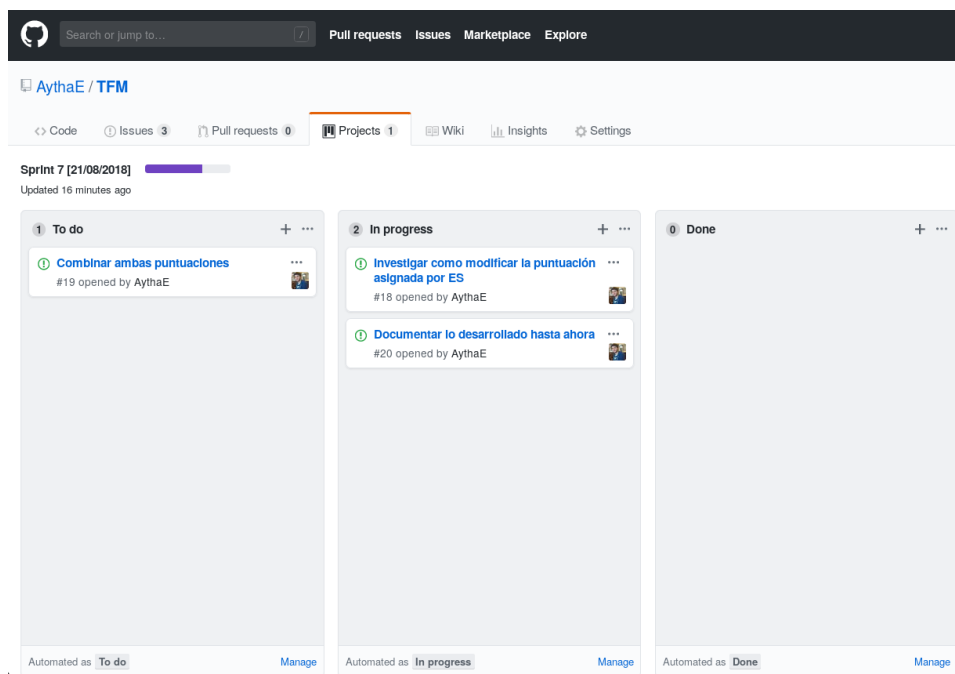


Figura 2.1: Ejemplo de tablero de un *sprint*

Capítulo 3

Contexto

3.1. Recuperación de información

La **Recuperación de Información (RI)** es una disciplina que trata de modelar, diseñar e implementar sistemas capaces de promocionar acceso basado en contenidos. [2]

En general un sistema de RI recibe una petición o consulta del usuario y debe devolver de entre su conjunto de información las unidades relacionadas con la consulta.

Estas unidades pueden representar cualquier tipo de elemento: ficheros de texto, imágenes, archivos de audio, etc. De forma genérica se denominan **documentos** así como al conjunto de información se le denomina **colección**.

3.1.1. Relevancia y similitud

La **relevancia** hace referencia a la relación entre la consulta de un usuario y los documentos recuperados. Por ello intuitivamente se entiende que un documento es relevante para una consulta concreta si contribuye a satisfacer la necesidad de información expresada por la misma. Aunque el concepto pueda parecer claro, la relevancia no es para nada absoluta, es una media subjetiva que depende de varios factores como quien la valore o como se haya planteado la consulta inicial.

Respecto a la **similitud** esta es una medida de semejanza entre documentos o entre documentos y consultas. Otra vez más nos encontramos ante una medida relativa y que se puede medir de diversas maneras: comparación de cadenas de texto, uso de un mismo vocabulario, que dos documentos pertenezcan al mismo autor, que dos documentos tengan múltiples referencias

comunes...[1]

3.1.2. Las tres dimensiones de la RI

Se puede decir que las tres dimensiones principales de la RI son:[2]

Acceso a la información

Como puede acceder un usuario a los datos. Existen diversos paradigmas de búsqueda:

- **Clasificación:** cada documento pertenece a clases y estas se pueden usar como jerarquías.
- **Agrupamiento:** documentos se agrupan en conjuntos.
- **Filtrado:** se selecciona un subconjunto de documentos.
- **Recomendación:** los documentos se presentan al usuario basados en su interacción previa con el sistema.
- **Resumen:** fragmentos de documentos utilizados para reducir la información presentada al usuario, muy típico de motores de búsqueda web.

Tipos de información

Hoy en día vivimos en una marea de información muy heterogénea y creciente lo que hace complicado definir los distintos tipos, por citar los principales actualmente:

- **Documentos textuales** como paginas web o PDF.
- **Partes de un documento**, capítulos o secciones de este.
- **Búsqueda de información multimedia:** como canciones o vídeos a partir de propiedades perceptibles o incluso de otros elementos multimedia, la búsqueda de imágenes en Google imágenes por ejemplo.
- **Búsqueda de e-mails:** implementado en cualquier cliente de correo web o nativo.
- **Búsqueda geográfica:** por nombre del lugar, sitios cercanos...

Colección

Guarda relación con los documentos que pueden ser buscados y se pueden clasificar tres tipos en función del tamaño:

- **Personal:** ficheros del dispositivo del usuario.
- **Corporativa:** documentos de una empresa, algo más compleja, supone búsqueda en múltiples ubicaciones conectadas en red.
- **Web:** cualquier documento web, el volumen de datos y la infraestructura es descomunal.

3.1.3. Componentes de un sistema de RI

De manera general un sistema de RI se compone de los elementos que se observan en la siguiente figura. Ha de contar una interfaz de usuario encargada de recoger las peticiones y mostrar los resultados. Será necesario interpretar estas consultas para convertirlas en términos que el sistema pueda entender.

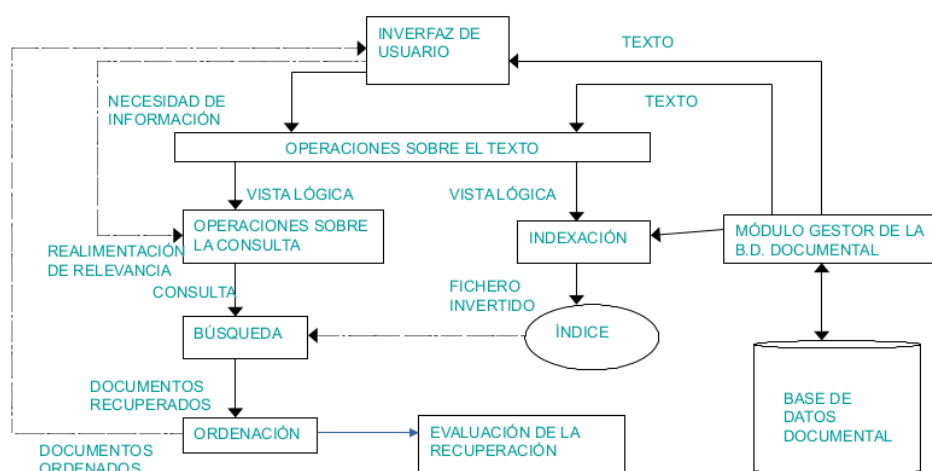


Figura 3.1: Componentes de un sistema RI [1]

Una vez hecho esto el algoritmo de búsqueda encontrará los documentos que sean relevantes para la consulta, tras esto será necesario puntuar estos documentos recuperados y devolver esta lista puntuada al usuario.

Para realizar este proceso de búsqueda y priorización se servirá del **índice**, una o varias estructuras de datos que permiten optimizar la búsqueda sobre una colección. En su forma más básica se trata de una estructura que relaciona términos con los documentos en los que aparecen, de modo que

si una consulta contiene esos términos se devuelven los documentos que los contienen.

3.1.4. Modelos

Se puede definir un modelo de RI como una especificación de la forma de representar documentos, consultas y realizar comparaciones entre ambos [1]. El objetivo final es calcular una puntuación para cada documento dada una consulta específica que determine el grado de relevancia de este, utilizando esa medida se puede llevar a cabo una ordenación o ranking de los documentos.

Los modelos clásicos a pesar de ser los más básicos sirven como base para crear otros más complejos como alguno de los que se hablará posteriormente. Estos modelos clásicos son:[2]

El **Modelo booleano** basado en teoría de conjuntos y lógica booleana. Se define el conjunto V como todas las palabras clave de la colección, $V = \{t_1, t_2, \dots, t_M\}$, así mismo se define el conjunto D como el conjunto de todos los documentos de la colección $D = \{d_1, d_2, \dots, d_n\}$.

Cada documento d_i se representa por tanto como un conjunto de términos que aparecen en él, este es un subconjunto de V . Las consultas en este modelo se representan mediante las operaciones booleanas típicas AND, OR y NOT.

El **Modelo vectorial** modela los documentos y consultas como vectores de términos en un espacio vectorial de dimensión definida por el número de términos de la colección.

Cada documento es por tanto un vector en dicho espacio vectorial. Usando los conjuntos descritos en el modelo previo se puede definir un documento d_i como el vector de términos $\vec{d}_i = (w_{1,i}, w_{2,i}, \dots, w_{M,i})$ donde $w_{i,j}$ representa el peso del término i en el documento j .

Queda por determinar el esquema de pesos y la función de similitud entre vectores.

El **Modelo probabilístico** pretende expresar la relevancia de los documentos utilizando la teoría de probabilidades, por tanto se define $P(Rel|d, q)$ como la probabilidad de que dado un documento d y una consulta q el documento sea relevante con cierta probabilidad.

Simplificando la notación a $P(Rel|d)$ y usando el Teorema de Bayes² podemos transformar el cálculo de esa probabilidad en:

$$P(Rel|d) = \frac{P(d|Rel)P(Rel)}{P(d)} \quad (3.1)$$

La función de similitud en el modelo probabilístico se expresa como:

$$\text{sim}(q, d) = \frac{P(\text{Rel}|d)}{P(\overline{\text{Rel}}|d)} \quad (3.2)$$

Es decir el ratio entre la probabilidad de que un documento sea relevante y no para q .

Aplicando la transformación anterior esta función queda como:

$$\text{sim}(q, d) = \frac{P(d|\text{Rel})}{P(d|\overline{\text{Rel}})} \frac{P(\text{Rel})}{P(\overline{\text{Rel}})} \approx \frac{P(d|\text{Rel})}{P(d|\overline{\text{Rel}})} \quad (3.3)$$

Donde $\frac{P(\text{Rel})}{P(\overline{\text{Rel}})}$ se conoce como característica independiente de consulta al suponer la relación entre las probabilidades de escoger un documento al azar y que este sea relevante o irrelevante, se suele eliminar como simplificación.

Por otro lado $\frac{P(d|\text{Rel})}{P(d|\overline{\text{Rel}})}$ supone la relación entre la probabilidad sabiendo que un documento es relevante este sea d y su inversa. Estas probabilidades resultan más fáciles de calcular y son las utilizadas en estos modelos.

3.1.5. Contexto histórico

La historia de la RI comienza antes de la era digital. Como ejemplo las tablas de contenidos e índices de un libro forman un sistema de RI a pequeña escala, los índices relacionan términos de indexación con su ubicación en el documento de forma similar a un glosario de términos, ver 3 .

Esto es lo que se conoce como búsqueda **pre-coordinada** donde los términos de búsqueda o consultas están definidas de antemano, esto hace que se pueda organizar muy bien la información (como se hace en una biblioteca por ejemplo), pero requiere que el usuario conozca estos términos y no supone un método muy escalable teniendo en cuenta el volumen de información manejado en sistemas actuales. Por ello surgió la **post-coordinación** en la década de 1950 que se basa en definir las consultas en el momento de la búsqueda, dándole libertad al usuario.

A este último enfoque pertenecen la mayoría de los sistemas actuales, los cuales recibieron un enorme impulso con el desarrollo de la web iniciado en 1989 por Tim Berners-Lee en el CERN. Los primeros buscadores web de la forma que los conocemos hoy surgieron entorno a 1994, sistemas como Lycos o Altavista.

La búsqueda de documentos en la web siempre ha resultado un reto debido a su naturaleza heterogénea, la propuesta de unos jóvenes estudiantes

de la universidad de Stanford en 1998 supuso una revolución y asentó lo que hoy conocemos como buscador. Esa propuesta fue el algoritmo *PageRank*[3] y esos chicos eran Larry Page, Sergey Brin, Rajeev Motwani y Terry Winograd; los dos primeros fundaron poco después una empresa llamada Google, que a día de hoy es el buscador más utilizado. [4]

Hoy en día existe una dependencia casi absoluta por los motores de búsqueda para navegar por internet, lo que hace de este un tema tan crucial en el que aún se sigue trabajando, con retos aún por delante como el enorme tamaño que ha alcanzado la web, la heterogeneidad de contenido con cada vez más contenidos multimedia o el acceso desde cualquier tipo de dispositivo y desde cualquier lugar.

3.2. Bibliometría

3.2.1. Definición

La **bibliometría** se puede definir como el análisis estadístico de publicaciones escritas. Sus métodos se suelen utilizar para ofrecer un análisis cuantitativo de la literatura académica [5]. Se relaciona mucho con la **cienciometría** que se puede entender como el estudio cuantitativo de la ciencia de forma general [6].

Específicamente para este trabajo resulta destacable el enfoque de poder medir la importancia de trabajos científicos de forma cuantitativa y como esto se puede utilizar para mejorar los resultados de un sistema RI. Yendo al uso más particular que se llevara a cabo durante el desarrollo de este trabajo ambas disciplinas proponen una serie de medidas particulares, algunas de las cuales comentaré en el siguiente apartado.

3.2.2. Medidas

Número de citas

Esta es una de las métricas más directas y sencillas, se basa la premisa de que un documento científico es más relevante si cuenta con mayor número de citas. Actualmente se encuentra disponible en casi cualquier plataforma bibliográfica como Google Scholar, Scopus o Web of Science.

Sobre esta métrica básica se han construido muchas posteriormente como el impacto de citas (media de citas por documento), el impacto de citas ponderado por el campo (la anterior pero ponderado por materia, tipo de publicación y año), *Highly cited papers* (documentos en el top 1% de citas ponderado por campo y año)[7], *CiteScore* (media de citas anuales para

todos los documentos de una revista concreta durante los últimos 3 años), *SCImago Journal Rank* (medida que pondera el número de citas de una revista científica con el prestigio de las que la citan) o *SNIP* (normalización del número de citas de un paper por su impacto en la materia) [8].

Análisis de co-citación

Medida de similaridad que establece que dos documentos serán semejantes si aparecen citados conjuntamente con frecuencia por otros documentos. Sobre esta premisa se puede considerar un índice de co-citación que es simplemente el número de citas co-citaciones de 2 documentos.

También existen métricas más complejas como el análisis de co-citación por proximidad que incorpora a la idea previa el hecho de si dos citas aparecen en la misma sección del texto estas estarán más relacionadas que si una aparece en la introducción y otra en las conclusiones por ejemplo. [9]

Índice h

Este índice se puede aplicar a un autor científico, el índice h de un autor es x si este tiene x artículos con al menos x citas [7]. Se utiliza ampliamente para medir la productividad de un investigador.

Tiene diversas variantes como el índice g (índice h para el número de citas medio), índice m (corrección del índice h con el tiempo) o el índice Py (número medio de publicaciones por año).

Acoplamiento bibliográfico

Muy relacionado con el análisis de co-citación, el acoplamiento bibliográfico o *bibliographic coupling* basa su concepto de similaridad en que 2 documentos serán semejantes si comparen citas.

En base a esta idea se puede crear una métrica simple que sea el número de referencias comunes entre dos documentos, o medidas más complejas como el acoplamiento bibliográfico entre autores que supone el acoplamiento entre el conjunto de citas de todos los trabajos de un autor con otro [10].

Altmetrics

Conjunto de medidas relativas al mundo *online*, veces que un documento ha sido descargado, compartido, mencionado en las redes sociales, blogs, wikipedia... [11]

Existen diversas compañías que ofrecen productos sobre estas métricas siendo una de las principales Altmetric. Entre sus productos destaca el "donut" que se puede apreciar en la siguiente imagen. Este corresponde con una medalla que puede colocar en la página de un artículo y permite de un vistazo la atención que está generando este trabajo. También incluye información en detalle de donde se está hablando del mismo incluyendo comentarios en redes sociales.

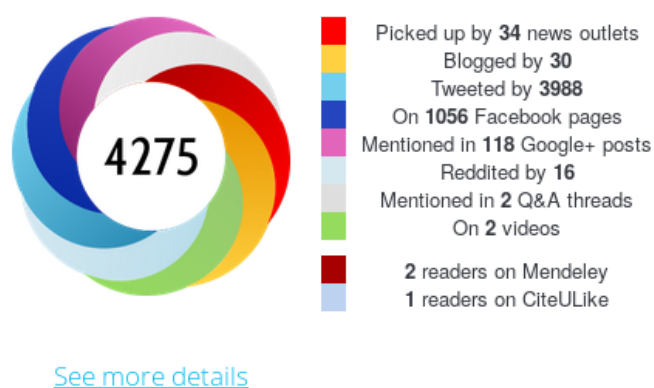


Figura 3.2: Donut de Altmetric

3.2.3. Limitaciones

Estas medidas ayudan a dar una idea de la importancia de un documento o de la productividad de un autor, pero no deben de ser tomadas como únicos criterios ya que presentan sus limitaciones.

Respecto a las medidas basadas en número de citas son muy dependientes de las fuentes de las que tomen datos así como su cobertura, por otro lado que un artículo sea muy citado no quiere decir que este sea muy influyente, se puede citar un trabajo para criticarlo. También hay que tener en cuenta las «auto citas» citas de un autor a otros trabajos suyos que pueden distorsionar estas medidas [7].

No se puede comparar categóricamente dos trabajos o autores de cualquier disciplina usándolas, ya que en distintos campos se cita de manera distinta.

En las medidas que dependan de citaciones (prácticamente todas) hay que considerar un factor temporal, para que un trabajo adquiera relevancia es necesario cierto tiempo, por ello se recomienda dejar fuera las publicaciones recientes (de los últimos tres años)[7].

3.3. Combinando ambas disciplinas

En los últimos años han surgido diversos trabajos que proponen combinar ambas disciplinas, utilizar la bibliometría para mejorar la recuperación de información. Esto no se puede aplicar de manera genérica y directa a cualquier sistema de RI ya que la bibliometría está muy centrada en el mundo de la académico y de la investigación, sin embargo con algunas adaptaciones las ideas de la bibliometría se encuentran muy presentes en los sistemas de RI, por ejemplo el algoritmo PageRank hace una analogía entre páginas web con artículos científicos así como hiperenlaces con citas en los mismos [3].

3.3.1. Sistemas de RI actuales con medidas bibliométricas

Donde resulta obvio que esta combinación puede ser beneficiosa es los sistemas RI especializados en literatura académica siendo los más importantes:

- Google Scholar: buscador gratuito especializado en literatura científica, contiene algunas medidas bibliométricas con el número de citas o el índice-h de un autor. Sin embargo ha sido muy criticado por intentar incluir el mayor número de artículos posibles sin importar la calidad de los mismos [12] o dar demasiada importancia al número de citas lo que hace que sea complicado descubrir nuevos trabajos que no son muy conocidos [13].
- Web of Science (WoS): Sistema de RI por suscripción más utilizado, habitualmente las instituciones académicas lo tienen contratado, como es el caso de la Universidad de Granada a través de la Fundación Española para la Ciencia y la Tecnología (FECYT). Agrupa múltiples bases de datos de diversas índoles llegando a tener más de 100 millones de documentos [14]. Fue uno de los primeros sistemas en aparecer y actualmente pertenece a Clarivate Analytics. A diferencia de Google Scholar los contenidos son revisados por expertos antes de ser incluidos [14].
- Scopus: Similar a WoS, en este caso pertenece a la editorial Elsevier. También se puede usar desde la UGR gracias a la FECYT. Este sistema es más reciente, cuenta con unos 70 millones de documentos [15] y también tiene un proceso de revisión de contenido. Dispone de algunas otras medidas de bibliometría además del número de citas o índice-h, como los otros sistemas descritos, cuenta con las medidas *CiteScore*, *SCImago Journal Rank* y *SNIP* (ver 3.2.2 para más información).

3.3.2. Trabajos relacionados

Desde un punto de vista académico resultan especialmente interesantes los trabajos de los talleres *Bibliometric-enhanced Information Retrieval* (BIR). Estos congresos se celebran anualmente desde 2014 y tienen como objetivo "hacer consciente a la comunidad de RI de sus posibles vínculos con la bibliometría" [16]. Voy a destacar algunos de ellos a continuación.

Ante el creciente volumen de trabajos científicos se hace complicado para investigador mantenerse al día en su propio campo porque resulta imposible leer todos los trabajos, por ello se suele utilizar algún sistema de recomendación [4]. Pero los sistemas existentes no contemplan bien el uso de medidas bibliométricas para distinguir los trabajos relevantes, por ello en [17] se propone reordenar la salida del sistema de recomendación Mr.DLib utilizando como criterio varias medidas derivadas del número de lectores de un artículo, un tipo de *altmetric* (ver 3.2.2). Las conclusiones de este trabajo demuestran que se encuentra una mejora utilizando las medidas de cienciometría solas o en combinación con el ordenamiento normal basado en texto que usando solo el ordenamiento del sistema RI, siendo la métrica que mejor resultado obtuvo el número absoluto de lecturas sin normalización. Esta mejora no es muy significativa como cabría esperar, los autores achacan esto a la falta de cobertura de datos bibliométricos en la colección.

Otro enfoque interesante es el de [18], donde se propone utilizar medidas bibliométricas como variables independientes de consulta o *static features*, ver el modelo probabilístico 3.1.4. Estas características se conocen como priores en el modelo probabilístico de lenguaje utilizado y modifican la probabilidad de que un documento sea relevante para una consulta dada, multiplicando dicha probabilidad por un factor. Para realizar su estudio han utilizado la colección bibliográfica de prueba iSearch, que desgraciadamente parece no estar disponible ya. De esta colección seleccionaron el subconjunto de artículos con al menos alguna cita dentro de la colección (para poder llevar a cabo un análisis de co-citación).

Como *static features* específicas han seleccionado la proporción de citas dentro del subconjunto de documentos entre el número total de estas, el *PageRank* calculado en el subconjunto y el clustering de co-citación ¹, siendo todas las variantes también suavizadas con una versión logarítmica. Aunque el concepto es interesante y el método exhaustivo los resultados obtenidos no son significativos, se achaca esto al bajo tamaño de la colección de prueba, con 863 documentos.

Siguiendo un modelo vectorial pero aplicando una variación del conocido

¹Creando un grafo no dirigido con peso de citas donde los nodos son documentos y las aristas citas, sobre el que se aplica un algoritmo de clustering y se calculan los priores del cluster mediante validación cruzada de 5 iteraciones.

esquema de pesos $tf*idf$ se encuentra [19]. En el esquema de ponderación para los términos original tf representa la frecuencia de un término en el documento e idf el número de documentos donde aparece el término a la inversa. Con ello se consigue dar mayor importancia a los términos que aparezcan menor número de veces ya que serán más discriminantes e importantes para una búsqueda.

El modelo propuesto en este paper apuesta por buscar documentos en función de otros, es decir los términos de consulta son otros documentos de la colección, los cuales se conocen como semillas. Para la ponderación de peso en cada documento ahora tf es el número de documentos co-citados con el documento semilla e idf la inversa del número total de citas entre documentos de la colección. Por último cita algunos ejemplos sobre colecciones de prueba y discute el modelo planteado sin llegar a evaluarlo realmente. De este, se dice que sería especialmente útil para usuarios con conocimiento previo del dominio, por ejemplo para llevar acabo una investigación bibliográfica sobre algún autor.

Basado parcialmente en el trabajo previo, el siguiente artículo [20] propone la creación de un *framework* 5 para llevar a cabo una investigación bibliográfica siguiendo un enfoque híbrido combinando información textual y de citación. A partir de la selección de algunos documentos semilla, el sistema crea el espacio de citaciones ², sobre el cual se filtra poniendo condiciones, el resultado de este filtrado se refina buscando términos y frases comunes con las semillas en el resumen de los documentos.

Utilizan su sistema para comparar con diversos trabajos de revisión bibliográfica realizados manualmente con el objetivo de lograr obtener el mismo conjunto final de documentos relevantes revisando menos trabajos con lo que se ahorraría un tiempo significativo. Sus resultados resultan prometedores, usando todas las combinaciones de 3 documentos semilla entre los seleccionados finalmente por cada revisión logran recuperar todos los documentos finales disponibles en Scopus (plataforma en la que realizan el estudio) reduciendo el número de documentos totales recuperados para esa revisión en hasta el 80% dependiendo de la revisión concreta.

²Documentos citados por las semillas, aquellos que las citan a ellas y documentos con relación de co-citación con estas.

Capítulo 4

Análisis

4.1. Enfoque planteado

Como se ha podido comprobar en el apartado previo existen numerosas propuestas para combinar la bibliometría y la RI, pero estos trabajos se realizan en un ámbito más académico y muchas veces no llegan a materializarse en sistemas reales. Los principales sistemas de RI para la recuperación de literatura académica integran algunas medidas básicas pero no llegan a implementar los modelos más complejos planteados desde la investigación a pesar de que sus resultados sean esperanzadores.

Esto hace que este campo siga estando en desarrollo y resulte interesante plantear nuevos modelos a la par que testarlos.

Por ello este TFM servirá para desarrollar un prototipo de modelo que combine ambas ramas de la teoría de la información y sirva para evaluar el rendimiento de estos planteamientos. El objetivo es comparar un sistema de RI clásico con uno que incorpore técnicas bibliométricas intentando medir su viabilidad y potencial mejora en los resultados recuperados.

En los últimos tiempos el acceso a artículos se ha incrementado exponencialmente gracias a algunas de las plataformas descritas previamente. A su vez, la información bibliográfica que incorporan también ha aumentado sustancialmente lo que hace cada vez más plausible la implementación de este tipo de sistemas. La irrupción de las *altmetrics* resulta destacable. A pesar de ser unas medidas bibliométricas bastante recientes, su capacidad para determinar la popularidad de los trabajos científicos es muy significativa.

Es especialmente interesante el enfoque de las técnica híbridas, como las planteadas en el último trabajo analizado, que permiten por un lado no divergir del modelo mental de sistema de RI del usuario, un buscador textual, pero incluir las potenciales ventajas de usar otro tipo de medidas.

El sistema propuesto utilizará como base un modelo clásico realizando una reordenación *a priori* de los resultados en función de algunas medidas directas como el número de citas o el número de lectores en una plataforma (incluyendo con ello alguna *altmetric*). Se plantea utilizar una realimentación inconsciente del usuario basada en considerar como relevantes los documentos del listado inicial que el usuario descargue tras leer el resumen y utilizándolos como semillas para refinar la búsqueda de manera transparente para él. Esta reordenación *a posteriori* se servirá del grafo de citación de estos documentos semilla para seleccionar los documentos que tengan una relación fuerte.

4.2. Historias de usuario

Siguiendo los enfoques de las metodologías ágiles como la empleada, en lugar de definir los requerimientos y alcance del software mediante una especificación de requisitos de usuario o casos de uso, me he decantado por utilizar historias de usuario.

El usuario que se mencionará en estas historias así como el del sistema final será un usuario relacionado con el mundo científico y que esté familiarizado con los conceptos típicos como las medidas bibliométricas empleadas.

En la siguiente lista se recogen las historias de usuario con las que se ha trabajado

- **Seleccionar el método de ordenación *a priori* de los resultados de búsqueda:** El usuario tendrá que poder elegir entre varios criterios de ordenación de resultados basados en medidas bibliométricas directas. Los criterios concretos dependen de la disponibilidad de datos bibliográficos por lo que se definirán más adelante.
- **Seleccionar el método de ordenación *a posteriori* de los resultados:** El usuario ha de tener la posibilidad de seleccionar entre varios métodos de ordenación de los resultados en base a su interacción con el sistema.
- **Realizar búsquedas de autores:** El usuario podrá ejecutar búsquedas de autores entre el colección de los mismos. Por motivos de comodidad y familiaridad se utilizarán como autores aquellos pertenecientes a la Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada.
- **Realizar búsquedas de artículos:** El usuario deberá tener la posibilidad de hacer búsquedas de artículos en el sistema. Al igual que

en la historia previa el conjunto de artículos estará constituido por los trabajos de los autores mencionados previamente.

- **Desplegar una vista detallada de un artículo:** El usuario dispondrá de la posibilidad de desplegar una vista detallada de cada uno de los artículos devueltos en la búsqueda. Dicha vista tiene como objetivo mostrar una información básica sobre los artículos que permita discernir su contenido y relevancia, así como permitir enlazar a otras fuentes de información más detalladas sobre el mismo. Contendrá al menos los siguientes datos:
 - Título
 - Autores
 - Resumen
 - Información bibliográfica
 - Palabras clave
- **Desplegar una vista detallada de un autor:** El usuario dispondrá de la posibilidad de desplegar una vista detallada de cada uno de los autores devueltos en la búsqueda. Al igual que en el caso anterior ha de contener un extracto de información sobre el autor así como enlaces a otras fuentes de datos o perfiles con más datos. Contendrá al menos los siguientes datos:
 - Nombre
 - Apellidos
 - Departamento / Grupo de investigación
 - Información bibliográfica

Capítulo 5

Diseño

5.1. Fuentes de datos

Al ser este un sistema RI el diseño ha de girar en torno a los datos, por ello comencé explorando diversas alternativas para utilizar como fuente de datos.

Como punto inicial para la extracción de los datos de los autores de la E.T.S.I.I.T. partí del Ranking UGRinvestiga [21]. En este se lista a todos los autores de la escuela junto con sus citas e índice h total y los últimos 5 años.

Sin embargo no hay modo de obtener datos de los artículos concretos por lo que barajé múltiples bases de datos bibliográficas, además de las ya comentadas en la sección Sistemas de RI actuales con medidas bibliométricas también exploré Research Gate.

Los resultados de mi análisis fueron los siguientes:

- Google Scholar y Research Gate parecen tener bastantes datos en comparación a otras opciones, sin embargo no ofrecen una API por lo que la extracción de datos solo se podría llevar a cabo mediante *web scraping* 6. Esto no es un proceso sencillo ya que muchas de estas web contienen mecanismos de detección de bots como los CAPTCHAs [22] y además es un método muy sensible al cambio: en caso de que produzca alguna modificación en la web en cuestión es probable que toque reescribir el método de extracción de datos.
- WoS y Scopus sí disponen de una API, tras realizar algunas pruebas y leer opiniones sobre ambas me decanté por utilizar Scopus como fuente de datos ya que su API me resultó más cómoda de utilizar y encontré datos fácilmente de algunos autores de la E.T.S.I.I.T. (cosa que me

costó bastante más en WoS).

5.2. Modelo de datos

Siguiendo el modelo de Scopus decidí separar la búsqueda de autores y la de artículos como apunté en los objetivos.

Por tanto mi modelo de datos está constituido por 2 entidades distintas pero relacionadas: Autores (*Author*) y Artículos (*Abstract*).

En las siguientes secciones se describen brevemente ambas entidades.

5.2.1. *Author*

Esta entidad es resultado de la combinación de la información disponible en el Ranking UGRinvestiga [21] y en Scopus, por eso podemos ver en el siguiente diagrama con el prefijo **ugr** las medidas de dicho ranking y con el de **scopus** los de la plataforma homónima.

Respecto a los atributos que finalizan en 5 estos hacen referencia a los valores de dichas medidas de los últimos cinco años mientras que este sufijo no aparece las medidas son totales.

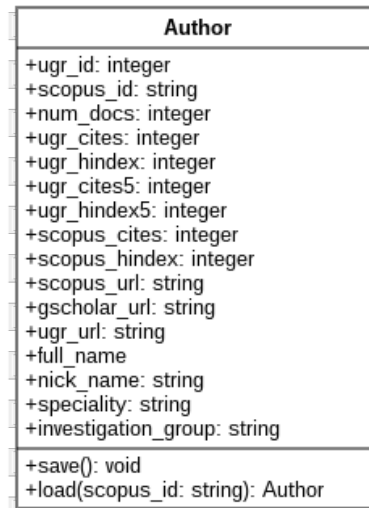


Figura 5.1: Clase *Author*

5.2.2. *Abstract*

La entidad *Abstract* corresponde casi directamente con los datos obtenidos a través de la API de Scopus, a excepción de:

- Las referencias han sido limitadas a citas internas, es decir entre los artículos que están disponibles en la colección, con la idea de usar esta información en el proceso de búsqueda.
- Los autores, que únicamente aparecían listados con su `scopus_id`, han sido enriquecidos con el nombre de los mismos así como el `ugr_id` en caso de estar disponible este último (este identificador estará disponible si el autor es alguno de los que forman parte de la colección de autores).

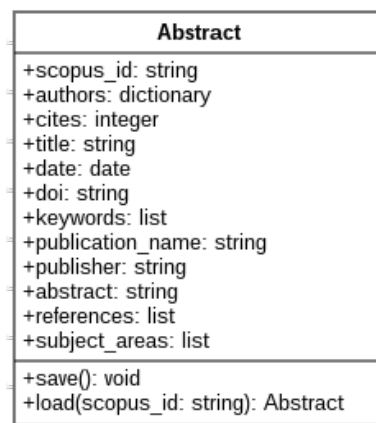


Figura 5.2: Clase *Abstract*

5.3. Arquitectura inicial del sistema

La arquitectura del sistema planteado se basará en el modelo cliente/-servidor con tres nodos:

- **Servidor de Base de Datos:** El cual almacenara la información correspondientes a autores y artículos como se ha modelado en el apartado previo.
- **Servidor de RI:** Donde se llevará a cabo la búsqueda en sí, se servirá del servidor de BD como almacenamiento de datos y atenderá las peticiones de búsqueda.
- **Cliente:** Parte del sistema que será visible para el usuario y que se encargará de recoger las peticiones de búsqueda, enviarlas al servidor de RI y presentar los resultados devueltos por el mismo



Figura 5.3: Diagrama de arquitectura inicial del sistema

Capítulo 6

Desarrollo

En este capítulo se detallara el proceso de desarrollo seguido, dividido en *sprints* como ya se ha comentado en la sección Metodología de desarrollo, así como un apartado inicial sobre las herramientas software empleadas y uno final con las modificaciones que ha sufrido la arquitectura planteada a lo largo del proceso de desarrollo.

6.1. Herramientas software empleadas

En esta sección describiré las herramientas software utilizadas a lo largo de todo el desarrollo del proyecto y el uso concreto para el que se han empleado

- **Debian:** Sistema Operativo (SO) utilizado por mi equipo personal durante todo el desarrollo.
<https://www.debian.org/index.es.html>
- **Python:** Lenguaje de programación usado en las fases exploratorias del proyecto, la indexación de documentos y que ejecuta el servidor *backend* 7.
<https://www.python.org/>
- **JavaScript:** Lenguaje de programación interpretado en el que se ha escrito el *frontend* 8 de la aplicación.
<https://developer.mozilla.org/es/docs/Web/JavaScript>
- **Elasticsearch:** Servidor de búsqueda utilizado como motor principal del sistema, este contiene los índices sobre los que se buscan y realiza la búsquedas por contenido.
<https://www.elastic.co/products/elasticsearch>

- **MongoDB**: Base de datos NoSQL empleada como almacén principal de datos a la cual se conecta el servidor *backend* para servir los datos.
<https://www.mongodb.com/es/what-is-mongodb>
- **ReactJS**: *Framework JavaScript* para el desarrollo de interfaces de usuario desarrollado por *Facebook* y utilizado como principal motor que potencia el *frontend*.
<https://reactjs.org/>
- **Searchkit**: *Framework* que incluye un conjunto de componentes *React* para la comunicación con *Elasticsearch*, algunos de los cuales se han utilizado para el desarrollo, incluyendo el tema y *layout* general de la interfaz de usuario.
<http://www.searchkit.co/>
- **TeXstudio**: Entorno integrado de escritura en \LaTeX utilizado para generación de esta memoria.
<https://www.texstudio.org/>
- **Docker**: Software de virtualización para basado en contenedores. Permite gestionar de forma simple la gestión y despliegue de una infraestructura software. Se ha empleado para gestionar la infraestructura a lo largo de todo el desarrollo ya que permite realizar pruebas rápidas de servicios, al no requerir instalar ninguna utilidad directamente en la maquina de desarrollo, si no simplemente descargar contenedores ya preparados y configurados para ser utilizados.
<https://www.docker.com/>
- **Visual Studio Code**: Editor de código multiplataforma creado por *Microsoft* y utilizado para toda la programación llevada a cabo en el proyecto del proyecto.
<https://code.visualstudio.com/>
- **Git**: Sistema de control de versiones empleado para versionar todos los artefactos del proyecto.
<https://git-scm.com/>
- **GitHub**: Plataforma de desarrollo colaborativo utilizada como servidor *git* donde almacenar todos los artefactos del proyecto.
<https://github.com/>
- **Pandas**: Librería de análisis de datos en *Python* que se ha convertido en estándar *de facto* para el manejo de datos en dicho lenguaje. Se ha utilizado para la extracción y limpieza de datos.
<https://pandas.pydata.org/>

- **PyMongo**: *Driver* oficial de *MongoDB* en *Python*, utilizado en todos los accesos programáticos a dicha BD.

<https://github.com/mongodb/mongo-python-driver>

- **Robo3T**: Cliente de *MongoDB* con interfaz gráfica de usuario empleado para realizar pequeñas consultas y poder ver los datos de manera más visual durante el proceso de desarrollo.

<https://robomongo.org/>

- **scopus-api**: Librería *Python* para la interacción con la API de Scopus utilizada para la obtención de datos.

<https://scopus.readthedocs.io/en/latest/>

- **Beautiful Soup**: Librería *Python* que facilita el *web scrapping* utilizada para extraer datos del ranking UGRinvestiga.

<https://www.crummy.com/software/BeautifulSoup/>

- **Matplotlib**: Librería *Python* para generar gráficos en dos dimensiones. Utilizada para crear algunos gráficos durante el análisis exploratorio de datos.

<https://matplotlib.org/>

- **MaterialUI**: *Framework JavaScript* que contiene un conjunto de elementos *React* implementando el estándar de diseño *Material Design* de *Google*. Utilizado para algunos componentes de la interfaz de usuario como los botones, vistas detalladas de artículos o autores y otros controles.

<https://material-ui.com/>

- **elasticsearch-py**: Cliente *Python* oficial de ES, este permite el control a bajo nivel de dicho servidor de búsqueda. Se ha empleado para realizar la indexación de los datos en ES

<https://elasticsearch-py.readthedocs.io/en/master/>

- **Star UML**: Programa de modelado de software utilizado para la generación de los diagramas de este proyecto.

<http://staruml.io/>

- **Cerebro**: Servidor de monitorización de ES empleado para gestionar el servidor durante el desarrollo, así como para realizar algunas pruebas ya que dispone de una pequeña interfaz que permite lanzar consultas contra dicho servidor.

<https://github.com/lmenezes/cerebro>

- **Flask**: *Framework* de desarrollo web *Python*, es el que potencia los servicios del *backend*.
<http://flask.pocoo.org/>
- **Gimp**: *Software* de manipulación de imágenes integrado en mi sistema operativo y que ha sido utilizado para editar algunas de la imágenes de esta memoria.
<https://www.gimp.org/>
- **Nginx**: Servidor web que recibe las peticiones del navegador, es encargado de servir los ficheros estáticos del *frontend* y derivar las peticiones al *backend*.
<https://www.nginx.com/>
- **uWSGI**: Servidor de aplicaciones *Python* encargado de ejecutar el *backend Flask* comunicándose con *Nginx*.
<https://uwsgi-docs.readthedocs.io/en/latest/>

6.2. *Sprint 1*

Este primer *sprint* tiene como objetivo general la investigación básica sobre la RI. Sus objetivos concretos son:

- Leer el libro Recuperación de Información: un enfoque práctico y multidisciplinar [2].
- Buscar los talleres BIR centrándose en sus editoriales para realizar un listado priorizado por interés de los diversos artículos.

Siguiendo la recomendación de mi tutor (co-autor del libro) me centré en los capítulos "*1 Introducción a la recuperación de información*", "*2 Indexación de documentos y procesamiento de consultas*", "*3 Modelos de recuperación de información clásicos*" y "*10 Técnicas de modificación de la consulta*".

Esta lectura me hizo adquirir unas bases más teóricas a lo que ya había estudiado en la asignatura del máster Gestión de Información en la Web donde obtuvimos una nociones básicas de lo que supone la RI y sus vertientes, realizando alguna práctica.

Los artículos de los talleres *Bibliometric-enhanced Information Retrieval* que encontré más destacados están detallados en el apartado Trabajos relacionados. Dichos trabajos se encuentran disponibles gratuitamente en <http://ceur-ws.org/> bajo el amparo de la Universidad Técnica de Aquisgrán (*RWTH Aachen University*) en Alemania.

Cada una de las ediciones de estos talleres se encuentra estructurada dividida en diversos trabajos. Por un lado un editorial que resume la edición y todos los trabajos aceptados en la conferencia así como los trabajos individuales, alguna *keynote* o presentación y algunas demostraciones.

En este *sprint* me dediqué a leer dichos editoriales clasificando por aparente interés los artículos de cada BIR generando una lista priorizada utilizada como orden en el que estudiar los trabajos. En la siguiente imagen se puede apreciar el aspecto de dicha lista generada en *Markdown*.

Lista priorizada de papers

- ✓ **BIR2014_1:** Meaning of citations, use of words in standard parts. Find interesting distribution of words
- ✓ **BIR2014_5:** iSearch analysis. Extract connection between citation based and topic relevance rankings checking if this collection is suitable to perform these kind of analysis
- ✓ **BIR2015_1:** Potentiality of interdisciplinary research between IR and bibliometrics. Ask several research questions summarizing other investigations in between of this disciplines.
- ✓ **BIR2015_4:** bibliometric-enhanced search features based on Marcia Bates model with 2 specific examples.
- ✓ **BIR2014_2:** use citations a mechanism for systematic IR of scientific literature. Helpful in situations in which one needs a comprehensive overview of a research topic.
- ✓ **BIR2014_4:** Use citation and co-citation as previous probabilities for relevancy. Measures: citation count, PageRank & co-citation clustering. Very query-dependent, suggestion to improve using this citation info & content features
- ✓ **BIR2017_11:** how to improve scientific recommender systems using bibliometry Measures.
- ✓ **BIR2016_7:** Alternative to the bag of words model (TF*IDF) used to find similar documents to a giving one based on citations. The tf and idf meares re-defined using citation information.
- ✓ **BIR2015_5:** Combining bibliographics and IR studying how polyrepresentation could be used to improve interactive search. Alternatives to ranked lists using bibliographics and contextual information. Based in clustering.

Figura 6.1: Fragmento de la lista priorizada creada

6.3. *Sprint 2*

Para el segundo *sprint* planteo el indagar en la RI e ir introduciendo la bibliometría. Sus objetivos concretos son:

- Leer los primeros papers de los BIR resumiendo y extrayendo ideas interesantes para el proyecto.
- Buscar información sobre medidas bibliométricas (Citas, índice h combinado...)

A partir de la lista priorizada fui leyendo los primeros artículos, el orden de esta lista lo fui alterando ya que con frecuencia al indagar en el trabajo este perdía o incrementaba su interés. Con el objetivo de poder aprovechar más estas lecturas fui creando una especie de resúmenes en los que anotaba los puntos más importantes que se trataban en el artículo, otros artículos

relacionados de los que se hablaba o los resultados obtenidos con su trabajo. En la siguiente imagen se puede apreciar uno de estos resúmenes”:

BIR2014_5

"On the Connection Between Citation-based and Topical Relevance Ranking: Results of a Pretest using iSearch"

Zeljko Carevic and Philipp Schaer

- **IR limitations.** Buckley, C.: Why current IR engines fail. *Inf. Retr.* 12, 6, 652–665 (2009).
- **Science Citation Index increase performance in scholarly IR systems.** Pao, M.L.: Term and citation retrieval: A field study. *Inf. Process. Manag.* 29, 1, 95–112 (1993).
- **iSearch test collection.** Lykke, M. et al.: Developing a Test Collection for the Evaluation of Integrated Search. In: Gurrin, C. et al. (eds.) *Advances in Information Retrieval*. pp. 627–630 Springer, Berlin, Heidelberg (2010).
- **Novel IR approach using Bibliometric & IR with a tf*idf schema.** White, H.: Some new tests of relevance theory in information science. *Scientometrics.* 83,3, 653–667 (2010).
- **TF*IDF schema using co-citation as measure of similarity:**
 - TF: #citations of the seed document together with the target document to analyze
 - DF: #overall citations for the target document in all iSearch collection
 - Method:
 - a. Given seed document A retrieve all documents that cite A
 - b. List of candidates created taken all citations from the documents in step 1
- **Result:** Using only internal references and relevant documents retrieve not enough documents to perform an analysis.
- **Further works:** Expand co-citation analysis using title, authors, journal and publication year along with the cites itself

Figura 6.2: Ejemplo de resumen de uno de los papers

En este momento me empecé a introducir en el mundo de las medidas bibliométricas, ya que no tenía conocimiento previo alguno de que existiera esta disciplina siquiera, mediante la lectura de artículos iba descubriendo distintas medidas así como sus posibles aplicaciones a la RI cuyo resultado final se encuentra sintetizado en la sección Bibliometría.

6.4. *Sprint 3*

Ya en este punto decidí que estaba listo para ir documentando todo lo que había aprendido por ello en este *sprint* me puse como objetivo escribir la introducción de este TFM. Los objetivos concretos son:

- Sintetizar mi estudio hasta este punto en los primeros capítulos de este TFM.
- Pensar un enfoque para el proyecto a desarrollar decidiendo que clase de sistema se desarrollaría.
- Continuar leyendo algunos artículos más de los BIR.

Para asimilar y reflexionar sobre todo lo que había leído me puse a escribir el capítulo Contexto e Introducción de este trabajo ya que ello me ayudaría a pensar un enfoque correcto. También quería poder mostrar algo a mi tutor para obtener algo de *feedback* por su parte.

Continué leyendo algunos trabajos más con lo que di por concluido mi proceso de investigación. Uno de esos últimos trabajos fue [20] el cual me gustó especialmente ya que pasaban de un modelo teórico a algo más práctico, accediendo a la API de Scopus e incluyendo ejemplos de su implementación en un repositorio de *GitHub*, lo que me llevó a plantear mi modelo híbrido combinando el sistema habitual de un sistema RI con reordenamiento de resultados *a priori* usando medidas bibliométricas y un ordenamiento *a posteriori* utilizando un grafo de citación entre los documentos.

Desgraciadamente estos enfoques dependen ampliamente de la cobertura de medidas bibliométricas disponibles. Por ejemplo, me hubiera encantado poder probar una reordenación previa utilizando alguna *altmetric* como el número de lecturas o descargas de un artículo, pero la plataforma que he utilizado para extraer los artículos no dispone de dichas medidas.

6.5. *Sprint 4*

Una vez sintetizado lo investigado y teniendo una idea aproximada de lo que pensaba desarrollar, en este *sprint* me dispuse a realizar diversas pruebas que fueran definiendo las tecnologías a emplear, en concreto:

- Buscar soluciones para montar sistemas RI.
- Investigar como conectarse a la API de Scopus o WoS.

Estuve haciendo alguna pruebas con algunos *frameworks* de búsqueda. Ya había utilizado *Lucene* como parte de las prácticas de la asignatura GIW, pero me pareció demasiado bajo nivel así que me centré en investigar otras alternativas. Encontré que las principales, que casualmente utilizaban por debajo *Lucene*, eran ***Solr*** y ***Elasticsearch (ES)***.

Buscando alguna comparativa [23] y comentarios de usuarios en plataformas tan reputadas como *StackOverflow* [24] parecía que se recomendaba ES por ser más sencillo de usar, esto me animó a realizar un breve tutorial [25] que me gustó bastante ya que resulta realmente simple de usar y basta únicamente con realizar consultas sobre una API REST por lo que basta con hacer peticiones HTTP con algún cliente simple como ***curl***. Todo esto hizo que me decidiera por este servidor de búsqueda.

Como ya comenté en el apartado 5.1 realicé un análisis de diversas fuentes de datos y me decanté por seleccionar alguna que dispusiera de API,

en concreto Scopus ya que me resultó más rápido encontrar artículos que me servirían para mi colección documental. Por lo que comencé buscando la implementación que habían realizado en el artículo [20], ya que en el mismo comentan que se encuentra disponible en *GitHub* [26]. Vi que dicha implementación era de muy bajo nivel, extraía los artículos mediante peticiones HTTP y manejaban el XML obtenido directamente. Pensé que tenía que haber un mejor método para esto, alguna librería que ya implementara los métodos de la API y abstrayera de esas tareas.

Así encontré la implementación oficial ofrecida por Elsevier, `elsapy` [27], un modulo en *Python* que cuenta con diversas clases para modelar los tipos de documentos y permite obtener y parsear los datos de la API de manera transparente. También encontré el módulo `scopus` [28] una implementación alternativa que había comenzado a desarrollarse algo antes que la oficial, parecía más activa y contaba con mejor documentación incluyendo ejemplos bastante útiles.

Funcionalmente `scopus` era muy similar a la versión oficial, pero estaba mejor modelada, se centraba solo en Scopus, que era lo que yo necesitaba, y tenía algunos añadidos muy útiles como uso de una caché por defecto con el objetivo de no repetir peticiones ya hechas, si no servir las desde un documento almacenado en disco, lo que hacía increíblemente veloz la recuperación de entidades ya consultadas previamente así como gestión transparente de API *key*³. Todo esto hizo que me decantara por esta última sobre la implementación oficial.

Para poder realizar mis pruebas tuve que solicitar una API *key* en el portal de Elsevier developers <https://dev.elsevier.com/myapikey.html> creando una cuenta gratuita. Para poder descargar los datos de Scopus desde fuera de la red de la universidad me ha sido necesario utilizar el servicio de VPN de la UGR siguiendo el siguiente tutorial [29].

En la siguiente tabla se puede contemplar las distintas APIs de Scopus así como sus límites. En concreto he utilizado las APIs: *Author Search* para realizar las búsquedas de autores, *Author Retrieval* para obtener los datos completos de los autores y *Abstract Retrieval* para obtener los artículos o *abstracts*. Como se puede ver los límites de las APIs son bastante elevados 5000 peticiones semanales para la búsqueda y recuperación de autores así como 10000 para los *abstracts*, por ello no han sido rebasados, pero en caso de sobrepasarlos bastaría con solicitar otra API *key* y cambiarla en el módulo `scopus`.

³Basta con introducirla en la primera ejecución y la propia librería la guarda en un fichero interno el cual usa para su consulta el resto de ocasiones.

Scopus APIs

#	API Name	Enabled or Disabled	Non-subscriber	Subscriber	Weekly Quota	Requests/second
1	Serial Title	Enabled	STANDARD, COVERIMAGE views / Default 25 results / Max 200 results	STANDARD, COVERIMAGE, ENHANCED Default 25 results / Max 200 results	20,000	3
2	Citations Count Metadata	Disabled	N/A	STANDARD view / Default 25 results / Max 200 results	50,000	18
3	Citations Overview	Disabled	N/A	STANDARD view / Default 25 results / Max 200 results	20,000	3
4	Subject Classifications	Enabled	No restrictions	No restrictions	N/A	N/A
5	Abstract Retrieval	Enabled	META view	All views, default FULL view	10,000	6
6	Affiliation Retrieval	Enabled	N/A	All views, default STANDARD view	5,000	6
7	Author Retrieval	Enabled	N/A	All views, default STANDARD view	5,000	3
8	Affiliation Search	Enabled	N/A	Default 25 results / Max 200 results	5,000	3
9	Author Search	Enabled	N/A	Default 25 results / Max 200 results	5,000	3
10	Scopus Search	Enabled	STANDARD view / Default 25 results	STANDARD view / Max 200 results COMPLETE view / Max 25 results COMPONENT view / Max 25 results	20,000	6
11	Author Feedback	Disabled	N/A	N/A	N/A	N/A

Figura 6.3: APIs de Scopus junto con sus límites

6.6. *Sprint 5*

Tras toda la documentación e investigación previas en este punto realicé el proceso de obtención de datos. Dicho proceso fue dividido en

- Extraer información de los autores del ranking UGRinvestiga.
- Usar esos autores obtenidos para buscarlos en Scopus.
- Descargar todos los *abstracts* de los autores obtenidos en el punto previo.
- Preprocesar y limpiar los datos obtenidos.

6.6.1. Obtención de datos

Teniendo en cuenta que no parecía haber opción de descarga de los datos del Ranking UGRinvestiga y que el formato parecía bastante estático con todos los datos en una tabla mi primer enfoque fue realizar un *web scrapping* utilizando la librería BeautifulSoup, ya comentada previamente. El aspecto de la tabla se puede ver en la siguiente imagen.

Tipo de Ranking: Disciplina Periodo: Histórico					
 Ingenierías Informática y de Telecomunicación					
Rank	Nombre	Citas	h-index	Perfil	Grupo
1	FRANCISCO HERRERA TRIGUERO	61092	123	Perfil	TIC186
2	ENRIQUE HERRERA VIEDMA	25156	76	Perfil	TIC186
3	JOSE LUIS VERDEGAY GALDEANO	12607	48	Perfil	TIC169
4	OSCAR CORDON GARCIA	12027	52	Perfil	TIC186
5	SALVADOR GARCIA LOPEZ	11163	37	Perfil	TIC186
6	MIGUEL DELGADO CALVO-FLORES	9288	47	Perfil	TIC111
7	MARIA AMPARO VILA MIRANDA	8639	45	Perfil	TIC174
8	MANUEL LOZANO MARQUEZ	6806	31	Perfil	TIC186
9	ALBERTO LUIS FERNÁNDEZ HILARIO	6725	32	Perfil	TIC186
10	JUAN JULIAN MERELO GUERVOS	6464	33	Perfil	TIC024

Figura 6.4: Aspecto tabla ranking UGRinvestiga

Realizando este proceso extraje datos de históricos y de los últimos 5 años de los autores de la escuela, además de aplicar este proceso al catálogo de grupos de investigación de Tecnologías de la Información y la Comunicación de la UGR para realizar una unión por código de grupo con los datos del ranking.

Posteriormente di con el proyecto OpenData de la Universidad de Granada el cual entre otros *datasets* ofrecía el mencionado ranking actualizado a fecha 28 de Mayo de 2018 [30] en formato Excel, fácilmente legible y con algunos datos adicionales como el departamento de los autores, la url de Google Scholar de cada autor o su *nick* el cual podía ser muy útil para buscarlos en Scopus.

Tras cargar estos últimos datos obtuve **214 autores** de la escuela. A continuación pasé al problema de buscar estos autores en Scopus, este problema no es nada trivial, ya que con frecuencia los autores no firman sus trabajos con su nombre completo, si no con parte de él o incluso de manera distinta entre unos trabajos y otros. Por ejemplo mi tutor, "JUAN MANUEL FERNÁNDEZ LUNA" suele firmar como "Fernández-Luna, Juan M". o como "Fernández-Luna, J.M".

El formato necesario para la búsqueda de autores en Scopus es nombre, apellidos y afiliación (la UGR en este caso) por lo tanto primero me dispuse a extraer el nombre y apellidos a partir de la cadena completa que disponía, esto puede parecer sencillo, pero la riqueza del lenguaje español hace que existan apellidos compuestos o personas con segundos nombres lo que dificulta la tarea.

Estos problemas con los nombres me hicieron montar el siguiente algoritmo para intentar buscar un autor en Scopus:

Algorithm 1 Obtiene los autores de Scopus a partir del ranking UGR

```

for all author in author_list do
   $f\_name, l\_name \leftarrow get\_name(author.nick)$ 
   $success, auth\_result \leftarrow author\_query(f\_name, l\_name)$ 
  if not success then
     $f\_name, l\_name \leftarrow get\_name(author.full\_name)$ 
     $success, auth\_result \leftarrow author\_query(f\_name, l\_name)$ 
    if not success then
       $f\_name, l\_name \leftarrow get\_name(author.name\_wo\_1\_lname)$ 
       $success, auth\_result \leftarrow author\_query(f\_name, l\_name)$ 
      if not success then
         $f\_name, l\_name \leftarrow get\_name(author.name\_wo\_2\_lname)$ 
         $success, auth\_result \leftarrow author\_query(f\_name, l\_name)$ 
        if not success then
          return No results found
  return  $auth\_result$ 

```

Donde f_name y l_name hacen referencia a nombre y apellidos respectivamente y $name_wo_2_lname$ hace referencia al nombre completo del autor eliminando el segundo apellido.

Con este algoritmo he logrado encontrar algún resultado en Scopus para **202 de los 214** autores buscados (**~ 94.39 % de los autores**), a pesar de esto el número total de autores recuperados son 397, por lo que habrá que realizar una limpieza de los mismos.

Limpieza de datos de autores

Si observamos la distribución de **nacionalidades de los autores** recuperados, la cual se ve en la siguiente imagen se aprecia que hay una mayoría abrumadora de Españoles (como cabría esperar) pero también algunas nacionalidades inesperadas como palestina o australiana. Por ello consideraré que todo autor cuya nacionalidad no sea española debe haber sido recuperado incorrectamente y será eliminado al tratarse de ruido, haciendo esto pasamos de 397 autores de Scopus representando a 202 autores de la UGR

a un total de 381 autores de Scopus representando a **198 autores de la UGR** ($\sim 98.02\%$ de los 202 previos).

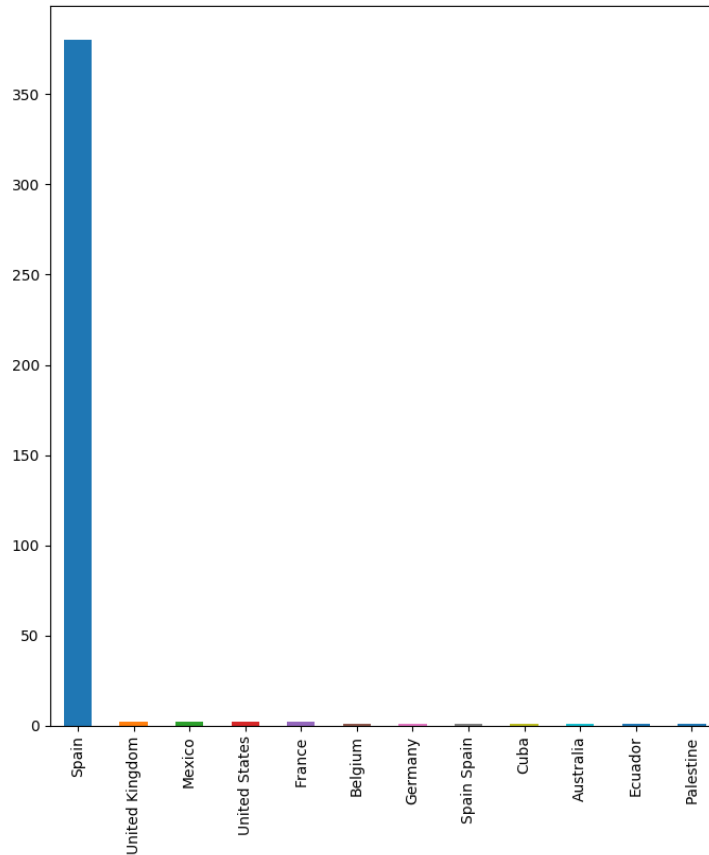


Figura 6.5: Distribución de nacionalidades entre los autores recuperados

Continuando el filtrado, si mostramos la distribución de **ciudades de los autores** restantes (ver la siguiente imagen) vemos que Granada o sus variantes aglutinan a la gran mayoría de los autores, por ello filtraré todos aquellos que no sean Granada lo cual me deja con un total de 334 autores de Scopus (**188 autores de la UGR** $\sim 94.95\%$ de los 198 previos).

Por último he realizado un filtrado por **área del conocimiento** descartando aquellos autores que no tengan al menos un trabajo englobado en el área de Ciencias de la Computación (*computer science*) con lo que finalmente me quedo con 199 autores de Scopus (**164 de la UGR**, $\sim 87.23\%$ de los 188 previos)

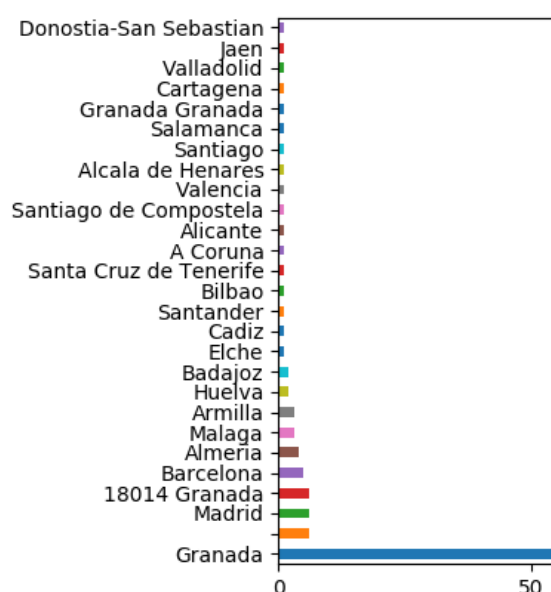


Figura 6.6: Fragmento de la distribución de ciudades entre los autores españoles recuperados

En la siguiente gráfica represento un *clustering de autores* UGR donde aparecen etiquetadas las siguientes clases: la clase *comp_spa_gr* representa a los autores que tienen algún autor en Scopus el cual es Español, de Granada y tiene algún artículo de Ciencias de la computación; la clase *spa_gr* igual con excepción del área del conocimiento, la clase *spa* que tiene algún autor solo español y la clase *other* que no cumple ningún criterio. Por tanto cada clase se ha ido eliminando en cada filtrado hasta quedarnos solo con la primera, la cual representa al $\sim 81.19\%$ de los autores totales de la UGR (164 de 202 autores posibles).

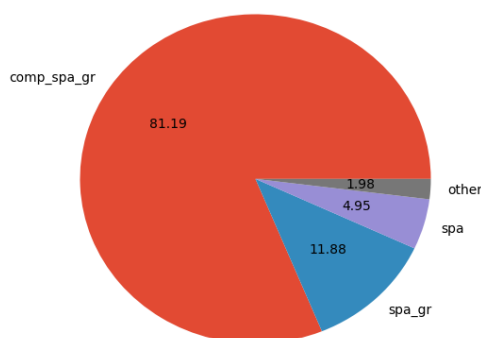


Figura 6.7: Representación del clustering de autores llevado a cabo en la limpieza de datos

Obtención de autores

Esta limpieza se llevó acabo usando los resultados devueltos por la búsqueda de autores de Scopus (usando la clase `AuthorSearch` del paquete `scopus` la cual implementa la API *Author Search* de Scopus). Estos resultados de búsqueda contienen un subconjunto de los datos de autores disponibles, por lo que el siguiente paso fue descargar la ficha de cada uno de los 199 autores de Scopus usando para ello la clase `ScopusAuthor` del paquete `scopus` la cual implementa la API *Author Retrieval* de Scopus.

Obtención de abstracts

Una vez hecho esto me dispuse a llevar a cabo el grueso de la obtención de datos, la descarga de todos sus *abstracts*. A partir de los autores de Scopus obtenidos previamente se obtienen todos los `eid` o `scopus_id` de sus artículos con la clase `ScopusAuthor`. Utilizando esos identificadores recuperaré todos los *abstracts* usando la clase `ScopusAbstract` del paquete `scopus`, la cual implementa la API *Abstract Retrieval*, en su vista completa o *full* [31], ya que es la única que incluye las referencias, las palabras clave y las áreas del conocimiento de cada *abstract*.

Los 199 autores recuperados cuentan con un total de 1263 artículos en Scopus, aunque este dato no es real, ya que frecuentemente los artículos tienen varios autores de entre esos 199, por lo que estos serían contados múltiples veces. El número final de artículos únicos asciende a **891**.

6.6.2. Preprocesado de los datos obtenidos

En este punto había obtenido tres conjuntos de datos relacionados pero algo inconexos:

- 164 autores del Ranking UGRinvestiga (tras la limpieza)
- 199 autores de Scopus (tras la limpieza)
- 891 *abstracts* de Scopus.

El primer paso como parece lógico fue combinar los autores de la UGR y Scopus. El numero de estos no coincide ya que Scopus a veces no es capaz de determinar que dos autores son la misma persona si firma de distinta manera, por ello crea dos autores distintos.

Para realizar esta combinación agrupé los autores de Scopus por el ID UGR sumando el número de documentos y de citas de los autores individuales seleccionando el mayor índice h individual entre ellos. A continuación

mezclé estos 164 autores combinados de Scopus con los datos provenientes del Ranking UGRinvestiga y del nombre del grupo de investigación, quedando finalmente constituida la entidad de datos descrita en la sección *Author* del capítulo Diseño. Dicha entidad se encuentra implementada en la clase del modelo **Author** y ha sido introducida en la BD *MongoDB* en la colección *author*.

En los *abstract* emulé el proceso de combinación de autores asignando los artículos al mismo autor UGR aunque originalmente fueran de distintos autores Scopus.

Tras ello limpié las referencias de cada uno de ellos para mantener solo las referencias a artículos que formaban parte de la colección, eliminando las referencias a otros trabajos externos. Esto puede suponer una pérdida de información, ya que limitaría el análisis de co-citación, pero hace más manejable el conjunto de referencias y como se comprueba en [20], cuanto más directa sea la relación de citación, más útil resulta para este análisis. Aplicando esta limpieza se mantienen un total de **742** referencias, quedando con esto la entidad descrita en la sección *Abstract* del capítulo Diseño. Dicha entidad se encuentra implementada en la clase del modelo **Abstract** y ha sido introducida en la BD *MongoDB* en la colección *abstract*.

Como curiosidad de este *sprint* me gustaría destacar que durante mi investigación de los datos devueltos por Scopus sobre los *abstracts* me percaté que la librería **scopus** no parseaba las palabras clave de los mismos aunque estas aparecían en los XML obtenidos de la API. Estas palabras clave podían funcionar bien en los sistema de búsqueda, como había leído durante mi proceso de documentación, por ello añadí esa funcionalidad a la librería y dicho cambio fue incorporado a la misma en su versión 0.9.

6.7. *Sprint* 6

Una vez tenía los datos preparados en este sprint me marqué como objetivo crear un sistema de búsqueda básico por contenido (sin aplicar bibliometría). Este objetivo se descompuso en:

- Indexar los datos en *Elasticsearch*.
- Crear un cliente web que permita la búsqueda usando ES.
- Crear un servidor *backend* como apoyo al proceso de búsqueda que permita recuperar los datos completos, no solo los indexados.

6.7.1. Indexación de los datos

Tras analizar un par de alternativas como motores de búsqueda (ver *Sprint 4*) me decanté por utilizar ES. Tras seguir un breve tutorial me pareció muy manual el proceso de introducir datos por lo que busqué alguna forma de hacerlo programáticamente. En mi búsqueda di con el tutorial [32], que utilizaba el cliente oficial de ES en *Python elasticsearch-py* [33]. Usando dicho tutorial como base y realizando un mapeo del índice ⁴[34] creé los índices *author* y *abstract*.

No todos los datos de dichas entidades han sido indexadas, ya que hay algunas que no aportaría nada a la búsqueda; es más solo la entorpecería al ocupar un espacio innecesario en el índice. Por ejemplo, los identificadores o URLs no aportan nada.

El índice *author* está constituido por los campos útiles para la búsqueda de contenido (de tipo *text*) y los campos de medidas bibliométricas para poder ordenar o aplicarlos en la búsqueda bibliométrica. En la siguiente tabla se recoge los campos concretos de la clase *Author* que forman parte del índice homónimo:

Campo	Tipo
<i>full_name</i>	<i>text</i>
<i>nick_name</i>	<i>text</i>
<i>scopus_cites</i>	<i>integer</i>
<i>scopus_hindex</i>	<i>integer</i>
<i>speciality</i>	<i>text</i>
<i>num_docs</i>	<i>integer</i>
<i>investigation_group</i>	<i>text</i>
<i>ugr_hindex</i>	<i>integer</i>
<i>ugr_cites</i>	<i>integer</i>
<i>ugr_hindex5</i>	<i>integer</i>
<i>ugr_cites5</i>	<i>integer</i>

Cuadro 6.1: Campos del índice *author*

⁴Forzar los datos y tipos de estos en un índice, no es necesario ya que ES es capaz de gestionarlo dinámicamente, pero con ello se consigue aplicar estrategias de búsquedas específicas para los tipos de datos y hace que no se pueda introducir un documento distinto por error.

Respecto al índice *abstract* los campos específicos son:

Campo	Tipo
<i>title</i>	<i>text</i>
<i>abstract</i>	<i>text</i>
<i>authors</i>	<i>text</i>
<i>subject_areas</i>	<i>text</i>
<i>keywords</i>	<i>text</i>
<i>publisher</i>	<i>text</i>
<i>publication_name</i>	<i>text</i>
<i>cites</i>	<i>integer</i>
<i>date</i>	<i>date</i>

Cuadro 6.2: Campos del índice *abstract*

6.7.2. Desarrollo del buscador por contenido

Teniendo en cuenta la características del proyecto cuyo objetivo es desarrollar un prototipo, el diseño no era un punto primordial, pero también quería crear un cliente que fuera fácilmente usable. Esto hizo que me decantara por crear un cliente web, ya que trabajo como desarrollador *full-stack* y es un ambiente en el que me siento cómodo y me permitiría desarrollar algo sin la necesidad de aprender tecnologías muy distintas.

Por eso comencé buscando librerías que facilitaran el desarrollo de interfaces de usuario para sistemas basados en ES. Me topé con dos destacadas *Searchkit* [35] y *Reactivesearch* [36], ambas basadas en el *framework* *ReactJS*. Básicamente suponen un conjunto de componentes visuales ya construidos (filtros, barras de búsqueda...), *layouts* y otras utilidades adicionales que facilitan la comunicación con ES.

Ojeando las documentaciones y ejemplos me dio la impresión de que *Reactivesearch* parecía más completa y adaptable, con más componentes y opciones de diseño, pero es desarrollada por *appbase.io*, una plataforma de DBaaS y la documentación de *Reactivesearch* se orienta mucho a que desarrolles aplicaciones de búsqueda utilizando su plataforma como fuente de datos, lo cual me echó algo para atrás. Por otro lado *Searchkit* parecía más simple, con menos componentes y opciones pero funcionales, ideal para mí ya que no buscaba realizar una interfaz gráfica compleja, si no simple. Contaba con ejemplos en vivo y proyectos de ejemplo demostrando la funcionalidad de cada uno de sus componentes.

Por todo ello decidí utilizar *Searchkit*. En la siguiente imagen se puede apreciar el aspecto de la demo de esta librería, como se puede observar cuenta con una interfaz bastante simple, muy inspirada en *Material Design*

[37] y con unas zonas claramente diferenciadas: En rojo podemos ver una barra superior con el nombre de la aplicación y la barra de búsqueda, a la izquierda en amarillo una zona de filtros, en recuadros azules un par de barras que muestran información sobre los resultados de búsqueda y permiten modificar la ordenación o visualización; por último en verde aparece la zona de resultados.

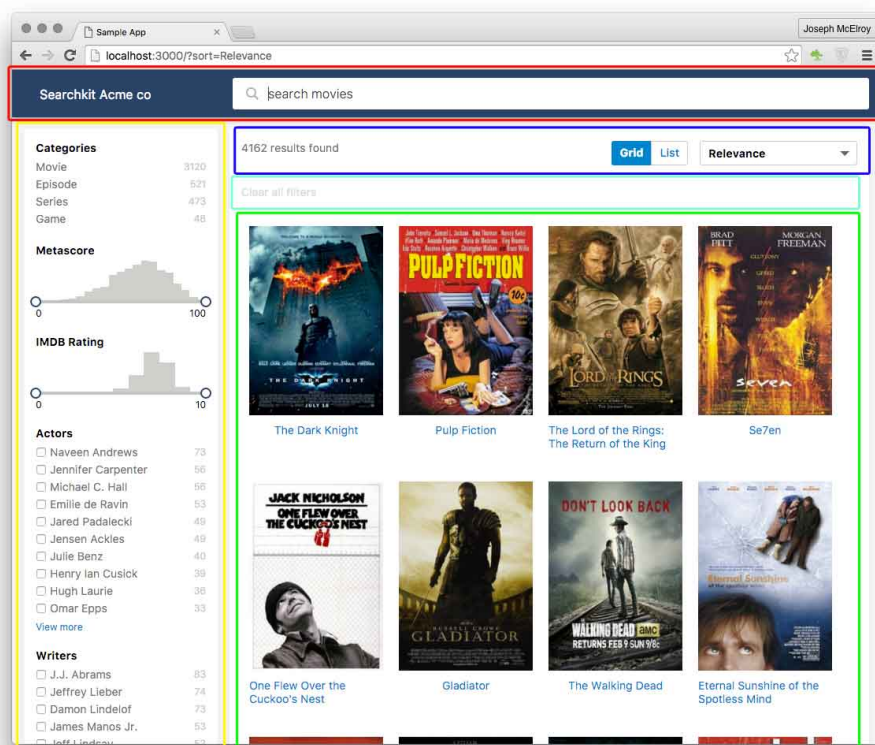


Figura 6.8: Aspecto demo *Searchkit*

Para mi aplicación seguiré este esquema a excepción de la zona de filtros amarilla ya que el pretendo que el la búsqueda sea lo más simple posible.

Habilitar la conexión entre ES y *Searchkit* es realmente sencillo, basta con instanciar un objeto `SearchkitManager` y pasarlo como argumento al componente `React SearchkitProvider` que englobe todos los componentes de *Searchkit* como si se tratara de un contexto. En el siguiente bloque de código se puede contemplar como se realiza la conexión con el índice de *author* de ES.

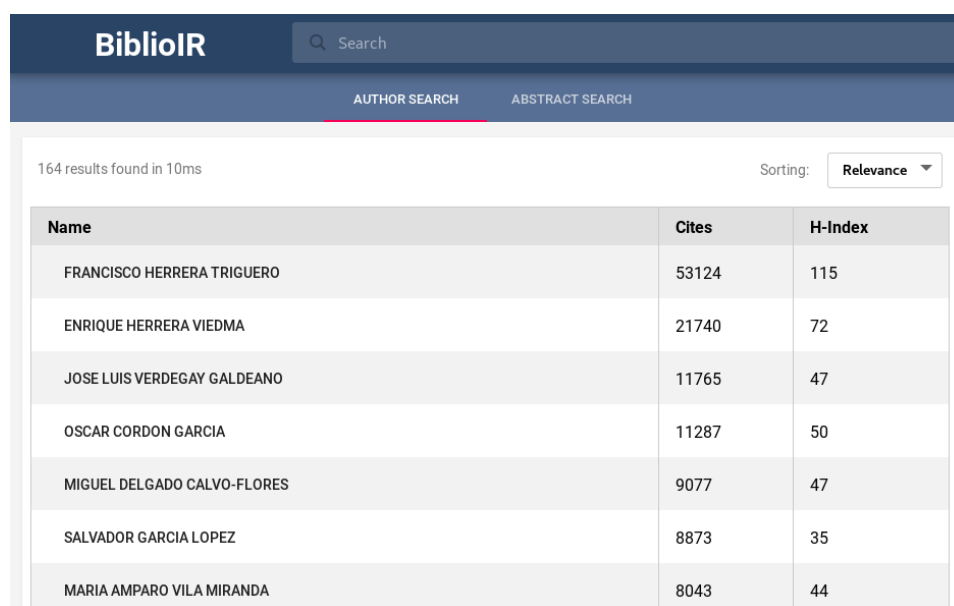
```
1 const searchkit = new SearchkitManager("http://
  localhost:9200/author")
2
3
4 <SearchkitProvider searchkit={searchkit}>
5   ...
6 </SearchkitProvider>
```

Listing 6.1: Conexión de *Searchkit* a una instancia ES local

Además de esto al conectarnos con una instancia local es necesario activar las opciones de ES relacionadas con CORS como la documentación oficial aclara [38].

Además de *Searchkit* también he utilizado el *framework MaterialUI* para algunos de los componentes visuales como las *tabs* de selección de búsqueda de autores o *abstracts* o los botones que aparecen en la aplicación.

Esta imagen muestra el aspecto que tiene el buscador de autores, como se puede ver la interfaz es realmente sencilla contando únicamente con una barra de búsqueda, unas pestañas para cambiar entre búsqueda de autores o artículos y las propias vistas de resultados así como un selector de orden de los mismos.



The screenshot shows the BiblioIR application interface. At the top, there is a dark blue header with the 'BiblioIR' logo on the left and a search bar with a magnifying glass icon and the text 'Search'. Below the header, there are two tabs: 'AUTHOR SEARCH' (which is active and underlined) and 'ABSTRACT SEARCH'. The main content area shows '164 results found in 10ms' and a 'Sorting: Relevance' dropdown menu. Below this is a table with three columns: 'Name', 'Cites', and 'H-Index'. The table lists seven authors with their respective citation counts and H-index values.

Name	Cites	H-Index
FRANCISCO HERRERA TRIGUERO	53124	115
ENRIQUE HERRERA VIEDMA	21740	72
JOSE LUIS VERDEGAY GALDEANO	11765	47
OSCAR CORDON GARCIA	11287	50
MIGUEL DELGADO CALVO-FLORES	9077	47
SALVADOR GARCIA LOPEZ	8873	35
MARIA AMPARO VILA MIRANDA	8043	44

Figura 6.9: Aspecto buscador de autores

Al seleccionar cualquiera de los autores haciendo click en su nombre podemos ver una ficha detallada con todos los datos de la clase **Author** incluyendo botones a los perfiles disponibles del autor.

JUAN MANUEL FERNANDEZ LUNA
General info

- **Nick name:** Juan Manuel Fernández Luna
- **Speciality:** Ciencias de la Computación e Inteligencia Artificial
- **Investigation group:** TRATAMIENTO DE INCERTIDUMBRE EN SISTEMAS INTELIGENTES



Bibliographic info

- **UGR information:**

Total Cites	Total H-Index	Last 5 years Cites	Last 5 years H-Index
1949	23	898	14
- **Scopus information:**

Number of documents	Cites	H-Index
92	743	13

Links

[UGR PROFILE](#)  [GOOGLE SCHOLAR PROFILE](#)  [SCOPUS PROFILE](#)

CLOSE

Figura 6.10: Aspecto vista de un autor

Por otro lado la vista del buscador de *abstracts* es algo diferente ya que no muestra los resultados en una tabla, si no a modo de lista. Para cada uno de los artículos se puede observar unos datos básicos: su título, autores, fecha de publicación, citas y palabras clave.

The screenshot displays the BiblioIR search interface. At the top, there is a search bar with the text 'BiblioIR' and a search icon. Below the search bar, there are two tabs: 'AUTHOR SEARCH' and 'ABSTRACT SEARCH'. The 'ABSTRACT SEARCH' tab is selected. The main content area shows 891 results found in 2ms. A sorting dropdown menu is set to 'ES default'. The results are listed as follows:

- Teranga Go!: Carpooling Collaborative Consumption Community with multi-criteria hesitant fuzzy linguistic term set opinions to build confidence and trust**
Ana M. Sanchez, Pedro Villar, Rosana Montes, Francisco Herrera
Date: 01/06/2018 Cites: 1
Keywords: Carpooling, Collaborative consumption, Hesitant fuzzy linguistic term set, Linguistic 2-tuples, Multicriteria decision making
- An aggregation approach for solving the non-linear fractional equality Knapsack problem**
Enrique Herrera-Viedma, Anis Yazidi, Tore M. Jonassen
Date: 15/11/2018 Cites: 0
Keywords: Dynamical system, Non-linear fractional equality knapsack, Rate limiting, Resource allocation
- A comparative study on consensus measures in group decision making**
Juan Miguel Tapia, Enrique Herrera-Viedma, Francisco Chiclana, María José del Moral
Date: 01/08/2018 Cites: 5
Keywords: consensus, decision support rules, fuzzy preferences, group decision making, similarity
- Optimization and Reoptimization in Fuzzy Linear Programming problems**
José Luis Verdegay, Behrouz Kheirfam
Date: 01/12/2013 Cites: 4
Keywords: Dual simplex method, Duality theory, Fuzzy linear programming, Trapezoidal fuzzy numbers.

Figura 6.11: Aspecto buscador de artículos

Como cabe esperar también se dispone de una vista detallada de cada artículo, donde se muestran datos más detallados, el *abstract* o en enlace del mismo en Scopus.

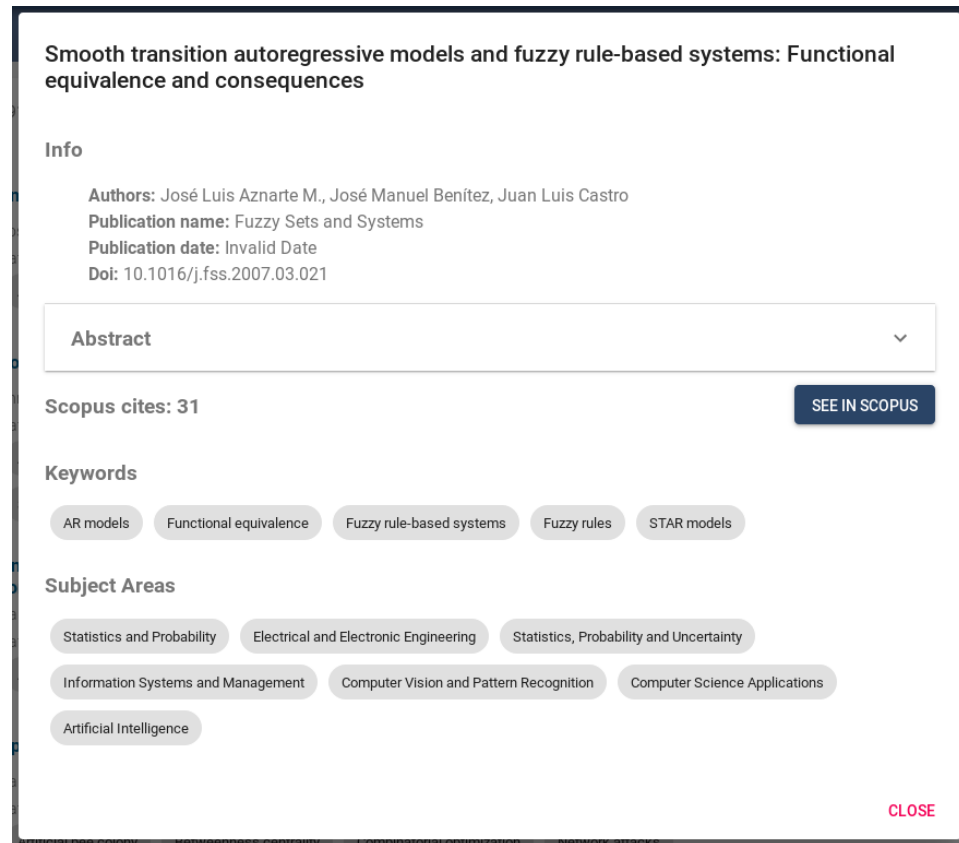


Figura 6.12: Aspecto vista de un artículo

Además de todo este *frontend* he desarrollado un pequeño servidor utilizando *Flask* que permite recuperar los datos completos de un autor o artículo por su id para combinarlos con los que vienen de ES y poder desplegar las vistas detalladas de cada elemento que se han mostrado antes. También cuenta con servicio para obtener un diccionario de artículos por id de Scopus y así poder mostrar una sección de referencias en los artículos que las tengan con enlaces directos a sus referencias.

6.8. Sprint 7

Con toda la infraestructura de búsqueda ya funcionando había llegado la hora de aplicar medidas bibliométricas para mejorar la búsqueda así como

documentar lo hecho hasta ahora. Este objetivo se divide por tanto en:

- Investigar como modificar la puntuación asignada por ES a los documentos.
- Combinar esa puntuación con medidas sobre las citas e índices h.
- Documentar lo desarrollado hasta ahora.

6.8.1. Aplicación de medidas bibliométricas y ordenación de resultados

Elasticsearch provee una puntuación o `_score` en cada documento devuelto por cada consulta. Dicha puntuación está basada en el propio modelo de *Lucene*, el cual se explica brevemente en la documentación oficial de ES [39]. Básicamente supone la combinación de varios modelos clásicos explicados en el apartado Modelos.

Primero se ejecuta un modelo booleano para descartar documentos que de ningún modo pueden formar parte del resultado de la consulta (por no tener ninguna relación en absoluto), tras esto para ordenar por relevancia este subconjunto de documentos se utiliza el modelo vectorial con un esquema de pesos $tf*idf$ modificado con una normalización en función de la longitud del campo (a campo más corto, más relevancia) además de algunas otras modificaciones, como los campos que el usuario haya decidido priorizar.

Todo esto hace que esta puntuación sea muy dependiente de la consulta y no tenga un límite superior, ya que eso significaría que se puede determinar la relevancia absoluta dada una consulta, cosa que no es posible como ya he discutido en los comienzos de esta memoria.

Por suerte en ES esta contemplada la posibilidad de modificar esta puntuación de manera interna usando lo que se conoce como *Function Score Query* [40], simplemente en la propia consulta se determina como se quiere modificar dicha puntuación. Dispone de bastante flexibilidad, permite reemplazarla por completo, combinarla mediante una suma, media o multiplicación con otras medidas como, por ejemplo, una función sobre un campo del documento (como puede ser las citas) o cualquier función calculable con un *script*.

Esta fue mi primera opción con la que estuve probando logrando realizar combinaciones. Logré probar algunos ejemplos prometedores pero no tenía claro como combinar ya que por ejemplo la puntuaciones que ES asignaba en mis pruebas a los documentos solían ser pequeñas (menores que 10) y aplicarle alguna modificación con el campo de las citas (que llega a superar las centenas) hacía que prácticamente diera igual la consulta, los resultados siempre eran un listado ordenado por citas ya que su peso era mucho mayor

que todo el mecanismo de puntuación interno de ES. Por ello me puse a buscar como normalizar las puntuaciones internas entre 0 y 1 para aplicar yo también una normalización a las citas y poder combinar ambas medidas con igual peso cada una. Tras navegar por varias preguntas en *StackOverflow* que aplicaban diversos trucos que no llegaban a funcionar di con una *feature request* en el propio repositorio *GitHub* de ES [41] donde los autores afirmaban que no se trataba de una buena idea y simplemente se limitaban a cerrar.

Esto hizo que me planteara un enfoque más manual, si no podía combinar a mi gusto las puntuaciones en ES podría obtener las puntuaciones de los documentos en el *frontend*, normalizar por la máxima puntuación para la consulta y tras esto combinar con mis medidas.

Para llevar a cabo este enfoque lo primero que realicé fue normalizar los números de citas e índices *h* para introducirlos como nuevos campos precalculados en los índices ya creados, por ello creé los campos `cites_norm` y `hindex_norm` en el índice de autores y los homólogos en el de artículos, con la excepción de que el índice *h* de un artículo no es algo real, ya que esta medida se aplica a los autores, pero para poder aplicarlo realicé una normalización de la media de los índices *h* de los autores del artículo.

Teniendo ya ambas medidas normalizadas tenía que establecer el método de combinación, siguiendo el consejo de mi tutor implementé dos combinaciones que han sido ampliamente utilizadas y estudiadas para este tipo de problemas: los algoritmos **CombMAX** y **CombSUM** [42]. El primero es básicamente seleccionar el valor máximo en *N* rankings de puntuación normalizados para cada uno de los documentos mientras que el segundo supone la suma de las *N* puntuaciones normalizadas para cada documento.

Estas son todas las piezas para la aplicación de medidas bibliométricas en mi sistema, el mecanismo por tanto es el siguiente:

Cada consulta *q* devuelve un conjunto de documentos ordenados por el mecanismo interno de ES $\{d_1, d_2, \dots, d_n\}$, cada uno de los cuales cuenta con una puntuación *s*, por lo que lo primero que hago tras recibir el resultado es buscar la puntuación máxima de entre todos los documentos *maxScore* y normalizar las puntuaciones de la manera siguiente:

$$s_{ni} = \frac{s_i}{maxScore} \forall i \in \{1, 2, \dots, n\} \quad (6.1)$$

Tras esto según el mecanismo de ordenación elegido entre: **CombMAX_cites**, **CombSUM_cites**, **CombMAX_hindex** y **CombSUM_hindex** se suman o calculan los máximos entre las puntuaciones ES normalizadas y las puntuaciones de citas o índice *H* asignando finalmente una puntuación *s'* a cada documento por la cual se reordena la lista de documentos.

Esto se aplica tanto para la búsqueda de autores como de artículos. Además de este mecanismo también se puede seleccionar la ordenación de ES (sin medidas bibliométricas ninguna) y ordenación únicamente por citas, índice h, nombre (en caso de autores) o título (en caso de artículos). La siguiente tabla recoge las ordenaciones de cada uno de los buscadores:

Autores	Artículos
ES Default	
Nombre	Título
Citas	
Índice H	
CombMAX Citas	
CombSUM Citas	
CombMAX Índice H	
CombSUM Índice H	

Cuadro 6.3: Resumen de las ordenaciones disponibles en cada módulo de búsqueda

6.9. *Sprint 8*

Esta última iteración del proyecto tuvo como objetivo refinar lo hecho hasta ahora. Esto se reduce a:

- Finalizar y optimizar la implementación de las ordenaciones
- Mejorar el despliegue del sistema para facilitarlo
- Ultimar la memoria del proyecto

Para ultimar el sistema desarrollado, estuve mejorando el mecanismo de selección de ordenaciones usando un selector propio, creado con *MaterialUI*, que me permitía un control más fino que los ya integrados en *Searchkit*.

Probando estas ordenaciones, me percaté de que la búsqueda dinámica según se escribe en el cuadro de búsqueda era demasiado pesada en el caso de los artículos. Dicha funcionalidad se sigue manteniendo para los autores, ya que son menos autores y el tamaño de un autor es bastante menor que el de un artículo. En el caso de la búsqueda de artículos deshabilité esa posibilidad, teniendo que pulsar **Enter** o cambiando el foco del cuadro de búsqueda para que se lance la petición contra ES.

Como ya he comentado, para el desarrollo del proyecto, he utilizado contenedores *Docker*, esto permite crear una infraestructura replicable de

forma sencilla. Pero tanto el *frontend* como el *backend* disponen de modos de depuración y desarrollo capaces, por ejemplo, de detectar cambios en el código de forma dinámica, recargando los servicios y permitiendo observar rápidamente los cambios. Estas funcionalidades resultan muy útiles durante el proceso de desarrollo, pero añaden una sobrecarga innecesaria para el uso real del sistema. Por ello se recomiendan desactivar las opciones de depuración y desarrollo antes de pasar cualquier sistema a producción.

Para conseguir ese objetivo creé un nuevo contenedor que producía y minificaba el código del *frontend* para producción. El mismo se encargaba de servir el *backend* utilizando un servidor de aplicación *uWSGI*. Para más detalles sobre la infraestructura final ver el siguiente apartado así como el anexo que explica el proceso de instalación y despliegue del sistema.

Por último, se finalizaron algunos apartados de esta memoria, como los resúmenes, conclusiones o anexos; así como la aplicación de correcciones por parte del tutor.

6.10. Arquitectura final del sistema

La arquitectura final del sistema es la de una aplicación web distribuida, algo diferente y más compleja que la propuesta inicialmente en 5.3:

- Un **cliente web** que se ejecuta en el navegador y constituye la interfaz de usuario, este se encarga de recoger las consultas, enviarlas al servidor de *Elasticsearch* (ES) y mostrar los resultados de las mismas sirviéndose de tecnologías como *Searchkit*, *ReactJS* o *MaterialUI*.
- Un **servidor Elastic** donde se ejecuta ES y el cual contiene los índices de autores y *abstracts* así como los datos indexados para la búsqueda (que no son todos los listados en el modelo de datos previamente).
- Un **servidor de monitorización** del servidor Elastic utilizando el proyecto de software libre Cerebro. Este permite monitorizar en tiempo real diversas medidas del servidor como la carga, espacio utilizado o estado además de múltiples funciones como un cliente REST que permite la consulta a bajo nivel sobre ES muy utilizado para depuración o durante el desarrollo para hacer pequeñas pruebas.
- Un **servidor de aplicación** que utiliza *uWSGI*, *Nginx* y *Flask* para montar algunos microservicios auxiliares que permiten recuperar el resto del modelo de datos que no se encuentra indexado así como aportar algunas estructuras de datos precalculadas para facilitar la búsqueda. También es el encargado de servir el código *frontend* al cliente (donde se ejecuta).

- Un **servidor mongo** que lleva una BD *MongoDB* la cual contiene la totalidad de los datos de ambas colecciones y sirve como fuente de datos para el servidor de aplicación.

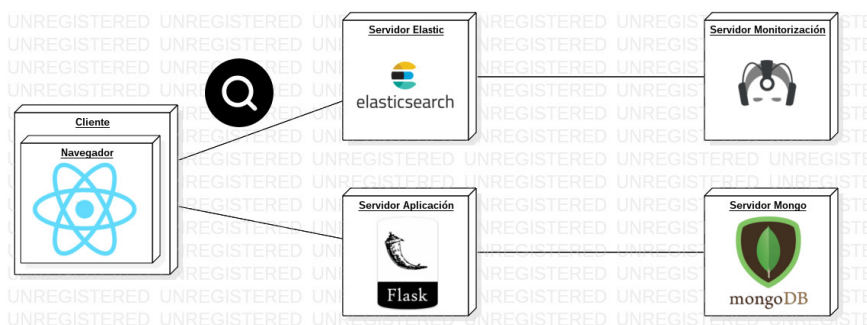


Figura 6.13: Diagrama de arquitectura final con las principales tecnologías usadas en cada nodo

Capítulo 7

Conclusiones y Trabajos Futuros

7.1. Relación del TFM con lo aprendido en el Máster

Para comenzar el último capítulo de la presente memoria destacaré las principales asignaturas del Máster que me han ayudado a llevar a cabo este proyecto.

- **Gestión de Información en la Web:** Como ya he ido comentado desde la introducción de esta memoria, esta asignatura fue mi principal motivación para elegir este proyecto. Ya que es donde adquirí las primeras nociones de lo que es un sistema de RI así como donde lleve a cabo alguna práctica construyendo uno por mi mismo. Esta asignatura me resultó la optativa más interesante del Máster ya que enseña los fundamentos de sistemas de información que son totalmente desconocidos para los estudiantes que no provienen de esa rama de la informática como yo.
- **Sistemas Software Basados en Web:** En ella se imparten los fundamentos de las aplicaciones web, utilizando herramientas extensamente empleadas a la par que punteras. Para mí supuso mis primeros acercamientos serios a *Python* o *JavaScript* y sus bases me resultan útiles a diario en mi trabajo. En este proyecto he aplicado los conocimientos adquiridos sobre arquitectura de aplicaciones web, gestión de infraestructuras con *Docker* o *ReactJS*.
- **Cloud Computing: Fundamentos e Infraestructuras:** A pesar de que en principio es la asignatura menos relacionada con la temática y desarrollo del proyecto de las tres. Durante sus clases he aprendido

realmente a utilizar *GitHub* y los sistemas de control de versiones, una habilidad realmente útil. También me enseñó los conceptos detrás de *Docker* así como otros sistemas de gestión y despliegue de infraestructuras.

7.2. Conclusiones

A título personal me ha resultado un proyecto muy interesante, que considero me ha enseñado bastante. Me siento satisfecho con el mismo, aunque me haya costado más tiempo y esfuerzo de lo esperado, mi curiosidad inicial sobre los buscadores ha quedado ampliamente satisfecha.

Aunque siempre se puede seguir aprendiendo tengo asimilados los fundamentos del funcionamiento de un sistema de recuperación de información y me considero capacitado para crear algún sistema de búsqueda en el futuro.

En mis pruebas con los distintos mecanismos de ordenaciones, me he encontrado que el algoritmo de combinación de *rankings* CombMAX no modifica demasiado los resultados, ya que depende mucho de que los datos se encuentren en la misma escala para poder combinarlos adecuadamente. A pesar de haber normalizado las puntuaciones dadas por ES, los números de citas e índice *h* para conseguirlo, la normalización de citas e índice *h* se ha hecho sobre la distribución total de estos valores en toda la colección, mientras que la normalización de la puntuación de ES no puede hacerse de forma absoluta. Por un lado porque dicha puntuación mide la relevancia y este no es un concepto absoluto, si no subjetivo; y por otro porque es dependiente de la consulta, por ello se normaliza la puntuación de forma dinámica.

Esto implica que la puntuación máxima de ES dada para una consulta se asigna al valor 1 y a partir de ahí se normaliza el resto de puntuaciones, pero si entre los documentos retornados no se encuentran algunos con un número significativo de citas o índice *h* sobre el total (como suele ser el caso), la comparación resulta totalmente desigual y la puntuación de ES resulta superior casi siempre por lo que no varía mucho el orden sobre la ordenación por defecto.

Este problema se hace más sutil en el algoritmo CombSUM, ya que simplemente se suman ambas puntuaciones, por lo que el número de citas o índice *h* puede variar la ordenación de forma más fácil. Como puede parecer lógico, lo que más se observan son reordenaciones locales, es decir, intercambios entre documentos consecutivos o muy cercanos en la ordenación por defecto, no grandes diferencias.

Como siempre suele pasar en estos proyectos, me he dejado bastantes cosas en el tintero que me gustaría haber probado. Si se analiza la funcionalidad del sistema definitivo se aprecia que no podido implementar un sistema

de ordenación *a posteriori* utilizando la información sobre citas recolectada. En parte por falta de tiempo, pero también por los datos disponibles sobre las citas, a pesar de contar con 742 citas en 891 artículos, estas citas se encuentran muy concentradas. Solo 370 artículos tienen alguna cita sobre alguno de esos 891 documentos ($\sim 41,53\%$ del total), este porcentaje resulta bastante bajo para utilizar esta información en alguna ordenación ya que de más de la mitad de artículos no se dispone de información alguna.

7.3. Trabajos futuros

La principal línea de trabajo que quedaría por seguir en este proyecto es la de **evaluar la mejora producida al aplicar medidas bibliométricas al sistema**. El hecho de que la colección documental del sistema esté constituida por autores de la Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación favorece que esta evaluación se pueda llevar a cabo, aunque esta no es una tarea sencilla.

La evaluación de sistemas RI se suele llevar a cabo mediante colecciones de prueba, donde existe un conjunto de documentos, un conjunto de consultas y un conjunto de resultados que deberían de retornar las mismas [43]. Al crear un sistema nuevo con una colección de documentos no utilizada previamente podemos comparar los resultados obtenidos con los que se esperarían, este problema se debe a la relatividad del concepto de relevancia. Por ello las colecciones de pruebas suelen incluir valoraciones de relevancia de expertos en la materia.

La forma más sencilla de llevar a cabo una evaluación sería realizar pruebas con el sistema y usuarios expertos como los propios profesores de la Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación. Realizando para ello algunas consultas y que ellos mismos comprueben las diferencias entre las distintas ordenaciones, finalmente mediante algún tipo de encuesta se podría recoger su opinión sobre la relevancia de las medidas en la recuperación llevada a cabo.

Otra posible línea sería la de incorporar **redes de citación** para realizar *relevance feedback* [44] utilizando las referencias, para ello, por los motivos comentados en el apartado previo, habría que incluir todas las referencias de los artículos en lugar de las propias internas. Esto aumentará significativamente en número de citas aunque muchas apunten a artículos externos, ese tipo de relación de citación conocida como co-citación es más débil que la citación directa pero también puede aportar información relevante, como se ha mencionado en algunos trabajos del apartado Trabajos relacionados.

Hay que tener en cuenta que este supone un procesamiento costoso al tener que operar sobre grandes grafos de artículos y citas como nodos y

aristas respectivamente. Especialmente en el contexto de la búsqueda donde el usuario busca la immediatez que le ofrecen gigantes como Google. No merecería la pena montar un esquema complejo que de resultados buenos pero tarde varios segundos en dar una respuesta. Por ello para explorar estos enfoques, habría que contar con un equipo más potente y adaptado a las necesidades de computo que el utilizado, así como, con alta probabilidad, utilizar estrategias de optimización *greedy* [45].

Bibliografía

- [1] J. M. F. Luna, “Apuntes de la asignatura Gestión de Información en la Web del Máster en Ingeniería Informática.”
- [2] J. F. H. G. Fidel Cacheda Seijo, Juan Manuel Fernández Luna, *Recuperación de Información: Un enfoque práctico y multidisciplinar*. 1ª ed.
- [3] L. Page, S. Brin, R. Motwani, and T. Winograd, “The pagerank citation ranking: Bringing order to the web.,” Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.
- [4] statcounter, “Search engine market share worldwide.” <http://gs.statcounter.com/search-engine-market-share#monthly-201705-201805-bar>. [Online; accedido 23-Junio-2018].
- [5] N. De Bellis, *Bibliometrics and Citation Analysis: From the Science Citation Index to Cybermetrics*. Scarecrow Press, 2009.
- [6] L. Leydesdorff and S. Milojevic, “Scientometrics,” *CoRR*, vol. abs/1208.4566, 2012.
- [7] L. Byl, J. Carson, A. Feltracco, S. Gooch, S. Gordon, T. Kenyon, B. Muirhead, D. Seskar-Hencic, K. MacDonald, M. T. Özsu, and P. Stirling, “White paper on bibliometrics, measuring research outputs through bibliometrics,” tech. rep., University of Waterloo, 01 2016.
- [8] University of Leeds, “What are bibliometrics?.” https://library.leeds.ac.uk/info/1406/researcher_support/17/measuring_research_impact/1. [Online; accedido 23-Junio-2018].
- [9] B. Gipp and J. Beel, “Citation Proximity Analysis (CPA) - A New Approach for Identifying Related Work Based on Co - Citation Analysis,” in *Proceedings of the 12th International Conference on Scientometrics and Informetrics (ISSI'09)* (B. Larsen and J. Leta, eds.), vol. 2, (Rio de Janeiro, Brazil), International Society for Scientometrics and Informetrics, Jul. 2009. ISSN 2175-1935.

- [10] Z. Dangzhi and S. Andreas, “Evolution of research activities and intellectual influences in information science 1996–2005: Introducing author bibliographic-coupling analysis,” *Journal of the American Society for Information Science and Technology*, vol. 59, no. 13, pp. 2070–2086.
- [11] University of Leeds, “What are altmetrics?.” https://library.leeds.ac.uk/info/1406/researcher_support/17/measuring_research_impact/3. [Online; accedido 23-Junio-2018].
- [12] J. Beall, “Google scholar is filled with junk science.” <https://web.archive.org/web/20141107175135/http://scholarlyoa.com/2014/11/04/google-scholar-is-filled-with-junk-science/>. [Online; accedido 23-Junio-2018].
- [13] J. Beel and B. Gipp, “Google Scholar’s Ranking Algorithm: An Introductory Overview,” in *Proceedings of the 12th International Conference on Scientometrics and Informetrics (ISSI’09)* (B. Larsen and J. Leta, eds.), vol. 1, (Rio de Janeiro, Brazil), International Society for Scientometrics and Informetrics, Jul. 2009. ISSN 2175-1935.
- [14] C. Analytics, “Web of science: It’s time to get the facts.” https://cdn.clarivate.com/wp-content/uploads/2017/05/d6b7faae-3cc2-4186-8985-a6ecc8cce1ee_Crv_WoS_Upsell_Factbook_A4_FA_LR_edits.pdf. [Online; accedido 23-Junio-2018].
- [15] ELSEVIER, “How scopus works - content.” <https://www.elsevier.com/solutions/scopus/how-scopus-works/content>. [Online; accedido 23-Junio-2018].
- [16] “Editorial,” in *Proceedings of the First Workshop on Bibliometric-enhanced Information Retrieval co-located with 36th European Conference on Information Retrieval (ECIR 2014)*, Amsterdam, The Netherlands, April 13, 2014., pp. 1–4, 2014.
- [17] S. Siebert, S. Dinesh, and S. Feyer, “Extending a research-paper recommendation system with bibliometric measures,” in *Proceedings of the Fifth Workshop on Bibliometric-enhanced Information Retrieval (BIR) co-located with the 39th European Conference on Information Retrieval (ECIR 2017)*, Aberdeen, UK, April 9th, 2017., pp. 112–121, 2017.
- [18] H. Zhao and X. Hu, “Language model document priors based on citation and co-citation analysis,” in *Proceedings of the First Workshop on Bibliometric-enhanced Information Retrieval co-located with 36th European Conference on Information Retrieval (ECIR 2014)*, Amsterdam, The Netherlands, April 13, 2014., pp. 29–36, 2014.

- [19] H. D. White, “Bag of works retrieval: Tf*idf weighting of co-cited works,” in *Proceedings of the Third Workshop on Bibliometric-enhanced Information Retrieval co-located with the 38th European Conference on Information Retrieval (ECIR 2016)*, Padova, Italy, March 20, 2016., pp. 63–72, 2016.
- [20] M. J. Sarol, L. Liu, and J. Schneider, “Testing a citation and text-based framework for retrieving publications for literature reviews,” in *Proceedings of the 7th International Workshop on Bibliometric-enhanced Information Retrieval (BIR 2018) co-located with the 40th European Conference on Information Retrieval (ECIR 2018)*, Grenoble, France, March 26, 2018., pp. 22–33, 2018.
- [21] UGR, “Rankings de investigadores ugrinvestiga según citación.” http://investigacion.ugr.es/ugrinvestiga/static/BuscadorRanking/*/buscar?tipo=&rama_c=&disciplina_c=TELE_D&especialidad_c=&indicador=&periodo=. [Online; accedido 3-Septiembre-2018].
- [22] H. Else, “How i scraped data from google scholar.”
- [23] Slant, “Solr vs elasticsearch.” https://www.slant.co/versus/355/358/~solr_vs_elasticsearch. [Online; accedido 3-Septiembre-2018].
- [24] S. Opel, “Solr vs elasticsearch - stack overflow.” <https://stackoverflow.com/a/10213568>. [Online; accedido 3-Septiembre-2018].
- [25] J. Abrahamsson, “Elasticsearch 101 – a getting started tutorial.” <http://joelabrahamsson.com/elasticsearch-101/>. [Online; accedido 3-Septiembre-2018].
- [26] M. J. S. (@janinaj), “lit-review-search.” <https://github.com/janinaj/lit-review-search>. [Online; accedido 4-Septiembre-2018].
- [27] E. Developers, “elsapy: A python module for use with elsevier’s apis: Scopus, sciencedirect, others.” <https://github.com/ElsevierDev/elsapy>. [Online; accedido 4-Septiembre-2018].
- [28] scopus api, “scopus: Python-based api-wrapper to access scopus.” <https://scopus.readthedocs.io/en/latest/>. [Online; accedido 4-Septiembre-2018].
- [29] CSiRC, “Configuración del servicio vpn ssl.” <https://csirc.ugr.es/informatica/RedUGR/VPN/ConfVPNSSL/>. [Online; accedido 4-Septiembre-2018].

- [30] O. UGR, “7ª edición de los rankings de investigadores ugrinvestiga según citación del vicerrectorado de investigación y transferencia tras la revisión realizada por la comunidad investigadora de la ugr.” <http://opendata.ugr.es/dataset/rankings-de-investigadores-ugrinvestiga-segun-citacion/resource/4937db08-d757-469d-9bee-96fee9f798f8>. [Online; accedido 4-Septiembre-2018].
- [31] E. Developers, “Abstract retrieval views.” <https://dev.elsevier.com/guides/AbstractRetrievalViews.htm>. [Online; accedido 5-Septiembre-2018].
- [32] S. C. (@sarahleejane), “creating an elasticsearch index with python.” <https://sarahleejane.github.io/learning/python/2015/10/14/creating-an-elastic-search-index-with-python.html>. [Online; accedido 5-Septiembre-2018].
- [33] Elastic, “elasticsearch-py: Official python low-level client for elasticsearch.” <https://github.com/elastic/elasticsearch-py>. [Online; accedido 5-Septiembre-2018].
- [34] Elastic, “Mapping - elasticsearch reference.” <https://www.elastic.co/guide/en/elasticsearch/reference/6.3/mapping.html>. [Online; accedido 5-Septiembre-2018].
- [35] T. Eleven, “Searchkit: Ui components for elasticsearch.” <http://www.searchkit.co/>. [Online; accedido 5-Septiembre-2018].
- [36] appbase.io, “Reactivesearch: React ui components for elasticsearch.” <https://opensource.appbase.io/reactivesearch/>. [Online; accedido 5-Septiembre-2018].
- [37] Google, “Material design.” <https://material.io/design/>. [Online; accedido 6-Septiembre-2018].
- [38] T. Eleven, “Searchkit: Configuring elasticsearch.” <http://docs.searchkit.co/stable/setup/elasticsearch.html#using-local-es-server>. [Online; accedido 6-Septiembre-2018].
- [39] Elastic, “Elasticsearch: Theory behind relevance scoring.” <https://www.elastic.co/guide/en/elasticsearch/guide/current/scoring-theory.html>. [Online; accedido 6-Septiembre-2018].
- [40] Elastic, “Elasticsearch: Function score query.” <https://www.elastic.co/guide/en/elasticsearch/reference/6.3/query-dsl-function-score-query.html#>

- query-dsl-function-score-query. [Online; accedido 6-Septiembre-2018].
- [41] Elastic, “Elasticsearch: Add the ability to normalize query scores.” <https://github.com/elastic/elasticsearch/issues/23834>. [Online; accedido 6-Septiembre-2018].
- [42] J. A. Shaw and E. A. Fox, “Combination of multiple searches,” in *Proceedings of The Third Text REtrieval Conference, TREC 1994, Gaithersburg, Maryland, USA, November 2-4, 1994*, pp. 105–108, 1994.
- [43] P. R. Christopher D. Manning and H. Schütze, “Information retrieval system evaluation.” <https://nlp.stanford.edu/IR-book/html/htmledition/information-retrieval-system-evaluation-1.html>.
- [44] Wikipedia, “Relevance feedback.” https://en.wikipedia.org/wiki/Relevance_feedback#cite_note-2. [Online; accedido 12-Septiembre-2018].
- [45] P. E. Black, “greedy algorithm,” in *Dictionary of Algorithms and Data Structures [online]*, Vreda Pieterse and Paul E. Black, eds. 2 February 2005.
- [46] Wikipedia, “Front-end y back-end.” https://es.wikipedia.org/wiki/Front-end_y_back-end. [Online; accedido 31-Agosto-2018].
- [47] Wikipedia, “Software framework.” https://en.wikipedia.org/wiki/Software_framework. [Online; accedido 31-Agosto-2018].
- [48] M. G. Software, “Scrum.” <http://www.mountaingoatsoftware.com/agile/scrum>. [Online; accedido 1-Septiembre-2018].
- [49] Wikipedia, “Web scraping.” https://en.wikipedia.org/wiki/Web_scraping. [Online; accedido 3-Septiembre-2018].
- [50] Elastic, “Install elasticsearch with docker.” <https://www.elastic.co/guide/en/elasticsearch/reference/current/docker.html>. [Online; accedido 10-Septiembre-2018].
- [51] Docker, “About docker ce: Supported platforms.” <https://docs.docker.com/install/#supported-platforms>. [Online; accedido 10-Septiembre-2018].
- [52] Docker, “Install docker compose.” <https://docs.docker.com/compose/install/>. [Online; accedido 10-Septiembre-2018].

Anexos

Apéndice A

Glosario

Scrum Metodología ágil de desarrollo de software basa en la descomposición del proyecto en sprints en los que definen los objetivos en lugar de como hacer cada paso del proyecto. Existen tres roles en un proyecto Scrum:

- Equipo de desarrollo: formado por varios integrantes habitualmente
- ScrumMaster: parte del equipo de desarrollo pero con un rol especial que puede ser entendido como el "entrenador" del equipo
- Product owner: representa el cliente o los usuarios finales que son los que finalmente usaran el software desarrollado.

Cada uno de los sprint cuentan con un conjunto de tareas definidas a cumplir conocidas como sprint backlog y se suele hacer uso de tableros para seguir el progreso de cada una de las mismas. [48]. 7

Teorema de Bayes Teorema perteneciente a la teoria de probabilidades que permite calcular la probabilidad condicional de dos eventos A y B en base su probabilidad condicional inversa y su probabilidad marginal

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Donde $P(A|B)$ es la probabilidad condicional de que dado un evento B se produzca A , $P(A)$ la probabilidad marginal o incondicional de que suceda A , $P(B|A)$ la probabilidad condicional de que dado A suceda B y $P(B)$ la probabilidad marginal de B . 14

términos de indexación concepto asociado a una serie de palabras donde el concepto es definido o discutido. 15

sistema de recomendación Tipo de sistema de RI en el que el usuario no expresa directamente su necesidad de información, si no que se le muestran items "similares" a los que ya ha consultado. 20

framework Abstracción que provee un entorno reutilizable y genérico que puede ser utilizado para facilitar el desarrollo software [47]. 21, 32, 33, 37, 47, 49

web scrapping Extracción de datos de una página web de forma automática manejando directamente los ficheros HTML de las páginas web [49]. 27, 33, 39

backend Parte de un sistema informático encargada del procesamiento o almacenamiento de información [46]. 31, 33, 34, 45, 55, 56, 75, 78

frontend Parte de un sistema informático encargada de la interacción con el usuario [46]. 31, 32, 34, 52, 54, 55, 56, 75, 78

Apéndice B

Lista de Acrónimos

API *Application Programming Interface*. III, 27, 29, 33, 37, 38, 44, 45

BD Base de Datos. 30, 32, 44, 45, 56

BIR *Bibliometric-enhanced Information Retrieval*. 19, 34, 35, 36

DBaaS *DataBase as a Service*. 47

E.T.S.I.I.T. Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación. 27

ES *Elasticsearch*. 5, 33, 37, 45, 47, 48, 49, 52, 53, 54, 55, 56, 60, 75, 78

FECYT Fundación Española para la Ciencia y la Tecnología. 19

GIW Gestión de Información en la Web. 1, 34, 37, 59

HTTP *HyperText Transfer Protocol*. 37, 78

Mr.DLib *Machine-readable Digital Library*. 20

PDF *Portable Document Format*. 1, 12

REST *Representational State Transfer*. 37

RI Recuperación de Información. I, 1, 2, 11, 12, 13, 14, 15, 16, 18, 19, 20, 23, 27, 30, 34, 35, 36, 37, 59, 61, 71

SO Sistema Operativo. 31

TFG Trabajo de Fin de Grado. 7

TFM Trabajo de Fin de Máster. II, 5, 6, 23, 36, 59

UGR Universidad de Granada. 19, 38, 40, 41, 42, 43, 44, 45

URL *Universal Resource Locator*. 46

VPN *Virtual Private Network*. 38

WoS Web of Science. 19, 27, 37

XML *Extensible Markup Language*. 37, 45

Apéndice C

Manual técnico de uso

C.1. Descripción de la arquitectura

El sistema desarrollado a lo largo de este proyecto se ha servido de *Docker* como forma de gestionar la infraestructura de los diversos nodos que lo conforman. La arquitectura final del sistema como se recoge en la figura 6.13 se constituye de 4 nodos o contenedores *Docker* concretamente.

- **app**: contenedor que contiene una imagen con un servidor *Nginx* encargado de servir el *frontend React* y el cual se comunica con servidor de aplicación *uWSGI* mediante un *socket UNIX*. Dicho servidor de aplicación sirve la aplicación *Flask* que constituye el *backend* del sistema.
- **elasticsearch**: contenedor en el cual se encuentra alojada una imagen de *Elasticsearch* en su versión *Open Source* siguiendo las instrucciones oficiales [50].
- **mongodb**: contenedor con la imagen de *MongoDB* utilizada
- **cerebro**: contenedor con el servicio de monitorización de ES *Cerebro*. Este contenedor no es necesario para el funcionamiento del sistema pero ha resultado muy útil durante el desarrollo para gestionar el contenedor de ES

C.2. `docker-compose.yml`

En el siguiente listing se observa la configuración del fichero `docker-compose.yml` que determina la infraestructura descrita arriba.

```
1 version: '2.2'
2 services:
3   app:
4     container_name: app
5     build: ./src
6     restart: always
7     ports:
8       - 80:80
9     networks:
10      default:
11        aliases:
12          - app
13   elasticsearch:
14     image: docker.elastic.co/elasticsearch/elasticsearch-oss:6.3.0
15     container_name: elasticsearch
16     environment:
17       - cluster.name=docker-cluster
18       - ES_JAVA_OPTS=-Xms512m -Xmx512m
19       - discovery.type=single-node
20     ulimits:
21       memlock:
22         soft: -1
23         hard: -1
24     volumes:
25       - ./data/es_data:/usr/share/elasticsearch/data
26       - ./config/elasticsearch.yml:/usr/share/elasticsearch/config/elasticsearch.yml
27     ports:
28       - 9200:9200
29       - 9300:9300
30     networks:
31      default:
32        aliases:
33          - elasticsearch
34   mongodb:
35     image: mongo:3.4
36     container_name: mongodb
37
38     ports:
39       - 27017:27017
40     volumes:
41       - ./data/mongodb_data:/data/db
42     command: mongod
43     networks:
44      default:
45        aliases:
46          - mongodb
47
48   cerebro:
49     image: lmenezes/cerebro
50     container_name: cerebro
51
```

```
52     ports:
53       - 9000:9000
54
55     networks:
56       default:
57         aliases:
58           - cerebro
```

Listing C.1: Fichero `docker-compose.yml` de la aplicación

C.3. Instalación

C.3.1. Docker y Docker Compose

Para poder desplegar la aplicación por tanto lo primero sera instalar *Docker* y *Docker Compose*. Para instalar *Docker* se ha utilizado su versión *Community Edition (CE)* por ello ver las instrucciones de instalación para el sistema en el que se desee instalar [51]. Respecto a *Docker Compose* recomiendo instalar la versión nativa para el sistema de destino en lugar de utilizar las opciones alternativas de instalación, ver la documentación oficial en [52].

C.3.2. Descripción del repositorio

Una vez instalado esto será necesario *clonar* el repositorio de *GitHub*, que recuerdo era <https://github.com/Aythae/TFM>, cuya estructura se observa a continuación. Para disponer de los datos de *MongoDB* y *Elasticsearch* será necesario descomprimir el fichero `data.7z`. Esto creará una carpeta `data`.

```

TFM
├── data/
│   ├── es_data/
│   └── mongodb_data/
├── config/
│   └── elasticsearch.yml
├── src/
│   ├── backend/...Código backend y fichero requirements.txt con las
│   │   dependencias de Python
│   ├── frontend/.....Código frontend y fichero package.json con las
│   │   dependencias de JS
│   ├── Exploring/
│   │   ├── elasticsearch/ ..... Código de indexación en ES
│   │   ├── Exploratory_analysis/ .. Ficheros y gráficos generados en el
│   │   │   análisis de datos
│   │   ├── investigadores/ .... Código extracción de datos del ranking
│   │   │   UGRinvestiga
│   │   └── scopus/ .....Código extracción de datos de Scopus
│   └── Dockerfile.....Fichero que dertermina la construcción del
│       contenedor app
└── docker-compose.yml

```

C.4. Despliegue

Con todo esto ya estamos listos para desplegar la aplicación. Para ello basta con ejecutar el siguiente comando desde el directorio TFM:

```
$ docker-compose -f "docker-compose.yml" up --build
```

Listing C.2: Comando para desplegar el sistema

Una vez finalice la construcción del contenedor **app** se mostrarán unos mensajes indicando que todos los contenedores han arrancado correctamente, tras esto podremos acceder a `http://localhost` donde se encontrará desplegada la aplicación.

En caso de algún problema en el despliegue esto seguramente se deba a los puertos disponibles, el sistema utiliza los puertos 80 (acceso HTTP

frontend), 27017 (*MongoDB*), 9000 (*Cerebro*), 9200 y 9300 (*Elasticsearch*). Por lo que si alguno de esos puertos se encuentra en uso el despliegue fallará. Dicho mapeo de puertos se puede cambiar fácilmente modificando los **ports** en el fichero **docker-compose.yml**.

Para parar la aplicación basta con aplicar un **Ctrl^C** sobre la consola donde se esté ejecutando el comando de despliegue.

