Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN COMPUTACIÓN

Desarrollo de algoritmos de pooling basados en re-ranking de contenidos por relevancia

Estudante: Jessica Penela Fernández
Dirección: Álvaro Barreiro García
Javier Parapar López

A Coruña, febrero, 2020.

A los que sonríen cuando solo tienen ganas de llorar

Agradecimientos

Me gustaría comenzar dando las gracias a mis directores de proyecto, tanto Álvaro como Javier, por la confianza depositada en mi, los consejos, las caras de seriedad cuando era necesario y las sonrisas cuando necesitaba un respiro, pero sobretodo, por el tiempo dedicado.

Quiero dar las gracias a mi familia. Empezando por mi madre, Mónica, que también ha hecho de padre y que se ha esforzado muchísimo para que yo tenga un futuro, sin ella no podría estar donde estoy. A mis abuelos, Elda y Carlos, también fueron papá y mamá cuando fue necesario y por ello forman parte de mis mejores recuerdos. A mi tío Carlos, ambos somos muy parecidos y las decisiones más importantes sobre mi futuro han sido gracias a sus consejos. A mi padrino, Pablo, la alegría de mi casa, una persona con carácter pero con un corazón inmenso. A mis tías Montse y Tania, totalmente diferentes, totalmente necesarias. Y por último, pero no menos importante a mis primos, Carlitos y Alejandra, ojalá la vida os depare cosas preciosas y yo pueda estar a vuestro lado en ellas.

Gracias también a Abel, no ha llegado hace mucho a mi vida, pero ha llegado para quedarse, me ha proporcionado la calma que no tenía y me ha ayudado mucho en esta recta final.

No quiero terminar sin dar las gracias a todos los amigos que he hecho estos años. Quiero hacer una mención especial a Laura por ser no sólo una compañera de aventuras sino que también mi compañera de piso. También a Borja, por celebrar conmigo los aprobados inesperados y por sufrir las prácticas a mi lado. Me queda mucha gente por nombrar, como Esther, Andrés, Eva, Bran, Ruth, Alejandro, Salva, Pablo e Iván los cuáles me han ayudado en múltiples factores de mi vida.

A todos y cada uno de vosotros, ***muchas gracias****.*

Resumen

En la actualidad, los Sistemas de Recuperación de Información son cada vez más importantes. Dichos sistemas proporcionan a un usuario información en base a una consulta previamente enviada, esas consultas pueden ser muy variadas, como por ejemplo el tiempo que va hacer mañana, la hora a la que emiten una película en el cine, la definición de una palabra, etc. La cantidad de información existente es extremadamente extensa y por lo tanto es muy importante que dichos sistemas sean lo más eficaces y eficientes posibles. Considerando la tarea central, encontrar información en un conjunto de documentos, la respuesta de un sistema de recuperación será una ordenación de los mismos, con el objetivo principal de proporcionar al usuario los documentos más relevantes para una consulta en las primeras posiciones del ranking.

Tradicionalmente, para evaluar los sistemas de recuperación se han usado benchmarks, formados por conjuntos de documentos, consultas y juicios de relevancia. Las primeras colecciones a evaluar tenían un tamaño pequeño y por lo tanto los asesores podían llevar a cabo la lectura de todos los documentos, lo cual hacía viable evaluar toda la colección. A partir de TREC(*Text Retrieval Conference*), las colecciones empezaron a crecer notablemente y por tanto la tarea de los asesores se complicó, volviendo inviable llevar a cabo una evaluación de toda la colección. Para solucionar este problema surge una técnica denominada pooling, la cual consiste en crear un pool con el top k de documentos pertenecientes a todos los sistemas participantes en una competición, dichos sistemas han ejecutado todos las mismas queries sobre los mismos documentos. Una vez creado el pool se presenta a los asesores en un orden arbitrario y los evalúan para determinar si son relevantes o no para la consulta enviada. TREC siempre ha usado la estrategia aquí mencionada, pero algunos investigadores han implementado otros algoritmos, los cuales no presentan los documentos en un orden arbitrario sino que lo hacen en un orden determinado, como por ejemplo MTF(*Move to Front*), los cuales no utilizan el contenido de los documentos, sino que el orden de los documentos en cada ranking.

A partir de esta idea es donde surge el proyecto que se presenta a continuación, donde vamos a crear un nuevo algoritmo de pooling, el cual utiliza el contenido de los documentos. El procedimiento a seguir es el siguiente: cada vez que un asesor marca un documento como relevante para una consulta, se produce un re-ranking de los documentos utilizando un modelo de relevancia y se presenta los documentos al asesor con el nuevo orden generado.

Abstract

These days, Information Retrieval Systems are becoming more and more important. These systems provide information to the user based on previous queries; these queries can be quite varied, from the weather there'll be tomorrow to the time a movie will be played or the definition of a word. The amount of available information is quite massive, which makes effectiveness and efficiency very important to these systems. Given the task at hand, finding information in a collection of documents, the information retrieval system would output the documents sorted so that those most relevant to the user's query are at the top.

Traditionally, information retrieval systems have been evaluated using benchmarks, consisting of documents, queries and relevant assessments. At first, the collections to be assessed were small, so it was viable for the assessors to read all the documents within, in order to assess said collection. From the TREC(*Text Retrieval Conference*) onward, collections grew noticeably in size, making the task of assessing them harder, which meant assessing the entire collection was no longer a viable approach. To solve this problem, a new technique, named pooling, appeared, which creates a pool of the top k documents belonging to all the systems involved in a competition, after these systems have all run the same queries on the same documents. Once the pool has been created, it is presented to the assessors in a random order so that their relevance to the query can be assessed. TREC has always followed the aforementioned strategy, but some other algorithms have been implemented by researchers, which do not present the documents sorted in a random order but do so in some specific order instead, such as MTF(*Move to Front*), and which don't use the content of the documents, but their order within each ranking.

This idea is the basis for this project, whose purpose is the creation of a new pooling algorithm that would make use of the content of the documents. The process is as follows: each time an assessor marks a document as relevant for a given query, the documents are reranked using a relevance model before being once again presented to the assessor.

Palabras clave:

- Recuperación información
- Relevancia
- Ranking
- Pooling
- Evaluación
- Indexación
- Modelos de lenguaje

Keywords:

- Information retrieval
- Relevance
- Ranking
- Pooling
- Evaluation
- Indexing
- Language models

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

En los últimos años los sistemas de recuperación de la información han ido adquiriendo cada vez más importancia. Dichos sistemas permiten resolver las necesidades de información de diversos usuarios. Para ello, los sistemas de recuperación de la información abarcan diferentes tareas:

- En primer lugar es necesario llevar a cabo una recogida de la información. Es importante destacar que la información puede ser proporcionada en diversos formatos, como por ejemplo vídeo, imagen, documentos, etc., y por tanto, para cada uno de estos tipos de información será necesario realizar un preprocesado en función de las características del mismo, lo cual luego permita al sistema trabajar con la información de forma cómoda y eficiente. Con la palabra preprocesado nos referimos tanto al parseado como al análisis de los documentos.
- El siguiente paso necesario consiste en llevar a cabo la indexación de la información, la cual permitirá realizar un acceso mucho más eficiente a la misma y por tanto proporcionará una mejora en las búsquedas futuras.
- A continuación es necesario realizar los algoritmos que permitan resolver las consultas planteadas por los usuarios de manera que proporcionen la información de la forma más satisfactoria posible.
- Para finalizar, es necesario realizar una evaluación del sistema para determinar si es eficiente y por lo tanto útil para resolver las necesidades de información de los usuarios.

1.1 Motivación

Como se ha mencionado anteriormente los sistemas de recuperación de la información son cada vez más importantes debido a la cantidad de uso que se hace de ellos habitualmente.

Debido a que existen muchos tipos de información, esto provoca que existan muchos sistemas de recuperación en base al tipo de información que se desea obtener, por ejemplo, un sistema para artículos periodísticos tendrá diferentes características que un sistema de recuperación de la información para obtener correos electrónicos o que uno que se emplee para la búsqueda en redes sociales. El caso paradigmático en este campo es el sistema de recuperación de información referente a web, ya que es un campo muy extenso, muy usado y engloba retos de recuperación muy complejos.

Existe una metodología clásica para realizar la evaluación de los sistemas, la cual consiste en construir colecciones para llevar a cabo la evaluación. TREC pone a disposición de los sistemas participantes una colección y unas necesidades de información. Los sistemas aportan los resultados de sus búsquedas y con eso se construyen los rankings, a través de los cuales, TREC hace un pool de documentos, formado por el top k de los documentos de todos los sistemas participantes. Una vez que dispone de ese pool, emplea la técnica de docId para hacer el pooling, que consiste en ordenar los documentos de manera alfanumérica por el identificador de los mismos. Una vez obtenido dicho orden, los documentos se presentan a un asesor para que realice una evaluación. Por tanto la motivación principal de este trabajo es realizar un nuevo algoritmo de pooling, igual que han realizado otros investigadores, pero en este caso, el algoritmo tendrá en cuenta el contenido de los documentos relevantes, para así intentar mejorar la query de forma iterativa con los términos más influyentes a la hora de encontrar documentos relevantes.

Este proyecto se enmarca en la actividad del IRLAB[1] en el que los últimos años se han realizado investigaciones en algoritmos de pooling, modelos de relevancia para PRF *(pseudo relevance feedback)* y para sistemas de recomendación. [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13] [14]

1.2 Objetivos

El objetivo es desarrollar un nuevo algoritmo de pooling el cual use el propio contenido de los documentos para ordenarlos, ya que los algoritmos existentes no usan dicha información, al contrario, o no usan ningún tipo de información o emplean el propio orden proporcionado por los sistemas participantes.
También se incluyen los siguientes requisitos no funcionales:

- **Eficacia** Nuestro algoritmo de pooling debe imponer un orden de los documentos, con el objetivo de proporcionar los documentos ordenados de manera que los documentos relevante se presenten de forma prioritaria a los asesores.

[1] http://irlab.org/

- **Eficiencia** Este algoritmo tiene que realizarse reduciendo en la medida de lo posible el uso de los recursos del sistema.

1.3 Estructura de la memoria

A continuación se explica la estructura de la memoria del presente proyecto:

- **Introducción:** Presenta el contexto del proyecto, la motivación por la cual ha sido impulsado y detalla los objetivos a lograr. También muestra la estructura de la memoria y el plan de trabajo llevado a cabo.
- **Conceptos:** Explica los conceptos necesarios para comprender el proyecto desarrollado, relacionados con la temática del mismo.
- **Tecnología:** Describe las tecnologías más relevantes que se han empleado en el desarrollo del proyecto en base a los requisitos del mismo.
- **Propuesta y desarrollo:** Explicación elaborada de las propuestas llevadas a cabo en el proyecto.
- **Experimentos y resultados:** Proporciona un análisis completo de los experimentos que se han realizado en el proceso de evaluación, la metodología y los resultados conseguidos. Esto se realiza a modo de comparativa con otras técnicas constituyentes del estado del arte.
- **Proceso de ingeniería:** Presenta el proceso de ingeniería llevado a cabo para garantizar el éxito del proyecto, así como las decisiones tomadas en cuanto al diseño para la construcción del mismo.
- **Conclusiones y trabajo futuro:** Proporciona una visión global de la calidad de los resultados obtenidos y propone líneas de trabajo futuro.
- **Apéndices**. Está compuesto por las siguientes secciones:
  - **Glosario**. Define los términos y acrónimos técnicos empleados en la memoria del proyecto.
- **Bibliografía**. Recoge la documentación bibliográfica sobre la que se basa el proyecto.

1.4 Plan de trabajo

El proyecto se ha llevado a cabo en diversas etapas, las cuales serán enumeradas de forma general a continuación:

- Estudio del campo de la recuperación de la información, el cual engloba conceptos básicos como la obtención de la información, el preprocesado de los documentos y la indexación de los mismos, la búsqueda de información y en mayor profundidad el estudio de las diferentes técnicas de pooling existentes y la evaluación de los sistemas.
- Determinar las colecciones sobre las cuales se va a implementar el sistema.
- Selección de las tecnologías más adecuadas para la realización del proyecto en función de la colección de datos seleccionada.
- Preprocesado y análisis de la colección seleccionada.
- Implementación de un indexador cuyo contenido sea la colección seleccionada ya preprocesada.
- Implementación del algoritmo de pooling DocId.
- Implementación del algoritmo de pooling MTF (*Move To Front*).
- Diseño e implementación de un algoritmo de pooling para realizar un re-ranking de los documentos en base al contenido de los mismos.
- Verificación de la implementación de los algoritmos.
- Realización de experimentos modificando los parámetros disponibles.
- Representación de los resultados en distintos formatos y análisis de los mismos.
- Elaboración de la memoria correspondiente al trabajo desarrollado durante el proyecto.

Capítulo 2

Conceptos

2.1 Recuperación de Información

Existen diferentes definiciones de lo que significa RI pero si las analizamos podemos apreciar que tienen algunos elementos en común. Un sistema de RI recibe una petición de un usuario, denominada consulta o $query$ y debe encontrar en un repositorio de información los pedazos de información necesarios, denominados típicamente documentos. La presentación de los resultados de búsqueda se presenta como una lista ordenada de documentos *(un ranking).*

Previamente se ha mencionado que el campo de recuperación de la información abarca muchos procesos intermedios, como por ejemplo el preprocesado de los documentos para llevar a cabo su indexación, o la implementación de los algoritmos de búsqueda, entre otras.

2.1.1 Crawling

La mayor parte de la gente, considera que crawling sólo comprende la búsqueda de información web, pero no es así, ya que el concepto crawling abarca todos los procesos de recogida de información independiente del campo en el que se emplee. En función de los requisitos de búsqueda funcionará de una forma o de otra. Realmente el web crawler es el más importante debido al tamaño y al gran uso que se hace de el diariamente y por lo tanto a continuación llevaremos a cabo su explicación de forma un poco más detallada:
Un crawler genérico comienza con un conjunto de URL*(Uniform Resource Locator)* iniciales. Para cada una de ellas se realiza una serie de comprobaciones antes de llevar a cabo la descarga de la página:

- Que la URL referencia a una página web
- No haya sido visitada previamente por el crawler

Si la página se descarga de forma correcta se almacena en un repositorio, se analiza el código HTML *(HyperText Markup Language)* en busca de enlaces a otros documentos, los cuales se añaden a la lista de URL y se repite el proceso recursivamente.

En nuestro proyecto no es necesario implementar un crawler debido a que utilizamos colecciones de documentos distribuidas por TREC, explicado más en detalle en las siguientes secciones, para las cuales el laboratorio IRLAB (*Information Retrieval Lab*) dispone de las licencias necesarias para su uso en dicha investigación.

2.1.2 Preprocesado y análisis de textos

Una vez que hemos obtenido los documentos podemos comenzar el análisis de los mismos. Realizar esto puede tener ciertas complicaciones, como por ejemplo:

- Los documentos pueden venir con diferentes tipos de codificación.
- Los documentos vienen en diferentes formatos, como pueden ser marcas que contienen etiquetas que no deseamos indexar, cómo las etiquetas HTML.

En primer lugar hay que ver cual es la codificación de los documentos, ya que es necesario que nuestro software sea capaz de comprender los documentos. El siguiente paso consiste en tokenizar los documentos, decidir cuales van a ser los separadores y en función de ello dividir los documentos en tokens. En tercer lugar debemos identificar cual es la información que nos interesa de los documentos disponibles, por ejemplo en nuestro caso los documentos son divididos en dos campos principales: un título y un texto. Las unidades indexables, también conocidas como términos o tokens pueden ser transformadas, lo cual es muy relevante ya que una misma palabra se puede escribir de múltiples formas y es muy importante que nuestro sistema de RI tenga en cuenta todas las variaciones posibles y las considere como el mismo término.

En este apartado resaltaremos algunas técnicas de preprocesado y análisis de texto, las cuales ayudarán a obtener mejores resultados en las búsquedas.

En primer lugar hablaremos de las **stopwords**, consideradas aquellas palabras muy frecuentes en un idioma y que por consiguiente no aportan información relevante en las búsquedas. Las stopwords pueden estar compuestas entre otras por artículos, preposiciones, conjunciones, etc. Si las eliminamos no perdemos información relevante pero sí que disminuimos notoriamente la cantidad de espacio necesario para el almacenamiento de los términos en la colección y por tanto también reducimos el tiempo de búsqueda empleado en el proceso de recuperación de la información. La eliminación de las stopwords no siempre es beneficiosa ya que algunas consultas pueden incluir muchas de las palabras que conforman este conjunto.

También disponemos de otra técnica de preprocesado denominada **stemming** que consiste en la eliminación de los sufijos de las palabras para así, reducirlas a su forma léxica o

lexema. Un ejemplo muy básico de stemming consiste en eliminar la letra "s" al final de las palabras, en el caso de que esa s se considere el plural de las mismas. Este proceso puede producir errores y es muy dependiente del idioma, por ejemplo en el árabe funciona muy bien debido a que tiene muchas variantes morfológicas.

Ya para finalizar nombraremos otra técnica más, denominada **lematización** que tiene como objetivo reducir y agrupar palabras usando lo que se conoce como lema o forma base.

2.1.3 Modelos de recuperación de la información

Un modelo de RI tiene como objetivo lo siguiente: Dada una necesidad de información, proporcionar un acceso a los documentos de forma que se facilite la obtención de los documentos relevantes, para ello existen distintos tipo de modelos de recuperación. Podemos clasificar los modelos clásicos de RI en booleano, vectorial y probabilístico, los cuales explicaremos a continuación:

Modelo booleano

El modelo booleano está basado en la teoría de conjuntos y la lógica booleana. Si lo definimos formalmente, dado del conjunto V como el conjunto de descriptores o términos presentes en la colección, también conocido como vocabulario.

$$V = \{t_1, t_2, ..., t_n\} \tag{2.1}$$

El conjunto D es el conjunto de todos los documentos de la colección. Cada documento d_i se representa mediante un subconjunto de V.

$$D = \{d_1, d_2, ..., d_m\} \tag{2.2}$$

En el modelo booleano las consultas se representan con expresiones booleanas que contienen los siguientes operadores.

- **AND**: Es un operador binario e indica que se desea encontrar los documentos que contengan ambos términos. Esta operación del modelo booleano se corresponde con la intersección de conjuntos.
- **OR**: Es un operador binario e indica que se desea encontrar los documentos que contengan al menos uno de los dos términos. Dicha operación del modelo booleano se corresponde con la unión de conjuntos.
- **NOT**: Es un operador unario e indica que se desea encontrar aquellos documentos que no contienen dicho término. Este operador se usa en combinación con los mencionados previamente, tanto AND como OR.

Entre sus ventajas encontramos:

- Es un modelo fácilmente formalizable e implementable.
- Es muy útil cuando los usuarios que lo usan son expertos, es decir, son capaces de formalizar las consultas con los operadores booleanos.

Entre sus desventajas encontramos:

- Un usuario experto si ve que una consulta devuelve muchos documentos es capaz de formularla de nuevo para que sea más estricta y de la misma manera tiene la capacidad de realizar una nueva consulta para que sea más laxa en caso de que sea necesario. Pero un un usuario normal no sería capaz de hacer esto, por tanto limita a los usuarios que pueden trabajar cómodamente con este modelo.
- Este modelo no tiene en cuenta las estadísticas de las palabras y existen otros modelos que si que son capaces de explotar las mismas.

Modelo vectorial

La base del modelo vectorial consiste en modelar los documentos y las consultas como vectores de términos. En primer lugar se crea un espacio vectorial cuya dimensión se corresponde con el número de términos de la colección donde cada documento y cada consulta son un vector de ese espacio vectorial. De esta manera se puede determinar qué documento es más parecido al vector que se corresponde con la consulta.

En este modelo el conjunto de todos los términos de la colección y el conjuntos de todos los documentos de la colección se define a través de las siguientes fórmulas, respectivamente:

$$\vec{V} = (t_1, t_2, ..., t_n) \tag{2.3}$$

$$\vec{D} = (d_1, d_2, ..., d_m) \tag{2.4}$$

Definimos un documento d_j como un vector de términos, donde $w_{i,j}$ se corresponde con el peso del término i en el documento j, de la siguiente forma:

$$\vec{d_j} = (w_{1,j}, w_{2,j}, ..., w_{M,j}) \tag{2.5}$$

En el caso de que el término no se encuentre en un documento tendrá peso 0. Una vez que disponemos de todos los vectores de términos se crea una matriz. Esta matriz representa en las columnas, los términos con los documentos en los que aparece. Por otro lado en las filas representa los documentos de la colección junto con los términos que contienen. A

continuación se puede apreciar un ejemplo:

$$\mathbf{M} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,j} & \cdots & w_{1,N} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,j} & \cdots & w_{2,N} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{M,1} & w_{M,2} & \cdots & w_{M,J} & \cdots & w_{M,N} \end{bmatrix} \tag{2.6}$$

A la hora de asignar los pesos podemos seleccionar diferentes métodos. Cabe destacar que el que se usa de forma más habitual es el que tiene en cuenta el número de ocurrencias de un término en un documento, así cuantas más veces aparezca el término más relevante será el mismo en dicho documento.

Este no es el único método posible, por ejemplo tenemos la especificidad del término en la colección, ya que si tenemos en cuenta que no todos los términos son igual de comunes, lo lógico es proporcionar un mayor peso a aquellos que son menos frecuentes ya que así se puede identificar de forma más sencilla los documentos más relevantes.

Uno de los esquemas de ponderación más conocido es el modelo tf*idf donde el tf representa la frecuencia del término en el documento y el idf el número de documentos en los que aparece el término, pero a la inversa. De esta manera conseguimos proporcionar más importancia a los términos que aparecen en un menor número de documentos.

Por tanto el modelo vectorial se centra en dos conceptos básicos, por un lado la asignación de los pesos y por otro la similitud de vectores de términos, por ejemplo el vector que representa un documento y el vector que representa una consulta. La manera más sencilla de determinar la similitud de dos vectores se hace a través de un producto interno de vectores como se muestra a continuación:

$$q * d = \sum_{i=1}^{M} w_{i,q} * w_{i,d} \tag{2.7}$$

El problema de esto es que no se tiene en cuenta el tamaño de de los documentos, por tanto aquellos documentos que sean más grandes obtendrán mejores resultados ya que tendrán más probabilidad de contener un mayor número de términos. La solución ante este problema consiste en introducir a la fórmula previa, la suma de los pesos de la colección, lo cual se proporciona a través de la siguiente fórmula:

$$q * d_{norm} = \frac{q * d}{|d|} \tag{2.8}$$

Como ventajas, podemos destacar las siguientes:

- Realiza una ordenación en base a la frecuencia de los términos.

- Se puede implementar en grandes colecciones de documentos.

Aunque también tiene desventajas, como pueden ser:

- No tiene en cuenta el orden de los términos.
- Asume que no hay relación entre dos términos.

Modelo probabilístico

Si tenemos en cuenta que el proceso de RI tiene un cierto porcentaje de incertidumbre, la manera más adecuada de modelarlo es la teoría de probabilidades.

Para llevar a cabo este modelo se parte de la base de que cada consulta tiene un conjunto ideal de documentos relevantes, por tanto el objetivo es determinar la probabilidad de que un documento d sea relevante para una consulta q. Una de las suposiciones en las que se basa el modelo probabilístico es el denominado **PRP** *(Probability Ranking Principle)* que determina que si el resultado de un sistema de RI es una lista ordenada de documentos de mayor a menor probabilidad de relevancia, la efectividad del sistema será la máxima posible. Por tanto el proceso consiste en ordenar los documentos de forma decreciente en cuanto a la probabilidad de relevancia.

Para calcular la probabilidad de que un documento sea relevante o no se emplea el Teorema de Bayes, del cual obtenemos la siguiente fórmula:

$$p(d|q) = \frac{p(q|d) * p(d)}{p(q)} \tag{2.9}$$

Siendo $p(q|d)$ la probabilidad de obtener la query, dado el documento. $p(d)$ es la probabilidad a priori de que el documento d sea relevante para la query. Y por último, siendo $p(q)$ la probabilidad de la query, la cual puede ignorarse porque para la misma query siempre tendrá el mismo valor para todos los documentos.

La principal ventaja de este modelo es su base teórica, que sirve como base a otros modelos más complejos.

2.1.4 Índices invertidos y procesado de consultas

La indexación de la información de los términos que hay en una colección de documentos permite entre otras cosas realizar búsquedas a partir de una consulta a una alta velocidad.

Un índice invertido es una estructura muy empleada en los sistemas de recuperación de la información, especialmente en los que tienen que manejar grandes cantidades de datos. Esta estructura contiene los términos de la colección y para cada uno de ellos asocia una lista con los documentos en los que aparece el mismo, dichos documentos se identifican a través de un identificador único.

A la hora de construir un índice invertido tenemos que seleccionar qué términos serán indexados, para esto se divide el texto en tokens. Idealmente se debe procesar los términos antes de almacenarlos con técnicas como stopping, stemming y lematización, de las cuales ya hemos hablado en las secciones previas.

Una vez que se ha creado el índice es necesario procesar los datos que hay en el para obtener resultados para una query, ya que su uso hace que se procesen más rápidamente. A continuación vamos a explicar los dos algoritmos más empleados.

Document-at-a-time

En la figura que se presenta a continuación se muestra el resultado de document at a time para la consulta ”tortuga camaleón zoo”. Las listas invertidas se muestran horizontalmente, las cuales contienen la cuenta de los términos. La puntuación se calcula a través de la suma de la cuenta de los términos en cada documento. En el primer paso se calcula el número de veces que aparece cada término en el primer documento, en el segundo paso se realiza para el segundo documento y así sucesivamente.

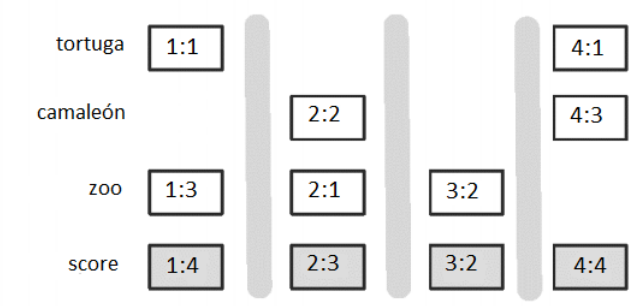

Figura 2.1: Ilustración del algoritmo document-at-a-time. [1]

Para comenzar la explicación debemos tener en cuenta que se emplea una cola de prioridad, cuyo objetivo es almacenar el *top-k* de documentos. Este algoritmo dispone de un bucle principal, el cual se realiza para cada documento de la colección. Si aparece el documento en una de las listas invertidas se evalúa una función que implementa la puntuación que un modelo de RI le da al documento. A continuación se calcula la puntuación del documento realizando la suma de las funciones ponderadas. Posteriormente el puntero de la lista invertida se mueve al siguiente posting. Al final de cada ciclo del bucle se ha calculado una nueva puntuación, la cual se agrega a una cola de prioridad.

La principal ventaja de este método es que el uso de memoria proviene de la cola de prioridad, la cual solo almacena k entradas, siendo esto el número de documentos evaluados pertenecientes a todo el conjunto de documentos.

Term-at-a-time

En la figura que se presenta a continuación se muestra el resultado de term at a time para la misma consulta utilizada en document-at-a-time: ”tortuga camaleón zoo”. Las líneas grises representan los pasos realizados en el algoritmo, de forma que en primer paso se analiza la lista invertida de ”tortuga” almacenando las puntuaciones parciales en acumuladores. En el siguiente paso las puntuaciones parciales calculadas previamente se combinan con las puntuaciones de ”camaleón” y así sucesivamente.

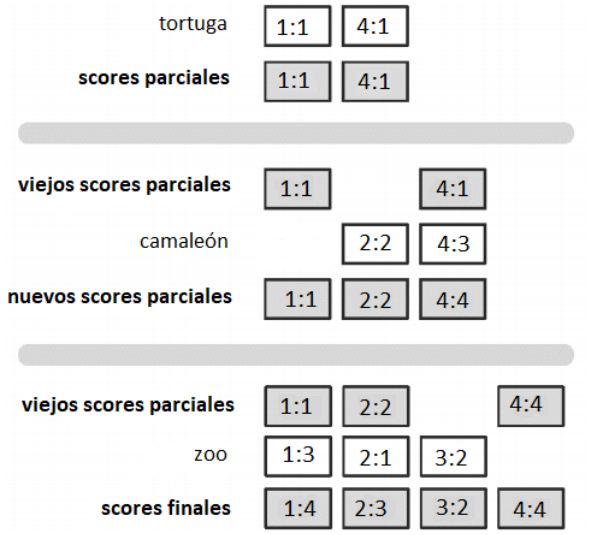

Figura 2.2: Ilustración del algoritmo term-at-a-time. [1]

En la práctica los acumuladores se almacenan en una tabla hash, la cual una vez que se hayan procesado todas las listas invertidas contiene las puntuaciones finales de los documentos. La principal desventaja de este algoritmo es el uso de memoria que se emplea para la tabla de acumuladores.

En la práctica, ambos algoritmos, document-at-a-time y term-at-a-time se implementan con optimizaciones adicionales, para así conseguir una gran mejora en la velocidad de ejecución.

2.1.5 Evaluación

La evaluación de los sistemas de RI se divide en dos partes:

- **La eficacia** del sistema, la cual se refiere a la capacidad del sistema para recuperar la información. En esta parte entra el concepto de **relevancia** que se asocia con la utilidad de la información otorgada al usuario. Las medidas más empleadas para la medición de

eficacia de un sistema son las siguientes:

- **Recall**, se define como la proporción de documentos relevantes recuperados de todos los documentos relevantes de la colección.

$$Recall = \frac{n^{\circ} \quad docs \quad rel \quad recuperados}{n^{\circ} \quad docs \quad rel \quad coleccion} \tag{2.10}$$

- **Precisión**, se define como la proporción de documentos relevantes recuperados en respuesta a una consulta.

$$Precision = \frac{n^{\circ} \quad docs \quad rel \quad recuperados}{n^{\circ} \quad docs \quad recuperados} \tag{2.11}$$

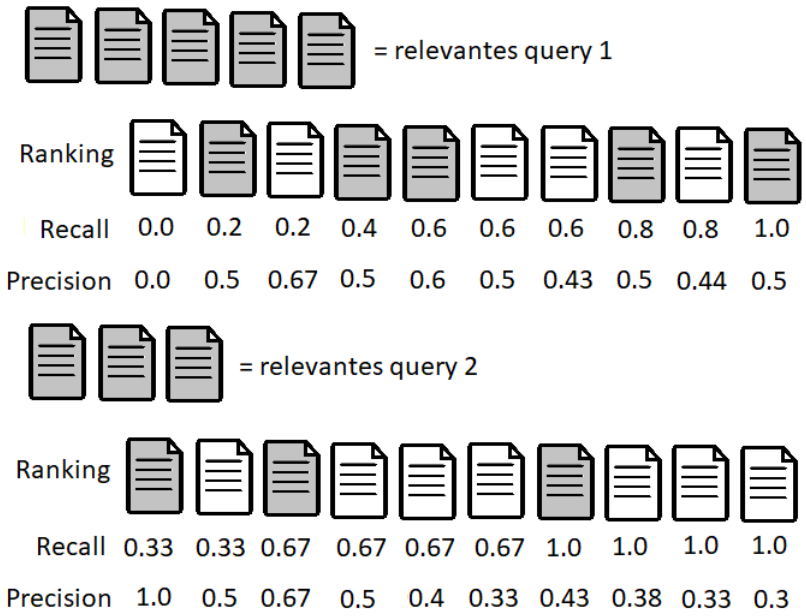

Figura 2.3: Ejemplo recall, precision y map.

Estas métricas son orientadas a conjuntos, las cuales se adaptaron a sistemas que devuelven rankings.

Disponemos de medidas como ***P@n*** (Precisión a n) que refleja el comportamiento del sistema de recuperación de información mediante el valor de la precisión en la posición n-ésima de la respuesta ordenada obtenida de una consulta.

De la misma forma también disponemos de ***R@n*** *(Recall a n)* que refleja el comportamiento del sistema de recuperación de la información mediante el valor del recall en la posición n-ésima de la respuesta ordenada obtenida de una consulta.

La precisión y la recuperación son métricas de valor único basadas en la lista completa de documentos devueltos por un sistema de recuperación. Hay sistemas que devuelven una lista ordenada de documentos, dicho orden tiene mucha importancia, para ello existe una métrica denominada precisión media *(average precision)* la cual consiste en realizar el promedio de los valores de precision en las posiciones donde se ha recuperado un documento relevante. Por ejemplo para la query 1 de la figura 2.3 el calculo sería el siguiente:

$$\frac{0.5 + 0.5 + 0.6 + 0.5 + 0.5}{5} = 0.52 \tag{2.12}$$

Si tenemos como objetivo determinar la efectividad de un sistema con un conjunto de queries que tienen diferente número de documentos relevantes, podemos emplear la técnica denominada **MAP** *(Mean average precision)* la cual realiza el promedio del average precision de cada query. A continuación se presenta un ejemplo con ambas queries de la figura 2.3:

$$\frac{\frac{0.5+0.5+0.6+0.5+0.5}{5} + \frac{1.0+0.67+0.43}{3}}{2} = 0.61 \tag{2.13}$$

- **La eficiencia**, la cual está relacionada con el tiempo necesario para encontrar una respuesta a una consulta y a la cantidad de recursos como disco, ancho de banda y energía empleados para ello.

2.2 Modelos de lenguaje

Los modelos de lenguaje (*language models*) son técnicas de RI en las cuales se supone que una query q se genera a partir de un modelo probabilístico de un documento d.

2.2.1 Formulación probabilística de los modelos de lenguaje

Dada una query $q = \{q_1, q_2, \dots, q_n\}$ donde q_i son los términos que forman la query y un documento d, debemos estimar $p(d|q)$, es decir la probabilidad de que se haya usado la query q para generar el documento d. Para esto se usa el teorema de Bayes mencionado anteriormente, donde la $p(q|d)$ es el *query likelihood*, que representa la probabilidad de que muestreando términos aleatoriamente a partir del language model del documento podamos generar exactamente el texto de la query.

$$p(q|d) = \prod_{i=1}^{n} p(q_i|d) \tag{2.14}$$

La estimación de $p(q_i|d)$ es la siguiente:

$$p(q_i|d) = \frac{f_{qi}, d}{|d|} \tag{2.15}$$

Siendo f_{qi}, d es la frecuencia del término en el documento y |d| la longitud del documento. Como se puede observar si algún término no se encuentra en el documento, la puntuación será 0. Es decir la puntuación será la misma independientemente de que falte 1 palabra o 5. Para evitar este problema se introduce la suavización.

2.2.2 Suavización

Las técnicas de suavizado (*smoothing*) mueven parte de la probabilidad de los términos que aparecen en el documento a los términos que por el contrario, no aparecen. Esto se suele realizar con un parámetro de suavización que a medida que aumenta proporciona más probabilidad a aquellos términos que no se encuentran en el documento.
Las dos técnicas más populares de suavización son las que se presentan a continuación:

- **Jelinek-Mercer**

$$p(w|d) = (1 - \lambda) * \frac{f_{qi}, d}{|d|} + \lambda * \frac{c_{qi}}{|c|} \tag{2.16}$$

- **Dirichlet**

$$p(w|d) = (1 - \frac{\mu}{|d| + \mu}) * \frac{f_{qi}, d}{|d|} + \frac{\mu}{|d| + \mu} * \frac{c_{qi}}{|c|} \tag{2.17}$$

Vamos a realizar en mas detalle la explicación y el desarrollo del suavizado denominado *Jelinek-Mercer* ya que será el que se aborde en el desarrollo del proyecto. Substituyendo esta estimación en la puntuación del documento con el modelo *query likelihood* obtenemos la siguiente fórmula:

$$p(q|d) = \prod_{i=1}^{n} ((1 - \lambda) * \frac{f_{qi}, d}{|d|} + \lambda * \frac{c_{qi}}{|c|}) \tag{2.18}$$

$c_{q,i}$ es la frecuencia del término en la colección de todos los documentos, y |c| es el tamaño de la colección. λ es un valor acotado entre 0 y 1, valores pequeños de λ producen menos suavizado y por tanto si una palabra está ausente en el documento, este se ve notoriamente penalizado. Este valor también debe ser distinto en función de la longitud de la query, ya que si es muy larga la ausencia de una de sus palabras en el texto no tendrá tanta importancia y por lo tanto un valor de λ alto proporcionara mejores resultados.

2.3 Pseudo-relevance feedback y re-ranking

La expansión de consultas tiene como objetivo agregar nuevos términos a la consulta original proporcionada por el usuario. Estas técnicas pueden mejorar el rendimiento del sistema de recuperación de la información.

2.3.1 Tareas de pseudo-relevance feedback

Pseudo-relevance feedback (*PRF*) [4] es una estrategia muy efectiva para mejorar la precisión de recuperación de información sin la intervención del usuario. En este proceso se asume que el top k de documentos recuperados para la query original son relevantes y produce una versión extendida de la query original usando la información de ese conjunto inicial de documentos. Con la query expandida se realiza una segunda búsqueda y los resultados obtenidos son los que se presentan al usuario. Las técnicas de pseudo-relevance feedback basadas en *language models* son las más destacadas debido a su base teórica.

Los modelos de relevancia (*RM*) introducen el concepto de relevancia en los modelos de lenguaje. El primer método conocido es $RM1$ y en el se emplea el query likelihood $p(q|D)$ como el peso del documento D y se realiza un promedio de la probabilidad de la palabra w dada por cada modelo de lenguaje de documento. La fórmula es la siguiente:

$$p(w|\theta_{RM1}) \propto \sum_{\theta_D \in \Theta} p(w|\theta_D) p(\theta_D) \prod_{i=1}^{m} p(q_i|\theta_D) \tag{2.19}$$

Siendo Θ el conjunto de documentos suavizados de la colección. En la práctica esto se computa sobre el top k del primer *retrieval*, ya que los que más afectan son los que se encuentran en las primeras posiciones.

Para mejorar el rendimiento se puede interpolar el modelo de relevancia $p(w|\theta_{RM1})$ con el modelo de query original θ_Q. Esto se puede hacer con la fórmula conocida como $RM3$ que se presenta a continuación:

$$p(w|\theta_{RM3}) = (1-\alpha) p(w|\theta_Q) + \alpha p(w|\theta_{RM1}) \tag{2.20}$$

2.4 Pooling

Una metodología basada en pooling consiste en que para cada query, se forma un *pool* con el top k (dicho valor suele ser 100) de documentos proporcionados por diversos sistemas de recuperación de la información, los cuales se presentan a los asesores para que juzguen si esos documentos son relevantes o no para la query. Esos rankings proporcionados por los

sistemas se denominan *runs* y el resultado proporcionado por los juicios de relevancia se conoce comúnmente como *qrels*.

Actualmente esta es la técnica empleada ya que para queda *query* solamente se juzga un subconjunto (*pool*) de toda la amplia colección de documentos. Todos aquellos documentos que no aparecen en el *pool* se consideran no relevantes.

2.4.1 El paradigma Cranfield de evaluación

En 1952, Cyril Cleverdon, un bibliotecario de la Facultad de Aeronáutica de Cranfield, en Estados Unidos realizo unos experimentos de recuperación de información para determinar la eficacia de los sistemas de indexación [15]. Consiguió una beca del NSF (*National Science Foundation*) por llevar a cabo la comparación de cuatro sistemas de indexación distintos. Dicho proyecto se conoce como *Cranfield-1*, el cual tuvo como requisitos indexar manualmente 18000 documentos de ingeniería aeronáutica en cada uno de los sistemas de indexación y evaluar el resultado de 1200 queries.

El siguiente paso, fue idear experimentos que permitieran realizar una evaluación más exhaustiva de cada sistema de indexación por separado. Para cada query, se examinaron todos los documentos de la colección y se determinó su relevancia, esto fue posible debido al pequeño tamaño de la colección ya que en otro caso el coste sería excesivo. Dicho experimento fue realizado con un total de 1400 documentos y 279 queries, para las cuales seis estudiantes se pasaron tres meses examinando cada documento para cada query. El resultado de este experimento se conoce como *Cranfield-2* y consiste en una colección de test formada por documentos, queries y juicios de relevancia para cada par documento-query. Dicho experimento estableció la base de la experimentación moderna de recuperación de la información ya que, el conjunto de documentos y de queries se puede usar para evaluar diferentes sistemas de ranking llevando a cabo una comparación con los juicios de relevancia hechos por los humanos.

En colecciones grandes no es posible realizar todas las comprobaciones par (documento-*query*) y debido a esto TREC ideó el procedimiento de pooling.

2.4.2 El procedimiento TREC de construcción de benchmarks de evaluación

TREC (*Text Retrieval Conference*) [16] tiene como objetivo apoyar la investigación referente a la recuperación de la información proporcionando la infraestructura necesaria para la evaluación a gran escala de metodologías de recuperación de texto. Para cada TREC, el NIST (*National Institute of Standars and Technology*) proporciona un conjunto de documentos de prueba y de consultas. Los participantes ejecutan sus sistemas de recuperación de información con dichos datos y proporcionan al NIST una lista de los documentos recuperados. El NIST agrupa los resultados individuales, juzga la corrección de los documentos recuperados y evalúa los resultados.

2.4.3 Procedimientos de pooling de TREC

El procedimiento de pooling de TREC funciona de la siguiente forma:

- Dada una colección de documentos los organizadores de la campaña definen una tarea para devolver documentos relevantes en base a un conjunto de queries.
- Los participantes envían los resultados de sus sistemas de recuperación de información.
- Para cada query, los primeros k documentos se agrupan en un único conjunto ordenado por el campo del identificador del documento de forma alfanumérica y se presentan a los evaluadores para que realicen su evaluación.

2.4.4 Otros algoritmos de pooling

Existen otros algoritmos de pooling como los mencionados a continuación:

- **MTF** (*Move to Front*) [17], dicho método dispone de una puntuación de prioridad para cada *ranking* del sistema participante. Inicialmente, dichas puntuaciones son uniformes. De forma aleatoria se selecciona una ejecución de las que disponen de una puntuación más alta (en la primera iteración, se selecciona una cualquiera de todas las disponibles, ya que parten todas de la misma puntuación) y se juzga el documento mejor clasificado (los documentos que ya han sido juzgados se omiten). Si el documento juzgado es relevante, se siguen juzgando los documentos del mismo *ranking* hasta llegar al punto en que se encuentra un documento que no es relevante. En dicho momento, la puntuación de ese *ranking* baja y de forma aleatoria se selecciona otro *ranking* de las que dispongan de una puntuación máxima.

  Todo el proceso explicado previamente se realiza para cada *query*.

- **Hedge**. Es un algoritmo empleado para combinar el asesoramiento de expertos. [5] [18]
- **Bayesian Bandits**. Una perspectiva bayesiana permite tratar formalmente la incertidumbre asociada a las probabilidades de ganar. Ya que cada máquina está caracterizada por su probilidad de ganar, es decir, de proporcionar un documento relevante.[5]

Capítulo 3
Tecnología

En este capítulo abordaremos el tema correspondiente a las tecnologías empleadas en el desarrollo del proyecto, el éxito del mismo se verá influenciado en una gran parte por las tecnologías seleccionadas, ya que si no disponemos de herramientas adecuadas a la resolución del problema nunca podremos hacer un buen desarrollo.

3.1 Plataforma

El primer paso es seleccionar la plataforma en la que se desarrollará dicho sistema. Debemos tener en cuenta entre otras cosas los siguientes factores:

- Grado de madurez
- Actividad de la comunidad. Cuanta más gente utilice la plataforma más fácil será resolver los problemas que vayan surgiendo a lo largo del desarrollo.
- Disponibilidad de librerías. Esto puede simplificar el trabajo a realizar siempre y cuando dichas librerías se encuentren bien documentadas.
- Curva de aprendizaje. Cuando una persona comienza a programar en un lenguaje nuevo esto requiere un tiempo de aprendizaje, cuanto menor sea ese tiempo más rápido empezará a desarrollar de forma eficiente.

Java

Java[1] es un lenguaje de programación de propósito general, concurrente, orientado a objetos y basado en clases. Una característica muy importante de este lenguaje es que una vez que hemos desarrollado el proyecto se podrá ejecutar en múltiples plataformas sin que sea necesario hacer una recompilación del mismo, esto ocurre así gracias a que el compilador Java

[1] https://www.java.com/es/

compila su código en un fichero independiente de la arquitectura de la máquina, por lo tanto cualquier máquina que disponga de un sistema de ejecución (run-time) puede llevar a cabo la ejecución.

Como se cita al comienzo de esta sección es un lenguaje concurrente o lo que se puede denominar multithread. Un thread es un flujo secuencial simple dentro de un proceso. Sin el uso de Threads habría tareas que serían prácticamente imposibles de ejecutar como por ejemplo aquellas que tienen amplios tiempos de espera entre etapas.

Podemos seguir destacando más facilidades proporcionadas por este lenguaje como puede ser el amplio número de APIs (*Application Programming Interface*) disponibles en el desarrollo de nuestro proyecto, lo cual permite realizar proyectos sin que sea necesario implementar todo el código desde cero. Una API hace referencia a los procesos, a las funciones y a los métodos que proporciona una bilioteca de programación a modo de una capa de abstracción para ser usada por otro programa informático. Un programador puede emplear la funcionalidad de una API para así evitar un amplio volumen de trabajo, el cual suele ser muy común en muchos proyectos.

Por otro lado disponemos de *Garbage Collector*, el cual administra la memoria automáticamente. En Java una vez que se crea un objeto, utiliza memoria, la cual permanece asignada hasta que se encuentran referencias para su uso. Cuando no hay referencias a un objeto esa memoria puede ser reclamada, pero en Java no es necesario la destrucción de la misma de forma manual ya que gracias a la recolección de basura que hace este lenguaje ya se realiza de forma automática. Si esto no se hiciera así un programa se podría quedar bloqueado al no quedar memoria en el sistema.

Lucene

Lucene[2] es una librería de recuperación de información escalable y de alto rendimiento. Es útil para aquellas aplicaciones o proyectos que requieran la realización de indexación y de búsqueda. A Lucene no le importa el origen de los datos o el formato, siempre y cuando se puedan convertir a texto. La arquitectura de Lucene se basa en el concepto de documento, el cual contiene campos de texto.

Koloboke

Koloboke[3] es una librería que extiende de *Java Collections Framework*, formada por un conjunto de clases e interfaces que sirven para almacenar listas, conjuntos y mapas. Koloboke

[2] https://lucene.apache.org/
[3] https://koloboke.com/

es más adecuado en este campo debido a que utiliza tipos primitivos los cuales no necesitan ningún tratamiento para ser usados.

Debido a la temática de este proyecto es necesario almacenar y analizar grandes cantidades de datos, aquí el motivo de seleccionar dicha librería ya que realiza estas tareas considerablemente más rápido, lo cual nos proporcionará más tiempo para la realización de diferentes experimentos buscando los resultados más óptimos.

3.2 Herramientas de soporte

No solo es necesario el uso de las herramientas más específicas para la elaboración del proyecto, también es necesario el uso de otras herramientas de soporte para la gestión del proyecto.

IDE: Eclipse

Eclipse [4] es una plataforma de desarrollo con una amplia cantidad de plugins como por ejemplo el plugin JDT que se encarga de dar soporte al lenguaje Java.

Un plugin añade funcionalidad adicional o una nueva característica al software. Se suele ejecutar a través de un software principal con el que interactúa a través de una interfaz.

Entre las características de Eclipse se puede destacar por ejemplo que tiene un editor de texto con resaltado de sintaxis donde puedes ver el contenido del fichero con el que estas trabajando en dicho momento. La compilación se realiza en tiempo real. Dispone de pruebas unitarias con JUnit, etc. Dichas características son bastante generales pero se pueden ampliar mediante los plugins como hemos mencionado previamente, por ejemplo añadiendo un control de versiones a través de Subversion.

Maven

Maven [5] es una herramienta que se emplea para gestionar y construir proyectos en Java. Contiene un fichero XML (*Extensible Markup Language*) llamado POM (*Project Object Model*), el cual describe el proyecto, incluyendo información de versiones, gestión de configuración, sus dependencias con otros módulos externos, el orden de construcción de los elementos constituyentes del proyecto y los plugins necesarios. Si el proyecto contiene módulos, estos se declaran dentro del pom.xml del directorio raíz y cada módulo contiene su propio pom.xml el cual hereda del pom.xml general. Maven permite redefinir la estructura de directorios es-

[4] https://www.eclipse.org/es
[5] https://maven.apache.org/

tándar pero es conveniente respetarla ya que hace que el proyecto sea más fácil de entender y facilita el mantenimiento futuro por otros programadores. En Maven un ciclo de vida de construcción de un proyecto se divide en fases:

- default: abarca todas las fases relacionadas con la generación de los artefactos o binarios del proyecto.
- clean: cubre las fases de limpieza del proyecto.
- site: abarca las fases para generar y distribuir un sitio web con la documentación asociada.

Cada vez que se ejecuta una fase del ciclo de vida, se ejecutan los plugins vinculados a ella. Por lo tanto, Maven contiene todo lo necesario para que a la hora de generar el fichero ejecutable contenga todo lo que necesita para su ejecución.

Bitbucket

Bitbucket [6] es una de las plataformas más conocidas para el control de versiones, lo cual permite llevar a cabo un mantenimiento del proyecto y poder volver atrás si se comete un fallo de forma sencilla. Bitbucket permite utilizar Git o Mercurial como controlador de versiones. Una ventaja muy importante es que dispone de un número ilimitado de repositorios privados. En las desventajas a destacar es que en cada proyecto sólo pueden colaborar un máximo de 5 personas (en la versión gratuita) lo cual en nuestro caso no es relevante.

Git

Git [7] es uno de los sistemas de versiones más utilizado hoy en día. Tiene una amplia comunidad de usuarios que lo usan lo cual proporciona facilidades a la hora de resolución de problemas. Entre las características mas destacables encontramos las siguientes:

- Casi todas las operaciones son locales.
- Tiene operaciones para deshacer cambios.
- Tiene tres estados posibles.
  - *Working directory*, es donde trabajamos en local.

[6] https://bitbucket.org/product/
[7] https://git-scm.com/

  - *Staging area*, donde se guardan los cambios preparados para subirse (se encuentra también en local).
  - *Repository*, donde se almacena el código.

- Es rápido.

Capítulo 4

Propuesta y desarrollo

En este capítulo del proyecto se detalla la propuesta a desarrollar en el mismo y el análisis y diseño llevado a cabo para permitir su implementación, incluyendo un análisis de requisitos, diseño de arquitectura y diseño de clases de cada uno de los componentes.

4.1 Propuesta

Este proyecto tiene como objetivo aplicar un algoritmo de *re-ranking* a las técnicas de pooling, de esta manera cada vez que se encuentra un nuevo documento relevante para la *query* se realiza un *re-ranking*.

Para ello partimos de un orden inicial, se va recorriendo dicho orden y realizando el modelo de relevancia RM3 ya explicado previamente.

4.1.1 Modelos básicos

A continuación se llevará a cabo una explicación del proceso a seguir de manera formal, explicando detalladamente los cálculos llevados a cabo en cada paso. Es importante tener en cuenta que este proceso de cálculo parte de un *retrieval* inicial.

Comenzamos recorriendo por orden dicho *retrieval* hasta encontrar el primer documento relevante, el cual pasará a formar parte del conjunto de documentos relevantes también conocido como *relevance set*. A continuación, calculamos la puntuación de la *query* para cada documento perteneciente a dicho conjunto. Esta puntuación se obtiene a través de la siguiente

fórmula, siendo n el número de términos que forman la *query*:

$$p(q|d) = \prod_{i=1}^{n} p(q_i|d) \tag{4.1}$$

Para realizar el cálculo de $p(q_i)$ emplearemos la fórmula de suavizado de Jelineck-Mercer correspondiente a la ecuación 2.16, la cual nos proporciona la siguiente aproximación:

$$p(qi|d) = (1-\lambda)\frac{f_{qi,d}}{|d|} + \lambda\frac{f_{qi,c}}{|c|} \tag{4.2}$$

Por tanto el desarrollo completo de la puntuación de una query se corresponde con la siguiente fórmula, resultante de la combinación de la ecuación 4.1 y 4.2:

$$p(q|d) = \prod_{i=1}^{n}(1-\lambda)\frac{f_{qi,d}}{|d|} + \lambda\frac{f_{qi,c}}{|c|} \tag{4.3}$$

Una vez que tenemos calculada la puntuación de la query en los documentos pertenecientes al conjunto de documentos relevantes, podemos proceder a realizar el cálculo de la puntuación de los términos pertenecientes a dicho conjunto. Para ello emplearemos la siguiente fórmula:

$$p(w|RS) = \sum_{i=1}^{n} p(w|d) * p(q|d) \tag{4.4}$$

Ya para finalizar, calcularemos $p(w|d)$ empleando también la fórmula de suavizado Jelineck-Mercer:

$$p(w|d) = (1-\lambda)\frac{f_{w,d}}{|d|} + \lambda\frac{f_{w,c}}{|c|} \tag{4.5}$$

Todo este proceso se realiza de forma iterativa, cada vez que el algoritmo proporcione un nuevo documento relevante, para así, recalcular la puntuación de los términos pertenecientes al conjunto de relevantes y lanzar la query con los nuevos top n de términos, los cuales independientemente de que cambie su puntuación no tienen porque ser términos diferentes. Al lanzar la *query* con los valores actualizados proporciona un nuevo *ranking*.

4.2 Construcción del sistema

En esta sección se lleva a cabo la explicación del análisis y diseño realizados para desarrollar el algoritmo propuesto.

4.2.1 Análisis de requisitos

Lo primero que se debe hacer es identificar los requisitos tanto funcionales como no funcionales del sistema.

Requisitos funcionales
Los requisitos funcionales son aquellos que especifican las funcionalidades que debe proporcionar nuestro sistema. En nuestro caso, son los siguientes:

- Implementación de un indexador con el contenido de los documentos a tratar.
- Reordenación de un ranking proporcionado por un sistema de recuperación de la información.
- Evaluación de dicha reordenación.
  - Comparación con otras técnicas como MTF y DocId.
  - Estabilidad de los parámetros.

Requisitos no funcionales
Por otro lado, los requisitos no funcionales definen los criterios que tiene que cumplir el sistema. Estos requisitos, son aquellos que condicionan las decisiones de diseño que se realizan a lo largo del desarrollo. En nuestro caso, son los siguientes:

- Escalabilidad: Como nuestro sistema emplea una gran cantidad de datos, tiene que escalar correctamente.
- Eficiencia: El tiempo de ejecución es muy importante, ya que si lo reducimos, podremos llevar a cabo un mayor número de experimentos con los diversos parámetros disponibles para determinar cuales son los mejores resultados.
- Extensibilidad: Es muy importante que nuestro proyecto disponga de líneas de trabajo futuro para continuar investigando mejores aproximaciones en algoritmos de reranking.

4.3 Arquitectura

En base a los requisitos especificados se propone una arquitectura compuesta por tres componentes: un indexador para almacenar la información de los documentos y realizar bús-

quedas, un componente de re-ranking y un componente de evaluación. El último componente, correspondiente a la evaluación, se divide en un comparador de calidad con otras técnicas conocidas y un componente de evaluación de estabilidad de parámetros. En la figura 4.1 se muestra un diagrama con la arquitectura del sistema implementado.

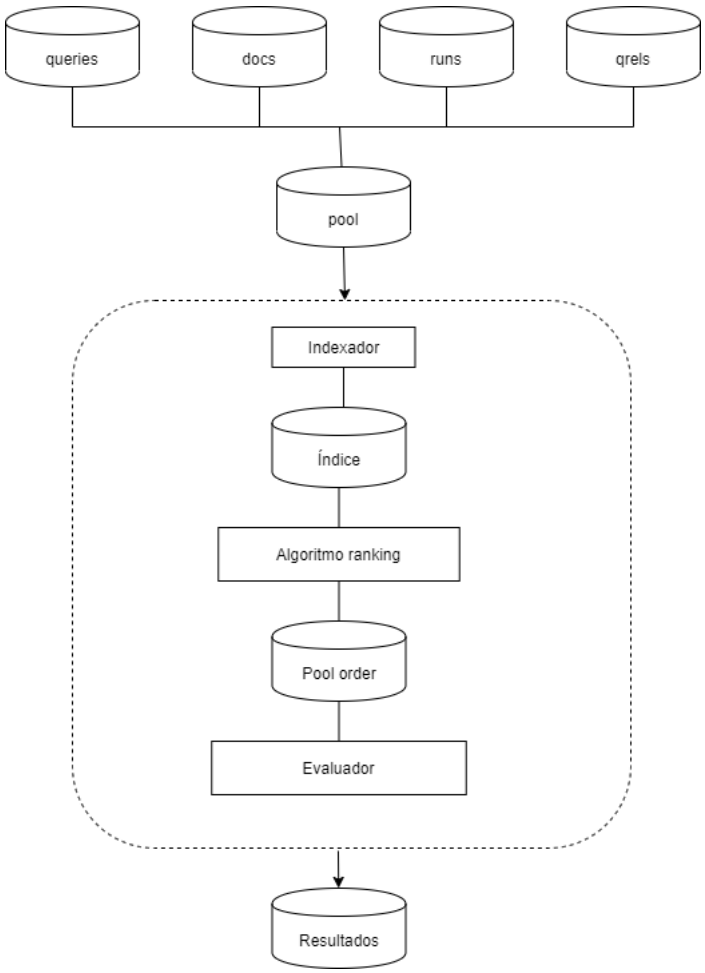

Figura 4.1: Esquema de la arquitectura del sistema.

Los componentes se han diseñado teniendo en cuenta los requisitos necesarios. El componente de indexado es el que se encarga de almacenar en una estructura de índice invertido la información correspondiente a los documentos para luego utilizarla de forma eficiente en las búsquedas. El componente de re-ranking, como su nombre indica, es el que realiza todo el proceso de re-ranking del pool, obteniendo los mejores términos de los documentos relevantes e incrementando las queries con los mismos. Y por último, el evaluador procesa los datos de los re-rankings y genera resultados de comparación visibles.

4.3.1 Casos de uso

El sistema contará con un único usuario, la desarrolladora que realiza los experimentos, la cual podrá llevar a cabo todas las funcionalidades especificadas por los requisitos funcionales. En la figura 4.2 se muestran los casos de uso disponibles.

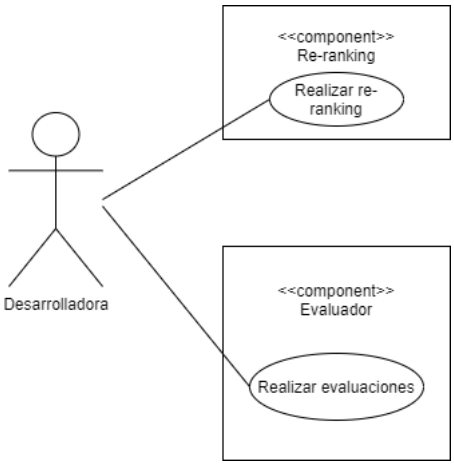

Figura 4.2: Diagrama de casos de uso.

4.3.2 Modelo de datos

El sistema trabaja sobre ficheros almacenados en disco, realizando una lectura de los mismos. Los ficheros correspondientes son los siguientes:

- Los *runs*, diversos ficheros que contienen resultados de sistemas de recuperación de la información ante diversas *queries*, los cuales proporcionan un *ranking* de documentos, en teoría por orden de relevancia. Cada fichero contiene el resultado de un sistema.
- Las *queries*, un único fichero el cual contiene toda la información de las mismas, compuesto por el identificador y el texto de cada una de ellas.
- Un fichero denominado *qrels*, el cual contiene los juicios de relevancia, es decir, para cada query indica que documentos son relevantes después de ser analizados por asesores.
- Los documentos, una gran cantidad de ficheros que almacenan todo el contenido de los documentos empleados por los sistemas, tanto los relevantes como los no relevantes. Estos documentos son almacenados en el indexador para luego acceder a su contenido

de forma eficiente y seleccionar los mejores términos para el incremento de la query original.

4.3.3 Diseño e implementación

A continuación, se lleva a cabo una explicación más extensa del diseño de cada uno de los componentes del sistema propuesto:

Indexador

En la figura que aparece en este apartado (4.3) se muestra un diagrama de clases UML que representa el diseño de clases del componente indexador. Cabe destacar que algunas de las clases que aparecen en este diagrama UML(*Unified Modeling Language*) para el componente de indexación, también se emplean en el componente de Ranking y por tanto se llevará a cabo la explicación de los mismos únicamente en este apartado. Para que las diversas clases con funcionamientos similares o relacionados fueran facilmente reconocibles, se ha agrupado dichas clases en paquetes. Por ejemplo por un lado tenemos un paquete que contiene las estructuras de datos que nos servirán en un futuro para llevar a cabo el funcionamiento de los componentes del sistema:

- La clase Qrels, contiene los datos pertenecientes a los juicios de relevancia, es decir, los documentos que son relevantes para cada *query*.
- La clase Run contiene la información proporcionada por cada sistema de recuperación de información participante. Entre los datos de los que dispone, encontramos por ejemplo un registro en donde se almacenan aquellos documentos que han aparecido en el *ranking* producido por el sistema para una *query*.
- La clase Documento contiene los campos de cada documento, como son el título y el texto entre otros.

Por otra parte, disponemos de un paquete que contiene las clases que procesan los ficheros. Dichas clases se emplean para obtener los datos de los ficheros que sean necesarios en nuestro algoritmo de *re-ranking* para emplearlos cuando sean necesarios de forma eficiente. Las clases empleadas para ello son las que se mencionan a continuación:

- La clase QrelsParser, es la encargada de procesar el fichero de juicios de relevancia.
- La clase RunsParser, que se encarga de procesar el fichero que contiene los *rankings* producidos por los sistemas de recuperación participantes.

- La clase DocumentsParser, que de la misma forma que las clases anteriores, se encarga de procesar el conjunto de documentos empleados en los sistemas de recuperación de información.

La clase principal de este componente se denomina Indexador. Dicha clase contiene los métodos necesarios para indexar la información que luego se empleará en el proceso de re-ranking. La clase aquí mencionada está asociada a otra, conocida como ThreadPool y a su vez esta se encuentra asociada a otra llamada WorkerThread, que implementa la clase Runnable.

La clase ThreadPool se usa principalmente para pasar datos necesarios como por ejemplo la información de los *runs*, de los juicios de relevancia, etc, a la clase WorkerThread que se encarga de indexarlos de forma pararela. Esto se hace así ya que trabajamos con una gran cantidad de datos y es muy importante reducir el tiempo empleado en su procesamiento.

Ranking

En este apartado llevaremos a cabo una explicación del diseño realizado para el componente de Ranking, el cual implica el uso de diferentes paquetes, algunos de ellos ya citados en el apartado anterior, para una mejor comprensión disponemos de la imagen 4.4 que muestra el UML correspondiente. Por ejemplo en el mismo paquete que se encuentran las clases Qrels y Run ya mencionadas, también disponemos de QueryInfo la cual se emplea en el componente actual.
QueryInfo es una clase en la cual se almacena la información correspondiente a la query, compuesta por el identificador de la misma y el texto que la compone. De la misma forma, en el paquete en el que se encuentran QrelsParser y RunsParser también disponemos de QueryInfoParser, que igual que en los otros caso lleva a cabo el procesado del fichero que contiene la información que será necesaria en el componente de Ranking.
La clase principal es AppRanking, esta clase se divide en 3 operaciones principales que explicaremos a continuación:

- En primer lugar disponemos de una operación encargada de generar el fichero que contiene los resultados provenientes de la media para todas las queries de los relevantes encontrados para cada una de ellas en cada iteración del algoritmo. Para ello se realizan todas las variaciones posibles con los parámetros disponibles, entre los que se encuentran:
  - Un parámetro n, que indica el número de términos con los que se va a incrementar la query. En nuestro caso, las posibilidades, son 3, 6, 9, 12 y 15.
  - Un parámetro λ, es el parámetro de suavización empleado en la fórmula de Jelineck-Mercer para el cálculo de la puntuación de los términos cuyo rango es entre 0 y

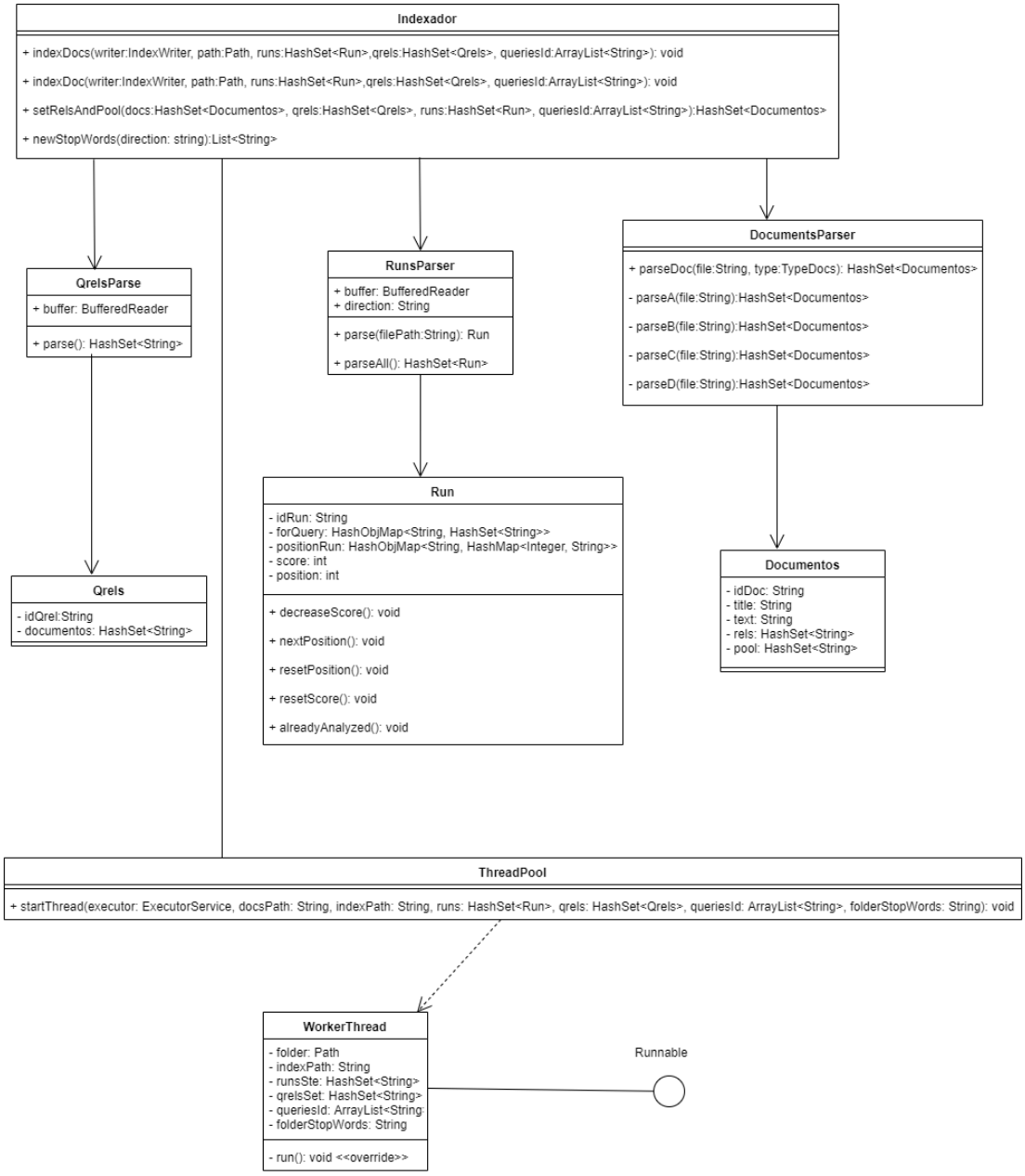

Figura 4.3: Diagrama UML del componente Indexador.

1.

– Un parámetro de interpolación, α, cuyo rango de también está comprendido entre 0 y 1. Dicho parámetro se emplea para dar más importancia a la query original o a la query expandida en función de su valor.

- Otra de las operaciones disponibles, se encarga de realizar un fichero el cual contiene los

valores del mejor resultado de la operación anterior (determina cual es) y los resultados tanto del algoritmo DocId como del algoritmo MTF, los cuales luego se compararan en el último componente de Evaluación.

- La otra operación disponible genera un fichero con diversos experimentos para luego evaluar la estabilidad de los parámetros que han proporcionado el resultado óptimo.

Como podemos apreciar la clase *AppRanking* se emplea principalmente para parsear los datos necesarios en función de los experimentos que se quieran realizar y de proporcionárselos a la clase *Ranking*. Esta clase, contiene una gran cantidad de métodos que en su conjunto desarrollan el algoritmo de re-ranking teniendo en cuenta el contenido de los documentos relevantes.

El procedimiento básico y necesario para comprender como funciona de forma general el algoritmo es el siguiente:

- En primer lugar se lanza la *query* original y se recupera una lista de documentos ordenados.
- Se va analizando uno por uno de forma ordenada si son relevantes, almacenando dicha información. Para saber si son relevantes o no, disponemos de los juicios de relevancia. En el momento que se encuentra el primer relevante se deja de recorrer dicha lista.
- Todos los documentos que se van analizando se almacenan en una estructura para que en las iteraciones venideras no se vuelvan a analizar.
- El documento relevante encontrado se añade al *relevance set*.
- En cada iteración se calcula la puntuación de la *query* en todos los documentos del *relevance set*. En la primera iteración con la *query* original y en las siguientes con la *query* expandida.
- A continuación se calcula la puntuación de los términos que se encuentran en los documentos pertenecientes al *relevance set* en combinación con la puntuación de las *queries*.
- Los términos se ordenan por la puntuación obtenida de mayor a menor.
- Se expande la *query* original con los n términos que se encuentran en las posiciones más altas.
- Se repite el proceso desde el primer paso, pero en este caso no se lanza la *query* original, sino que la expandida, con la información de los n primeros términos obtenidos en los cálculos previos.

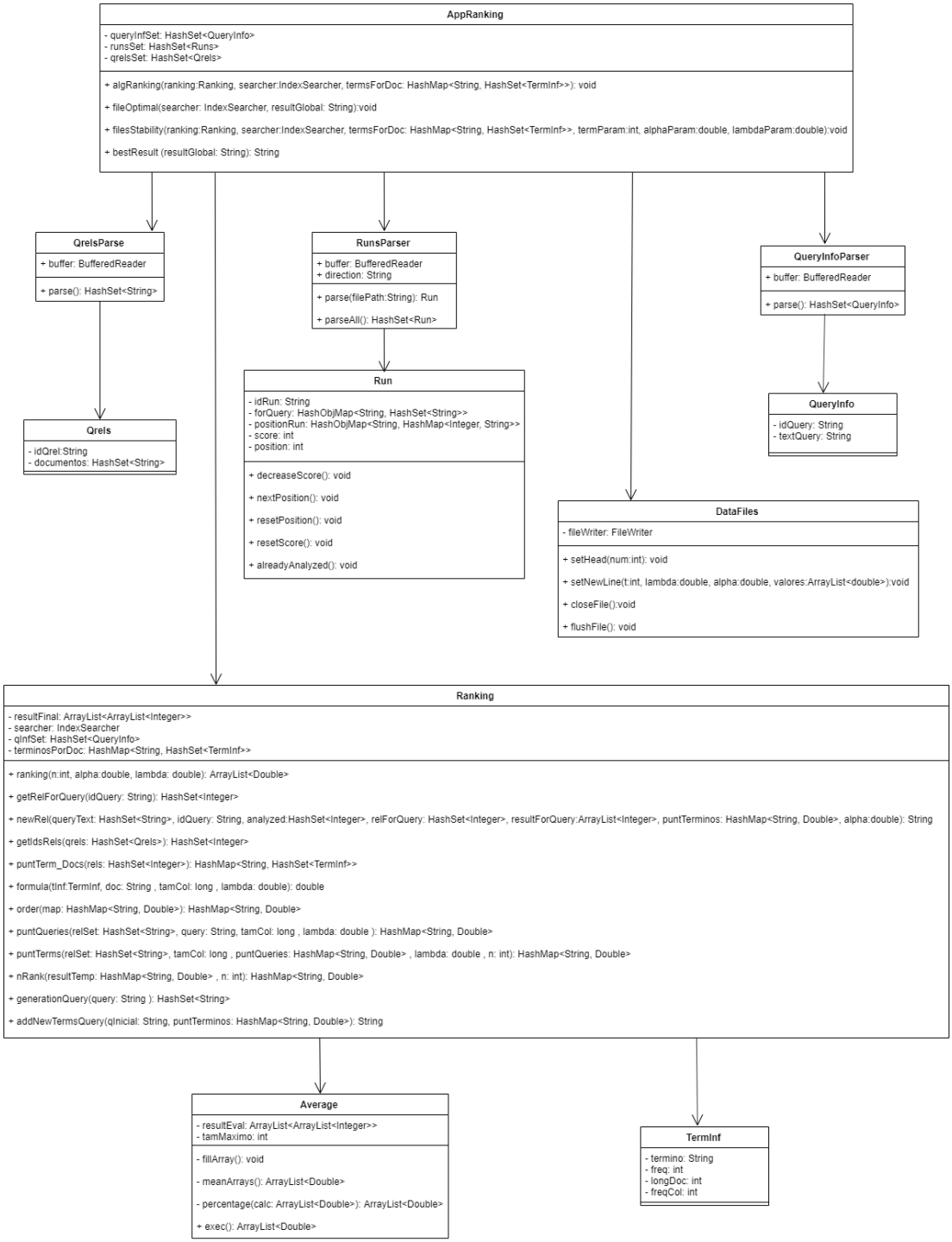

Figura 4.4: Diagrama UML del componente Ranking.

Todo el proceso previamente mencionado se realiza hasta el momento en el que se han encontrado todos los documentos relevantes para la *query* original.

Es importante destacar que para realizar las comparaciones de los resultados del algoritmo se realiza el promedio de los resultados de todas las *queries*, esto se realiza con la clase denominada *Average* visible en el diagrama de clases.

En el diagrama UML también disponemos de una clase llamda DataFiles, la cual se emplea para escribir los ficheros que contienen todos los resultados de los experimentos para luego realizar una comparación y evaluación de los mismos.

Evaluation

El último componente que vamos a representar y explicar es el denominado *Evaluation* para el cual disponemos del UML correspondiente a la figura 4.5.

La clase principal es *Evaluation*, su objetivo es recibir los parámetros necesarios para decidir si se va a hacer una comparación gráfica de el resultado óptimo de nuestro algoritmo junto con DocId y MTF (de esto se encarga la clase Graph), o si por el contrario se va a representar gráficamente para llevar a cabo la evaluación de la estabilidad de los parámetros que se han seleccionado para el resultado óptimo. En cuyo caso, se encargan las clases *Stab_Term_Fixed* y *Stab_Alpha_Lambda_Fixed* dependiendo de los parámetros seleccionados.

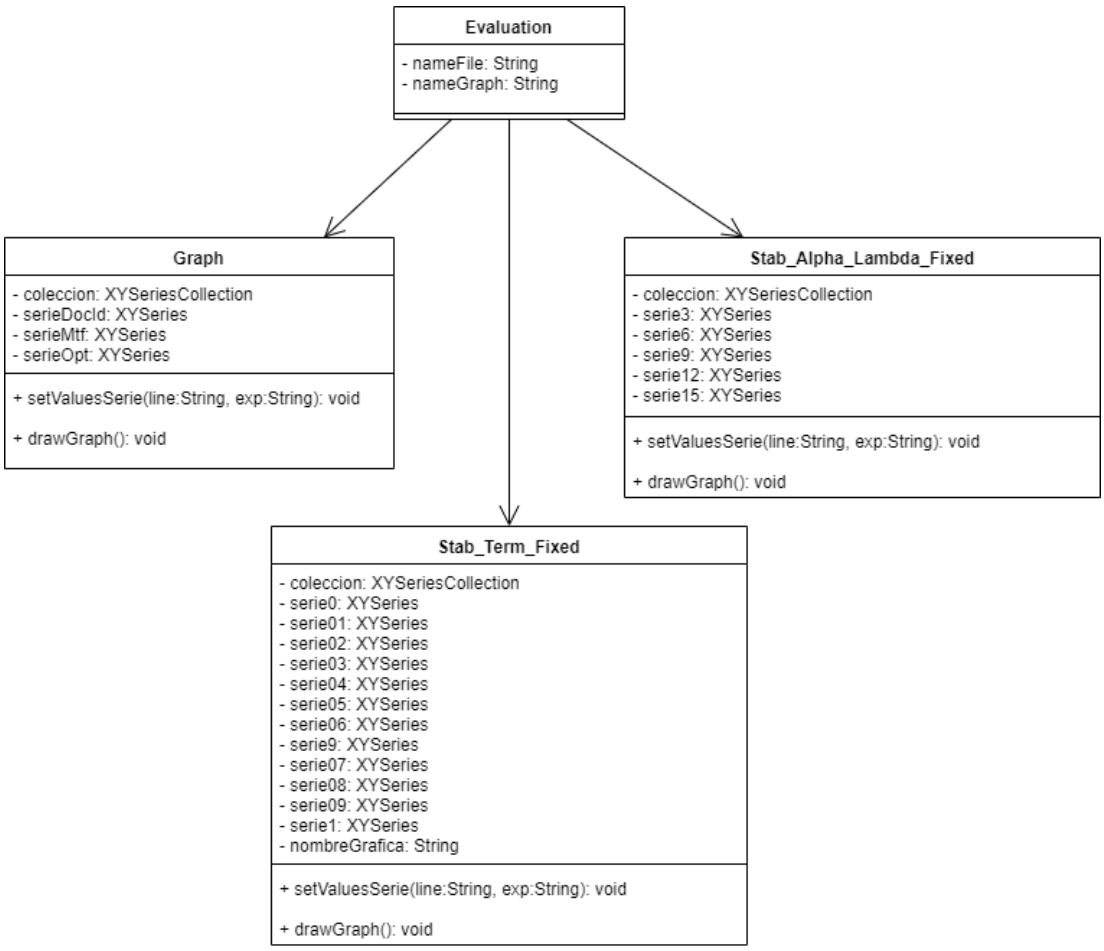

Figura 4.5: Diagrama UML del componente Evaluation.

Capítulo 5

Experimentos y resultados

En este capítulo analizaremos los resultados obtenidos en los experimentos. Comenzaremos hablando de las características del proceso de evaluación, tales como la elección de datos sobre las que realizar los experimentos, las métricas seleccionadas para la evaluación o la metodología de evaluación. A continuación, mostraremos y analizaremos los resultados obtenidos, comparándolos con una serie de algoritmos de referencia.

5.1 Proceso de evaluación

En esta sección explicaremos los detalles y las elecciones abarcadas por la evaluación de los resultados: Las colecciones de datos empleadas, las métricas seleccionadas para la evaluación del sistema, la metodología de evaluación seguida y los algoritmos seleccionados a modo de referencia.

5.1.1 Colecciones

Para la realización de los experimentos se ha seleccionado la colección de datos TREC 5, teniendo en cuenta las siguientes características:

- Documentos: El número de documentos a analizar estimado es de 133681.
- *Queries*. En total TREC5 contaba con un total de 50 *queries* a procesar.
- *Pooled runs*: La cantidad de sistemas de recuperación de la información participantes son 77 automáticas y 24 manuales.

5.1.2 Métricas de evaluación

Para realizar la evaluación de un sistema de recuperación de información disponemos de métricas que miden la cantidad de elementos relevantes y no relevantes se han recuperado, y con ellos la calidad del ranking que se genera. Entre estas métricas disponenos por ejemplo, de la precisión y la exhaustividad (*recall*). Para su cálculo se basan en un conjunto de juicios de relevancia que indican si un documento es relevante o no para la *query*.

5.1.3 Metodología de evaluación

Para realizar la evaluación de nuestro algoritmo de *re-ranking* (determinar cual de los experimentos realizados modificando las variables posibles es el mejor) y llevar a cabo la comparación de los resultados obtenidos con los ya existentes proporcionados por otros algoritmos de pooling previamente implementados debemos seleccionar la metodología más adecuada. En nuestro caso hemos seleccionado la métrica conocida como *recall*, ya explicada en el capítulo de conceptos, la cual indica la proporción de documentos relevantes recuperados del total de documentos relevantes. Cuanto más se aproxime dicho valor a 1 mejor será la implementación realizada y por lo contrario cuanto más se acerque a 0 peores serán los resultados obtenidos.

En la primera fase que consiste en determinar cual de los múltiples experimentos realizados con el intercambio de los parámetros disponibles proporciona un mejor resultado, inicialmente optamos por quedarnos como mejor resultado aquel que obtenía todos los relevantes lo antes posible, lo cual comprobamos que no era válido ya que muchos resultados alcanzaban dicha meta en el mismo punto. La conclusión final fue obtener aquel que contuviera la máxima área bajo la curva, es decir, que la suma del *recall* en todos los puntos hasta encontrar el 100% de los documentos relevantes fuera lo más alta posible.

Para que la comparación de nuestro algoritmo con otros ya existentes fuera más visual se realizaron unas representaciones gráficas de los experimentos en conjunto, las cuales se muestran en las secciones venideras. Es importante comprender que se representa en cada eje:

- En el eje de las X (abscisas) se muestra el número de documentos juzgados en cada punto.
- En el eje de las Y (ordenadas) se muestra el valor de *recall* en cada punto.

5.1.4 Algoritmos de referencia

Para evaluar correctamente el re-ranking obtenido para los sistemas de recuperación de la información, es necesario realizar una comparación con otros algorimtos que sirven como referencia, o lo que también se conoce como *baselines*. Para este proyecto se han escogido dos algoritmos de pooling, los cuales no emplean el contenido de los documentos para incrementar la query. A continuación se describen los *baselines* elegidos:

- **DocId**: Es un algorimto de pooling en el cual los documentos se ordenan de forma alfanumérica a través del identificador de la misma. A continuación, se juzgan de uno en uno en dicho orden.
- **MTF**: Este método ya se ha explicado en el capítulo de conceptos de forma más extensa. Es un método de pooling el cual dispone de una puntuación de prioridad para cada sistema de recuperación de información participante, las cuales son uniformes. Se selecciona una sistema de recuperación de información participante de aquellos que cuentan con la puntuación más alta de forma aleatoria y se juzga el primer documento en la clasificación. Si es relevante, se continúan juzgando por orden el resto de documentos hasta el punto en el que se encuentra uno que no es relevante en el cual se disminuye la puntuación de prioridad de ese sistema de recuperación de información participante y se selecciona otro de forma aleatoria de los que tienen mayor puntuación de prioridad.

5.2 Resultados

En esta sección nos centramos en exponer los resultados obtenidos, comparándolos con los algoritmos de referencia y analizando el impacto de los diferentes parámetros del sistema.

5.2.1 Parámetros del sistema

El sistema dispone de dos tipos de parámetros, los que son únicamente para la lectura de los ficheros necesarios para su ejecución y aquellos que proporcionan cambios en los resultados de las ejecuciones. En el primer grupo, para la lectura de los datos necesarios disponemos de los siguientes parámetros:

- qrelsPath, sitio de disco en donde se encuentra el fichero con los juicios de relevancia

para proceder a su lectura.

- runsPath, sitio de disco en donde se encuentran los ficheros que contienen la información correspondiente a los sistemas de recuperación de información participantes.
- queryInfoPath, sitio de disco en donde se encuentra el fichero con el identificador y el contenido de cada query.
- indexPath, la ubicación del indexador donde está almacenado todo el contenido de los documentos para acceder a ellos de forma eficiente.
- typeExec, el tipo de ejecución que queremos realizar, esto indica si deseamos realizar el cálculo del resultado óptimo, o por ejemplo el cálculo correspondiente a la estabilidad de parámetros. La estabilidad de parámetros determina como influye la variación única de uno de los parámetros disponibles en los resultados obtenidos en los experimentos.

Por otro lado, vamos a enumerar los parámetros que afectan al resultado de los experimentos:

- **Valor de suavización**: El valor de suavizado empleado para calcular la puntuación de los términos, dicho valor oscila entre cero y uno, ambos inclusive.
- **Valor de interpolación**: Este valor se usa para proporcionar más o menos importancia a la query expandida con los nuevos términos y de forma opuesta proporcionar menos o más importancia a la query original.
- **Número de términos**: Este valor consiste en el número de términos con el que vamos a expandir la query original. En nuestro caso hemos seleccionado como posibilidades, los números tres, seis, nueve, doce y quince.

5.2.2 Resultados preliminares

Antes de realizar los experimentos más avanzados, se realizaron una serie de experimentos previos, con una escala de datos más manejables que los experimentos finales. En estos primeros experimentos se redujo el número de *queries* empleadas. Reducir las *queries* empleadas también reducía de forma notoria los documentos procesados, ya que solo era necesario trabajar con aquellos documentos que aparecían para dicha *query* en alguno de los sistemas de recuperación de la información participantes.

También destacar que en las primeras pruebas no empleamos el parámetro de interpolación, y como parámetro de suavizado solo usábamos 0.7 que es el valor que se suele usar más comúnmente.

Gracias a esta reducción tanto de posibles valores de parámetros como de cantidad de tiempo se redujo el tiempo de las primeras ejecuciones.

5.2.3 Resultados finales

A la vista de que los cálculos de las puntuaciones de los términos y la obtención de los documentos se realizaba de forma satisfactoria, se decidió obtener ya los resultados finales, combinando todos los parámetros posibles, tanto λ, α como el número de términos explicados en la sección 5.2.1.

Es importante destacar que se tuvo en cuenta la importancia de las *stopwords*, ya que una buena elección de las mismas puede provocar cambios notorios en los resultados de la experimentación, a continuación se va a llevar a cabo una explicación de los experimentos realizados y de los resultados obtenidos:

Inicialmente realizamos una prueba donde para la indexación de los documentos se empleó una colección de *stopwords* disponible a través de una librería de Lucene, conocida como *EnglishAnalyzer*. Al finalizar la indexación, obteníamos un total de 2022600 términos indexados, siendo los que más frecuentes en cuanto a número de ocurrencias los siguientes: *from, wich, have, said*, etc.

El resultado óptimo (considerado aquel que tiene mayor área bajo la curva de *recall*) se obtuvo con los parámetros n = 9, λ = 0.7 y α = 0.6, cuyos resultados comparados con las técnicas docId y MTF se muestran en la figura 5.1.

Si analizamos la gráfica podemos comprobar que el resultado del algoritmo que hemos implementado es bastante bueno, siendo el peor DocId. Al comienzo MTF obtiene mejores resultados, pero a medida que avanzan las iteraciones nuestro algoritmo supera la calidad de MTF. Para ser más exactos cuando se han analizado 2000 documentos, nuestro algoritmo tiene un porcentaje de *recall* de aproximadamente un 0.94% mientras que MTF dispone de un 0.85%. Como hemos mencionado previamente cuanto más se acerque a 1 mejores serán los resultados obtenidos, por esto mismo nuestro algoritmo es mejor.

Una vez se realizó la primera prueba decidimos comprobar si realmente la elección de las *stopwords* afectaba tanto como aparentaba en la teoría, para ello realizamos la indexación de los documentos con una colección de palabras elegidas de forma manual. Al finalizar dicha indexación obtuvimos un total de 2351066 términos indexados, siendo los más frecuentes en cuanto a número de ocurrencias los siguientes: *will, year, new, last,*etc.

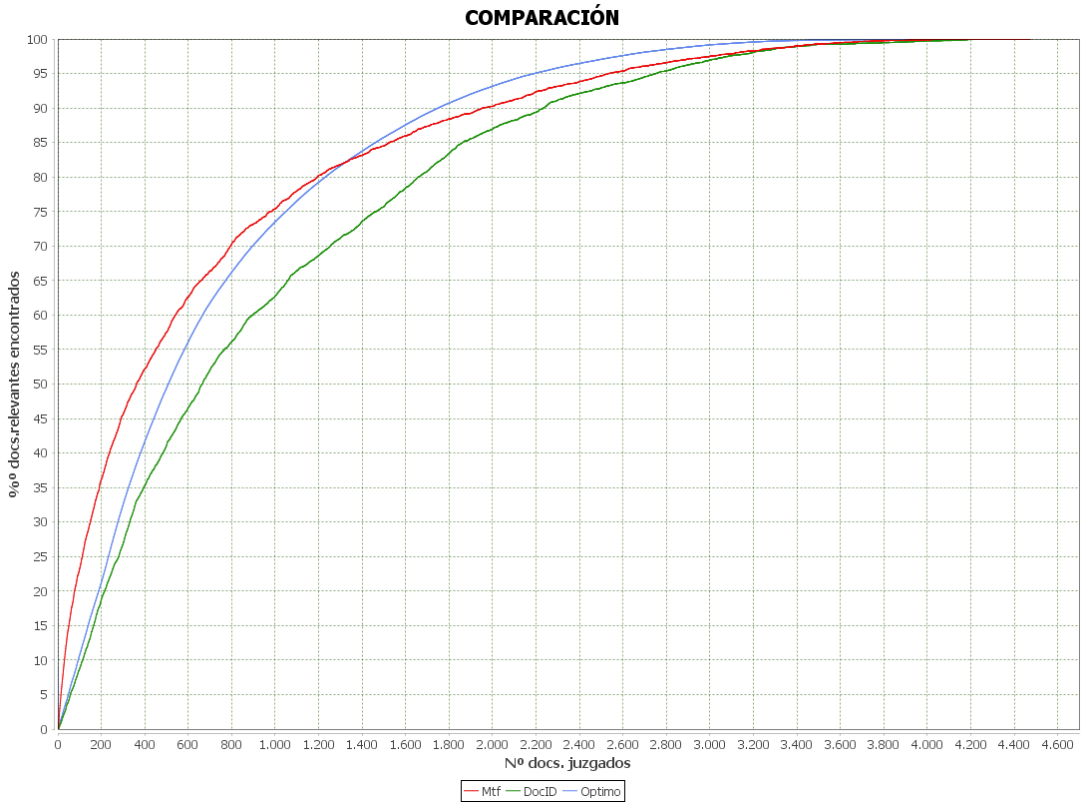

Figura 5.1: Comparación resultado óptimo del algoritmo con DocId y MTF.

En el segundo experimento el resultado óptimo se obtuvo con los parámetros n = 12, λ = 0 y α = 0, cuyos resultados comparados con las técnicas docId y MTF se muestran en la figura 5.2

Al analizar la gráfica podemos comprobar que el resultado proporcionado por el algoritmo que hemos implementado es todavía mejor, superando a MTF cuando ha juzgado solamente unos 600 documentos (en el experimento anterior es necesario analizar algo menos de 1400 documentos para que esto suceda). También cabe destacar que una vez analizados 2000 documentos ha mejorado de un 0.94% a un 0.95% en valores de *recall*.

Si analizamos detalladamente el resultado obtenido por el segundo experimento podemos apreciar que el mejor resultado es aquel en el que no se realiza ningún tipo de suavización y en el que no se le da nada de importancia a la *query* expandida. Es decir, la propia query original produce muy buenos resultados, lo cual se debe a que muchos términos que provocaban interferencias en la *query* original han sido eliminados en la indexación de los documentos.

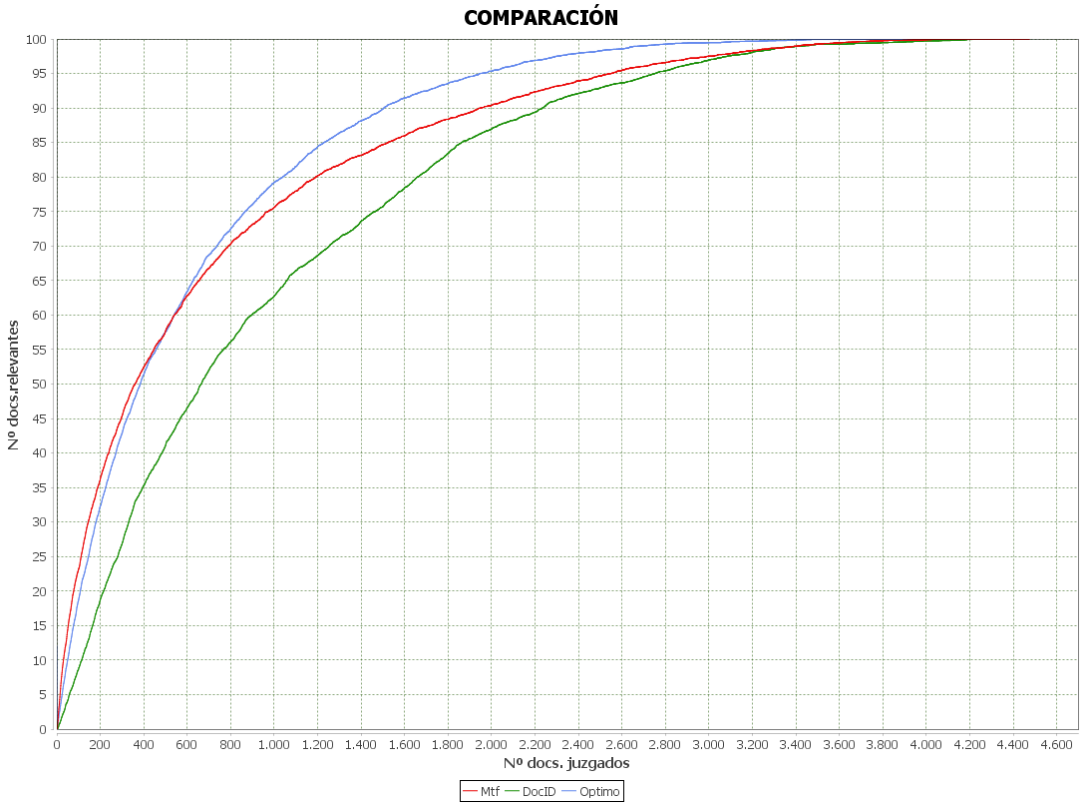

Figura 5.2: Comparación resultado óptimo del algoritmo con DocId y MTF.

5.2.4 Estabilidad de parámetros

Una vez se han encontrado los parámetros óptimos, se realiza un análisis de comportamiento del sistema variando un sólo parámetro de cada vez, mientras los demás se mantienen fijos. Los resultados que se presentan a continuación son con el resultado óptimo obtenido para el primer experimento con las *stopwords* proporcionadas por la librería *EnglishAnalyzer* ya que los resultados de ambos experimentos a simple vista son prácticamente idénticos y con analizar uno de ellos es suficiente.

En la figura 5.3 podemos observar como varían los resultados variando el parámetro α (interpolación). En la parte inferior de la gráfica se puede apreciar una leyenda la cual indica que color se corresponde con el resultado de la variación producida por cada valor posible de α en saltos de 0,1. La variación de dicho parámetro es la más inestable de todas a pesar de que no presenta una variación demasiado notoria.

En la figura 5.4 podemos observar como varían los resultados variando el parámetro λ (suavización). En la parte inferior de la gráfica se puede apreciar una leyenda la cual indica

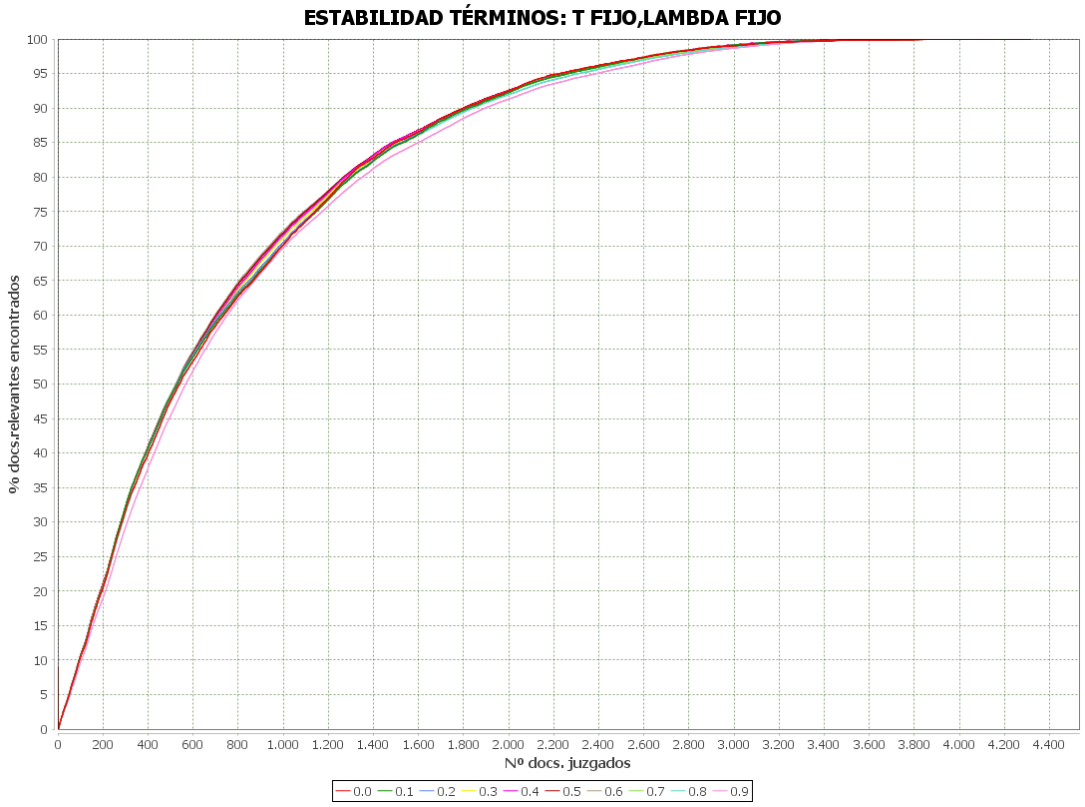

Figura 5.3: Variación del parámetro α, manteniendo λ y n fijos

que color se corresponde con el resultado de la variación producida por cada valor posible de λ en saltos de 0,1. Dicho parámetro es bastante estable.

Por último, en la imagen 5.5 podemos observar como varían los resultados variando el parámetro n (número de términos añadidos a la *query* original). En la parte inferior de la gráfica se puede apreciar una leyenda la cual indica que color se corresponde con el resultado de la variación producida por cada valor posible de n. Dicho parámetro es muy estable llegando al punto de que es prácticamente imposible distinguir las variaciones que se producen a simple vista en la gráfica.

5.2.5 Discusión

Observando los resultados comprobamos la importancia que tiene el preprocesado de la información, en este caso las stopwords seleccionadas, que como ya se ha mencionado previamente son palabras muy comunes que no nos interesa almacenar ya que no nos van a proporcionar información relevante.

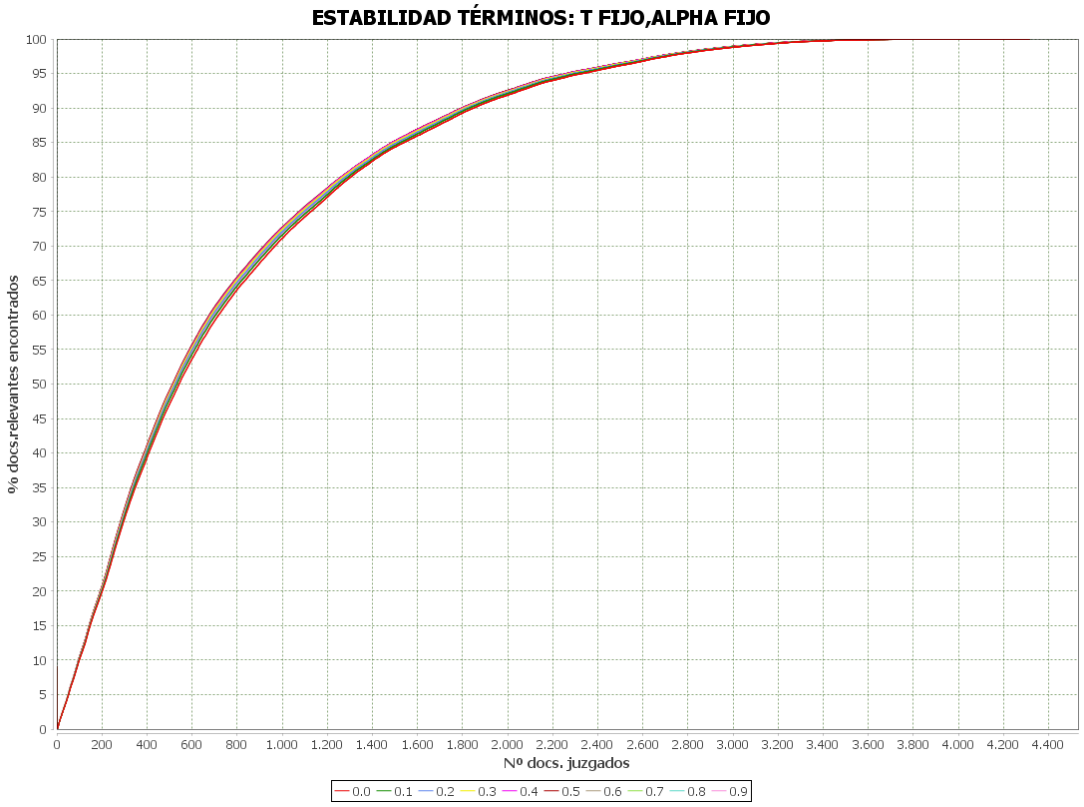

Figura 5.4: Variación del parámetro λ, manteniendo α y n fijos

Concretamente el mejor resultado en la segunda prueba es el que le proporciona toda la importancia a la query original y por lo tanto aunque añadimos en teoría doce términos no los tiene en cuenta, eso es debido a que la query original esta funcionando muy bien por si misma ya que muchos términos innecesarios ya no aparecen en los documentos y por tanto no influyen en la clasificación de los documentos.

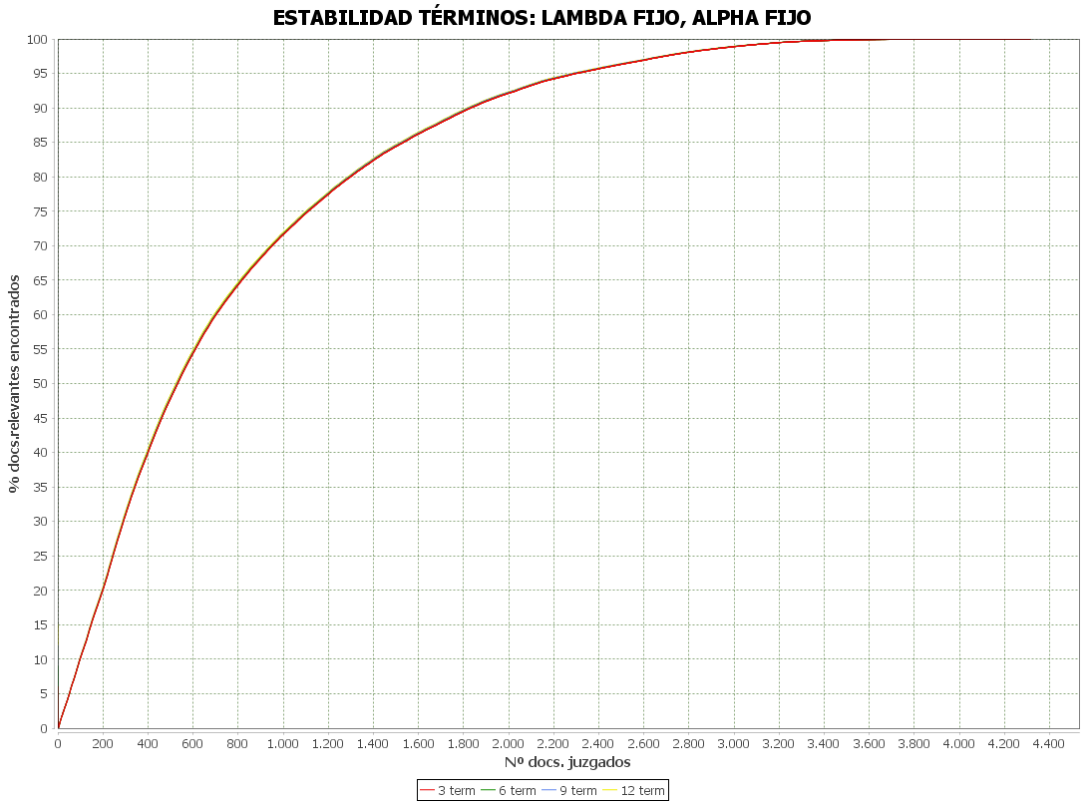

Figura 5.5: Variación del parámetro n, manteniendo α y λ fijos

Capítulo 6

Proceso de Ingeniería

En este capítulo se detalla el proceso de ingeniería seguido en el desarrollo del proyecto. Para comenzar explicamos la metodología de desarrollo elegida, una metodología ágil conocida como Scrum. A continuación, se lleva a cabo la explicación de la gestión del proyecto, como la descomposición en historias de usuario, la estimación de tiempo, los recursos y la gestión de riesgos.

6.1 Elección de la metodología

Si deseamos que el desarrollo del proyecto se realice de forma exitosa tenemos que comenzar eligiendo que metodología se empleara en ello.
El proyecto aquí desarrollado es de investigación y por tanto la metodología debe adaptarse a los requerimientos de un proyecto de estas características. Los requisitos metodológicos para un proyecto de este estilo son los siguientes:

- Desarrollo iterativo: Los pasos llevados a cabo en una investigación se deciden en función de los resultados obtenidos, por ello es necesario dividir el desarrollo del mismo en iteraciones. Así, cuando finaliza una iteración se pueden estudiar los resultados obtenidos para adaptar los siguientes pasos de forma adecuada, incorporando el conocimiento adquirido.

- Cambios en los requisitos: Los requisitos de este tipo de proyectos varían con mucha frecuencia en función de los resultados previos. Por tanto, es necesario seleccionar una metodología que proporcione medios para la gestión de cambios en los requisitos.

- Gestión temprana de riesgos: Es necesario detectar los riesgos que pueden surgir a lo largo de la investigación cuanto antes para minimizar su impacto y evitar que no se pueda llevar a cabo el proyecto de forma exitosa.

Todas las restricciones pueden satisfacerse empleando una metodología ágil de desarrollo. Dichas metodologías definen una aproximación iterativa e incremental, con el trabajo dividido en ciclos de desarrollo cortos enfocados en conseguir un producto que sea funcional.

El desarrollo incremental proporciona la posibilidad de detectar rápidamente los problemas minimizando por lo tanto el riesgo que ellos conllevan.

Por tanto la metodología Scrum ha sido la seleccionada para el desarrollo de este proyecto, la cual detallaremos a continuación.

6.2 Scrum

Scrum[1] es un *framework* de desarrollo ágil, iterativo e incremental desarrollado por Ken Schwaber y Jeff Sutherland. Como introducción, un *framework* es un conjunto de conceptos, prácticas y criterios para enfocar un tipo de problema en particular.

6.2.1 Características

En esta sección se explican las características más importantes de Scrum y los beneficios que se derivan de ellos. Todo esto es lo que ha determinado la elección de dicha metodología entre otras muchas existentes.

Metodología iterativa e incremental

Como ya hemos comentado previamente Scrum realiza una división del trabajo en iteraciones. Las cuales suelen abarcar una duración de entre una y cuatro semanas, de tal forma que cuando finaliza cada una de ellas se obtiene un incremento del producto. Permite gestionar los cambios de manera más eficaz y los riesgos que van surgiendo.

Gestión de cambios

Scrum determina que el problema a resolver no puede ser totalmente definido desde el principio del desarrollo, esto puede ser por cambios en los requisitos del cliente o porque los

[1] https://www.scrum.org

resultados pueden proporcionar nuevos retos a resolver. Por tanto se centra en mejorar la rapidez de entrega para así trabajar con nuevos requisitos si es necesario. En los proyectos de este tipo, es decir, investigación, se parte de una idea base a desarrollar, pero donde a lo largo de la misma pueden aparecer nuevos caminos hacia donde orientarse.

Gestión de riesgos

Gracias a la metodología en Scrum, dividida en ciclos cortos de desarrollo, disponemos de una gestión temprana de riesgos, lo que nos permite tomar medidas antes de que se produzcan grandes desviaciones las cuales pongan en peligro el éxito de la investigación.

Transparencia

El principio de transparencia requiere que los aspectos más importantes del desarrollo se definan con un estándar común, de forma que todos los observadores sean capaces de comprenderlos de la misma forma.

Inspección

Los usuarios de Scrum deben inspeccionar el progreso hacia el objetivo de la iteración para detectar variaciones no deseadas. Esto ayuda a realizar una revisión de la calidad del producto.

6.2.2 Estructura y adaptación

En Scrum disponemos de tres tipos de elementos conocidos como roles, eventos y artefactos, los cuales están unidos a través de reglas.
A continuación, en la figura 6.1 se muestra el flujo de trabajo de la metodología Scrum. El *Product Backlog* es donde se recoge el trabajo que se realizará, el cual se divide en *sprints* o interacciones que abarcan entre una y cuatro semanas. Cuando comienza un *sprint* se realiza una reunión de equipo (*Sprint Planning*) en la cual se decide cuales de los elementos constituyentes del *Product Backlog* se van a realizar en el sprint correspondiente. Esto conlleva a que los elementos que se realizarán en dicho *sprint* se retiran del *Product Backlog* y se mueven al *Sprint Backlog*.

A medida que transcurre el *sprint*, los desarrolladores constituyentes del proyecto se autoorganizan para implementar las funcionalidades objetivo, a mayores de forma paralela realizan unas breves reuniones diarias (*Daily Scrum*) para mantenerse al día de los avances que realiza cada uno de los miembros. Una vez que el *sprint* finaliza, se presenta el trabajo terminado a los interesados en una reunión (*Sprint Review*) y se realiza una reflexión sobre la

iteración concluida (*Sprint Retrospective*), en la cual cada uno de los participantes comenta su perspectiva respecto a los siguientes puntos:

- Que ha funcionado bien.
- Que hay que mejorar en la siguiente iteración.
- Que se ha aprendido.
- Que le habría gustado hacer.

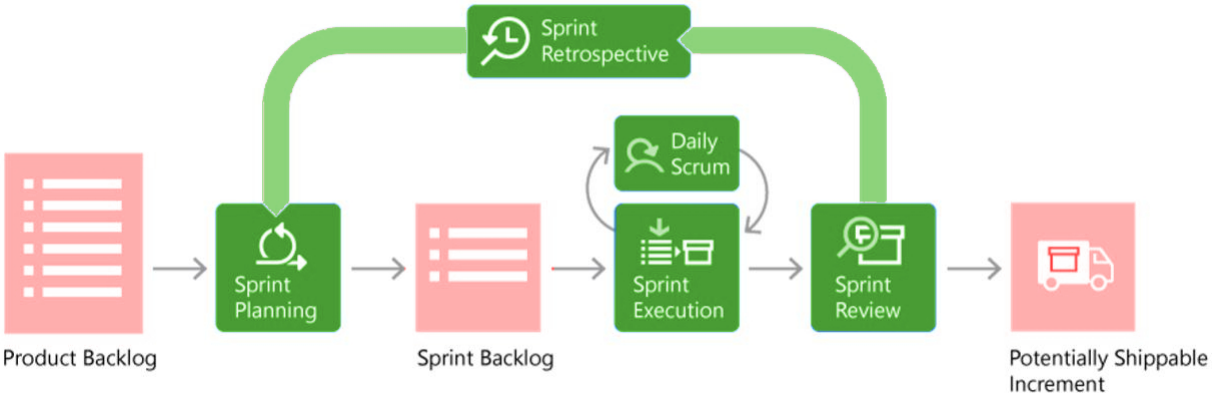

Figura 6.1: Flujo de trabajo en Scrum.

A continuación, se lleva a cabo una explicación de los elementos constituyentes de Scrum.

Roles Scrum

El equipo se encarga de entregar los incrementos en cada iteración, los cuales deben ser funcionales y reutilizables. El equipo esta formado por tres roles que explicaremos en breve. El equipo se caracteriza por ser auto-organizativo y multifuncional lo cual hace que no dependan de nadie externo al equipo teniendo todas las competencias necesarias. Los roles son los siguientes:

- Dueño del producto (*Product Owner*): Hace de puente de comunicación entre el cliente y el equipo. Se encarga de mantener el *Product Backlog*:
  - Crear historias de usuario, descripciones informales en lenguaje natural de la funcionalidad del sistema entendible para el usuario.
  - Ordenación y priorización de los elementos constituyentes de *Product Backlog*, es decir, las historias de usuario.

  - Asegurarse de que el *Product Backlog* es comprensible y transparente para todos.

  En dicho proyecto, el rol *Product Owner* ha sido asignado a Álvaro Barreiro y Javier Parapar, directores del proyecto, ya que son personas experimentadas en este tipo de proyectos y estan capacitados para definir las historias de usuario y la dirección de la investigación.

- Equipo de desarrollo: Formado por las personas que realizan las entregas potencialmente utilizables en cada *sprint*. Suele estar formado por un grupo de entre tres y nueve personas, aunque el número óptimo son siete. Dicho grupo trabaja como un todo, compartiendo las responsabilidades independientemente de las habilidades concretas de cada uno.

  En dicho proyecto el equipo de desarrollo esta formado por única desarrolladora, la alumna de ingeniería informática Jessica Penela Fernández, debido a esto se han tenido que realizar adaptaciones en algunos elementos de la metodología.

- *Scrum Master*: Facilita el trabajo del equipo eliminando los impedimentos y las distracciones. Se encarga también de que se cumplan las prácticas, valores y reglas de la metodología, animando al equipo a mejorar. Este rol no se tiene en cuenta en dicho proyecto ya que es desarrollado por una única persona.

Artefactos

Los artefactos de Scrum están pensados para maximizar la transparencia de la información clave, para que todos lo entiendan de la misma forma. Los artefactos son los siguientes:

- *Porduct Backlog*: Una lista ordenada de todo lo que es necesario realizar, las funcionalidades, los requisitos, las mejoras y las correcciones. Esta lista está formada por historias de usuario, ya citadas anteriormente.

  Inicialmente esta formado por una primera versión pero que no será la definitiva ya que ira evolucionado a medida que el producto y el entorno lo requieran. En nuestro proyecto está formado por las ideas que constituyen el anteproyecto, pero a medida que se van implementando cosas, va cambiando de manera enriquecedora y por lo tanto se van definiendo nuevas historias de usuario y creando nuevos *sprints*.

- *Sprint Backlog*: Es la lista de elementos del Product Backlog que se desarrolla en cada *sprint* para alcanzar el objetivo fijado y crear el siguiente incremento. Dicha lista se crea en el *Sprint Planning* y una vez creada solo se puede modificar si es necesario por el equipo de desarrollo.

- Incremento: Es la suma de todos los elementos que contituyen el *Product Backlog* que se han completado a lo largo del *sprint* e integrados con las anteriores iteraciones. Los incrementos deben ser funcionales y estar en una condición utilizable.
- Gráfica de evolución del proyecto (*burn down chart*): Muestra de forma gráfica el trabajo restante al inicio de cada *sprint* y el que debería quedar en caso ideal.

Eventos Scrum

En Scrum disponemos de una serie de eventos predefinidos con el objetivo de regular y minimizar la necesidad de reuniones los cuales tienen una duración máxima. Los *sprints* como excepción, no se pueden alargar ni acortar una vez que se empiezan, en cambio, el resto de eventos se pueden terminar una vez finalicen o se alcance el objetivo de los mismos. De esta forma, no se desperdicia el tiempo.

- *Sprint*: También conocido como iteración, es una unidad básica de desarrollo con una duración fija una vez que se inicia. Durante dicho tiempo el equipo trabaja para alcanzar un objetivo. El trabajo se detiene al llegar a un tiempo límite y en ese momento se evalúa la cantidad de trabajo conseguido. El tiempo de duración suele abarcar entre una y cuatro semanas, siendo dos semanas la opción más común.
  Esta duración es bastante breve con el objetivo de reducir la complejidad y el riesgo. Los *sprints* permiten analizar resultados y reconducir la investigación.
  En cada *sprint* se realiza una reunión de planificación, denominada *Sprint Planning*, una reunión diaria denominada *Daily Scrum*, la revisión del *sprint* denominada *Sprint Review* y una reunión retrospectiva denominada *Sprint Retrospective*. La explicación de todos los eventos aquí citados se realiza a continuación.
- *Sprint Planning*: Cuando comienza un *sprint* se hace una planificación del mismo. Esta reunión se realiza con el objetivo de seleccionar que elementos del *Product Backlog* se realizarán creando con ellos el *Sprint Backlog*. Estos elementos son seleccionados por los miembros del equipo de desarrollo.
  En nuestro caso, las reuniones de *Sprint Planning* se han realizado al principio de cada iteración con los dueños del producto y la única persona perteneciente al equipo de desarrollo.
- *Daily Scrum*: Consiste en una reunión realizada de forma diaria de unos quince minutos para evaluar el progreso del proyecto. Todos los miembros del equipo tienen que intentar responder a las siguientes preguntas:
  - ¿Qué trabajo hiciste ayer para contribuir al alcance del objetivo del *sprint*?

  - ¿Qué planeas hacer hoy para contribuir al alcance del objetivo del *sprint*?
  - ¿Hay algún problema que podría llevar a cabo el impedimento de que algún miembro del equipo no sea capaz de cumplir los objetivos?

  Como el equipo está constituido por una única desarrolladora no se han realizado dichas reuniones.

- *Sprint Review*: Es una reunión con una duración de no más de cuatro de horas, realizada en la finalización del *sprint*, en ella se realiza una revisión tanto del trabajo completado como del que no. Se presenta el trabajo completado al cliente y los participantes realizan un debate sobre que es lo que se debería realizar a continuación.
  En la realización de este proyecto, las reuniones han consistido en presentaciones del trabajo realizado por la desarrolladora a los dueños del producto, los cuales han llevado a cabo una revisión del *Product Backlog*.

- *Sprint Retrospective*: Una reunión de como mucho tres horas realizada entre el *Sprint Review* y el siguiente *Sprint Planning*. En dicha reunión el equipo reflexiona sobre la iteración finalizada identificando mejoras, errores, cosas buenas, etc.
  En nuestro proyecto no se han realizado estas reuniones, al ser un equipo formado por una única desarrolladora.

6.3 Gestión del proyecto

Con la finalidad de obtener el éxito del proyecto debemos realizar una buena gestión del mismo. Dicha gestión permite realizar el trabajo cumpliendo con los tiempos y el coste estimados y alcanzando un amplio nivel de calidad. Esto también permite realizar un seguimiento para ir analizando el progreso en cada momento, detectando desviaciones indeseadas. A continuación se lleva a cabo una explicación de dicho proceso, explicando estimaciones de historias, asignación de recursos y gestión de riesgos.

6.3.1 *Product Backlog*

A continuación se indican las historias de usuario que forman parte del *Product Backlog* de este proyecto, agrupadas por categorías, para facilitar la comprensión de las mismas.

- **Estudios iniciales**

Es necesario realizar un primer estudio sobre los temas que se tratarán para poder abordar la construcción del sistema.

  - **H1: Análisis general de la literatura de los sistemas de recuperación de información**
  - **H2: Análisis general de la literatura de los algoritmos de pooling**

- **Procesado, lectura y almacenamiento de datos**
  - **H3: Lectura de las queries solicitadas a los sistemas de recuperación**
  - **H4: Lectura de los documentos pertenecientes a TREC 5 empleados por los sistemas de recuperación**
  - **H5: Lectura de los resultados de las ejecuciones proporcionadas por los sistemas de recuperación participantes en TREC 5**
  - **H6: Lectura de los juicios de relevancia**
  - **H7: Almacenamiento de la información necesaria en un índice invertido, aproximación inicial**
    Una vez se han leído todas las colecciones necesarias, es necesario almacenarlas en una estructura eficiente para poder acceder a ellas posteriormente de forma eficiente. En el primer caso se realiza un índice empleando las soptwords pertenecientes al *EnglishAnalyzer*
  - **H8: Almacenamiento de la información necesaria en un índice invertido, optimización**
    Para obtener un resultado más eficiente en los algoritmos se crea un segundo índice con una selección concreta de las stopwords.

- **Implementación de técnicas de pooling ya existentes**
  Se lleva a cabo una implementación de dos algoritmos de re-ranking existentes para posteriormente llevar a cabo una comparación de nuestro algoritmo con los mismos y comprobar si se produce una mejora en los resultados obtenidos.
  - **H9: Implementación del algoritmo de pooling DocId.**
  - **H10: Implementación del algoritmo de pooling MTF.**

- **Implementación y evaluación de técnicas propuestas**
  - **H11: Implementación de suavizado Jelineck-Mercer**
  - **H12: Implementación del algoritmo de re-ranking con contenido de los documentos, aproximación inicial**

Inicialmente partimos de un algoritmo incorporando el top n de términos sin ponderar la puntuación de lo mismos.

  - **H13: Implementación del cálculo de la puntuación de la query en cada documento**
  - **H14: Implementación del cálculo de la puntuación de los términos en combinación con las queries para cada documento**
  - **H15: Implementación del algoritmo de re-ranking con contenido de los documentos, aproximación avanzada**
    Se modifica el algoritmo de re-ranking para tener en cuenta el top n de documentos con la puntuación asociada de cada uno de ellos en cada documento.
  - **H16: Elaboración de experimentos**
    Una vez desarrolladas todas las técnicas propuestas, se realizan los experimetos oportunos, usando los resultados como guías para enfocar los siguientes.

- **Elaboración de la memoria**
  La redacción de la memoria puede dividirse en los capítulos que la forman o incluso en las secciones.

  - **H17: Capítulo Introducción**
  - **H18: Capítulo Conceptos**
  - **H19: Sección Plataforma - Capítulo Tecnología**
  - **H20: Sección Herramientas de Soporte - Capítulo Tecnología**
  - **H21: Secciones Propuesta y construcción del sistema - Capítulo Propuesta y desarrollo**
  - **H22: Sección Arquitectura - Capítulo Propuesta y desarrollo**
  - **H23: Sección Proceso de evaluación - Capítulo Experimentos y resultados**
  - **H24: Sección Resultados - Capítulo Experimentos y resultados**
  - **H25: Capítulo Proceso de Ingeniería H26: Capítulo Conclusiones**

6.3.2 Recursos

Para el desarrollo del presente proyecto son necesarios los siguientes recursos:

- Dueño del producto
- Jefe de Proyecto

Historia	Horas	Historia	Horas	Historia	Horas
H1	12	H11	4	H21	10
H2	4	H12	24	H22	18
H3	3	H13	8	H23	6
H4	8	H14	6	H24	6
H5	8	H15	10	H25	6
H6	2	H16	30		
H7	9	H17	14		
H8	8	H18	18		
H9	8	H19	8		
H10	12	H20	8		

Cuadro 6.1: Horas estimadas para cada tarea.

- Analista
- Diseñador
- Programador

Al ser un proyecto reducido no se dispone de una única persona para cada puesto. Y por lo tanto los directores de proyecto asumen los roles de Dueño del producto, mientras que la alumna que realiza el proyecto es la encargada de asumir todos los roles restantes.

Cabe destacar que también se emplean recursos que no son humanos, como por ejemplo un ordenador en donde realizar el proyecto.

6.3.3 Estimación de tiempos

En los proyectos *Scrum*, es habitual estimar el esfuerzo de las historias de usuario en puntos de historia, los cuales representan la cantidad de trabajo que supone realizar la historia de usuario por el equipo en conjunto. Debido a que solo existe una persona desarrollando en el equipo cada punto de historia se corresponde con una hora real.

Existe una técnica muy habitual para estimar en *Scrum*, llamada *Planning Poker*, en la cual los miembros del equipo juegan con una carta la estimación que consideran boca abajo. A continuación, se da la vuelta de todas las estimaciones y se realiza un debate. En este caso, este procedimiento no tiene sentido realizarse, por tanto las estimaciones las ha hecho únicamente la alumna. Dichas estimaciones se pueden ver en la tabla 6.1.

Sprint	Historias
0	H1, H2, H3, H4
1	H5, H6, H7
2	H9, H10
3	H11, H12
4	H13, H14, H15
5	H16
6	H17, H19, H20
7	H18, H8
8	H21,H22
9	H23, H24, H25

Cuadro 6.2: División de sprints.

6.4 Desarrollo del proyecto

Las historias de usuario ya enumeradas anteriormente se han dividido en varios *sprints* en función del tiempo disponible, los cuales se muestran de forma más clara en la tabla 6.2. La idea inicial era realizar *sprints* de 35 horas de trabajo, pero como se puede apreciar los ciclos de trabajo no se ciñen a dicha duración, sobretodo por imprevistos externos al proyecto, principalmente relacionados con otros estudios paralelos y con la incompatibilidad con dos puestos de trabajo. El computo del proyecto se ha retrasado en cuanto a la entrega final, debido a estos imprevistos mencionados que han provocado tres grandes parones los cuales serán explicados más en profundidad a continuación.

En la figura 6.2 se muestra el diagrama de Gantt del proyecto donde se puede ver la distribución temporal de los *sprints* que forman el desarrollo. Como se puede apreciar hay *sprints* que aunque en carga de horas son similares temporalmente son muy diferentes, ocupando unos mucho más que otros, esto se debe a la poca disponibilidad temporal de la programadora provocada por la incompatibilidad con diversos temas externos al proyecto, lo cual implica que no todos los *sprints* se pueda dedicar el mismo tiempo diario al desarrollo, incluso llegando al punto de algún día no poder realizar ningún avance.

Algunos *sprints*, como por ejemplo el número 5 que abarca los experimentos se alargó más de lo esperado debido a problemas en el tiempo de ejecución de los experimentos, el cual fue solventado a lo largo de dicho sprint.

Desde finales de octubre a mediados de febrero se ve un parón muy grande en cuanto al desarrollo producido por la realización de la programadora de un curso a nivel deportivo, junto con el desempeño en dos trabajos, uno por la mañana y otro por la tarde. A continuación,

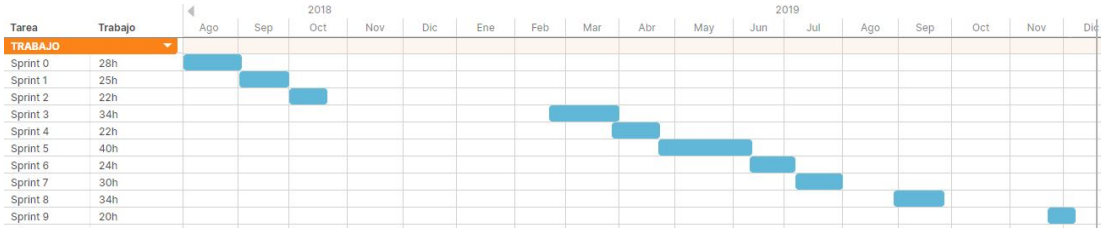

Figura 6.2: Diagrama de Gantt.

en agosto hay un parón debido a las vacaciones tanto de la investigadora como de los dueños del producto. Y para finalizar se puede ver un parón, a partir del mes de septiembre producido por un problema de salud severo de la desarrolladora con su consecuente baja laboral.

Capítulo 7

Conclusiones y trabajo futuro

Con el paso de los años los Sistemas de Recuperación de Información han adquirido una gran importancia, ya que proporcionan a millones de usuarios una gran cantidad de información muy útil en el día a día. La demanda de información es cada vez mayor y de forma paralela también aumenta la información disponible. Si disponemos de un *pool* de documentos con determinada información y sabemos que un asesor desea encontrar la información necesaria de la forma más sencilla posible, debemos proporcionarle aquellos documentos que se consideran más importantes en primer lugar.

Suponiendo que nuestro sistema dispone de un *ranking* de documentos los cuales proporcionará en dicho orden al asesor, nuestro objetivo es que ese orden tenga la máxima calidad posible, es decir, que analizando pocos documentos el asesor encuentre la información que mas se ajusta a su consulta.

7.1 Investigación

En el proyecto realizado se ha desarrollado un algoritmo de *pooling* que pretende proporcionar a los usuarios que lo empleen el mejor orden posible de los documentos solicitados ante una *query*, para ello se ha empleado el contenido de los propios documentos relevantes, determinando cuales son los términos que proporcionan mayor información e incorporándolos a la *query* original.

El resultado obtenido en dicho desarrollo es muy bueno y por lo tanto competitivo contra otros algoritmos ya existentes como los mencionados en la memoria (MTF y docId entre otros).

7.2 Trabajo futuro

Una vez finalizado dicho proyecto, se pueden abrir nuevas líneas de investigación:

- Realización de experimentos con otras técnicas de suavizado, como puede ser Dirichlet, mencionado en el capítulo correspondiente a las técnicas más conocidas de suavización.
- Investigación de la eficacia de dicho algoritmo en otras colecciones para comprobar si puede ser reutilizado con múltiples colecciones.
- Mejora de las técnicas de preprocesado de los documentos para así obtener mejoras en los resultados con el algoritmo de *pooling* implementado .

Glosario de acrónimos

TREC *Text REtrieval Conference.*

MTF *Move To Front*

PRF *Pseudo Relevance Feedback*

IRLAB *Information Retrieval Lab*

RI *Recuperación Información*

URL *Uniform Resource Locator*

HTML *HyperText Markup Language*

PRP *Probability Ranking Principle*

AP *Average Precision*

MAP *Mean Average Precision*

LM *Language Models*

RM *Relevance Model*

NSF *National Science Foundation*

NIST *National Institute of Standars and Technology*

API *Application Programming Interface*

IDE *Integrated Development Environment*

XML *Extensible Markup Language*

POM *Project Object Model*

UML *Unified Modeling Language*

Bibliografía

[1] W. B. Croft, D. Metzler, and T. Strohman, "Search engines - information retrieval in practice," 2009.

[2] D. E. Losada, J. Parapar, and A. Barreiro, "When to stop making relevance judgments? A study of stopping methods for building information retrieval test collections," *JASIST*, vol. 70, no. 1, pp. 49–60, 2019. [Online]. Available: https://doi.org/10.1002/asi.24077

[3] ——, "A rank fusion approach based on score distributions for prioritizing relevance assessments in information retrieval evaluation," *Information Fusion*, vol. 39, pp. 56–71, 2018. [Online]. Available: https://doi.org/10.1016/j.inffus.2017.04.001

[4] D. Valcarce, J. Parapar, and Á. Barreiro, "Lime: linear methods for pseudo-relevance feedback," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC 2018, Pau, France, April 09-13, 2018*, H. M. Haddad, R. L. Wainwright, and R. Chbeir, Eds. ACM, 2018, pp. 678–687. [Online]. Available: https://doi.org/10.1145/3167132.3167207

[5] D. E. Losada, J. Parapar, and A. Barreiro, "Multi-armed bandits for adjudicating documents in pooling-based evaluation of information retrieval systems," *Inf. Process. Manage.*, vol. 53, no. 5, pp. 1005–1025, 2017. [Online]. Available: https://doi.org/10.1016/j.ipm.2017.04.005

[6] D. Valcarce, J. Parapar, and Á. Barreiro, "Item-based relevance modelling of recommendations for getting rid of long tail products," *Knowl.-Based Syst.*, vol. 103, pp. 41–51, 2016. [Online]. Available: https://doi.org/10.1016/j.knosys.2016.03.021

[7] D. Valcarce, J. Parapar, and A. Barreiro, "Efficient pseudo-relevance feedback methods for collaborative filtering recommendation," in *Advances in Information Retrieval - 38th European Conference on IR Research, ECIR 2016, Padua, Italy, March*

20-23, 2016. Proceedings, ser. Lecture Notes in Computer Science, N. Ferro, F. Crestani, M. Moens, J. Mothe, F. Silvestri, G. M. D. Nunzio, C. Hauff, and G. Silvello, Eds., vol. 9626. Springer, 2016, pp. 602–613. [Online]. Available: https://doi.org/10.1007/978-3-319-30671-1_44

[8] ——, "Language models for collaborative filtering neighbourhoods," in *Advances in Information Retrieval - 38th European Conference on IR Research, ECIR 2016, Padua, Italy, March 20-23, 2016. Proceedings*, ser. Lecture Notes in Computer Science, N. Ferro, F. Crestani, M. Moens, J. Mothe, F. Silvestri, G. M. D. Nunzio, C. Hauff, and G. Silvello, Eds., vol. 9626. Springer, 2016, pp. 614–625. [Online]. Available: https://doi.org/10.1007/978-3-319-30671-1_45

[9] D. E. Losada, J. Parapar, and A. Barreiro, "Feeling lucky?: multi-armed bandits for ordering judgements in pooling-based evaluation," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing, Pisa, Italy, April 4-8, 2016*, S. Ossowski, Ed. ACM, 2016, pp. 1027–1034. [Online]. Available: https://doi.org/10.1145/2851613.2851692

[10] D. Valcarce, J. Parapar, and A. Barreiro, "A study of smoothing methods for relevance-based language modelling of recommender systems," in *Advances in Information Retrieval - 37th European Conference on IR Research, ECIR 2015, Vienna, Austria, March 29 - April 2, 2015. Proceedings*, ser. Lecture Notes in Computer Science, A. Hanbury, G. Kazai, A. Rauber, and N. Fuhr, Eds., vol. 9022, 2015, pp. 346–351. [Online]. Available: https://doi.org/10.1007/978-3-319-16354-3_38

[11] ——, "A study of priors for relevance-based language modelling of recommender systems," in *Proceedings of the 9th ACM Conference on Recommender Systems, RecSys 2015, Vienna, Austria, September 16-20, 2015*, H. Werthner, M. Zanker, J. Golbeck, and G. Semeraro, Eds. ACM, 2015, pp. 237–240. [Online]. Available: https://doi.org/10.1145/2792838.2799677

[12] J. Parapar, M. A. P. Quindimil, and A. Barreiro, "Score distributions for pseudo relevance feedback," *Inf. Sci.*, vol. 273, pp. 171–181, 2014. [Online]. Available: https://doi.org/10.1016/j.ins.2014.03.034

[13] J. Parapar, A. Bellogín, P. Castells, and A. Barreiro, "Relevance-based language modelling for recommender systems," *Inf. Process. Manage.*, vol. 49, no. 4, pp. 966–980, 2013. [Online]. Available: https://doi.org/10.1016/j.ipm.2013.03.001

[14] J. Parapar and A. Barreiro, "Promoting divergent terms in the estimation of relevance models," in *Advances in Information Retrieval Theory - Third International Conference,*

ICTIR 2011, Bertinoro, Italy, September 12-14, 2011. Proceedings, ser. Lecture Notes in Computer Science, G. Amati and F. Crestani, Eds., vol. 6931. Springer, 2011, pp. 77–88. [Online]. Available: https://doi.org/10.1007/978-3-642-23318-0_9

[15] C. Cleverdon, *The Cranfield Tests on Index Language Devices*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, p. 47–59.

[16] E. Voorhees and D. Harman, "Trec: Experiment and evaluation in information retrieval," *The MIT Press*, 2005.

[17] G. V. Cormack, C. R. Palmer, and C. L. A. Clarke, "Efficient construction of large test collections," in *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '98. New York, NY, USA: Association for Computing Machinery, 1998, p. 282–289. [Online]. Available: https://doi.org/10.1145/290941.291009

[18] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119 – 139, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S002200009791504X