RECUPERACIÓN DE INFORMACIÓN

Sistema de Recuperación de Información Tradicional

Curso 2019-2020

Víctor Miguel Peñasco Estívalez - 741294
Rubén Rodríguez Esteban - 737215

Índice de contenidos

Introducción

En este documento se va a proceder a describir la creación de un sofisticado sistema de recuperación, concretamente un sistema de recuperación tradicional, que trabaja con una colección de documentos facilitada por el profesorado de la asignatura, y que ha sido descargada a través del portal *Zaguán*.

Para poder llevar a cabo la creación y posterior evaluación del sistema de recuperación tradicional se han tomado como referencia un conjunto de necesidades de información predefinidas, y que están caracterizadas por englobar un amplio espectro de áreas relacionadas con los documentos del corpus.

Haciendo uso de la librería Lucene, que permite crear una capa de abstracción sobre el sistema de recuperación de información a través de la integración de funciones de indexación y búsquedas de información textual, se ha llevado a cabo el proceso de indexado de los documentos de la colección, así como el correspondiente parseo de las necesidades de información y el cálculo del ránking.

Arquitectura del sistema

En este apartado del documento se procede a describir la arquitectura del sistema de recuperación tradicional propuesto (véase *Figura 1*). Dicha arquitectura consta de tres clases principales:

- **SearchFiles:** clase que se encarga de llevar a cabo diversas labores como el parseo de las necesidades de información, la transformación de dichas necesidades en consultas que pueden interpretarse por el sistema de recuperación, la ejecución de tales consultas y la devolución a los usuarios de los documentos del corpus obtenidos tras ejecutar cada una de ellas.
- **IndexFiles:** clase que se encarga de llevar a cabo la indexación de los documentos sobre los que se efectúan las consultas.
- **CustomSpanishAnalyzer:** clase que sirve como instrumento para la analizar las consultas y los documentos por parte del procesador de consultas y del indexador de documentos, respectivamente.

Adicionalmente hay una clase más, denominada **DefaultSetFolder**, que es una clase estática alojada dentro de la clase **CustomSpanishAnalyzer.** Dicha clase se encarga de cargar atómicamente el contenido del fichero de las stop words la primera vez que éste se emplea por la clase **CustomSpanishAnalyzer.** La ruta se almacena en el atributo *DEFAULT_STOPWORD_FILE*.

El almacenamiento de las necesidades de información se ha gestionado a través de un HashMap, de tal forma que cada elemento es una tupla <clave, valor> donde la clave es el identificador de la necesidad de información y el valor es su contenido. A continuación se

muestra una imagen del diagrama de clases que refleja toda la información explicada anteriormente.

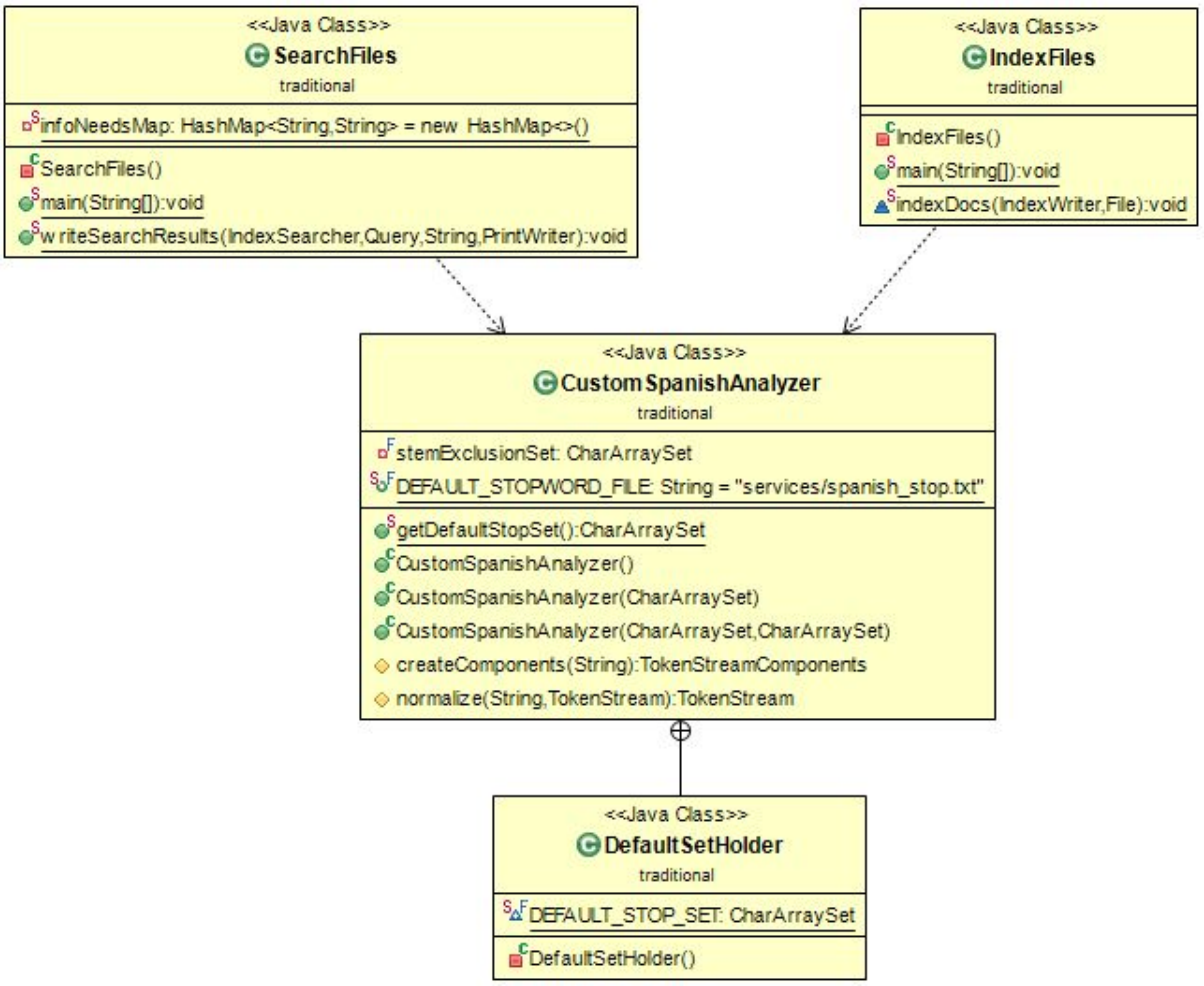

Figura 1. Diagrama de clases de MiniTrecTraditional.

Funcionamiento del sistema

En esta sección del documento se van a describir los aspectos generales del funcionamiento del sistema de recuperación, atendiendo al proceso que se sigue para la indexación de documentos, el parseo de consultas y el cálculo del ránking.

Analizador

En este sistema se ha utilizado un analizador personalizado, el cual se encuentra en la clase **CustomSpanishAnalyzer**. Este analizador se basa en el analizador *SpanishAnalyzer* disponible en Lucene.

A la hora de configurar el analizador, se ha utilizado el mismo fichero de **stop words** en español que utiliza el SpanishAnalyzer. Este fichero se encuentra en services/stopwords.txt.

Para el proceso de **tokenización** utiliza *StandardTokenizer*. Aplica al flujo de tokens una serie de **filtros**: *StandardFilter*, *LowerCaseFilter* y *StopFilter*. Posteriormente también hace uso del filtro de **stemming** *SpanishLightStemFilter*.

Indexación

El proceso de indexación toma los documentos de un directorio proporcionado como argumento al ejecutar, y crea un índice en una ruta también proporcionada como argumento. El indexador hace uso del analizador *CustomSpanishAnalyzer* mencionado anteriormente.

Los campos introducidos en el índice son: path (ruta del fichero), title (título del trabajo), creator (autores), date (año de publicación) y type (indicador de si se trata de TFG o tesis).

Parseo de consultas

El parseo de consultas se realiza en la clase *SearchFiles*. Para cada una de estas consultas, inicialmente en forma de cadena, se realiza el siguiente procesado:

- Se crea una consulta booleana vacía a la que se le irán añadiendo subconsultas con la restricción *SHOULD*. Dichas subconsultas son a su vez procesadas, de la forma explicada anteriormente, por el analizador *CustomSpanishAnalyzer* una vez son añadidas.

- Eliminar los caracteres '*' y '?' para que luego no se confundan con caracteres que indiquen wildcard queries.

- Tokenizar la cadena.

- Identificar si en la consulta se piden específicamente cierto tipo de documentos (bachelorThesis o masterThesis). En ese caso, añadir una *BoostQuery(2)* sobre ese campo.

- Localizar años, para lo cual se utiliza un post-tagger para identificar números. El modelo utilizado es específico de español, aunque para este propósito concreto se podría utilizar uno estándar. Tras localizar un número, se comprueba si es mayor que 2000 y menor o igual que el año actual (más probable que se trate de un año), y en ese caso se añade una consulta sobre el campo date del índice. Dado que todos los documentos del corpus son posteriores al año 2000, no es necesario reconocer años anteriores. En el caso de que se hayan detectado exactamente dos años en la consulta, se añade una consulta de rango entre esos dos números sobre el campo date. También se realizan consultas de rangos sobre el campo date para consultas que incluyen cadenas como "últimos 3 años" o "a partir del año 2015".

- Localizar nombres, para lo cual se usa un *Name Finder* especializado en español. Para los nombres encontrados, se añade una *BoostQuery(2)* sobre el campo creator.

- Para cada uno de los tokens obtenidos de la cadena inicial de la consulta, se añade una query sobre el campo description y una *BoostQuery(2)* sobre title. Se añaden todos los tokens, salvo en los siguientes casos:
    - Se ha encontrado un nombre relacionado con el campo creador, entonces no se añaden palabras como profesor, alumno, tutor, etc.
    - Se ha encontrado una referencia a tesis, TFG o TFM, entonces no se añaden palabras como tfg, trabajo, fin, grado, etc.

Cálculo del ránking

Para el cálculo del ránking de resultados, se ha utilizado el modelo probabilístico Okapi BM25, el cual es utilizado por defecto en la versión 7.0.0 de Apache Lucene.

Resultados

La valoración de los resultados proporcionados por el sistema de recuperación de información desarrollado no es fácil, ya que valorar la relevancia o irrelevancia de los documentos recuperados se basa en criterios subjetivos, algo poco fiable teniendo en cuenta que por el momento sólo existen dos jueces (los autores del trabajo).

Pese a esto, se han llegado a diversas conclusiones:

- Para la mayoría de las necesidades de información planteadas, devuelve bastantes resultados que se podrían considerar relevantes, al menos en los primeros puestos del ránking.
- Existen dos necesidades de información (de las cinco planteadas) que no devuelven exactamente los resultados esperados. La primera de ellas es la que hace referencia a trabajos dirigidos por un profesor llamado Javier. Esta necesidad de información es algo ambigua y posiblemente sería difícil devolver resultados satisfactorios en cualquier sistema de recuperación de información tradicional.
- La segunda necesidad de información que no devuelve resultados satisfactorios es la que hace referencia a la evolución de las corrientes ideológicas en la segunda mitad del siglo XX. La principal dificultad de esta consulta reside en que la palabra cine es el término clave, y que cualquier documento que no trate esta temática difícilmente se puede considerar relevante. El problema es cómo centrarse en ese término. Esto es fácil si se conoce la consulta, pero bastante complicado (al menos sin utilizar técnicas de aprendizaje automático) para que una máquina lo realice a priori sin mucha información más allá de la cadena que representa la necesidad de información.