Sistema Tradicional

Recuperación de información

Martín Gascón Laste
Eduardo Ruiz Cordón

Tabla de contenido

1. Introducción

Durante el desarrollo de este trabajo se va a diseñar un sistema de recuperación de información tradicional la búsqueda de información en la colección de metadatos DublinCore del repositorio Zaguan a cerca de documentos académicos de la universidad de Zaragoza.

Con el diseño de dicho sistema se pretende profundizar en el aprendizaje de las diferentes técnicas de indexación y búsqueda mediante el uso de la API del software Lucene.

2. Arquitectura del software

La base de la arquitectura de este software, se basa en la arquitectura seguidas en la práctica 1 y 2 de la asignatura. Hay 2 clases ejecutables que son: *IndexFiles* y *SearchFiles*. *IndexFiles* se encarga de realizar las acciones de creación e indexación de los campos que se explican en el siguiente apartado, mientras que *SearchFiles* se encarga de realizar las búsquedas a partir de una necesidad de información.

Como se observa en la imagen 1, la clase *IndexFiles* hace uso del *CustomAnalyzer* y del enumerado *FieldType*. *CustomAnalyzer* es el analizador creado por nosotros usado para realizar la indexación. El enumerado *FieldType* se usa para poder encapsular en el método *indexField(FieldTypes, String, String, Document, Document)* la indexación haciendo uso de diferentes de tipos de campo en la indexación.

En la imagen 1 se puede ver como la clase *SearchField* hace uso de *CustomAnalyzer* que al igual que *IndexFiles* la usa como analizador. También se puede observar cómo usa varias clases que sus nombres terminan en *Detector* eso es debido a que estas clases se encargan de detectar patrones en la necesidad de información en el apartado 5. A diferencia de *DateDetector* y *TypeDetector*, las clases *SpanishNamesDetector* y *SpanishLocationDetector* se inicializan al comienzo de la búsqueda debido a que hay que cargar una gran cantidad de información (se explica en detalle en el apartado 5).

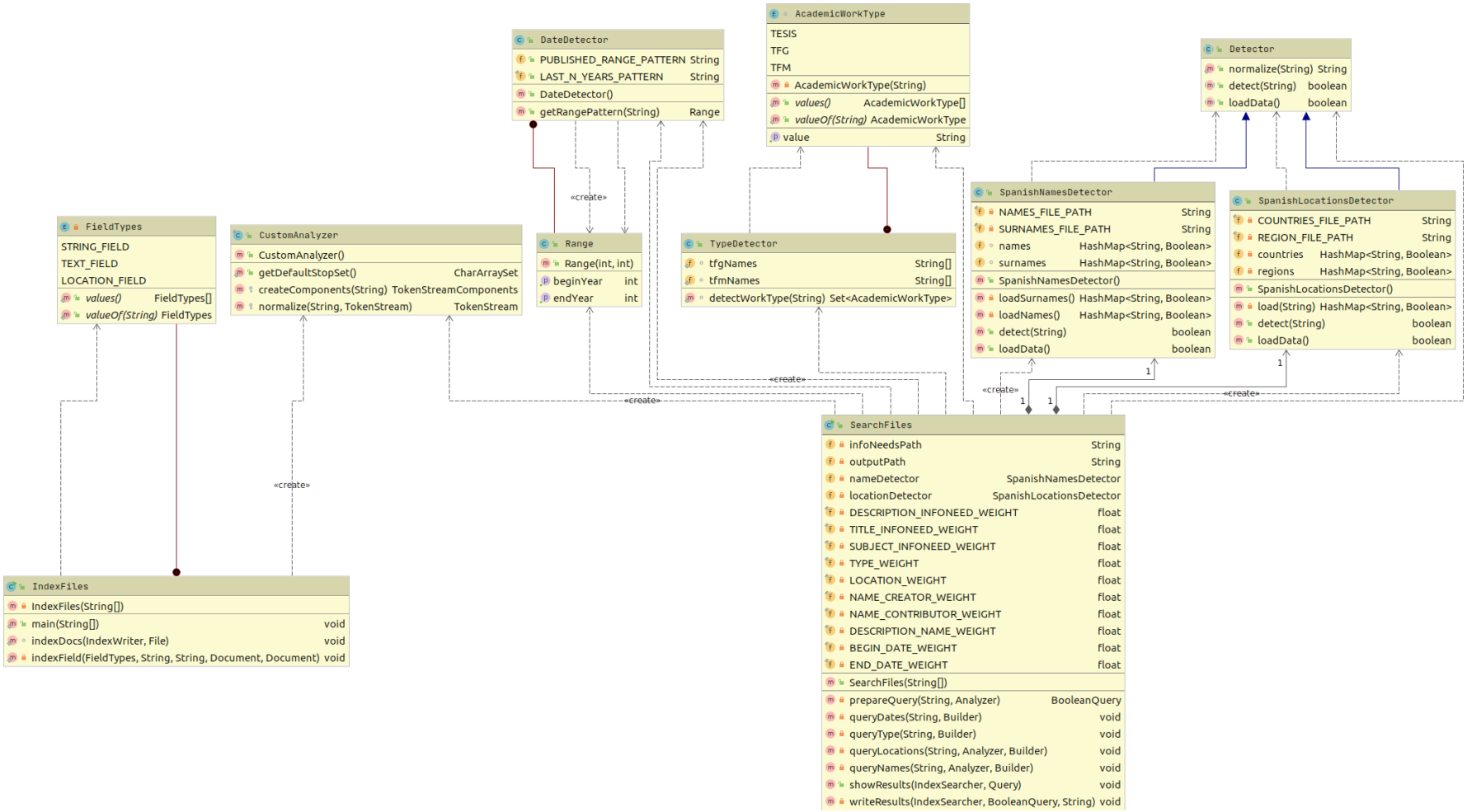
DateDetector
PUBLISHED_RANGE_PATTERN String
LAST_N_YEARS_PATTERN String
DateDetector()
getRangePattern(String) Range
AcademicWorkType
TESIS
TFG
TFM
AcademicWorkType(String)
values() AcademicWorkType[]
valueOf(String) AcademicWorkType
value String
Detector
normalize(String) String
detect(String) boolean
loadData() boolean
«create»
FieldTypes
STRING_FIELD
TEXT_FIELD
LOCATION_FIELD
values() FieldTypes[]
valueOf(String) FieldTypes
CustomAnalyzer
CustomAnalyzer()
getDefaultStopSet() CharArraySet
createComponents(String) TokenStreamComponents
normalize(String, TokenStream) TokenStream
Range
Range(int, int)
beginYear int
endYear int
TypeDetector
tfgNames String[]
tfmNames String[]
detectWorkType(String) Set<AcademicWorkType>
SpanishNamesDetector
NAMES_FILE_PATH String
SURNAMES_FILE_PATH String
names HashMap<String, Boolean>
surnames HashMap<String, Boolean>
SpanishNamesDetector()
loadSurnames() HashMap<String, Boolean>
loadNames() HashMap<String, Boolean>
detect(String) boolean
loadData() boolean
SpanishLocationsDetector
COUNTRIES_FILE_PATH String
REGION_FILE_PATH String
countries HashMap<String, Boolean>
regions HashMap<String, Boolean>
SpanishLocationsDetector()
load(String) HashMap<String, Boolean>
detect(String) boolean
loadData() boolean
«create»
«create»
«create»
«create»
«create»
1
1
1
1
SearchFiles
infoNeedsPath String
outputPath String
nameDetector SpanishNamesDetector
locationDetector SpanishLocationsDetector
DESCRIPTION_INFONEED_WEIGHT float
TITLE_INFONEED_WEIGHT float
SUBJECT_INFONEED_WEIGHT float
TYPE_WEIGHT float
LOCATION_WEIGHT float
NAME_CREATOR_WEIGHT float
NAME_CONTRIBUTOR_WEIGHT float
DESCRIPTION_NAME_WEIGHT float
BEGIN_DATE_WEIGHT float
END_DATE_WEIGHT float
SearchFiles(String[])
prepareQuery(String, Analyzer) BooleanQuery
queryDates(String, Builder) void
queryType(String, Builder) void
queryLocations(String, Analyzer, Builder) void
queryNames(String, Analyzer, Builder) void
showResults(IndexSearcher, Query) void
writeResults(IndexSearcher, BooleanQuery, String) void
«create»
IndexFiles
IndexFiles(String[])
main(String[]) void
indexDocs(IndexWriter, File) void
indexField(FieldTypes, String, String, Document, Document) void

Imagen 1

3. Analizador

Se ha creado un analizador personalizado llamado *CustomAnalyzer que* extiende de la clase *StopwordAnalyzerBase* permitiendo tokenizar y procesar las cadenas de texto durante el proceso de indexado y de búsqueda. Este analizador permite realizar transformaciones sobre las cadenas de caracteres aplicando los siguientes filtros:

- Eliminación de stopwords en el idioma español.
- Proceso de stemming con un stemmer en lenguaje español (*SpanishLightStemFilter*).
- Conversión a minúscula de las cadenas de caracteres.

Durante el proceso de búsqueda se realizan estas transformaciones a todos los campos salvo a los campos *date* y *type*. Se ha valorado la utilización de N-gramas mediante el filtro *NGramTokenFilter* utilizando la configuración para la búsqueda de Bi-gramas y N-gramas de tamaño tres, cuatro y cinco. Sin embargo, sea verificado que los resultados obtenidos parecen ser menos precisos, por tanto, se ha descartado su utilización.

4. Indexación

Los índices que se han decidido crear son los siguientes:

- *title*: se ha creado este índice porque en los títulos aparecen palabras clave sobre el contenido del documento, lo que ayudará a obtener un mejor resultado al realizar las consultas.
- *subject*: este índice se ha creado debido a que contiene los temas principales sobre los que tratan los trabajos.
- *description*: se ha decidido crear el índice porque al igual que en el título contiene palabras clave sobre el contenido del documento, lo que ayudará a distinguir documentos válidos en las búsquedas.
- *contributor*: se ha creado este índice para que se puedan realizar consultas sobre los directores de los documentos.
- *creator*: se ha creado este índice para que se puedan realizar consultas sobre los autores de los documentos.
- *date*: este índice se ha creado para que se puedan al realizar consultas en las que se acoten los años de publicación de los documentos.
- *type*: se ha creado este índice para que se puedan realizar consultas sobre un tipo de documento en específico, por ejemplo, sobre las TESIS
- *publisher*: se ha decidido crear este campo para que se puedan realizar consultas sobre Departamentos de la universidad o Áreas de estudio.

5. Consultas

Para mejorar la eficacia de las consultas se han creado varias clases encargadas de detectar diferentes patrones detectados al diseñar el sistema en las necesidades de información. *DateDetector* se encarga de detectar si en la necesidad de información se especifica que se obtengan los documentos que se hayan realizado en un rango de años. *TypeDetector* detecta si en la necesidad de información se pide obtener los documentos que sean de un tipo concreto, por ejemplo, Trabajos de Fin de Grado.

Las clases *SpanishNamesDetector* y *SpanishLocationsDetector* comprueba si la necesidad de información cumple con el patrón detectado (empezar por mayúscula en ambos casos) pero también comprueba si la palabra que cumple el patrón es un nombre o una localización. Como se ha dicho en el apartado 2 estas dos clases cargan una gran cantidad de información. *SpanishNamesDetector* carga una lista con un amplio conjunto de nombres en español. *SpanishLocationsDetector* carga una lista con los países del mundo en español, las comunidades autónomas y provincias de España.

Se ha usado el modelo probabilístico BM25 como algoritmo de ranking. Este algoritmo permite que los documentos que contengan palabras que estén en la necesidad de información, y tengan poca frecuencia en la colección de documentos tenga un mejor puesto en el ranking. Esto se debe a que la fórmula que usa Lucene para calcular el score (puntuación que determinará la posición en el ranking) es: *boost * idf * tf*. El *idf* es la frecuencia inversa en los documentos, el tf la frecuencia del término en los documentos y el boost un peso otorgado a la consulta. En la imagen 2 se puede ver un ejemplo de cómo se calcula el score de una consulta.

```
6.3358846 = weight(description:diagnostic in 2635) [BM25Similarity], result of:
  6.3358846 = score(freq=1.0), computed as boost * idf * tf from:
    5.0 = boost
    2.8154247 = idf, computed as log(1 + (N - n + 0.5) / (n + 0.5)) from:
      1309 = n, number of documents containing term
      21868 = N, total number of documents with field
    0.45008373 = tf, computed as freq / (freq + k1 * (1 - b + b * dl / avgdl)) from:
      1.0 = freq, occurrences of term within document
      1.2 = k1, term saturation parameter
      0.75 = b, length normalization parameter
      128.0 = dl, length of field (approximate)
      124.971695 = avgdl, average length of field
```

Imagen 2

Se ha hecho uso del boost para mejorar el ranking de nuestras búsquedas. Si se detecta que una palabra es un nombre debe tener más peso al realizar la búsqueda que el resto de palabras por lo que se le ha dado mayor boost.

Poder influir en el score a través del boost también nos ha servido para contrarrestar un poco el problema que da la frecuencia de los términos. Por ejemplo, en el campo *type* solo pueden aparecer 4 opciones posibles, por lo que la frecuencia de estos términos es muy alta y el peso que va a tener las consultas sobre este campo es muy bajo, sin embargo, con el boost se consigue suavizar este problema.

6. Resultados

Para la valoración de las diferentes métricas relativas a los pesos de las consultas y relativas a las diferentes técnicas utilizadas en la elaboración del sistema, se ha tenido en cuenta la relevancia de los primeros 10 documentos recuperados, así como su orden.

Los mejores resultados se han obtenido en las búsquedas de las necesidades de información con identificador 107-2 y 109-2, en el caso de la necesidad 107-2 se cree que es debido a la gran cantidad a la gran cantidad de documentos relevantes para esta necesidad de información, sin embargo, para asegurar esta hipótesis habrá que realizar un análisis mas en profundidad.

El motivo por el que la necesidad 109-2 tiene tan buenos resultados puede deberse a que se están realizando búsquedas sobre *Bioinformática* lo cual es un campo muy concreto, esto ha ayudado a detectar los documentos relevantes, debido a que la frecuencia invertida de los términos relevantes es mayor.

Al contrario de la necesidad de información con identificador 107-2 es probable que existan pocos documentos que satisfagan la necesidad de información con identificador 105-5, aunque como se ha comentado anteriormente es necesario un análisis en profundidad para determinarlo.

En la necesidad de información con identificador 106-4 no se han obtenido unos buenos resultados. Esto se debido a que a diferencia que la necesidad 109-2 las palabras que más peso han tenido en la búsqueda (*Alzheimer* y *Parkinson*) no han sido las palabras clave para encontrar información relevante.

Los resultados que se han obtenido de la ultima necesidad de información han sido poco precisos. Este resultado se cree que se debe a que todas las palabras que se usan en la necesidad de información tienen una frecuencia muy alta. Esto hace que la diferencia de score entre resultados sea pequeña y que no cubran la necesidad de información.