Recuperación de Información

Modelo Semántico

Cristina Oriol García - 755922

Néstor Monzón González - 735418

Índice

SemanticGenerator

En cuanto al *SemanticGenerator*, hemos basado su implementación en el *IndexFiles.java* de nuestro sistema de Recuperación Tradicional. Dado el directorio con los documentos, se itera cada fichero y se parsea cada campo mediante el parser estándar de *javax.xml.parsers*. A continuación, creamos los recursos correspondientes a cada campo en el recurso original (por ejemplo, cada campo *creator* en el xml pasa a ser un recurso *creator* con sus correspondientes propiedades de nombre y apellido, etc., que se enlaza al documento mediante la propiedad *creado*), y damos valor a los literales de acuerdo al texto en los campos de este tipo (*title*, *description*, *subject*…).

Por otra parte, para dar el *tema* (conceptos de nuestro tesauro de *SKOS*), se han tokenizado (mediante una función propia simple basada en expresiones regulares, *SemanticGenerator.tokenizar*) los campos de tipo texto, donde consideramos que pueden aparecer los temas, y se ha comprobado la coincidencia entre estas palabras y las etiquetas (tanto *prefLabels* como *altLabels*) definidas en nuestro tesauro. Así, a un documento que contiene “biología” en la descripción se le asignará el concepto *nuestraraiz:biologia* ya que su prefLabel en español es también “biologia” (coincide con la tokenización de “biología”, que le elimina la tilde).

La búsqueda en el tesauro se ha realizado con la API de Jena del modelo, no hemos considerado necesaria la utilización de SPARQL en este caso.

Finalmente, los tres modelos (el modelo OWL en el que se definen nuestros documentos, el SKOS con el tesauro, y el de salida, con las instancias de dichos documentos), se unen en el de salida.

Se ha intentado aplicar la inferencia de distintas formas, pero ha resultado imposible porque el modelo se hacía tan grande que sobrepasaba la memoria de heap que hemos podido asignar en nuestros ordenadores. En todo caso, habría sido interesante aplicar la regla de OWL propertyChanAxiom especialmente para conseguir que los conceptos se generalicen correctamente (se nos recomendó que realizaramos una reducción del modelo lo suficientemente importante como para que solo contenga esta sentencia que expande los temas y los campos sobre los que realizamos consultas, pero como en el caso de nuestras consultas estas se realizan sobre más campos como *title, subject, description, date, autor, contributor*, aún quedan demasiados componentes en el modelo y nuestros ordenadores siguen sin ser capaces de procesarlo). Por ejemplo, que un documento cuyo tema sea nuestro concepto *biomedicina* también debería considerarse *medicina*, ya que el segundo es *skos:broader* del primero.

Generación de consultas

La generación de las consultas en SPARQL ha sido realizada manualmente mediante una traducción. En esta se ha dado una especial importancia al que sería el *Tema* de los documentos, el campo generado por nosotros como previamente se ha comentado, realizándose sobre los temas del tesaurio, de tipo skos.

Para hacer la búsqueda de los temas se realiza la traducción del siguiente modo:

```
SELECT ?x WHERE{ ?x ruta:Tema ?tema . (?tema ?score) text:query (skos:prefLabel "represion") . }
```

En este ejemplo podemos ver cómo se recupera un documento que corresponde a *?x* que mediante *ruta:Tema* le indicamos que debe tener un tema al que nos referimos como *?tema* y que este mediante la instrucción *text:query* queremos que en su campo prefLabel (campo que posee ya que los temas son de tipo skos Concept) tiene que aparecer la palabra *represión*.

Además como también se puede observar hemos añadido el campo *?score,* posible gracias a los índices, para la generación de un ranking en los resultados obtenidos de las consultas. Estos score se han aplicado a los campos opcionales de la consulta, ya que si son obligatorios hemos considerado que como se utilizan para descartar documentos no era necesario ponerlo ya que todos los devueltos deben cumplir esas condiciones si o si.

```
SELECT DISTINCT ?x WHERE { ?x ruta:Tema ?tema . {{(?tema ?score) text:query (skos:prefLabel "represion")} UNION {(?tema ?score) text:query (skos:prefLabel "dictadura")} UNION {(?tema ?score) text:query (skos:prefLabel "caciquismo")} }OPTIONAL {{(?x ?score3) text:query (ruta:Subject "siglo 20" "XX")} UNION {(?x ?score3) text:query (ruta:title "siglo 20" "XX")} UNION {(?x ?score2) text:query (ruta:description "siglo 20" "XX")}} bind(coalesce(?score,0)+coalesce(?score2, 0) as ?scoreTot)     } order by desc(?scoreTot)
```

En este ejemplo tenemos la unión de varias búsquedas en el campo tema, lo que significa que el tema del documento devuelto debe contener alguno de los términos buscados en el tema, a esta consulta sobre el tema se le asigna un score. Además se realiza una búsqueda en los campos *title, subject y description* para ver si contienen los términos siglo 20 o XX, a esta búsqueda se le asigna otro score. El score total será calculado mediante la suma de los dos y los resultados se ordenan en orden decreciente respecto a este.

Por otro lado también ha sido realizar consultas más allá del tema, como por ejemplo:

```
SELECT DISTINCT ?x  WHERE {?x rdf:type ruta:TFG . ?x ruta:date ?date . FILTER ( ?date >= "2012"^^xsd:gYear )
```

En este ejemplo se ha realizado un filtrado del campo *date* que corresponde a la fecha de publicación indicando que debe ser mayor o igual a 2012. También tenemos la condición de que el documento devuelto sea de tipo TFG exclusivamente.

Semantic Searcher

Para la realización del programa de búsqueda primero realizamos la indexación del modelo para poder aplicar búsquedas sobre los índices. Para ello hemos definido una nueva entidad sobre la que creabamos las propiedades de los campos que queremos indexar y la configuración que queremos.

```
EntityDefinition entDef = new EntityDefinition("uri", "name",
ResourceFactory.createProperty("http://nuestraraiz/","title"));

entDef.set("firstName",
ResourceFactory.createProperty("http://xmlns.com/foaf/0.1/","firstNa
me").asNode());

...
```

Una vez realizado este proceso, creamos un directorio y realizamos la indexación del modelo en él, por motivos de tiempo al procesar hemos utilizado un *RAMDirectory* que carga los índices en la RAM ya que al intentar indexarlos en un directorio permanente tardaba demasiado tiempo en ejecutarse.

Una vez tenemos los documentos del modelo indexados leemos linea a linea cada necesidad y para cada una realizamos el procesamiento de la consulta.

Para ello leemos del fichero el string que corresponderá a la consulta ya realizada en SPARQL y la ejecutamos, guardando para cada resultado devuelto el número de la necesidad a la que correspondía y su identificador tipo 'oai_zaguan.unizar.es_documento.xml'.

Evaluación

Para la evaluación, se ha utilizado el programa implementado para el sistema tradicional, así como los juicios de relevancia que se emitieron en ese momento. Se ha modificado ligeramente el programa de evaluación ya que había ciertos errores en el cálculo de la precisión. Las medidas que se muestran son con el evaluador actualizado (incluídas las del tradicional). A continuación, se muestra la comparación de estas medidas obtenidas con este sistema y el tradicional implementado anteriormente.

Medida	Tradicional	Semántico
precision	0.378	0.373
recall	0.375	0.252
F1	0.366	0.284
prec@10	0.640	0.500
MAP	0.591	0.493

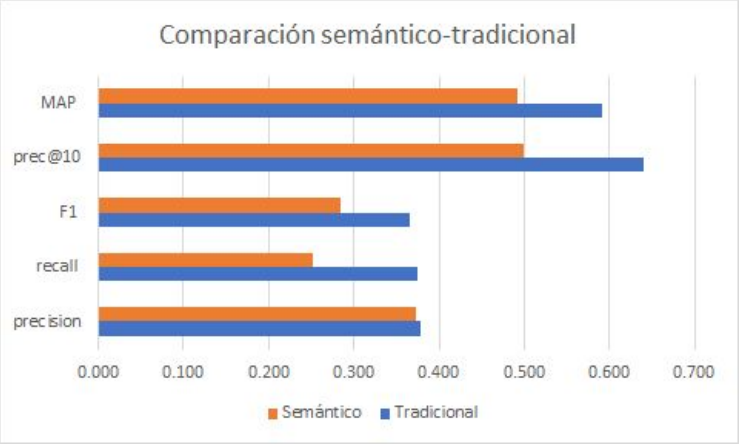

Figura 1: Tabla e histograma con la comparación de la evaluación del sistema tradicional y el semántico.

Además, a continuación se comparan las curvas precisión-recall globales con ambos sistemas (azul tradicional, naranja semántico):

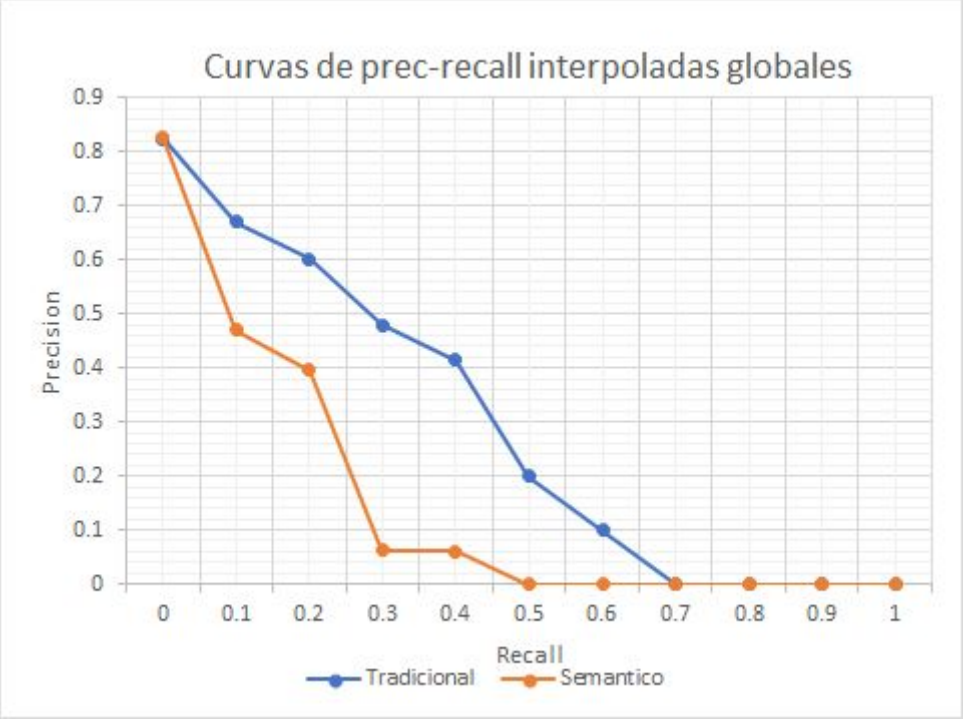

Figura 2: Curvas de prec-recall interpoladas globales con cada sistema

Como se puede observar en la figura 1, el sistema tradicional es superior en todas las métricas. La única medida ajustada es la precisión global, es decir, ambos sistemas recuperan una proporción similar de documentos "relevantes" entre los recuperados (de acuerdo a los juicios de relevancia que generamos en su día) que el tradicional. No obstante, el tradicional recupera una mayor proporción de los relevantes respecto al total (mayor recall), la relevancia de sus 10 primeros resultados es mayor (mayor prec@10, especialmente importante si se usara como sistema de recuperación web para usuarios, por ejemplo), y, sobre todo, su MAP y F1-score, las medidas que mejor combinan la precisión y el recall con todos los umbrales posibles, son significativamente mayores.

La diferencia entre los sistemas se puede apreciar visualmente en las curvas de precision-recall interpoladas globales de la figura 2. Por ejemplo, para conseguir un 0.2 de recall en el sistema semántico se da una precisión del 0.4, mientras que en el tradicional es del 0.6.

En parte, consideramos que esta diferencia se da por cómo se generaron los juicios en su momento. Con cada sistema de recuperación tradicional se obtuvieron una serie de documentos clasificados como relevantes. A continuación, entre todos los grupos, se revisaron y clasificaron manualmente estos documentos como relevantes o no relevantes, dando lugar a los juicios que se han utilizado (omitiendo detalles de las dobles valoraciones, etc.). Con esto, los documentos relevantes en los juicios son un subconjunto de los documentos clasificados como relevantes por los sistemas tradicionales de todos los grupos. Por tanto, cualquier documento que no fuera clasificado como relevante por ningún sistema tradicional se considera irrelevante en los juicios (concretamente, cualquiera que no estuviera en el top 30 de ningún sistema). Esto puede penalizar injustamente a los sistemas semánticos, que es más probable que devuelvan otros documentos por su funcionamiento completamente distinto.

Aun así, es probable que el sistema semántico sea efectivamente inferior al tradicional en este caso. Creemos que esto se debe fundamentalmente a que el modelo semántico es mucho más simple de lo que debería, ya que sus principales ventajas, debidas a la relación semántica entre distintos conceptos, con las distintas inferencias de información que estas permiten, no se han aplicado a la escala que se daría en un modelo real (al que se le dedicaría mucho más tiempo y recursos que los razonables para una asignatura), por lo que no llega a alcanzar la masa crítica de recursos y reglas que mostrarían su verdadero potencial.

Otro de los motivos por los que nuestro sistema tradicional funcione mejor que el semántico es por su utilización del analizador con stemming *NuestroSpanishAnalyzer* (que aplica el *SnowBallFilter*). Este analizador, como explicamos en la memoria del sistema tradicional, permite hacer matching entre palabras que solo comparten la raíz léxica ("ingenieril" con "ingeniería", por ejemplo).

Por otra parte, el sistema semántico tiene la ventaja de que cada consulta (en SPARQL) ha sido elaborada manualmente realizando distintas pruebas (como se ha explicado anteriormente), mientras que en el semántico la generación de las consultas era automática a partir del texto de cada necesidad de información. Esto debería favorecer a los resultados del analizador semántico, aunque evidentemente como sistema es más práctico el anterior, ya que para buscar una nueva necesidad solo se debe escribir en lenguaje natural.