Tecnologías de la Información 2015-16: Práctica P1

Objetivos de las Clases Prácticas

Durante las prácticas de la asignatura vamos a desarrollar un motor de recuperación completo para páginas HTML. La funcionalidad del motor será triple: crear un índice a partir de un corpus de documentos, ejecutar consultas en batch (en lotes), y ejecutar consultas en modo interactivo. El desarrollo se dividirá en 5 prácticas:

P1) Implementación de recuperación según el modelo vectorial y función coseno para un índice dado.
P2) Implementación de la función de indexación a partir de un corpus de documentos dados.
P3) Implementación del procesamiento de documentos HTML para extraer sus términos.
P4) Incorporación de pseudorelevance feedback al recuperador.
P5) Implementación de recuperación en modo interactivo, con generación de snippets.

En cada una de las entregas se tendrá que evaluar la efectividad del motor de búsqueda, y compararlo con las entregas anteriores para ver va evolucionando. El desarrollo será 100% en Java, y estará simplificado considerablemente (eg., el índice se carga completamente en memoria principal). Cada parte es independiente de las demás, y no será necesario aprobar cada una por separado.

Funcionamiento y Estructura del Motor

Se suministra por AulaGlobal el esqueleto del código del motor. Éste ofrece tres funcionalidades principales al ejecutarse:

```
> java ti.SearchEngine
Usage: ti.SearchEngine <command> <options>

where <command> and <options> are one of:
  - index <path-to-index> <path-to-collection> [<path-to-stopwords>]
  - batch <path-to-index> <path-to-queries>
  - interactive <path-to-index>
```

La clase `SearchEngine` sirve de punto de entrada al programa. Simplemente comprueba los parámetros de la consola y delega la ejecución en las clases `Indexer`, `Batch` e `Interactive`, que son las encargadas de ejecutar el motor. La clase `Index` contiene las estructuras de datos del índice. Un índice es creado por el `Indexer`, y será utilizado para recuperación desde `Batch` y desde `Interactive`.

Además, la interface `DocumentProcessor` define operaciones básicas de un procesador de documentos para extraer los términos a indizar. Se proporciona la clase `SimpleProcessor` para las primeras prácticas, pero será sustituida por `HtmlProcessor` para procesar páginas HTML correctamente desde la práctica P3. La interface `RetrievalModel` define operaciones básicas para un modelo de recuperación. Durante la práctica P1 se implementará la clase `Cosine`, y durante la P4 se implementará `CosineWithFeedback`.

Todo el código está documentado, y se entrega también la documentación javadoc. Se recomienda encarecidamente revisar la documentación para entender bien la estructura del motor.

Tuplas

La mayor parte de los datos almacenados en el índice y utilizados por el motor se modelan en forma de tuplas, es decir, de pares ordenados de valores. Por ejemplo, hay tuplas (documento, norma), (término, IDF), (término, peso), (documento, similitud), etc. La clase `Tuple<T1,T2>` implementa esta estructura para dos valores de tipo arbitrario. Dado que se usa en todo el motor, es importante entender cómo funciona:

```
int docId = 34;
double sim = 0.843;
// Crear una tupla (int, double)
Tuple<Integer, Double> resultado = new Tuple<>(docId, sim);
System.out.println(resultado.item1); // 34
resultado.item2 = 0.23;
System.out.println(resultado.item2); // 0.23
```

HashMap<K,V>

Otra estructura de datos que se utilizará en todo el motor es el `HashMap<K,V>` de Java, que mapea claves de tipo `K` a valores de tipo `V` (pueden por tanto entenderse como tuplas `Tuple<K,V>`). La característica principal es que su complejidad de acceso es O(1), por lo que se utilizan para saber qué valor se corresponde con una cierta clave. De nuevo, es muy importante entender cómo funcionan:

```
// Mapear Strings a int
HashMap<String, Integer> hm = new HashMap<>();
hm.put("coche", 4);
hm.put("rojo", 32);
Integer a = hm.get("coche"); // 4
Integer b = hm.get("rojo"); // 32
Integer c = hm.get("casa"); // null
boolean d = hm.containsKey("coche"); // true
// Recorrer todos los pares clave-valor
for (Map.Entry<String, Integer> e : hm.entrySet()) {
    String clave = e.getKey();
    int valor = e.getValue();
    e.setValue(90);
}
```

Index

El índice del motor contiene varias estructuras de datos en la clase `Index`:

- `HashMap<String, Tuple<Integer, Double>> vocabulary`: mapea un término a una tupla (termID, IDF).
- `ArrayList<Tuple<String, Double>> documents`: el elemento i-ésimo se corresponde con el documento de docID=i. El valor es una tupla (docName, norma).
- `ArrayList<ArrayList<Tuple<Integer, Double>>> invertedIndex`: el elemento i-ésimo se corresponde con el término de termID=i. El valor es el postings list de ese término: cada elemento es una tupla (docID, peso), identificando el peso de ese término en ese documento. Esta estructura se utilizará para recuperación.
- `ArrayList<ArrayList<Tuple<Integer, Double>>> directIndex`: el elemento i-ésimo se corresponde con el documento de docID=i. El valor es el postings list de ese documento: cada elemento es una tupla (termID, peso), identificando el peso de ese término en ese documento. Esta estructura se utilizará para recuperación con pseudorelevance feedback.

Por tanto, tanto términos como documentos se identificarán de forma numérica con un `int`.

Además, el índice ofrece funcionalidad de cache para asignar y obtener la versión limpia de un documento ya procesado, que se utilizará después durante la recuperación interactiva para mostrar snippets. La versión cache de un documento es un objeto `Tuple<String, String>` que contiene el título y el contenido principal del documento.

Práctica P1

En la primera práctica se proporciona el índice (`2011-index`) creado ya previamente para un corpus de documentos (`2011-documentos`), y se debe implementar un recuperador según el modelo vectorial y la función coseno. Los pasos a seguir son:

1. Cargar el índice desde disco.

2. Cargar consultas desde disco.
3. Por cada consulta:
    a. Extraer los términos de consulta.
    b. Calcular la similitud con los documentos.
    c. Generar la salida con los primeros 500 resultados.
4. Evaluar la efectividad del motor.

Implementación

Esta funcionalidad se implementa principalmente en la clase `Cosine`, que es llamada desde `Batch` y utiliza la información de `Index`. Por cada consulta se ejecutará el método

```
ArrayList<Tuple<Integer, Double>> runQuery(String queryText,
                                           Index index,
                                           DocumentProcessor docProcessor)
```

que devuelve tuplas (docID, similitud).

Internamente, primero extrae los términos de la consulta, y después llama al método

```
ArrayList<Tuple<Integer, Double>> computeVector(ArrayList<String> terms,
                                                Index index)
```

que recibe lista de términos y devuelve el vector consulta, representado como tuplas (termID, peso).

Después, llama al método

```
ArrayList<Tuple<Integer, Double>> computeScores(ArrayList<Tuple<Integer, Double>> queryVector,
                                                Index index)
```

que devuelve la similitud entre los documentos del índice y el vector consulta. De vuelta en la clase `Batch`, se mostrarán por pantalla, y en formato TREC, todos los resultados para todas las consultas:

```
> java ti.SearchEngine batch 2011-index 2011-topics.xml
Loading index...done. Statistics:
  - Vocabulary: 209732 terms (5,05 MB).
  - Documents: 2088 documents (43,04 KB).
  - Inverted: 17,95 MB.
  - Direct: 0,01 MB.
2011-001  Q0  2011-72-101  1    0.2041027796949601    sys
2011-001  Q0  2011-69-107  2    0.20122488707165842   sys
...
2011-001  Q0  2011-33-104  373  0.003256827817572941  sys
2011-001  Q0  2011-08-055  374  0.001941189195305561  sys
2011-002  Q0  2011-90-092  1    0.1768618051073748    sys
2011-002  Q0  2011-96-020  2    0.16986359165389944   sys
...
2011-023  Q0  2011-49-058  499  0.007816253059873348  sys
2011-023  Q0  2011-40-071  500  0.007776422197533339  sys
```

El formato TREC se explica más adelante.

Pesos

La formulación de pesos usada sigue los principios TFxIDF:

- $tf_{td} = 1 + \log(f_{td})$
- $idf_t = \log\left(1 + \frac{n_d}{c_t}\right)$

Por tanto, los pesos son $w_{td} = tf_{td} \cdot idf_t$.

Similitud

La similitud se calculará según la función coseno:

$$sim(\vec{d}, \vec{q}) = \frac{\sum_t w_{td} \cdot w_{tq}}{\sqrt{\sum_t w_{td}^2} \sqrt{\sum_t w_{tq}^2}}$$

Los pesos de los términos en los documentos ya están calculados en el índice (en `invertedIndex`), y sus normas también (en `documents`). Los pesos en la consulta y su norma se calculan en tiempo de consulta.

Sugerencias y Requisitos

- Nótese que en el numerador del coseno sólo hay que preocuparse por los términos de la consulta; la contribución de los otros términos es cero.
- El IDF de un término aparece una vez en el peso en el documento y otra vez en el peso en la consulta.
- No puede usarse el índice directo para esta práctica.
- Qué documentos contienen un término de la consulta puede saberse inmediatamente a través de `invertedIndex`.
- No se pueden usar librerías externas para nada.
- Buscar comentarios `// P1` para ver rápidamente qué partes del código hay que completar.

Evaluación

Una vez implementado el recuperador, se debe evaluar su efectividad. Para ello se proporciona una herramienta de evaluación, un archivo de consultas y un archivo de juicios de relevancia. Primero, se debe generar un archivo con los runs del sistema:

```
> java ti.SearchEngine batch 2011-index 2011-topics.xml > 2011.run
Loading index...done. Statistics:
  - Vocabulary: 209732 terms (5,05 MB).
  - Documents: 2088 documents (43,04 KB).
  - Inverted: 17,95 MB.
  - Direct: 0,01 MB.
> cat 2011.run
2011-001  Q0  2011-72-101  1    0.2041027796949601    sys
2011-001  Q0  2011-69-107  2    0.20122488707165842   sys
...
2011-023  Q0  2011-49-058  499  0.007816253059873348  sys
2011-023  Q0  2011-40-071  500  0.007776422197533339  sys
```

Por último, se debe ejecutar la herramienta de evaluación:

```
> java -jar ireval.jar
usage: java -jar ireval.jar TREC-Ranking-File TREC-Judgments-File
```

que recibe el archivo run del sistema y el archivo de juicios de relevancia. Como salida, mostrará la efectividad por cada una de las consultas, y después la media sobre todas las consultas. Por ejemplo:

```
> java -jar ireval.jar 2011.run 2011.qrel
num_ret                         2011-001          126
num_rel                         2011-001           70
num_rel_ret                     2011-001           70
P@5                             2011-001       0.8000
P@10                            2011-001       0.8000
P@15                            2011-001       0.6667
...
nDCG@100                        2011-001       0.8479
nDCG@200                        2011-001       0.8764
nDCG@500                        2011-001       0.8764

num_ret                         2011-002          500
```

```
num_rel                         2011-002            47
num_rel_ret                     2011-002            47
P@5                             2011-002        0.2000
P@10                            2011-002        0.2000
P@15                            2011-002        0.2000
...
nDCG@50                         2011-023        0.3366
nDCG@100                        2011-023        0.3422
nDCG@200                        2011-023        0.3422
nDCG@500                        2011-023        0.3422
...
num_q                             all       23
num_ret                           all    10487
num_rel                           all     1152
num_rel_ret                       all      975
P@5                               all   0.5043
P@10                              all   0.4609
P@15                              all   0.4319
P@20                              all   0.4348
P@30                              all   0.4275
P@50                              all   0.3896
P@100                             all   0.3200
P@200                             all   0.1889
P@500                             all   0.0848
R@5                               all   0.0654
R@10                              all   0.1077
R@15                              all   0.1486
R@20                              all   0.1934
R@30                              all   0.2829
R@50                              all   0.4130
R@100                             all   0.6687
R@200                             all   0.7796
R@500                             all   0.8694
iP@0.00                           all   0.7110
iP@0.10                           all   0.6188
iP@0.20                           all   0.5215
iP@0.30                           all   0.4915
iP@0.40                           all   0.4223
iP@0.50                           all   0.3988
iP@0.60                           all   0.3470
iP@0.70                           all   0.3260
iP@0.80                           all   0.2994
iP@0.90                           all   0.2279
iP@1.00                           all   0.1351
RP                                all   0.4045
RR                                all   0.6056
AP                                all   0.3837
nDCG@5                            all   0.3929
nDCG@10                           all   0.3869
nDCG@15                           all   0.3840
nDCG@20                           all   0.3923
nDCG@30                           all   0.4019
nDCG@50                           all   0.4227
nDCG@100                          all   0.5476
nDCG@200                          all   0.6030
nDCG@500                          all   0.6403
```

Se suministra el archivo `2011-ejemplo.run` como ejemplo de salida de sistema, para ver cómo funciona la herramienta de evaluación.

Formatos TREC

Un archivo `.run` de salida de un motor tiene el siguiente formato:

```
2011-001  Q0  2011-72-101  1    0.2041027796949601    sys
2011-001  Q0  2011-69-107  2    0.20122488707165842   sys
```

```
...
2011-023  Q0  2011-25-036  99   0.029122264017657634  sys
2011-023  Q0  2011-62-113  100  0.02901768957593081   sys
```

Las columnas son: ID del topic, el literal `Q0` (siempre), el ID de documento, la posición del documento, la similitud con la consulta, el ID de sistema (en la práctica siempre `sys`).

Un archivo .qrel de juicios de relevancia tiene el siguiente formato:

```
2011-001        0       2011-51-039     2
2011-001        0       2011-64-042     1
...
2011-023        0       2011-88-050     0
2011-023        0       2011-82-066     2
```

Las columnas son: ID del topic, el literal `0` (siempre), el ID de documento, el juicio de relevancia. En nuestro caso, la relevancia es gradual con niveles 0, 1 y 2.

Opcional

Podrán obtener puntuación extra aquellos grupos que implementen otros modelos y funciones, como por ejemplo Okapi BM25, siempre que estén correctamente evaluados y comparados con la funcionalidad básica requerida.

Para Depurar Código

Se suministra también el índice (`teoria-index`) y el archivo de topics (`teoria-topics.xml`) correspondiente al ejemplo que vimos en clase de teoría, y cuya solución está en AulaGlobal. Al ser un corpus de sólo 4 documentos, puede resultar útil para depurar el código del coseno.

Nótese que la formulación de idf es distinta en la práctica y en el ejemplo de teoría. El índice de la práctica (`2011-index`) está calculado con la formulación de este enunciado, mientras que el de teoría (`teoria-index`) está calculado con la formulación que usamos aquél día. Además, para que pueda funcionar la recuperación, tendríais que cambiar la línea 33 de `SimpleProcessor` para que quedase así:

```
if (token.length() > 0)
```

Es importante recordar que estos cambios sólo son necesarios para el ejemplo de teoría. A la hora de entregar la práctica debe seguirse el código original y la formulación de este enunciado.

Entrega

Se debe entregar lo siguiente por AulaGlobal:

- Un archivo ZIP/RAR/TGZ con todos los archivos del código del motor.
- El archivo `.run` del motor resultante de ejecutarlo con los topics proporcionados (`2011-topics.xml`).
- Un documento PDF, de 3 páginas como mucho, con la evaluación de la efectividad del motor.
    - ¿Cómo de bien funciona el motor? ¿Bajo qué circunstancias?
    - Discutir los números y hacer algún gráfico.
    - Discutir resultados medios para todos los topics.
    - Escoger un topic que se estime oportuno y analizarlo en detalle.

Las fechas límite son:

- P102: 29 de abril
- P101: 1 de mayo

Sólo hace falta que entregue un miembro de cada grupo.