Búsqueda y Minería de Información 2020-2021

Universidad Autónoma de Madrid, Escuela Politécnica Superior

Grado en Ingeniería Informática 4º curso

Práctica 1 – Implementación de un motor de búsqueda

Fechas

- Comienzo: lunes 15 / martes 16 de febrero
- Entrega: lunes 1 / martes 2 de marzo (14:00h)

Objetivos

Los objetivos de esta práctica son:

- La iniciación a la implementación de un motor de búsqueda.
- Una primera comprensión de los elementos básicos necesarios para implementar un motor completo.
- La iniciación al uso de la librería *Whoosh* en Python para la creación y utilización de índices, funcionalidades de búsqueda en texto.
- La iniciación a la implementación de una función de ránking sencilla.

Los documentos que se indexarán en esta práctica, y sobre los que se realizarán consultas de búsqueda serán documentos HTML, que deberán ser tratados para extraer y procesar el texto contenido en ellos.

La práctica plantea como punto de partida una pequeña API general sencilla, que pueda implementarse de diferentes maneras, como así se hará en esta práctica y las siguientes. A modo de toma de contacto y arranque de la asignatura, en esta primera práctica se completará una implementación de la API utilizando Whoosh, con lo que resultará bastante trivial la solución (en cuanto a la cantidad de código a escribir). En la siguiente práctica el estudiante desarrollará sus propias implementaciones, sustituyendo el uso de Whoosh que vamos a hacer en esta primera práctica.

En términos de operaciones propias de un motor de búsqueda, en esta práctica el estudiante se encargará fundamentalmente de:

a) En el proceso de indexación: recorrer los documentos de texto de una colección dada, eliminar del contenido posibles marcas tales como html, y enviar el texto a indexar por parte de Whoosh.
b) En el proceso de responder consultas: implementar una primera versión sencilla de una o dos funciones de ránking en el modelo vectorial, junto con alguna pequeña estructura auxiliar.

Material proporcionado

Se proporcionan:

- Varios módulos Python en un archivo comprimido src.zip, de las que el estudiante partirá para completar código e integrará las suyas propias. En particular, el módulo `main.py` (en el paquete `bmi.search`) incluye un programa main que deberá funcionar con el código a implementar por el estudiante.
- Una pequeña colección docs1k.zip con aproximadamente 1.000 documentos HTML, y un pequeño fichero urls.txt. Ambas representan colecciones de prueba para depurar las implementaciones y comprobar su corrección.
- Un documento de texto "output.txt" con la salida estándar que deberá producir la ejecución del programa `main.py`.

Los profesores corregirán las prácticas con Python 3.9. El código entregado deberá ejecutar correctamente con esta versión. Además, tal y como se entrega sólo depende del siguiente requisito (que se puede instalar via pip, por ejemplo): whoosh. Las dependencias adicionales se tendrán que justificar y reflejar en la memoria. Para el parsing de HTML se sugiere por ejemplo la librería BeautifulSoup.

Ejercicios

1. Implementación basada en Whoosh

Implementar las clases y módulos necesarios para que el programa main funcione. Se deja al estudiante deducir alguna de las relaciones jerárquicas entre las clases Python.

1.1 Indexación (3pt)

Definir las siguientes clases:

- `Index` en el módulo `bmi.search.index`.
- `WhooshIndex`, como subclase de `Index`, en el módulo `bmi.search.whooshy`.
- `WhooshBuilder`, como subclase de `Builder`, (en el módulo `bmi.search.whooshy`.

Se sugiere utilizar dos "fields" en el esquema de documentos de Whoosh: la ruta del documento (dirección Web o ruta en disco), y el contenido del documento.

La entrada para construir el índice (método `Builder.build()`) podrá ser a) un fichero de texto con direcciones Web (una por línea); b) una carpeta del disco (se indexarán todos los ficheros de la carpeta, sin entrar en subcarpetas); o c) un archivo zip que contiene archivos comprimidos a indexar. Supondremos que el contenido a indexar es siempre HTML.

1.2 Búsqueda (1.5pt)

Implementar la clase `WhooshSearcher` como subclase de `Searcher`, en el módulo `bmi.search.whooshy`.

2. Modelo vectorial

Implementar dos modelos de ránking propios, basados en el modelo vectorial.

2.1 Producto escalar (2pt)

Implementar un modelo vectorial propio que utilice el producto escalar (sin dividir por las normas de los vectores) como función de ránking, por medio de la clase `VSMDotProductSearcher`, como subclase de `Searcher` (en el módulo `bmi.search.search`).

Este modelo hará uso de la clase `Index` y se podrá probar con la implementación `WhooshIndex` (puedes ver un ejemplo de esto en el módulo `main.py`).

Además, la clase `VSMDotProductSearcher` será intercambiable con `WhooshSearcher`, como se puede ver en `main.py`, donde la función test_search utiliza una implementación u otra sin distinción.

Para simplificar, aplicar a las consultas simplemente una separación de palabras por espacios en blanco y normalización a minúsculas.

Añadir a mano un documento a la colección docs1k.zip de manera que aparezca el primero para la consulta "obama family tree" para este buscador. Documentar en la memoria cómo se ha conseguido y por qué resulta así.

2.2 Coseno (1.5pt)

Refinar la implementación del modelo para que calcule el coseno, definiendo para ello una clase `VSMCosineSearcher`. Para ello se necesitará extender `WhooshBuilder` con el cálculo de los módulos de los vectores, que deberán almacenarse en un fichero, en la carpeta de índice junto a los ficheros que genera Whoosh. Pensad en qué parte del diseño interesa hacer esto, en concreto, qué clase y en qué momento tendría que calcular, devolver y/o almacenar estos módulos.

Añadir a mano un documento a la colección docs1k.zip de manera que aparezca el primero para la consulta "obama family tree" para este buscador. Documentar en la memoria cómo se ha conseguido y por qué resulta así.

3. Estadísticas de frecuencias (2pt)

Utilizando las funcionalidades de la clase `Index`, implementar una función `term_stats` en un nuevo módulo `statistics.py` que calcule a) las frecuencias totales en la colección de los términos, ordenadas de mayor a menor, y b) el número de documentos que contiene cada término, igualmente de mayor a menor. Visualizar las estadísticas obtenidas en dos gráficas en escala log-log (por cada colección –seis gráficas en total), que se incluirán en la memoria.

Indicaciones

Se podrán definir clases adicionales a las que se indican en el enunciado, por ejemplo, para reutilizar código. Y el estudiante podrá utilizar o no el software que se le proporciona, con la siguiente limitación:

- No deberá editarse el software proporcionado más allá de donde se indica explícitamente.
- El programa **main deberá ejecutar** correctamente.

Entrega

La entrega consistirá en un único fichero zip con el nombre **bmi-p1-XX.zip**, donde XX debe sustituirse por el número de pareja (01, 02, …, 10, …). Este fichero contendrá:

- Una carpeta **src/** con todos los ficheros .py con las implementaciones de los ejercicios 1, 2 y 3. Estos ficheros se ubicarán en la ruta apropiada de subcarpetas según sus módulos.
- En su caso, un fichero **readme_lib.txt** indicando las librerías adicionales que fuesen necesarias (no se incluirán Whoosh ni BeautifulSoup) así como su forma de instalarlas (comando pip, por ejemplo). Se recomienda no obstante consultar con el profesor antes de hacer uso de librerías adicionales.
- Una **memoria bmi-p1-XX.pdf** donde se documentará:
  - Qué version(es) del modelo vectorial se ha(n) implementado en el ejercicio 2.
  - Cómo se ha conseguido colocar un documento en la primera posición de ránking, para cada buscador implementado en el ejercicio 2.
  - El trabajo realizado en el ejercicio 3.
  - Y cualquier otro aspecto que el estudiante considere oportuno destacar.

  La calidad de la memoria representará orientativamente el 10% de la puntuación de los ejercicios que se documentan en la misma.

No se deberán incluir en el .zip las colecciones proporcionadas por el profesor, ni los archivos de proyecto del entorno de desarrollo que se esté usando.

El fichero de entrega se enviará por el enlace habilitado al efecto en el curso **Moodle** de la asignatura.

Calificación

Esta práctica se calificará con una puntuación de 0 a 10 atendiendo a las puntuaciones individuales de ejercicios y apartados dadas en el enunciado. El peso de la nota de esta práctica en la calificación final de prácticas es del **20%**.

La calificación se basará en el **número** de ejercicios realizados y la **calidad** de los mismos. La puntuación que se indica en cada apartado es orientativa, en principio se aplicará tal cual se refleja pero podrá matizarse por criterios de buen sentido si se da el caso.

Para dar por válida la realización de un ejercicio, el código deberá funcionar (a la primera) integrado con las clases que se facilitan, tal cual se facilitan, **sin ninguna modificación**. El profesor comprobará este aspecto ejecutando el programa `main` así como otros main de prueba adicionales.