UNIVERSIDADE DA CORUÑA

Lenguajes Naturales

Curso 2012/2013

My Little Trivial Player

Memoria de la Práctica

Autores:

Garabato Míguez, Daniel <daniel.garabato@udc.es>
López Beade, Vanesa <vanesa.lopezb@udc.es>
Valcarce Silva, Daniel <daniel.valcarce@udc.es> (Portavoz)

Índice

1. Arquitectura del sistema

Bla, bla, bla...
En la Figura 1, se puede observar la arquitectura general del sistema.

Figura 1: Arquitectura general del sistema

1.1. Proceso de búsqueda

El proceso de búsqueda de respuestas del sistema... Se puede ver un diagrama de secuencia en la Figura 2.

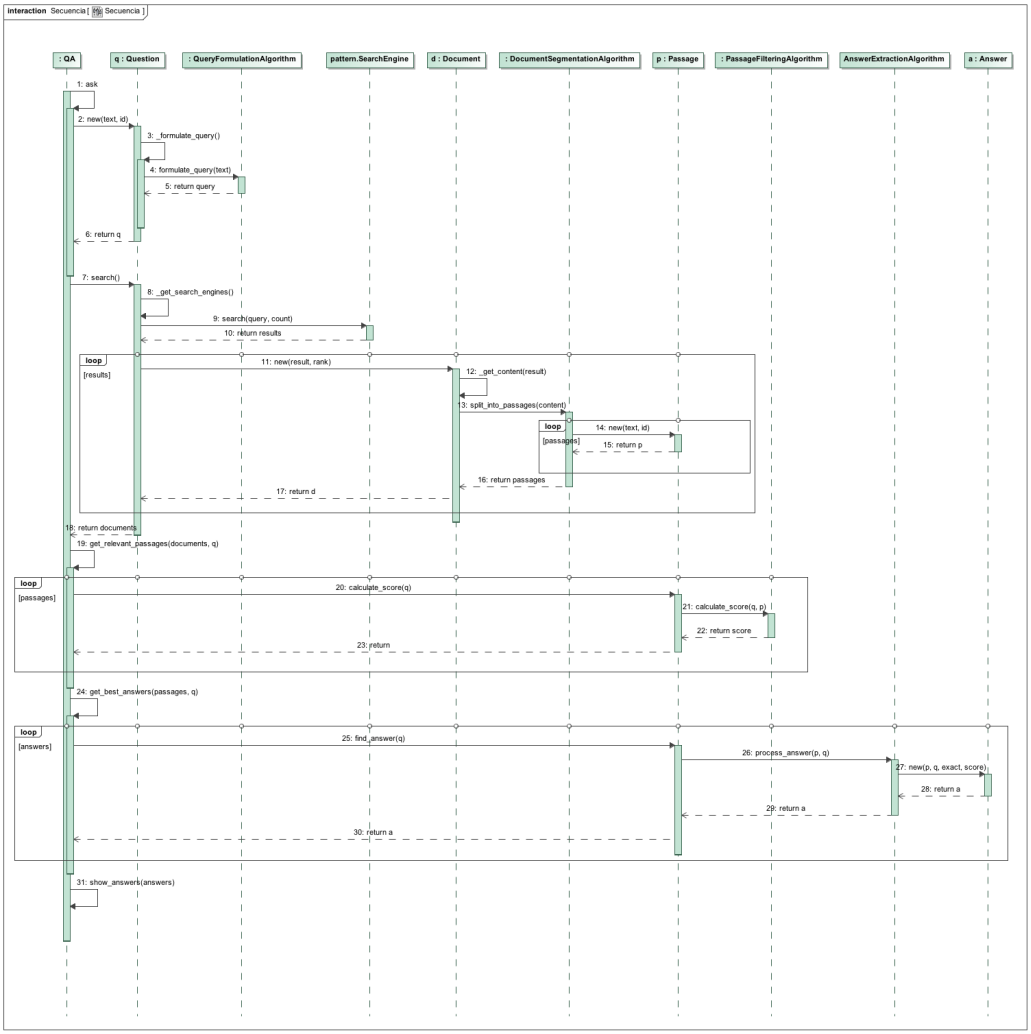

Figura 2: Diagrama de secuencia del proceso de búsqueda de respuestas

2. Herramientas empleadas

2.1. Toolkits

NLTK.

2.2. Sistema de control de versiones

Git y Bitbucket

3. Manual de instalación y uso

3.1. Instalación del sistema

3.1.1. NLTK

sudo pip install -U nltk nltk.download()

3.1.2. Google API

sudo pip install -U google-api-python-client sudo pip install -u pdfminer

3.1.3. Bing API

3.2. Uso del sistema

4. Diario de trabajo

A continuación se muestra un resumen del trabajo realizado a lo largo de la elaboración de la presente práctica. Las entradas se ordenan por su fecha cronológica y describen brevemente las decisiones y las acciones tomadas.

2012/10/26

En la primera reunión del grupo, hemos discutido sobre los primeros pasos a la hora de enfrentar la práctica. Tras un análisis de los diferentes *toolkits* que se nos presentan en el enunciado de la práctica, nos decidimos a usar NLTK por dos razones principales. El primer motivo es la gran cantidad de módulos que posee y su amplia documentación. En segundo lugar, porque Python nos parece un lenguaje muy cómodo para el desarrollo del proyecto.

Profundizando más en el desarrollo del trabajo, decidimos usar Git como sistema de control de versiones y apoyarnos en un repositorio privado de Bitbucket. Esto nos permitirá tener nuestro código bien organizado y documentado así como proporcionarnos un respaldo de los datos.

Por último, acordamos documentarnos más sobre el uso de Python en el procesamiento de lenguaje natural en general y con NLTK en particular. Para ello recurriremos a la bibliografía recomendada por los creadores del toolkit [1]. También optamos por estudiar las APIs de Google y de Bing para realizar consultas.

2012/10/29

Hemos instalado y configurado NLTK. Hemos estudiado utilizar el módulo Pattern (en Python) para la implementación de las consultas en los buscadores (Google y Bing). Hemos solicitado unas claves para poder utilizar las APIs de dichos buscadores.

Por otro lado, hemos comenzado a estudiar la formulación de la consulta (*query formulation*). Debemos eliminar aquellas palabras innecesarias (*stopwords*) por lo que utilizamos el diccionario de Porter ya integrado en NLTK. Consideramos mantener las comillas y los apóstrofes ya que dan mejores resultados en los buscadores. Optamos por no eliminar la interrogación final puesto que es irrelevante para ellos.

2012/11/05

Se ha dedicado la tarde del presente día a la implementación de la búsqueda de información en los buscadores web (a saber, Google y Bing). Hemos

comenzado empleando la biblioteca Pattern[2] la cual nos aporta una interfaz adaptador entre el API de los buscadores y nuestro sistema. No obstante, tras las pruebas iniciales comprobamos que el servidor de Google devolvía un error HTTP400 Bad Request a nuestras peticiones.

Al no encontrar solución a este problema, dedujimos que sería un bug en la biblioteca Pattern por lo que comenzamos a desarrollar nuestra propia biblioteca para integrar las APIs de los buscadores. Tras la integración del API de Google, verificamos en las pruebas que se producía el mismo error. Buscando posibles soluciones en la web, averiguamos que el origen del error 400 no era una petición HTTP mal formada si no un error de autenticación: la clave del API que estábamos utilizando era incorrecta.

Comprobamos que efectivamente ese era el error y decidimos volver a utilizar la biblioteca Pattern en vez de seguir implementando la nuestra porque esta ya nos proporciona una acceso unificado a la información de los buscadores.

2012/11/06

Se ha realizado la planificación de la arquitectura del sistema de búsqueda de respuetas con el objetivo de orientar el desarrollo de los componentes que faltan. Hemos concluido crear las clases `QA`, `Query`, `Document`, `Passage` y `Answer` tras un esbozo del diagrama de clases que se ha construido a partir de un estudio de los casos de uso.

2012/11/12

Durante la sesión de hoy hemos continuado refinando nuestro sistema realizando una adaptación de las funcionalidades ya implementadas a la arquitectura del sistema esbozada en la sesión previa.

Además se ha implementado una sencilla interfaz para la realización de las consultas. Por un lado, se ofrece la posibilidad de introducir una única consulta manualmente en el sistema y, por otro lado, permitiremos el uso de un fichero externo para realizar múltiples consultas mediante el paso de un fichero como parámetro en nuestro programa principal. El tratamiento de dicho fichero de preguntas se realiza mediante una expresión regular con el fin de proporcionar mayor robustez al sistema.

En último lugar desarrollamos la lógica necesaria para gestionar un fichero de configuración.

2012/11/13

En primer lugar, hemos decidido crear una clase `MyConfig` cuyo objetivo es encapsular la forma de acceder al fichero de configuración.

A continuación hemos utilizado el módulo pickle de Python para serializar los documentos resultantes de realizar la consulta en los buscadores.

Por último, se ha desarrollado código capaz de manejar distintos tipos de documentos que nos podemos encontrar por la red:

- Los ficheros HTML se tratan con un procesador dedicado a ello que elimina las etiquetas, el código CSS o JavaScript, etc.
- Los archivos en texto plano no necesitan procesamiento.
- Aquellos documentos cuyo formato no reconocemos[1] son manejados de una forma genérica: mediante una expresión regular extraemos los caracteres legibles de los mismos.

2012/11/19

Hemos comenzado con la división de los documentos en pasajes. Consideramos oportuno que los fragmentos sean solapados para evitar la posible pérdida de información. En principio hemos decidido segmentar los documentos por líneas —el número de ellas será especificado en el fichero de configuración—.

Posteriormente, se ha implementado el uso de logs mediante el módulo logging de Python. Hemos creado un fichero de configuración para tal efecto.

2012/11/20

En primer lugar, hemos reestructurado el proyecto para utilizar el patrón estrategia para los algoritmos de formulación de consultas, puntuación de pasajes y procesamiento de respuestas.

Hemos desarrollado nuestra primera heurística para la evaluación de la relevancia de los pasajes en la búsqueda de respuestas. El algoritmo se basa en la similitud entre el pasaje y la pregunta. Se limpia la pregunta y el pasaje de símbolos y stopwords. A continuación se calcula el número de palabras coincidentes. Dividimos dicha cantidad entre el número de palabras en el pasaje y lo ponderamos por la relevancia del resultado según el buscador.

[1] En un inicio pensamos en tratar de forma especial documentos ofimáticos (.doc, .docx, .xls, .xlsx, .ppt, .pptx, OpenDocument...); sin embargo, la probabilidad de encontrárselos como resultado de la búsqueda generada por una pregunta factual es muy baja y el tratamiento genérico que hemos implementado da unos resultados satisfactorios.

Por otro lado, se ha refinado el algoritmo de formulación de consultas eliminando símbolos no deseados.

2012/11/26

Decidimos utilizar otro patrón estrategia para la segmentación del texto en pasajes. Creamos una estrategia nueva que segmenta el texto en párrafos, complementando asi a la estrategia de segmentar por un número fijo de líneas.

Mejoramos el algoritmo de filtrado de pasajes relevantes por similitud aplicando *stemming* a la pregunta y al pasaje mediante el uso del Stemmer de Porter. Probamos además el uso de un lematizador basado en WordNet, pero tras su prueba descartamos su utilización ya que discriminaba menos casos similares.

Documentamos el fichero de configuración con los posibles algoritmos a emplear (formulación de consultas, segmentación en pasajes, búsqueda de pasajes relevantes y procesamiento de respuestas).

Implementamos lógica adicional para la gestión de las respuestas y su correcta visualización de acuerdo a las normas del enunciado de la práctica.

2012/11/27

Durante el día de hoy se ha diseñado el gráfico de arquitectura del sistema, estableciendo los diferentes módulos que componen el mismo:

- Query Formulation
- Document Retrieval
- Passage Retrieval
  - Document Segmentation
  - Passage Filtering
- Answer Processing
  - Answer Extraction
  - Answer Filtering

Así mismo hemos elaborado un diagrama de secuencia en el que se expone el funcionamiento global del sistema (puesto que éste solo contempla un único caso de uso: realizar consultas).

5. Bibliografía

Referencias

[1] S. Bird, E. Klein, E. Loper: *Natural Language Processing with Python — Analyzing Text with the Natural Language Toolkit*, O'Reilly Media (2009)

[2] CLIPS: Pattern. http://www.clips.ua.ac.be/pages/pattern