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. Introducción

El objetivo de la presente práctica es realizar la implementación de un sistema de búsqueda de respuestas (*question answering*). Para ello, en primer lugar, diseñaremos una arquitectura general del sistema. A continuación, se especificará el funcionamiento de cada uno de los módulos y como se implementará.

Consideraremos diferentes estrategias a la hora de resolver los diferentes problemas y las evaluaremos para quedarnos con las que nos den mejores resultados.

2. Arquitectura del sistema

Basándonos en varios modelos conceptuales de un sistema de búsqueda de respuestas ([1], [2]) y el dado en la asignatura, hemos desarrollado nuestra propia arquitectura que puede verse en la Figura 1.

Query Formulation A partir de una pregunta en lenguaje natural genera una consulta (*query*) que será posteriormente interpretada por un motor de búsqueda web.

Document Retrieval Obtiene una lista de documentos tras realizar la consulta pertinente en los motores de búsqueda.

Passage Retrieval Devuelve los pasajes relevantes de la lista de documentos anterior.

- **Document Segmentation** Divide cada documento en pasajes.
- **Passage Filtering** Selecciona los pasajes más relevantes.

Answer Procesing Genera las respuestas asociadas a los pasajes relevantes.

- **Answer Extraction** Extrae las respuestas asociadas a cada pasaje.
- **Answer Filtering** Filtra las mejores respuestas.

2.1. Proceso de búsqueda

Para ilustrar el proceso de búsqueda de respuestas del sistema, hemos elaborado un diagrama de secuencia que puede verse en la Figura 2.

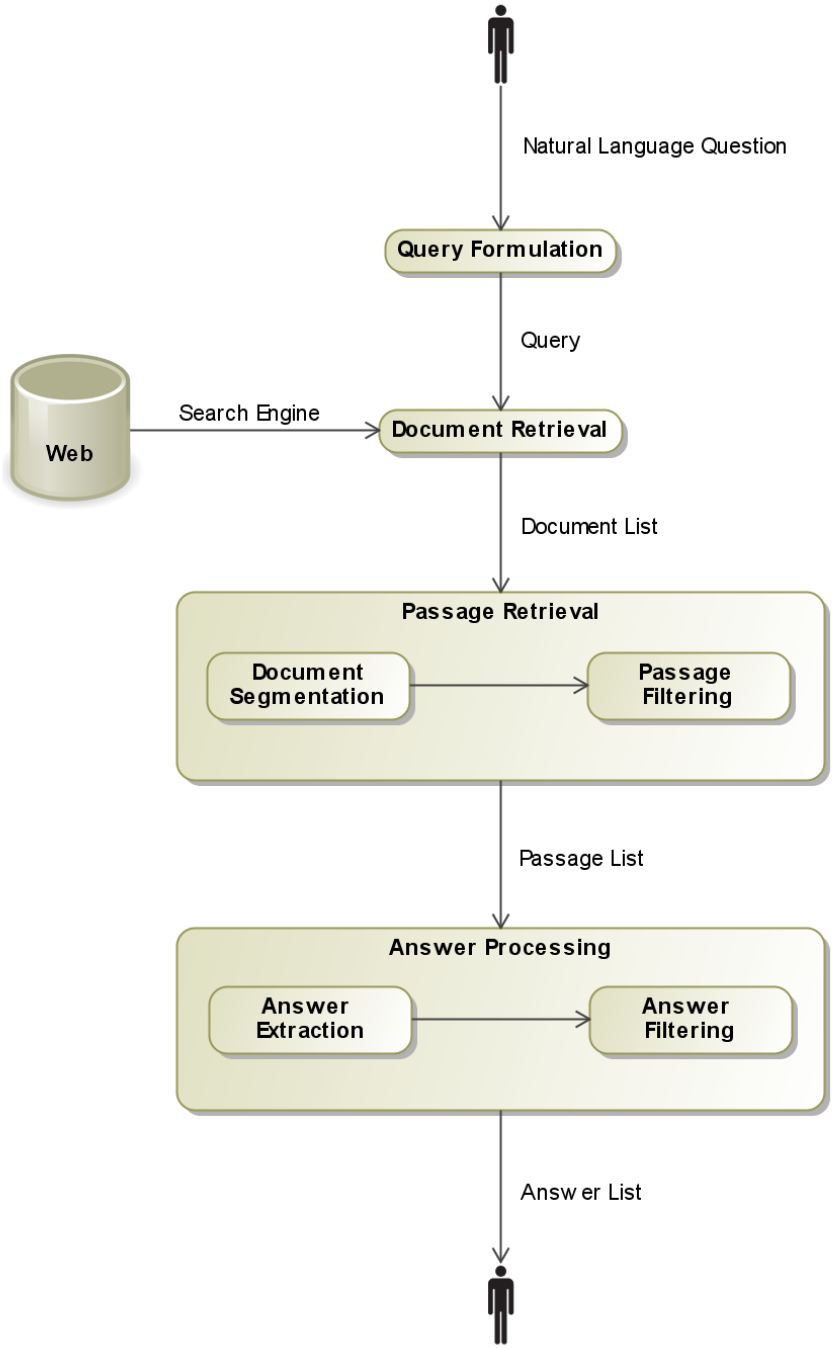

Figura 1: Arquitectura general del sistema

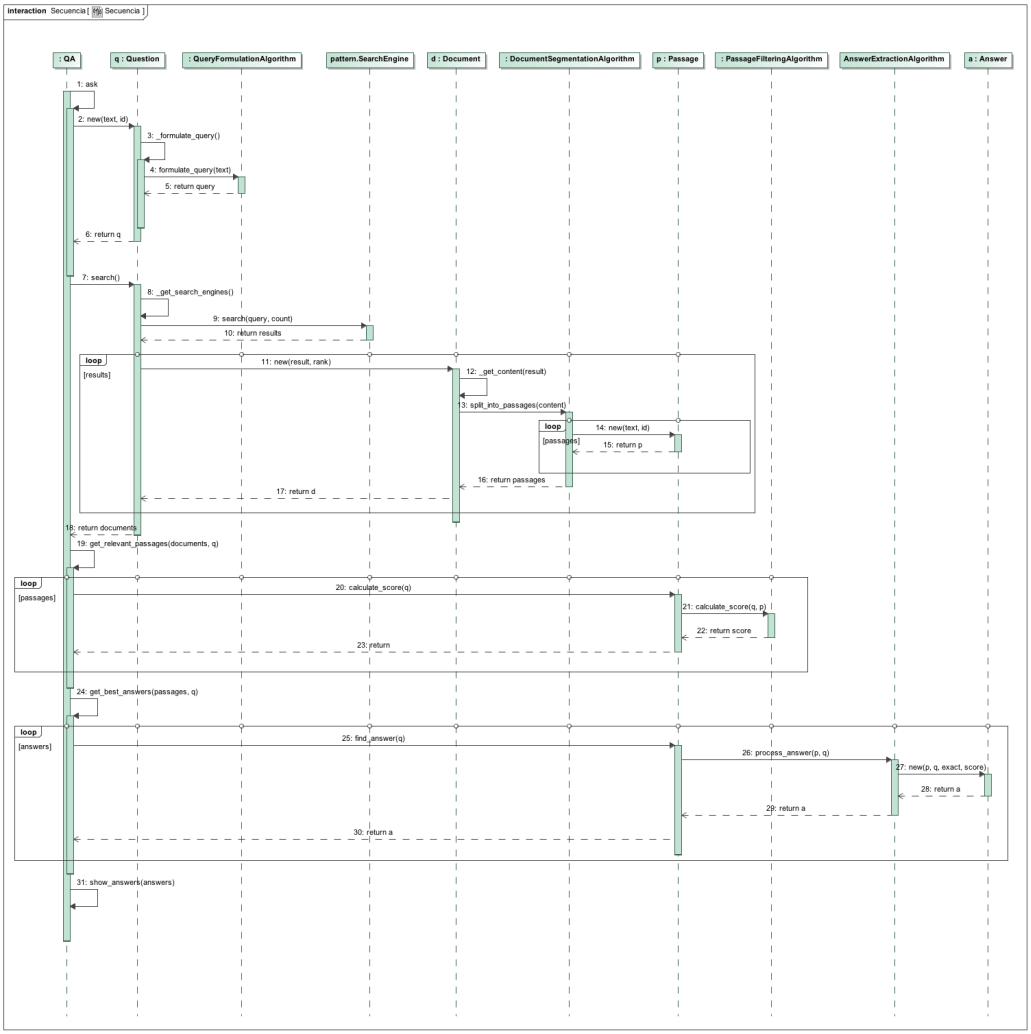

Figura 2: Diagrama de secuencia del proceso de búsqueda de respuestas

3. Herramientas empleadas

NLTK Framework libre de procesamiento de lenguaje natural para Python. Puede descargarse en [3].

Pattern Módulo de *web mining* libre para Python. Puede descargarse en [5].

PDFMiner Módulo de Python para la extracción de información de documentos PDF. Más información en [6].

lxml Módulo de Python...

Stanford NER

Stanford Parser

Git Sistema de control de versiones distribuido libre. Puede descargarse en [7].

BitBucket Repositorio online para git. Puede accederse a él en [8].

LaTeX Sistema de composición de textos con el que se ha elborado el presente documento. Más información en [9].

4. Manual de instalación y uso

4.1. Instalación del sistema

4.1.1. NLTK

sudo pip install -U nltk nltk.download()

4.1.2. Google API

sudo pip install -u pdfminer

4.1.3. Bing API

4.2. Uso del sistema

5. Algoritmos

5.1. Query Formulation

El objetivo de estos algoritmos es formular una consulta para los buscadores web a partir de la pregunta introducida por el usuario.

5.1.1. Stopwords

Este algoritmo nos devuelve una query. Es llamado cada vez que se crea un objeto de tipo Question y recibe el texto de la pregunta. Su funcionamiento consiste en primero pasar el texto de la pregunta a minúsculas y , a continuación, obtenemos una lista con los caracteres de la pregunta. De esta lista eliminamos los símbolos que no nos interesan (comas, interrogaciones, exclamaciones, etc.) puesto que no nos influyen en la búsqueda. Tenemos así el texto de la pregunta filtrado.

Finalmente obtenemos otra lista con las palabras de la pregunta y, utilizando el corpus del NLTK con las *stopwords*, eliminamos las palabras irrelevantes a la hora de realizar la búsqueda. Tenemos así nuestra query lista para pasársela a los buscadores.

5.2. Document Segmentation

Los siguientes algoritmos son llamados cada vez que se crea un objeto de tipo Documento; les pasaremos un documento y nos devolverán la lista de pasajes de dicho documento.

5.2.1. Split into Lines

En este primer algoritmo obtenemos pasajes compuestos por un número fijo de líneas (a determinar por el usuario, en caso de fallo al obtenerlo, dicho número será 5) del documento superponiendo dichos pasajes. Para ello, se divide el contenido del documento en líneas y se itera sobre estas líneas de manera que cada pasaje estará formado por el número de líneas fijado. En cada nueva iteración, un pasaje estará formado por dicho número de líneas menos una del pasaje de la iteración anterior más una nueva línea. Por último se añade también como pasaje la descripción que nos ofrece el motor de búsqueda.

5.2.2. Split into Paragraphs

En este segundo caso, la forma de obtener pasajes cambia para obtener en cada pasaje un único párrafo. Se divide el contenido del documento en líneas y cada pasaje estará formado por una de ellas. Añadiendo además, como en el caso anterior, el pasaje formado por la descripción ofrecida por el motor de búsqueda.

5.2.3. Split into Sentences

5.3. Passage Filtering

El siguiente algoritmo fijará una determinada puntuación a cada pasaje que necesitamos saber cuando queremos obtener los pasajes más relevantes para una determinada pregunta.

5.3.1. Similarity

En primer lugar, al texto de la pregunta y del pasaje les eliminamos las stopwords con el algoritmo de Query Formulation: Stopwords y dividimos ambos textos en palabras obteniendo dos listas. A continuación, aplicamos stemming sobre ambas listas, obtenemos las palabras que están en ambas listas y este número de palabras coincidentes será nuestra puntuación inicial.

Por último, los pasajes que pertenezcan a documentos obtenidos de respuestas situadas en las posiciones más altas del ranking de respuestas devueltas por los motores de búsqueda obtendrán mayor puntuación por considerarse que esto indica que son mejores. Normalizamos el ranking del documento al que pertenece el pasaje que estamos puntuando y multiplicamos la puntuación inicial anterior por esta cantidad, obteniendo así, la puntuación final del pasaje.

5.3.2. Proximity

5.3.3. Mixed

5.4. Answer Extraction

5.4.1. Entity Recognition

6. 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.

6.0.2. 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 [4]. También optamos por estudiar las APIs de Google y de Bing para realizar consultas.

6.0.3. 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.

6.0.4. 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[?] 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.

6.0.5. 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.

6.0.6. 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.

6.0.7. 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.

6.0.8. 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.

6.0.9. 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.

6.0.10. 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.

6.0.11. 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).

6.0.12. 2012/12/03

Corrección de múltiples errores en diferentes módulos de la práctica.

Hemos considerado que una consulta en diferentes motores de búsqueda nos puede dar los mismos documentos como resultado; por tanto, se ha decidido controlar la posible repetición de documentos para evitar duplicados.

6.0.13. 2012/12/04

Añadimos la escritura de resultados en un fichero.

Implementamos la lógica necesaria para gestionar la posible falta de respuestas. Por un lado, si los motores de búsqueda no nos devuelven resultados deolvemos NIL con la puntuación máxima. Por otro lado, si no encontramos una buena respuesta en los documentos de la web devolvemos NIL con la puntuación mínima. Por último si el número de respuestas admisibles es menor que tres, las mostramos y añadimos un NIL con puntuación mínima.

En el fichero de configuración, podemos especificar el umbral de lo que consideramos una respuesta admisible.

6.0.14. 2012/12/10

Creamos un algoritmo de Answer Extraction basado en la técnica de Named Entities Recognition explicado en la Sección 5.4.1. Queda por completar la parte de Query Classification (de momento solo admitimos preguntas sobre personas).

Mejoramos nuestra metodología de Answer Filtering ponderando las puntuaciones de las respuestas por su frecuencia relativa de aparición en los pasajes relevantes.

Realizamos las primeras pruebas sobre un sistema ya funcional y obtenemos buenos resultados —en principio— para preguntas relacionadas con personas. Las respuestas están motivadas correctamente y la tasa de aciertos ronda el 20 %. Consideramos esto un hito en el desarrollo del sistema ya que tenemos una primera versión preliminar que proporciona resultados coherentes.

6.0.15. 2012/12/11

Dedicamos el día de hoy a implementar técnicas de Question Classification. Utilizamos dos modelos: un clasificador bayesiano ingenuo y un modelo de máxima entropía proporcionados por NLTK.

Modificamos el corpus qc del NLTK para que se adapte a nuestras entidades y entrenamos ambos sistemas. Obtenemos una precisión algo más elevada en el caso del modelo de máxima entropía.

Consideramos realizar para el próximo día la integración del Named Entity Recognizer de Stanford puesto que los resultados del reconocedor de entidades del NLTK no son muy satisfactorios en algunos casos.

6.0.16. 2012/12/17

Hemos implementado dos técnicas para detectar números y otros tipos de entidades (NUMBER y OTHER, respectivamente). Los números se reconocen como caracteres numérico o como sustantivos y adjetivos numerales. El reconocimiento de otro tipo de entidades se realiza mediante la extración de sustantivos (exceptuando aquellos que son entidades).

Realizamos también pequeñas refactorizaciones del código. Optimizamos la técnica de Question Classification mediante el uso de una caché de preguntas.

6.0.17. 2012/12/18

En el día de hoy, se ha integrado el Named Entity Recognition de Stanford con el código actual mediante comunicación via un socket TCP. Ahora podemos utilizar ambos reconocedores de entidades (será necesario realizar una comparativa entre ellos).

6.0.18. 2012/12/26

Concertamos una reunión con el profesor responsable de las prácticas para realizar un seguimiento de la misma y plantearle unas dudas sobre el reconocimiento de entidades tipo Other.

Realizamos correciones menores en el soporte de Unicode, el timeout de descarga de ficheros, el salvado de resultados en ficheros, la selección mediante el fichero de configuración del tipo de motor de NER o la falta de clasificadores de preguntas.

Por otro lado, hemos mejorado el Passage Retrieval consdirando también los snippets que nos proporcionan los buscadores como posibles pasajes relevantes. También modificamos la lógica del Answer Extraction para eliminar aquellos términos que ya aparecen en la pregunta como posibles respuestas (no nos van a preguntar de qué color es el caballo blanco de Santiago).

Tras realizar unas pruebas para evaluar la calidad de nuestro sistema, observamos que si el Question Classification falla, es imposible obtener una respuesta válida. Por tanto, nos hemos propuesto mejorarlo con la inclusión

de más *features* a nuestros clasificadores. Hemos probado con añadir un segundo sustantivo, pero vimos que empeoraba el rendimiento en el conjunto de test. A continuación, intentamos aproximarnos al problema utilizado chunking para obtener la *head word* más característica; sin embargo, la calidad del análisis sintáctico superficial no era suficiente para obtener buenos resultados. Por último, hemos contemplado la posibilidad de usar análisis sintáctico completo. En el trabajo de Babak Loni [10] se comenta el uso del *full parser* de Stanford en combinación con un algoritmo diseñado por él para la obtención de la *head word*. Queda pendiente dicho trabajo para el próximo día.

6.0.19. 2013/01/02

Hemos adaptado el *full parser* de Stanford para que se comunique mediante *pipes* con nuestra práctica. Implementamos el algoritmo descrito en [10] y [11] para obtener *head words* usando el analizador sintáctico antes descrito. Entrenamos nuevos clasificadores de preguntas con diferentes características y comprobamos que el que mejor se comporta es el clasificador bayesiano ingenuo que tiene como características de entrada la primera palabra, el primer sustantivo y la *head word*.

Se ha elaborado un nuevo algoritmo de Document Segmentation que divide un texto en pasajes de n oraciones solapadas. Por otro lado, se ha diseñado un algoritmo de Passage Filtering que examina la proximidad de los términos de la pregunta en el pasaje con el fin de puntuarlo. Por último, se ha creado un método híbrido para el filtrado de pasajes que combina el algoritmo por "similitud" y el de "proximidad" para ponderar ambas aproximaciones.

Se ha refinado el tratamiento de excepciones especialmente en lo referido a problemas en la obtención de documentos y en el tratamiento de caracteres Unicode.

En último lugar, hemos realizado pruebas con diferentes parámetros en nuestros algoritmos.

6.0.20. 2013/01/11

Hemos dedicado el día de hoy a realizar pruebas de la práctica con diferentes parámetros para obtener los valores óptimos de los mismos. También hemos corregido ciertos fallos que nos han surgido tras la búsqueda de respuestas en serie.

Por restricciones de cuota en los motores de búsqueda de Google y Bing, no se pueden realizar las 50 búsquedas consecutivas. Para evitar errores en

la obtención de documentos, lo mejor es ejecutar la búsqueda sobre ficheros de hasta 30 preguntas espaciados temporalmente por, al menos, una hora.

7. Bibliografía

Referencias

[1] J. Lin, B. Katz: *Question Answering Techniques for the World Wide Web*. EACL (2003). `http://www.umiacs.umd.edu/%7Ejimmylin/publications/Lin_Katz_EACL2003_tutorial.pdf`

[2] B. Magnini: *Open Domain Question Answering: Techniques, Resources and Systems*. RANLP (2005). `http://lml.bas.bg/ranlp2005/tutorials/magnini.ppt`

[3] NLTK: `http://www.nltk.org`.

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

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

[6] PDFMiner: `http://www.unixuser.org/~euske/python/pdfminer/index.html`.

[7] Git: `http://www.git-scm.com`.

[8] BitBucket: `http://www.bitbucket.org/daniel_garabato/ln`.

[9] LaTeX: `http://www.latex-project.org`.

[10] B. Loni: *Enhanced Question Classification with Optimal Combination of Features*. Ed. Delft University of Technology, 2011.

[11] J. Silva et al: *From symbolic to sub-symbolic information in question classification*. Ed. Springer Science, 2010.