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 en la que se especificará el objetivo de cada uno de los módulos.

Para implementar las distintas unidades funcionales de la práctica, implementaremos nuestros propios algoritmos, pero también haremos uso de múltiples herramientas de terceros. Describiremos dichas herramientas y para qué las hemos utilizado.

Consideraremos diferentes estrategias a la hora de resolver las diferentes problemáticas. Implementaremos aquellas que estén a nuestro alcance y las evaluaremos para quedarnos con las que nos den mejores resultados.

Se incluye en esta memoria un manual de instalación de la aplicación puesto que al hacer uso de múltiples herramientas de diferentes tecnologías es necesario la instalación de numerosos paquetes. Por otro lado, la memoria también contiene unas instrucciones de uso de la práctica.

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. Vista estática

Con el objetivo de representar los distintos elementos que dan soporte a nuestro sistema se ha elaborado un diagrama de clases que puede observarse en la Figura 2.

En la vista estática se representan las clases que se desarrollarán para implementar las funcionalidades requeridas de la misma y sus propiedades, tanto sus atributos y métodos como sus relaciones con otras clases del dominio. Además, se muestra también la conexión entre los elementos propios que hemos desarrollado y los módulos o paquetes de terceros que se han utilizado.

2.2. Vista dinámica

Para ilustrar el proceso de búsqueda de respuestas del sistema, hemos elaborado un diagrama de secuencia que puede verse en la Figura 3. En dicho diagrama se puede observar el flujo del programa durante la búsqueda de respuestas ante una pregunta realizada por el usuario por línea de comandos.

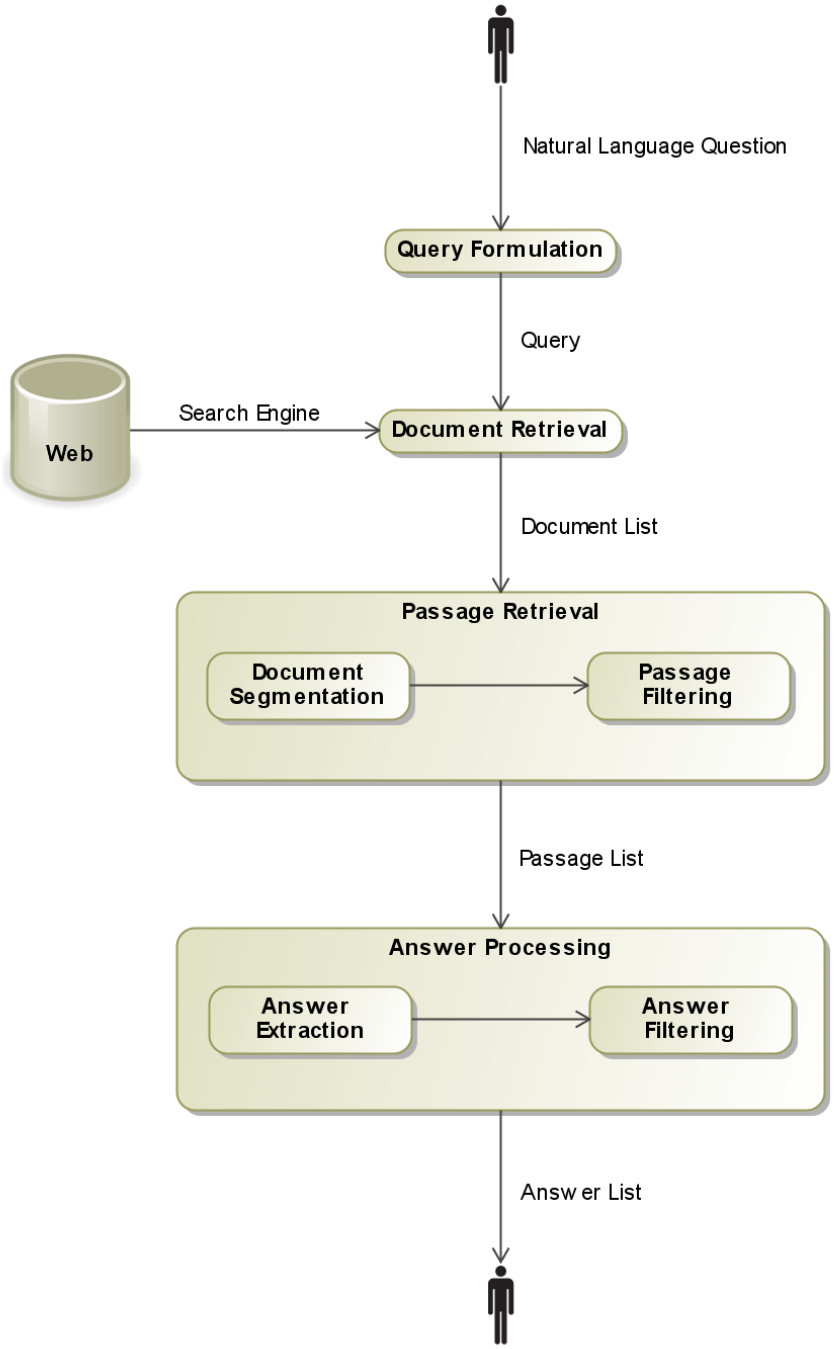

Figura 1: Arquitectura general del sistema

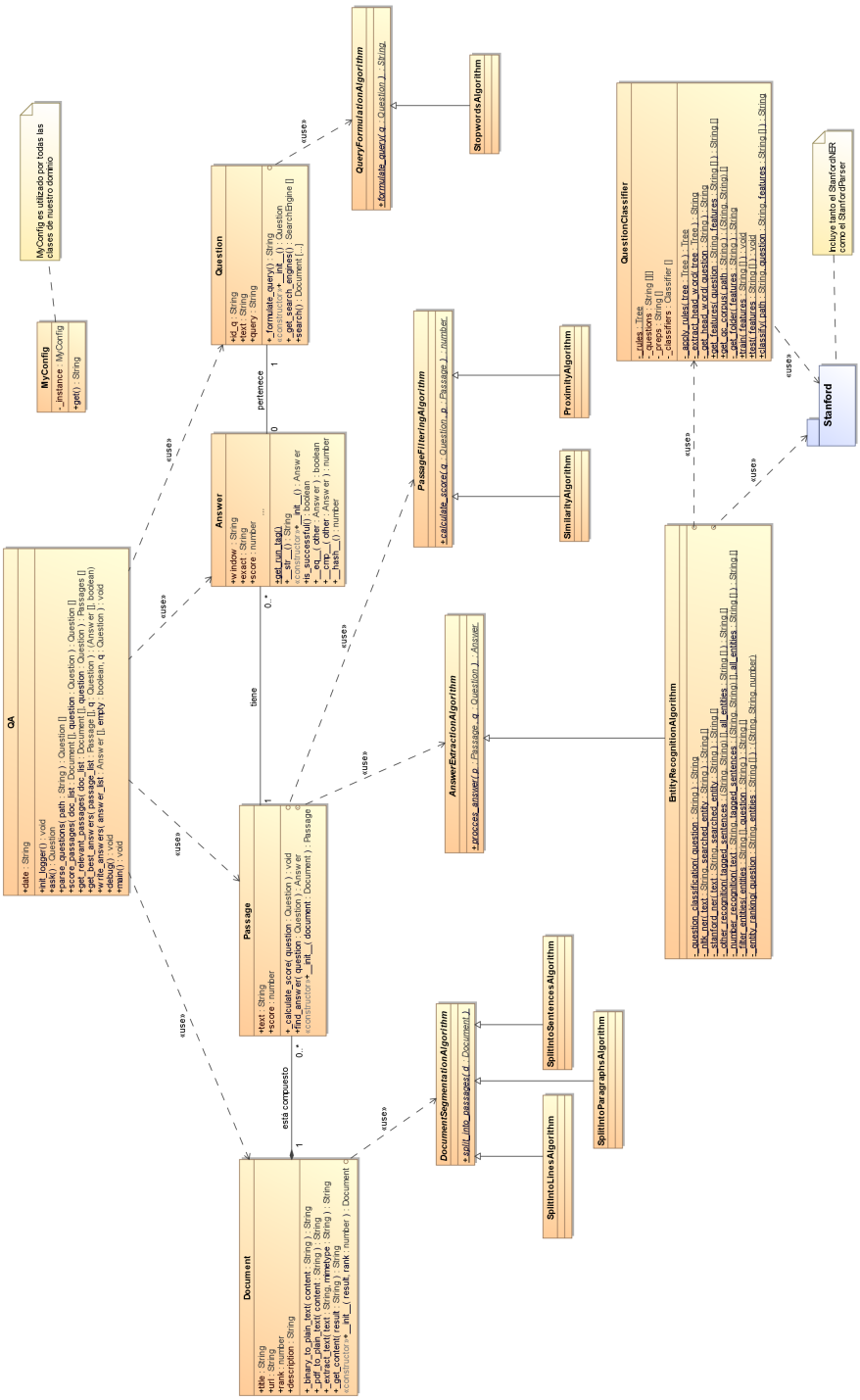

Figura 2: Diagrama de clases de la práctica

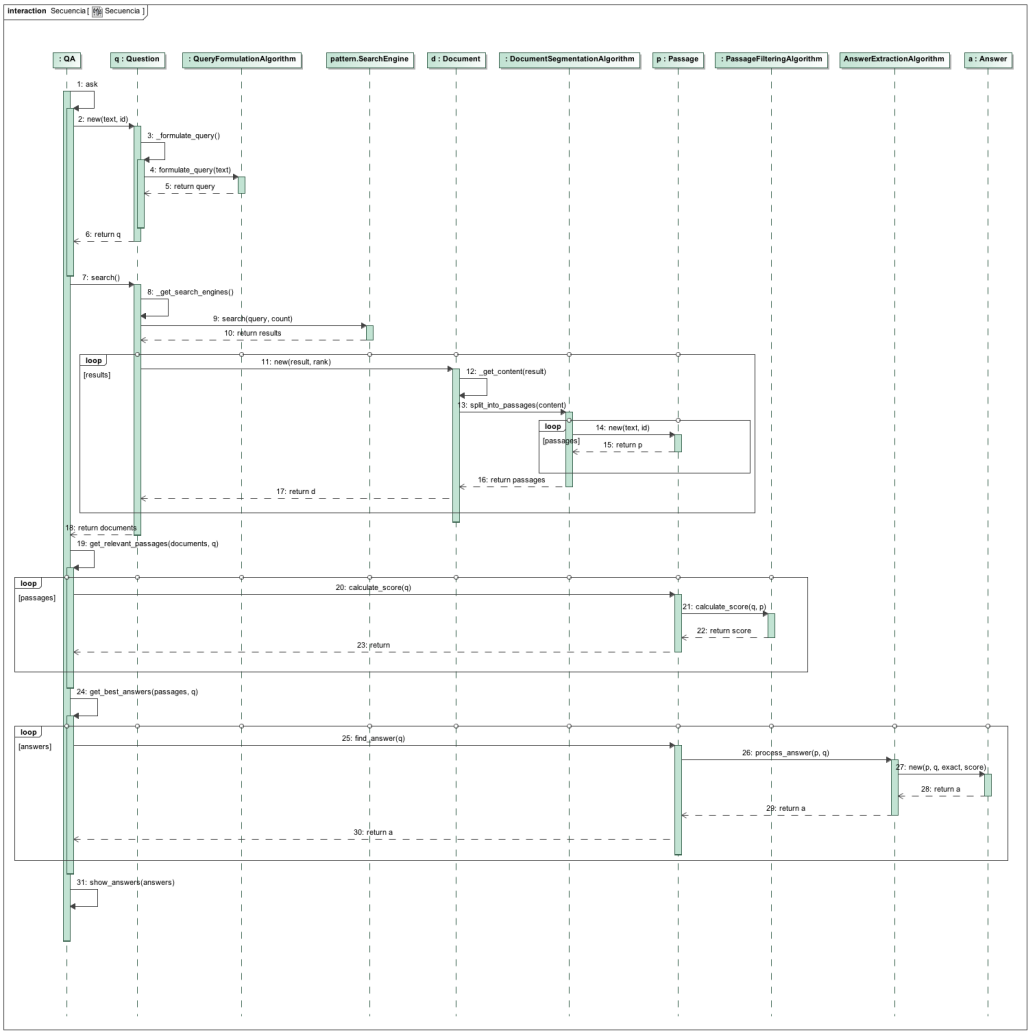

Figura 3: Diagrama de secuencia del proceso de búsqueda de respuestas

3. Herramientas empleadas

Durante el desarrollo de la práctica se emplearon diversas utilidades de terceros que han permitido facilitar el desarrollo de la misma y simplificar, parcialmente, su complejidad mediante la incorporación de módulos ya desarrollados que posibilitan su elaboración, puesto que sin ellos el alcance de la implementación sería inabordable. A continuación se especifican cuales son las herramientas principales en las que nos hemos basado.

NLTK Se trata de una plataforma libre para el desarrollo de aplicaciones en lenguaje Python que permite llevar a cabo un procesamiento del lenguaje natural.

NLTK proporciona interfaces para tratar con múltiples *corpus* y diversos recursos léxicos, como por ejemplo *WordNet*, que permiten llevar a cabo su propósito de tratar el lenguaje natural permitiendo funcionalidades como la realización de análisis sintácticos, tokenización o la clasificación de textos, que han sido algunos de los pilares fundamentales sobre los cuales se ha construído nuestra práctica.

Puede obtenerse más información a cerca del mismo, así como descargarse, en [5].

Pattern Consiste en un módulo libre de *web mining* para Python. Su principal funcionalidad es la extracción de información mediante la realización de consultas a diversos búscadores web como *Google*, *Bing* o *Wikipedia*.

La utilización de este módulo nos ha permitido manejar las API's de los diferentes buscadores a través de la interfaz que proporciona el propio módulo, logrando encapsular el funcionamiento y las particularidades de cada API.

Se puede encontrar más información en [7].

PDFMiner Módulo de Python que permite llevar a cabo la conversión de un documento en formato PDF a texto plano. Esta herramienta nos ha facilitado el tratamiento de los documentos PDF recuperados de la web, ya que, por su propia naturaleza, contienen información adiccional que es irrelevante para nuestro propósito y que, a través de esta utilidad, conseguimos obviar.

Se puede obtener más información en [8].

lxml Consiste en un módulo que posibilita el manejo de datos estructurados en XML y HTML a través de una sencilla interfaz en Python. Puede encontrarse más información sobre el mismo en [9].

Stanford NER También conocido como *CRFClassifier*, se trata de un reconocedor de entidades implementado en Java que permite clasificar las diferentes entidades de un texto según su dominio: nombres, organizaciones, lugares, etc.

Su funcionamiento a través de *sockets* nos ha permitido incorporarlo a nuestra implementación de la práctica. Puede obtenerse más información sobre esta herramienta en [10].

Stanford Parser Se trata de un analizador de lenguaje natural que permite, mediante una etiquetación, identificar los diferentes elementos que componen un texto: frases, sujeto, verbo, etc.

Del mismo modo que el *Stanford NER* se comunica a través de *sockets* con nuestro programa en Python. Se puede encontrar más información sobre el mismo en [11].

Git Sistema de control de versiones distribuido que facilita las tareas de desarrollo del software proporcionando soporte a diferentes versiones del mismo. Puede obtenerse más información en [13].

BitBucket Utilidad online que permite la creación de repositorios centralizados para sistemas de control de versiones como *git* o *mercurial*. Puede accederse a nuestro repositorio en [14].

LaTeX Sistema de composición de textos con el que se ha elborado el presente documento. Más información en [15].

4. Manual de instalación y uso

En esta sección se describen los pasos a seguir para instalar correctamente todos los componentes necesarios para ejecutar nuestra práctica. Asimismo, también se adjuntan las intrucciones detalladas de uso.

A pesar de que las tecnologías utilizadas son multiplataforma, se recomienda utilizar un sistema operativo UNIX. Concretamente, se ha comprobado el correcto funcionamiento de la práctica en ArchLinux 2012.10.06, Debian 7.0, Mac OS X 10.8 y Ubuntu 11.04.

4.1. Instalación del sistema

A continuación se detallan los paquetes necesarios y cómo instalarlos.

4.1.1. Python y sus módulos

Es necesario disponer de una versión moderna de Python 2 (al menos Python 2.6, pero se recomienda Python 2.7). En el sitio web de Python se puede descargar el intérprete [3]. Hemos utilizado únicamente el intérprete recomendado (CPython) con lo que no podemos garantizar el correcto funcionamiento en otras implementaciones como PyPy.

Una vez obtenido Python, es necesario instalar diversos módulos. Para simplificar el proceso, es preciso disponer de las SetupTools de Python. Hay scripts disponibles para diversas plataformas que pueden descargarse de su sitio web [4].

```
# sh path/to/downloads/setuptools-{version}.egg
```

Es preciso instalar Pip para instalar el resto de módulos:

```
# easy_install pip
```

A continuación, instalamos los módulos Numpy, PyYAML, NLTK, PDFMiner, lxml y Pattern con Pip:

```
# pip install -U numpy pyyaml nltk pdfminer lxml
  pattern
```

Es necesario bajarse los corpus del NLTK para que funcionen correctamente algunas de sus características como el PoS Tagger o el Named Entity Recognition Tagger:

```
$ python
>>> import nltk
>>> nltk.download('all')
```

4.1.2. Java

Aunque nuestra práctica está realizada en Python, utilizamos herramientas del grupo Stanford NLP que están desarrolladas en Java. Estas requieren Java 6 o superior. En la web de Oracle se puede descargar el JDK [12].

Los componentes de Stanford que se son necesarios ya vienen incluidos con el código de la práctica por lo que no es necesario descargarse nada más.

4.1.3. Claves de buscadores

El módulo de obtención de documentos de la web requiere unas *API Keys* de los buscadores a utilizar. En el fichero de configuración de la práctica ya se incluye por defecto unas claves gratuitas de Google y Bing que pueden ser sustituidas por otras claves válidas sin dificultad.

La clave de Google permite realizar 100 consultas por hora, mientras que la de Bing da acceso a 5000 consultas al mes.

4.2. Uso del sistema

El sistema tiene dos modos de funcionamiento principales así como un tercero con objetivos de *debugging*. Con el objetivo de facilitar el uso de nuestro programa, se ha codificado un script en Bash (`launch.sh`) localizado en la raíz del proyecto que permite lanzar fácilmente la práctica.

En cualquiera de los tres modos de ejecución, las respuestas se almacenan en un fichero en la carpeta `res`. El nombre de dicho fichero se obtiene a partir del *timestamp* de la hora de ejecución del programa.

4.2.1. Modo interactivo

El modo interactivo permite al usuario introducir una pregunta por la entrada estándar. El sistema buscará procesará la pregunta y devolverá la respuesta en el fichero de respuestas.

Para iniciar este modo, debemos ejecutar el script con la opción *interactive*:

```
$ ./launch.sh interactive
```

O bien ejecutar directamente el código Python:

```
$ cd src
$ python QA.py
```

4.2.2. Modo *batch*

Para el procesamiento de preguntas por lotes, nuestra práctica permite especificar un fichero en el que se leerán una serie de preguntas que serán procesadas secuencialmente. Los resultados de cada pregunta se encuentran todos en el mismo fichero.

El formato del fichero de preguntas es el siguiente. Cada pregunta se encontrarán en una línea y estará precedida por un código alfanumérico (sin blancos). A continuación de dicho código, el resto de la línea será considerado el cuerpo de la pregunta.

Este modo de ejecución puede ser invocado desde el script con la opción *batch* seguido de la ruta del fichero de preguntas: *interactive*:

```
$ ./launch.sh batch <file>
```

También se puede ejecutar directamente el código Python:

```
$ cd src
$ python QA.py <file>
```

4.2.3. Modo *debug*

Por último, nuestra práctica permite ejecutar el proceso de búsqueda de respuestas a partir de un conjunto de documentos ya obtenidos de los motores de búsqueda. El objetivo de esta modalidad de ejecución es limitar el número de consultas web durante la fase de pruebas debido a los límites de las claves gratuitas de Bing y Google.

Para que funcione correctamente este módulo, es necesario realizar en primer lugar una pregunta de modo interactivo o *batch* con la opción *persistence.document* activada en el fichero de configuracion (más información sobre cómo modificar la configuración de la práctica en la Sección 4.3). Esto nos generará un fichero llamado `documentos.pkl` que contiene los documentos obtenidos serializados mediante el módulo Pickle de Python y que es necesario para la ejecución en modo *debug*.

A continuación, solo se necesita ejecutar el script la opción *debug*:

```
$ ./launch.sh debug
```

Alternativamente, se puede ejecutar directamente el código Python:

```
$ cd src
$ python QA.py pickle
```

4.3. Fichero de configuración

Para dotar de mayor flexibilidad a la práctica, se han creado dos ficheros de configuración que permiten modificar el comportamiento de los diversos módulos y del sistema de *logging*.

4.3.1. `src/conf/config.conf`

Permite modificar los parámetros de los diversos módulos del sistema de búsqueda de repuestas. Las posibles opciones se encuentran en forma de comentario dentro del fichero de configuración y tienen nombres autodescriptivos.

4.3.2. `src/conf/logging.conf`

El fichero de configuración del módulo Logging de Python permite establecer qué tipo de *logs* se registran en ficheros de texto o por salida estándar. Para más información, consultar la referencia oficial del módulo [16].

5. Algoritmos

En esta sección se detallan los algoritmos utilizados en el sistema de búsqueda de respuestas.

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 ya que puede resultar relevante según la lógica del buscador.

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

Por último, este algoritmo obtiene pasajes formados por el número de frases indicadas por el usuario. El contenido del documento se divide en frases utilizando el sent_tokenize del NLTK. Se añade además, como en los dos casos anteriores, el pasaje formado por la descripción ofrecida por el motor de búsqueda.

5.3. Passage Filtering

Estos algoritmos fijarán una determinada puntuación a cada pasaje según su relevancia para una determinada pregunta para así poder seleccionar los mejores.

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

Como en el caso del algoritmo anterior primero eliminamos las stopwords con el algoritmo de Query Formulation: Stopwords, dividimos ambos textos en palabras y aplicamos stemming tanto sobre la lista de palabras de la pregunta como la de palabras del pasaje.

A la hora de puntuar los pasajes se valora que estos tengan palabras de la pregunta cercanas entre si, asignándole una puntuación de acuerdo a lo próximas que estén entre sí.

5.3.3. Mixed

Consideramos en este algoritmo una estrategia mixta, asignándole a cada pasaje la media de la puntuación obtenida con los dos algoritmos anteriores (Similarity y Proximity).

5.4. Answer Extraction

Los algoritmos de extracción de respuestas tienen como objetivo hallar una respuesta dados una pregunta y un pasaje relevante.

5.4.1. Entity Recognition

El funcionamiento de este algoritmo se basa en la extracción de entidades del texto de un pasaje. Para ello, en primer lugar, obtenemos la clasificación de la pregunta y, a continuación, recuperamos las entidades adecuadas a dicha clasificación.

Para cada pasaje, devolvemos la entidad que más se repite en la pregunta con una puntuación que determina su frecuencia de aparición en el pasaje.

5.4.1.1 Question Classification

Realizamos una clasificación de la pregunta para saber qué tipo de entidades debemos obtener del pasaje. Para ello hemos entrenado diferentes clasificadores a partir de un corpus de entrenamiento formado por preguntas y su clasificación en Other, Number, Person, Location, Time, Date, Money, Percent y Organization.

Hemos utilizado tres tipos de clasificadores: de Bayes ingenuo, de árbol de decisión y de máxima entropía. A su vez, hemos utilizado diferente combi-

naciones de las siguientes características: primera palabra, primer sustantivo y *head word*.

Para obtener la *head word*, es decir, la palabra clave que indica el tipo de pregunta hemos utilizado el algoritmo descrito en [17] y [18].

Tras probar todas las combinaciones posibles de clasificadores y características, el que mejor resultados da en el conjunto de test ha sido el clasificador bayesiano ingenuo con todas las características.

5.4.1.2 Named Entity Recognition

Una vez obtenida la clasificación de la pregunta, debemos buscar las entidades de ese tipo. Para ello, utilizamos el NER Parser de Stanford que permite la extracción de entidades tipo Time, Location, Organization, Person, Money, Percent y Date; sin embargo, debemos definir estrategias diferentes para los Number y los Other.

Para reconocer entidades Number utizamos el Part-of-Speech Tagger del NLTK obteniendo los numerales cardinales y ordinales (CD y JJ) y una expresión regular que reconoce caracteres numéricos.

Por último, la extracción de entidades tipo Other se realiza obteniendo los sustantivos comunes que no son entidades tras un análisis léxico por parte del NLTK.

Descartamos aquellas respuestas que son palabras que ya aparecen en la pregunta.

También es posible emplear el extractor de entidades del NLTK (se puede indicar en el fichero de configuración); no obstante, hemos visto que proporciona peores resultados.

6. Resultados

En las siguientes tablas se muestran los resultados obtenidos para las pruebas realizadas. Nos quedamos con la cuarta configuración puesto que es con la que obtenemos un mayor MRR.

Id	Document Retrieval	Document Segmentation	Passage Filtering
1	10 resultados	sentences/5/100	mixed
2	10 resultados	paragraph/-/100	mixed
3	10 resultados	sentences/1/200	mixed
4	20 resultados	sentences/1/500	mixed
5	20 resultados	sentences/1/500	proximity
6	20 resultados	sentences/1/500	similarity
7	20 resultados	sentences/5/100	mixed
8	20 resultados	sentences/5/100	proximity
9	20 resultados	sentences/5/100	similarity

Id	run-tag	MRR estricto	MRR permisivo	Respuestas R
1	plnaex031ms	0,24333	0,24333	15
2	plnaex031ms	0,19667	0,19667	11
3	plnaex031ms	0,31	0,31	19
4	plnaex031ms	0,32333	0,32333	18
5	plnaex031ms	0,26	0,26	16
6	plnaex031ms	0,29	0,29	17
7	plnaex031ms	0,28	0,28	16
8	plnaex031ms	0,19	0,21	11
9	plnaex031ms	0,22667	0,22667	13

Id	R o U	NILs	NILs correctos	% R	% R o U
1	15	1	0	30,00 %	30,00 %
2	11	1	0	22,00 %	22,00 %
3	19	1	0	38,00 %	38,00 %
4	18	1	0	36,00 %	36,00 %
5	16	1	0	32,00 %	32,00 %
6	17	1	0	34,00 %	34,00 %
7	16	1	0	32,00 %	32,00 %
8	12	1	0	22,00 %	24,00 %
9	13	2	0	26,00 %	26,00 %

7. 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.

7.1. Jornada del 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 [6]. También optamos por estudiar las APIs de Google y de Bing para realizar consultas.

7.2. Jornada del 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.

7.3. Jornada del 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[7] 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.

7.4. Jornada del 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.

7.5. Jornada del 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.

7.6. Jornada del 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.

7.7. Jornada del 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.

7.8. Jornada del 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.

[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.

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.

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

7.9. Jornada del 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.

7.10. Jornada del 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).

7.11. Jornada del 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.

7.12. Jornada del 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.

7.13. Jornada del 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

7.14. Jornada del 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.

7.15. Jornada del 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.

7.16. Jornada del 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).

7.17. Jornada del 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 [17] 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.

7.18. Jornada del 2013/01/02

Hemos adaptado el *full parser* de Stanford para que se comunique mediante *pipes* con nuestra práctica. Implementamos el algoritmo descrito en [17] y [18] 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.

7.19. Jornada del 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.

7.20. Jornada del 2013/01/24

Ultimamos los últimos detalles de la práctica (solución de algunos bugs, refactorización de código, etc.) y de la memoria (digramas, instrucciones y resultados) para preparar la defensa.

8. Bibliografía

Referencias

[1] J. Lin, B. Katz: *Question Answering Techniques for the World Wide Web*. EACL (2003). http://www.umiacs.umd.edu/~jimmylin/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] Python: http://www.python.org.

[4] Python SetupTools: http://pypi.python.org/pypi/setuptools.

[5] NLTK: http://www.nltk.org.

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

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

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

[9] lxml: http://lxml.de/.

[10] Stanford NER: http://nlp.stanford.edu/software/CRF-NER.shtml

[11] Stanford Parser: http://nlp.stanford.edu/software/lex-parser.shtml

[12] Oracle Java: http://www.oracle.com/us/technologies/java/overview/index.html.

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

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

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

[16] Configuración del módulo Logging de Python: http://docs.python.org/2/library/logging.config.html.

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

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