Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Question answering de dominio abierto y de dominio cerrado

Tesis presentada para optar al título de
Licenciado en Ciencias de la Computación

Julián Peller

Director: José Castaño

Buenos Aires, 2016

QUESTION ANSWERING DE DOMINIO ABIERTO Y DE DOMINIO CERRADO

Question answering es un área de ciencias de la computación que busca generar respuestas concretas a preguntas expresadas en algún lenguaje natural. Es un área compleja que combina herramientas de búsqueda y recuperación de la información (*information retrieval*), de procesamiento del lenguaje natural (*nlp*) y de extracción de información (*information extraction*). Por poner un ejemplo: para el input *"¿Cuándo nació Noam Chomsky?"* un sistema de question answering debería devolver algo como "*el 7 de diciembre de 1928*". Esta área representa el paso lógico posterior a los sistemas de recuperación de documentos y logró en los último años una serie de hitos impulsados por el proyecto general de la web semántica. Watson, el sistema desarrollado por IBM que derrotó a los mejores competidores de Jeropardy! es el ejemplo más visible, pero incluso buscadores como Bing! y Google comienzan a incorporar este tipo de algoritmia.

En esta tesis investigamos los distintos problemas que se subsumen bajo el concepto de question answering y reseñamos diferentes soluciones y modelos aplicados para resolverlos, bajo el proyecto de la implementación de dos sistemas básicos de question answering. El primer sistema implementado es un modelo de dominio cerrado (específico) y datos estructurados solo para inglés. El segundo modelo es un sistema multilingüe, de dominio abierto y que utiliza como corpora las wikipedias de diferentes idiomas. Para el primer modelo orientamos nuestro desarrollo de acuerdo al modelo teórico del paper [Popescu et al., 2003a] e implementamos soluciones para un conjunto restringido de preguntas. Para el segundo modelo utilizamos un subconjunto de los problemas de la competencia CLEF '07 y desarrollamos el sistema utilizando como baseline el framework Qanus, adaptándolo para utilizar herramientas de procesamiento de lenguaje multilingües de la librería Freeling.

Palabras claves: Question Answering, Closed Domain, Open Domain, Multilingual, Freeling, Qanus, CLEF, Semantic Tractability

OPEN AND CLOSED DOMAIN QUESTION ANSWERING

Question answering is a computer science area that aims to generate concrete responses to questions posed in some natural language. It's a complex area that combines information retrieval, natural language processing and information extraction tools. For example, for the input *"'When was Noam Chomsky born?"*, a question answer system should return something like "*December 7th, 1928*". This area represents a logical step beyond the standard information retrieval systems and in the recent years it has achieved a serie of important milestones, driven by the general project of semantic web. Watson, the system developed by IBM which defeated the best human competitors of Jeopardy! is the most visible example, but even search engines like Bing! and Google have started to incorporate this kind of algorithmics.

In this thesis we research the different problems subsumed under the concept of question answering and we review different solutions and models applied to resolve them, under the project of the implementation of two basic systems of question answering. The first implemented system is a closed (specific) domain model with structured data only for English. The second model is an open domain multilingual system which uses as corpora wikipedias in different languages. For the first model we oriented our develpmnet following the theoretical framework exposed in the paper [Popescu et al., 2003a] and we implemented solutions for a restricted set of questions. For the second model, we used a subset of problems of the competition CLEF '07 and we developed the system using as baseline the framework Qanus, adapting it to use the multilingual natural language processing tools of the library Freeling.

Keywords: Question Answering, Closed Domain, Open Domain, Multilingual, Freeling, Qanus, CLEF, Semantic Tractability

AGRADECIMIENTOS

A Susana, a Mariela y a Diego - primeros, definitivamente, por estar cuando hacía falta estar.

A Juan, que me abrió las puertas de su hogar y, siempre a su manera, me apoyó.

A mis sobrinos, Ema, Lisandro y Antonio, que renombraron la alegría.

A Diana, que sin saberlo fue un ejemplo y hoy ya no está, pero sé que se pondría contenta.

A mi seide, que me regaló mi primera compu.

A mi bobe, que se le pasaban los churrasquitos y me marcó a fuego el gusto.

A mi primo Luciano, descubrimiento vital esencial y tardío; por lo cotidiano familiar.

A mi primo Matías, por su buena onda y su apoyo en mis baches de pragmatismo.

A Laila y Mariano, a mis otros tíos Alfredo, Cati y Guille, mis otras primas, Tatiana y Luna, y mis otros abuelos, oma y ota: por todos los momentos compartidos y el afecto que me brindaron desde pequeño.

A mis amigos de siempre: Loi, Ale, Uri y Lacho, que perduran en el tiempo, como imaginé la amistad.

A mis otros amigos (Mariano y Joaquín, bah), por las barreras que rompimos juntos.

A mis compañeros de diferentes momentos de la facu, con los que compartí trabajos, tardes de estudio o simplemente buenos momentos: Hernán, Juan Manuel, Adrián, Curtu, Brian, Iván, Luciano, Andrés, Leo, Facu, Doc, Manu, Fede, Martín, Felipe, Juan, Carolina, Nico, Alejandro, Martín, Tomás y seguro me olvido de varios nombres por la distancia.

A los docentes que me inspiraron y encendieron mi curiosidad intelectual durante la carrera.

A mi director de tesis, José Castaño, por el aguante en este proyecto.

A las demás personas que me acompañaron durante estos 10 años y para las que el protocolo de agradecimientos no tiene un lugar.

¡Gracias a todos!

A la política universitaria del kirchnerismo, que me permitió terminar la carrera.

Y a mi madre, que hizo otro tanto.

Índice general

1.. Introducción 1
1.1. Qué es question answering 1
1.2. Proyecto de la tesis 2
1.3. Estructura de la tesis 4

2.. Marco teórico 7
2.1. Terminología 7
2.2. Information Retrieval 9
2.2.1. Métricas 11
2.2.2. Índice invertido de búsqueda 11
2.3. Procesamiento de Lenguaje Natural 12
2.3.1. Part-of-speech (POS) tagging 12
2.3.2. Reconocimiento de Entidades Nombradas (NER) 16
2.3.3. Clasificación de Preguntas (QC) 19
2.3.4. Otras tareas 20

3.. Estado de Arte 23
3.1. Introducción general 23
3.1.1. Historia y Competencias 24
3.1.2. Métricas de evaluación 25
3.2. Literatura y sistemas 29
3.2.1. Enfoques sobre open domain 29
3.2.2. QA como interfaz a una base de datos 32
3.2.3. IBM-Watson 41

4.. Implementación de datos estructurados 49
4.1. Base de datos 49
4.2. Implementación 52
4.2.1. Código 52
4.2.2. Ejemplos 63
4.2.3. Corridas 71
4.2.4. Conclusiones, limitaciones y trabajo futuro 74

5.. Implementación sobre Qanus 77
5.1. Ejercicio de CLEF 77
5.1.1. Tareas 78
5.1.2. Análisis de las preguntas 80
5.2. Sistema 82
5.2.1. Implementación baseline 85
5.2.1.1. Base de conocimiento 85
5.2.1.2. Anotado de la pregunta 86
5.2.1.3. Generación de Respuestas 87
5.2.2. Modificaciones realizadas 93

5.2.2.1. Inferencia del tópico del grupo de preguntas 93
5.2.2.2. Generación de queries 93
5.2.2.3. Generación de respuestas 94
5.2.3. Experimentación 95
5.2.4. Conclusiones, limitaciones y trabajo futuro 104

Apéndices 107

A.. Herramientas 109
A.1. Herramientas de Stanford 109
A.2. Freeling 111
A.3. Apache Lucene 113

Bibliografía 115

1. INTRODUCCIÓN

1.1. Qué es question answering

Question answering es un área de ciencias de la computación que busca generar respuestas concretas a preguntas expresadas en algún lenguaje natural. Es un área compleja que combina herramientas de búsqueda y recuperación de la información (*information retrieval*), de procesamiento del lenguaje natural (*PLN*) y de extracción de información (*information extraction*). Por poner un ejemplo: para el input *"¿Cuándo nació Noam Chomsky?"* un sistema de question answering podría devolver "*7 de diciembre de 1928*".

Podríamos pensar esta área como un paso lógico posterior, o un refinamiento, de los sistemas de recuperación de información usuales (por ejemplo, los buscadores web). En estos sistemas, el input no es una pregunta en lenguaje natural sino una serie de palabras preseleccionadas por el usuario para el motor de búsquedas, y el output no es una respuesta concreta sino una serie de documentos relevantes según el sistema, que luego el usuario deberá evaluar y revisar por su cuenta para encontrar la información que quiere.

Question answering logró en los últimos años una serie de hitos impulsados por el proyecto general de la web semántica. Watson, el sistema desarrollado por IBM que derrotó a los mejores competidores de Jeropardy! en tiempo real[1] es el ejemplo más visible, pero incluso buscadores como Bing! y Google comienzan a incorporar este tipo de algoritmia.

Los sistemas de question answering suelen ser sistemas complejos que abordan distintas problemáticas: por un lado deben definir y optimizar la base de conocimiento para el dominio dado, por otro deben realizar un análisis de las preguntas en lenguaje natural a fin de volverlas tratables computacionalmente y, finalmente, deben poder buscar -o generar- y decidir la mejor respuesta para la pregunta ingresada, si es que esa respuesta existe. Ejemplos de sistemas de question answering abundan: por un lado, los ya mencionados gigantes comerciales Google[2] y Bing![3] incorporan día a día mayor soporte para question answering, también Siri[4] y Google Now[5], los asistentes personales para iPhone y Android respectivamente incorporan un motor de question answering y, también, el ya mencionado IBM-Watson[6]. Por otro lado, existen desarrollos no comerciales ni tan conocidos de envergadura como, por ejemplo, el proyecto de la universidad Carnegie Mellon, Ephyra[7][Pires, 2012] (ahora discontinuado), el sistema basado en web START[8], del MIT, WolframAlpha[9], LASSO y Falcon [Lampert, 2004][Moldovan et al., 2000], YagoQA [Adolphs et al., 2011], [Chung et al., 2004], Qanus [Jun-Ping Ng, 2010] entre una infinidad de otros trabajos destacables.

Algunos de los subproblemas de question answering tienen nombre propio en la litera-

[1] En esta url está disponible el programa en el que el sistema vence a sus competidores humanos: `http://www.youtube.com/watch?v=WFR3lOm_xhE`

[2] `http://www.google.com`

[3] `http://www.bing.com`

[4] `http://www.apple.com/ios/siri/`

[5] `https://www.google.com/search/about/learn-more/now/`

[6] `http://www.ibm.com/watson/`

[7] `http://www.ephyra.info/`

[8] `http://start.csail.mit.edu/`

[9] `www.wolframalpha.com/`

tura y son una sub-área específica. Por ejemplo, dependiendo de la amplitud de la base de conocimiento, un sistema es de dominio cerrado (*closed domain*), si la base es acotada a un dominio de la realidad específico; por el contrario se llama de dominio abierto (*open domain*), si no lo es, es decir, si se espera que sepa responder preguntas de cualquier dominio. Por su parte, dominios de conocimiento más pequeños, en general, requieren y permiten un modelado más exhaustivo de los datos y un análisis más estructurado, mientras que dominios de conocimiento más abiertos suelen tener un enfoque apoyado más fuertemente en el análisis lingüístico cuantitativo.

Otra distinción usual contempla el tipo de datos de la base de conocimiento: puede ser estructurado, como en una base de datos relacional, semi-estructurado, como los documentos XML, o también sin estructura, como el texto plano. Cada tipo de datos tiene su enfoque: los datos estructurados definen una ontología acotada que limita qué cosas se pueden preguntar y, en consecuencia, qué cosas se pueden responder: el problema en este caso consiste en traducir la pregunta formulada en un lenguaje humano a una consulta válida definida en el modelo de datos. Por otro lado, si los datos no tienen estructura, no es posible definir una ontología rígida y se hace necesario otro tipo de enfoque más difuso y basado en análisis lingüísticos del corpus de datos mismo contra la pregunta. No siempre, pero en general una base de conocimiento closed domain está asociada a un tipo de datos estructurado o semi-estructurado, mientras que las bases open domain suelen ser no estructuradas.

Otro tipo de clasificación se centra en el tipo de preguntas que los sistemas saben responder. Los tipos más conocidos son las factoids -o fácticas-, que refieren a hechos fácticos (¿Cuándo ocurrió ...? ¿Cuántos hijos tiene...? ¿En dónde vivía Y en el año ...?), las listas (¿Qué libros escribió Nietzsche entre 1890 y 1900?) y las definiciones (¿Qué es un huracán?). Otros tipos usuales son las preguntas por un modo (¿Cómo...?), por una razón (¿Por qué...?) y, en general, pueden agregarse subclasificaciones como dependencias temporales explícitas o dependencias semánticas con otras preguntas.

En 3 Estado de Arte veremos con detalle los enfoques arquitecturales usuales del área y los componentes típicos de cada módulo.

1.2. Proyecto de la tesis

En esta tesis investigamos los distintos problemas que se subsumen bajo el concepto de question answering y reseñamos diferentes soluciones y modelos aplicados para resolverlos, bajo el proyecto de la implementación de dos sistemas básicos de question answering: uno de dominio cerrado, específico, y datos estructurados en inglés, por un lado, y otro sistema multilingüe, de dominio abierto y que utiliza como corpora las wikipedias de diferentes idiomas, por el otro. Para el primer modelo orientamos nuestro desarrollo de acuerdo al modelo teórico del paper [Popescu et al., 2003a] e implementamos soluciones para un conjunto restringido de preguntas. Para el segundo modelo utilizamos un subconjunto de los problemas de la competencia CLEF '07 y desarrollamos el sistema utilizando como baseline el framework Qanus[10], adaptándolo para utilizar herramientas de procesamiento de lenguajes multilingües de la librería Freeling[11].

Mientras el enfoque estructurado es complicado de evaluar debido a su dominio acotado

[10] `http://www.qanus.com/`
[11] `http://nlp.lsi.upc.edu/freeling/`

y específico, es más sencillo realizar experimentos sobre las herramientas al utilizarlas en el modelo no estructurado, gracias a las distintas competencias cuya razón de ser es, justamente, la creación de criterios de evaluación al nivel de la comunidad de investigadores del área. El aporte concreto de esta tesis son dos modelos de software con diferentes soportes para idiomas: un modelo básico de question answering como interfaz a la base de datos en inglés, montado sobre una base de datos con información de países y ciudades de ejemplo, y un modelo no estructurado de dominio abierto, completamente multilingüe, que resuelve algunas subtareas de la competencia Clef '07.

Con un poco más de detalle, la implementación del modelo de dominio cerrado está basada en los trabajos de Ana María Popescu et. al. [Popescu et al., 2003a] y [Popescu et al., 2003b] en donde se define la noción de pregunta *semánticamente tratable* en el contexto de una base de datos concreta. Proponen allí, así mismo, un modelo teórico para identificar este tipo de preguntas y una forma de transformarlas en consultas de SQL, así como también un sistema concreto que implementa el modelo. Popescu argumenta que una interfaz a una base de datos en lenguaje natural puede no identificar una pregunta, pero que jamás debe mal interpretarla activamente, es decir, interpretar algo distinto a lo preguntado y dar una respuesta que no se condice con la necesidad de información del usuario. Una mala interpretación de una pregunta reducirá la confianza del usuario en el sistema, volviéndolo inusable. En cambio, identificando una pregunta como intratable, se puede disparar un proceso de reformulación o especificación asistida de la pregunta, lo cual no es tan costoso en términos de confianza en el sistema. La idea principal que guía a sus trabajos es proponer una clase de preguntas específica que sea 1) suficientemente sencilla para ser interpretada por un modelo computacional y, a la vez, suficientemente abarcadora de las preguntas generalmente hechas por humanos. La intuición detrás de este enfoque es que, mientras, en el caso general, una pregunta puede ser compleja, ambigua y difícil de comprender (incluso por un humano), también hay preguntas simples, unívocas y con una interpretación sencilla incluso para una máquina (por ejemplo: "¿Qué restaurantes de comida china hay en Belgrano?"). La *tratabilidad semántica*, cualidad de una pregunta para una base de datos dada, define esta clase de preguntas simples. Nosotros implementamos una versión limitada de este modelo teórico en 4 Implementación de datos estructurados sobre un la base de datos World[12] (ver 4.1 Base de datos), provista por MySQL, que consiste en tres tablas (countries, cities y languages), y posee cierta información básica de geografía internacional.

Por su parte, para desarrollar y evaluar mecanismos de dominio abierto resolvimos algunos ejercicios de la competencia de question answering organizada por CLEF en 2007. CLEF (de *Cross-Language Evaluation Forum*) es una organización que busca fomentar la investigación en sistemas de information retrieval cross-language. En particular, una vez por año CLEF lanza una competencia de Question Answering multilingüe, con diferentes corpora y diferentes tipos de ejercicios. Estas competencia permiten obtener un patrón estándar de comparación entre distintos desarrollos y una idea general del estado de arte alcanzado en cada área. Por ejemplo, la competencia ya finalizada del año 2013, QA4MRE@CLEF2013, (Question Answering for Machine Reading Evaluation) se enfoca principalmente en Machine Reading, tarea que incluye un grado de razonamiento elevado para la computadora[13]. Existen distintas conferencias de evaluación de sistemas QA o

[12] Ver `http://dev.mysql.com/doc/world-setup/en/index.html` y `http://dev.mysql.com/doc/index-other.html`

[13] `http://celct.fbk.eu/QA4MRE/`

de subtareas asociadas (por ejemplo TREC - Text Retrieval Conference [14]-, TAC - Text Analysis Conference [15]) - y, a su vez, estas distintas competencias ofrecen distintos llamados a competencias. Elegimos resolver una tarea de la competencia Clef '07 por varias razones (Ver [Clef, 2007a] y [Clef, 2007b] para un detalle exhaustivo de la conferencia en cuestión). La razón principal fue la pertinencia de la tarea a evaluar al scope de esta tesis. Muchas competencias exigen un grado de complejidad que excede por mucho lo que puede alcanzarse en el tiempo estimado de una tesis de licenciatura y, si bien tuvimos que recortar ciertos aspectos de las tareas a fin de implementar este proyecto en tiempo y forma, estos aspectos fueron pocos. Otra razón fue la disponibilidad y el atractivo de la base de conocimiento para estos ejercicios: utilizan imágenes de wikipedia. La competencia del '07 ofrece dos tipos de tareas:

- Mono-lingual: donde el idioma de la pregunta y el idioma de la fuente de información son el mismo.
- Cross-lingual: donde el idioma de la pregunta y el idioma de la fuente de información difieren.

Las tareas consideran los siguientes idiomas: inglés, búlgaro, alemán, español, italiano, francés, holandés, rumano y portugués. Por su parte, algunos problemas utilizan fuentes de datos privados de la competencia y otros utilizan como fuente las distintas wikipedias. De los problemas que utilizan wikipedia, implementamos un sistema que responde las preguntas en español, mono-idioma, es decir, ejercicios con preguntas formuladas en español que se responden en base a la wikipedia en español e hicimos lo mismo para el portugués. A su vez, implementamos esta misma solución para el inglés, dado que estaban disponibles las preguntas y señalados los links a las imágenes de wikipedia en inglés, pero no fue posible evaluar sus resultados debido a que las respuestas esperadas no estaban disponibles online y no obtuvimos respuesta de los organizadores de la competencia. El uso estructural de la librería freeling permite la implementación de soluciones para otros idiomas mediante el set-up del corpus en el idioma y una pequeña configuración. Los ejercicios elegidos constan de 200 preguntas agrupadas. Los grupos de preguntas refieren a un tema, inferible a partir de la primer pregunta. Por ejemplo, el primer grupo de preguntas es:

- ¿En qué colegio estudia Harry Potter?
- ¿Cuál es el lema del colegio?
- ¿En qué casas está dividido?
- ¿Quién es el director del colegio?

Es decir, para cada grupo se debe inferir el "tema" en la primer pregunta para arrastrarlo a la hora de responder las siguientes. Más allá de esta particularidad, las preguntas son preguntas simples. Más adelante haremos un análisis de las mismas con más detalle.

1.3. Estructura de la tesis

La tesis se articula de la siguiente manera: en la Introducción (1), que estamos concluyendo en estos párrafos, realizamos en primer lugar una introducción mínima al área del

[14] `http://trec.nist.gov/`
[15] `http://www.nist.gov/tac/`

question answering (1.1), en segundo lugar mencionamos los alcances de esta tesis (1.2), y en tercer lugar, aquí, damos una estructura general de la tesis (1.3).

En el siguiente capítulo (2) recorreremos los conceptos generales de algunas áreas en las que se apoya question answering, repasando primero la terminología básica (2.1) para pasar luego a comentar estructuras típicas de recuperación de la información (2.2) y, finalmente, de procesamiento de lenguajes (2.3) aplicado a problemas de question aswering.

En el capítulo III (3) pasamos revista general del estado de arte del área. Primero realizamos una introducción general (3.1), considerando la historia de la disciplina, las competencias en las que la investigación se nucleó (3.1.1) y las métricas utilizadas por estas competencias para evaluar a los competidores (3.1.2), luego hacemos un recorrido de diferentes enfoques, considerando el estado de arte académico para dominio abierto (3.2.1) y dominio cerrado (3.2.2) y finalmente, el comercial, reseñando el funcionamiento de Watson de IBM (3.2.3).

En los siguientes capítulos presentamos los modelos implementados:

En el IV (4) comentamos el modelo de dominio cerrado para la base de datos World, presentando primero la base de datos (4.1) y luego la implementación de nuestro sistema (4.2), pasando revista de su código (4.2.1), dando ejemplos (4.2.2) y presentando conclusiones, limitaciones y trabajo futuro(4.2.4).

En el V (5) presentamos el modelo de dominio abierto para las wikipedias de diferentes idiomas, presentando en primer lugar los problemas seleccionados de la competencia Clef '07 (5.1), en segundo lugar la implementación de nuestro sistema (5.2), presentando el sistema baseline de basado en Qanus (5.2.1), las adaptaciones multilingües y demás mejoras realizadas (5.2.2), las diferentes corridas que realizamos para evaluarlo (5.2.3) y, finalmente, las conclusiones, los límites y potenciales mejoras del sistema (5.2.4).

2. MARCO TEÓRICO

En este capítulo vamos a recorrer algunos términos básicos y problemas de procesamiento de lenguaje e information retrieval que deben conocerse para comprender mejor lo que discutiremos sobre question answering en los siguientes capítulos. En la primer sección (2.1) vamos a introducir terminología básica como *token*, *n-grama*, *ontología* y *feature*. En la segunda (2.2) comentaremos qué es information retrieval, cuales son las métricas de evaluación del área y qué es un índice invertido de búsqueda. Finalmente (2.3), describiremos los principales problemas propios de procesamiento de lenguajes relacionados con QA (POS tagging o etiquetado gramatical, NE tagging o reconocimiento de entidades y clasificación de preguntas) y mencionamos algunas otras herramientas y conceptos de uso frecuente.

Para referencias en general sobre temas de recuperación de la información puede consultarse el texto de Manning[Manning et al., 2008], mientras que para temas de procesamiento de lenguajes en general puede consultarse el de Jurafsky[Jurafsky, 2000]. Por otro lado, para temas específicos de POS tagging puede verse la web de Van Guilder [1] y los papers utilizados [Toutanova et al., 2003] [Toutanova and Manning, 2000], sobre Question Classification los papers [Li and Roth, 2002] [Hermjakob, 2001] [Manning and Klein, 2003] [Lehnert, 1986] y para reconocimiento de entidades [Bach and Badaskar, 2007] [Nadeau and Sekine, 2007] [Finkel et al., 2005].

2.1. Terminología

Tokens

Un token es una palabra o un signo de puntuación o, más en general, una cadena de caracteres con algún significado en el contexto de un texto. Por ejemplo, el texto "El sol brilla." tiene cuatro tokens: {'El', 'sol', 'brilla', '.' }. Las herramientas que generan una lista de tokens a partir de un texto se llaman *tokenizers* y suelen permitir distintos tratamiento de ciertas palabras o signos de puntuación. (Por ejemplo: pasar todo a minúsculas, eliminar signos de puntuación, considerarlos aparte, etc). Las bibliotecas que proveen tokenizers suelen proveer también herramientas para dividir un texto en oraciones (*Sentence Segmentation*). Estos dos procesos suelen ser el primer paso de todos los análisis de procesamiento de lenguaje natural.

N-Gramas

Un n-grama es una subsecuencia continua de n tokens de un string.
Por ejemplo, la oración: "Hoy está nublado", tiene los siguientes n-gramas:

Tres unigramas	:	{ "Hoy", "está", "nublado" }
Dos bigramas	:	{ "Hoy está", "está nublado" }
Un sólo trigrama	:	{ "Hoy está nublado" }

[1] `http://ccl.pku.edu.cn/doubtfire/NLP/Lexical_Analysis/Word_Segmentation_Tagging/POS_Tagging_Overview/POS%20Tagging%20Overview.htm`

Los n-gramas son útiles para recorrer un texto con distintos tamaños de ventana en busca de algún patrón conocido, por ejemplo: buscando entidades nombradas que no fueron reconocidas por otras herramientas. Cabe notarse que la definición general de n-grama es la de una secuencia continua de n items cualesquiera (por ejemplo, caracteres; y no solo de tokens como mencionamos aquí).

Ontología

El término 'ontología' es un término originalmente filosófico, que refiere al estudio de lo que hay, de lo que es, o, dicho de otro modo, a la definición y al estudio de los entes que existen en la realidad última y de sus cualidades esenciales y sus relaciones intrínsecas. Aplicado a ciencias informáticas, principalmente dentro áreas como inteligencia artificial y representación del conocimiento, esta noción original se sostiene, acotando la noción de realidad a uno o varios dominios de problemas. Más concretamente, la noción de ontología en informática refiere a la definición formal y exhaustiva de un conjunto de conceptos que representan entidades, tipos o clases de entidades, propiedades y relaciones entre estas entidades relevantes para el modelado de un dominio de problemas dado. Esta ontología formal define un vocabulario inicial fijo que determina el tipo entidades, atributos de entidades y relaciones entre ellas que existen en un dominio dado, permitiendo definir formalmente los problemas que se pueden plantear (y resolver) para ese dominio en esa ontología.

Algunos ejemplos de ontologías populares son dbPedia[2], Yago[3] y Freebase[4], bases de conocimiento estructuradas extraídas de diferentes fuentes de datos (entre ellas: wikipedia, wordnet) que ofrecen una interfaz de consultas basada en RDF[5], un estándar de modelado de meta-datos para recursos web.

Features

En machine learning[6] en general, un *feature* puede definirse como una propiedad medible de un fenómeno siendo observado. En las aplicaciones de procesamiento de lenguaje natural (que son un aplicaciones de machine learning), el dominio de "fenómenos observados" está restringido a los fenómenos lingüísticos y las propiedades medibles también. Un ejemplo de feature es: un booleano indicando "la palabra comienza con mayúscula". En general, los features toman valores numéricos o booleanos, aunque también es posible utilizar features más complejos (como strings, o listas). Los features se agrupan en vectores. Durante un proceso de análisis, cada entidad procesada tiene un vector de features que la representa de manera tratable. Por ejemplo, podemos asignarle a cada palabra de una oración el siguiente vector de tres features:

[2] `http://dbpedia.org/`

[3] `http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/`

[4] `http://www.freebase.com/`

[5] `https://www.w3.org/RDF/`

[6] Machine learning es una rama del estudio sobre inteligencia artificial que busca desarrollar técnicas que permitan a las computadoras generalizar comportamientos a partir de una fuente de información no estructurada que sirve como ejemplo para esta generalización, que le permite al programa inducir conocimiento.

v_1:*bool*	indica si la palabra comienza con mayúscula
v_2:*int*	indica la longitud de la palabra
v_3:*string*	representa la palabra en minúsculas

Encontrar features (o "características") discriminantes e independientes entre sí es una forma en la que se modelan correctamente muchos problemas de PLN, como la asignación de diferentes etiquetas a las palabras de una oración (por ejemplo: pos tags, ner tags), la clasificación de preguntas y la extracción de tópicos.

Los features aparecen en muchos contextos, por ejemplo: en 5 Implementación sobre Qanus nosotros construimos una serie de features a mano en la sección de answer retrieval, para subir la posición de la respuesta esperada entre la lista de pasajes rankeados. En las herramientas lingüísticas que veremos a continuación (2.3), los features son normalmente extraídos automáticamente de un corpus. En estos casos, el volumen de features suele ser relativamente alto (10.000, 100.000) y se incorporan patrones estadísticos complejos alejados de la simplicidad del vector de tres features de nuestro ejemplo.

A modo de ilustración, en la tabla 2.1, reproducimos los ejemplos que se pueden encontrar en
[Nadeau and Sekine, 2007] sobre dimensiones de features típicas para el proceso de reconocimiento de entidades. Veremos más sobre reconocimiento de entidades en breve (2.3.2), pero, rápidamente, lo que se busca es encontrar entidades nombradas como "Buenos Aires" o "Juan Pérez" en oraciones. En este proceso podrían usarse, por ejemplo, tres tipos de features basados en propiedades de las palabras mismas, su ocurrencia en listas o corpora conocidos y en propiedades de los documentos del corpus.

2.2. Information Retrieval

El problema conocido como recuperación de la información (o *information retrieval*) consiste, formalmente, en retornar *información relevante* para una *consulta* (*information need*) a partir de una *base de conocimiento*.[Manning et al., 2008]

Como la generalidad de esta definición sugiere, information retrieval es un área de trabajo amplia que abarca disciplinas variadas y problemas diversos (como por ejemplo: el almacenamiento físico de la información y su calidad, la representación de la información en modelos formales, el modelado semántico de las nociones de "similaridad" y "relevancia" entre las consultas de usuarios y los diferentes documentos de la base de información).

El caso de uso socialmente más relevante y conocido son los motores de búsqueda web: en los motores de búsqueda, la base de conocimiento son las páginas de la web, la consulta es la cadena de texto que ingresamos en el input del buscador, y la información relevante es una lista de documentos priorizados que, se espera, satisfará la necesidad de información del usuario. En estos sistemas, las consultas se interpretan como una serie de tokens seguidos (con la opción de agregar diferentes operadores lógicos, pero, en general, tomándolos como una serie concatenada por un OR inclusivo). El núcleo del sistema es un *índice de búsqueda* de los documentos de la base de conocimiento, que en principio podemos pensar como un diccionario indexado de palabras en documentos (más adelante veremos índices invertidos). El proceso de creación y mantenimiento de estos índices depende del tipo y del dinamismo de la base de conocimiento, y oscila entre un modesto setup a mano

Features a nivel palabra	
Features	Ejemplo
Case	Empieza con mayúscula La palabra está en mayúsculas Mezcla mayúsculas y minúsculas (por ejemplo: eBay, laTex)
Puntuación	Termina con punto, tiene un punto en el medio (por ejemplo: Sr., I.B.M.) Contiene un apóstrofe
Morfología	Prefijo, sufijo, stem Terminaciones comunes
Part-of-speech	nombre propio, sustantivo, verbo, etc
Funcional	Sólo letras, sólo símbolos, n-gramas Versión en mayúsculas / minúsculas Patrones (matches con expresiones regulares) Cantidad de tokens, cantidad de caracteres Terminaciones comunes
Features sobre listas conocidas	
Features	Ejemplo
Lista General	Diccionario general Stop words Sustantivos capitalizados (por ejemplo: Octubre, Lunes) Abreviaciones comunes (por ejemplo: Sr., PhD.)
Lista de Entidades	Organizaciones, gobiernos, aerolíneas, etc Nombre típico, apellido, celebridad Estrella, continente, país, calle
Lista de pistas	Palabras típicas de una organización (“asociados”) Títulos, prefijos de nombres Palabras típicas de lugares (“río”, “puerto”)
Features extraídos de documentos	
Features	Ejemplo
Ocurrencias múltiples	Entidades en el contexto Ocurrencias en mayúsculas o minúsculas Anáforas, correferencia
Posición en documento	Posición según oración, párrafo, documento
Meta datos Frecuencia en corpus	URI, mail headers, sección del xml imágenes asociadas Frecuencia de la palabra o la frase Co-ocurrencia Permanencia de una unidad de varias palabras

Tab. 2.1: Diferentes tipos de features

de una lista de documentos estáticos hasta los ejércitos de *spiders* de Google, indexando la web.

La consulta no es la necesidad de información del usuario sino lo que el usuario puede expresar de su necesidad dentro de la interfaz de consultas que el sistema de information retrieval sabe interpretar. Question answering puede pensarse como un sistema de recuperación de la información que busca presentar una interfaz de consultas más cercana a la representación lingüística usual de una pregunta concreta y que entiende como información relevante una respuesta concreta en lugar de una lista de documentos.

2.2.1. Métricas

Las métricas de rendimiento de los sistemas de information retrieval articulan de diferentes maneras los documentos totales devueltos por el sistema, la cantidad de documentos relevantes y la cantidad de documentos irrelevantes para una consulta dada. En la práctica existen diferentes maneras de adjudicar relevancia a un documento.

$$\text{Precisión} = \frac{|\{\text{documentos relevantes}\} \cap \{\text{documentos recuperados}\}|}{|\{\text{documentos recuperados}\}|} \tag{2.1}$$

$$\text{Recall} = \frac{|\{\text{documentos relevantes}\} \cap \{\text{documentos recuperados}\}|}{|\{\text{documentos relevantes}\}|} \tag{2.2}$$

Las dos métricas principales son la precisión y el recall. La precisión (2.1) mide cuántos de los documentos devueltos son relevantes, es decir, la proporción de documentos relevantes en el total de documentos devueltos para la consulta. La precisión puede evaluarse sobre el total de documentos devueltos o sobre un corte arbitrario, considerando solamente los primeros n resultados obtenidos. Por su parte, el recall(2.2) (traducido como *exhaustividad*), mide cuantos de los documentos relevantes disponibles en la base de conocimiento fueron efectivamente recuperados. En general, la precisión es la métrica privilegiada en casos de uso de sistemas de information retrieval. En QA, como veremos, las condiciones son diferentes.

Otras métricas conocidas son la familia de métricas F, que representan distintos trade off entre precisión y recall. La formula general (2.3) introduce el parámetro β que pondera recall sobre precisión. F_β pondera β veces más el recall que la precisión. Las más utilizadas son F_1 (o solo F), F_2 y $F_{0,5}$. F_2 (2.4) pondera 2 veces más el recall que la precisión, por lo que es muy utilizada para evaluar subtareas de information retrieval en question answering.

$$F_\beta = \frac{(1+\beta^2) \cdot (\text{Precision} \cdot \text{Recall})}{(\beta^2 \cdot \text{Precision} + \text{Recall})} \tag{2.3}$$

$$F_2 = \frac{5 \cdot (\text{Precision} \cdot \text{Recall})}{(4 \cdot \text{Precision} + \text{Recall})} \tag{2.4}$$

2.2.2. Índice invertido de búsqueda

Un índice invertido es la estructura de datos típica utilizada en problemas de information retrieval. Consiste en un mapeo de términos en documentos que los contienen. Es decir, para un término dado, un índice invertido devuelve una lista de los documentos cargados que lo contienen. Este tipo de estructuras invierte la relación normal en la cual a

partir de un documento se accede a la lista de términos que este documento contiene (de allí su nombre). Por ejemplo, para los textos:

T[0] = “qué es esto"
T[1] = “esto es un ejemplo"
T[2] = “qué gran ejemplo"

Un índice invertido contendría las siguientes entradas (dónde el número n es un puntero al texto T[n]):

“qué" : {0, 2}
“es" : {0, 1}
“esto" : {0, 1}
“un" : {1}
“ejemplo" : {1, 2}
“gran" : {2}

Además, un índice suele contener otra información relevante, como por ejemplo: la frecuencia de aparición de cada término en los respectivos documentos.

2.3. Procesamiento de Lenguaje Natural

El problema de question answering puede pensarse como un problema de information retrieval, pero en realidad excede este dominio: su objetivo no es devolver una lista de documentos para una serie de palabras, sino una respuesta puntual a una pregunta puntual. Para esto incorpora una dimensión semántica, esto es, toda una serie de problemas de procesamiento de lenguaje natural que se abordan con varias herramientas, que aportan distintos modelos de análisis lingüístico a nivel *oración*, tanto de la consulta del usuario, como de los documentos indexados. Por ejemplo: el pos-tagger es un analizador que genera la secuencia de etiquetas gramaticales de la oración, mientras que el reconocimiento de entidades nombradas (NER) identifica entidades como “Buenos Aires”, “José Pérez”, etc- y las clasifica (como *lugar* y *persona*, respectivamente) . Los sistemas de QA utilizan distintas herramientas para modelar la consulta y para buscarla, con algún grado de comprensión semántica, dentro de su base de conocimiento. En lo que sigue de esta sección comentaremos algunas herramientas de procesamiento de lenguajes relevantes para question answering en general, y para nuestra tesis en particular.

2.3.1. Part-of-speech (POS) tagging

El POS-tagging o *etiquetado gramatical* consiste en asignar a los diferentes tokens una etiqueta con el rol o categoría gramatical que cumplen en su contexto de emisión (por lo general, una oración o un párrafo). El POS-tagging cumple un rol fundamental en áreas como el reconocimiento de voz, el procesamiento de lenguaje natural e information retrieval. El input de un pos tagger es una lista de tokens y un tagset (conjunto de etiquetas) específico, mientras que su output es un tag para cada uno de los tokens. Como ejemplo introductorio, consideremos la siguiente oración y un etiquetado gramatical posible:

“El hombre bajó la escalera.”

Token	Etiqueta Gramatical (POS-tag)
El	Determinante, artículo definido, masculino, singular
hombre	Nombre común masculino singular (sustantivo)
bajó	Verbo principal indicativo pasado tercera persona del singular (genero indefinido)
la	Determinante, artículo femenino singular
escalera	Nombre común femenino singular (sustantivo)
.	Punto final

Tab. 2.2: Ejemplo de resultado de un POS-tagger

Categoría General	Ejemplos
Determinante	aquel, este, mi, sus, nuestras
Pronombre	yo, tú, él, mí, nos, aquéllos, suyos
Preposición	a, ante, bajo, con, contra
Conjunción	y, aunque, pero, incluso
Interjección	ah, eh, ejem
Sustantivo	chicos, tesis, Pedro, cortapapeles
Verbo	cantamos, corrió, bailarán
Adjetivo	alegre, bonita, pésimos, desnuda
Adverbio	despacio, hábilmente, posteriormente

Tab. 2.3: Categorías gramaticales

El resultado de un POS-tagger para esta oración podría ser el que presentamos en la tabla 2.2.

La asignación de etiquetas no es un proceso trivial debido a la ambigüedad de roles posibles que tienen muchas palabras. Por ejemplo, la palabra 'ayuda' puede funcionar como sustantivo (en "La ayuda llegó a tiempo") o como verbo (en "Batman ayuda a los ciudadanos de Ciudad Gótica"). Un algoritmo de pos-tagging debe resolver estas ambigüedades seleccionando la categoría adecuada según el contexto. Más allá de estos casos, muchas palabras tienen una sola categoría posible (por ejemplo: 'ventana' es siempre un sustantivo, mientras que 'lentamente' es un adverbio y 'corrió' un verbo, etc).

Existen diferentes formas de clasificar gramaticalmente a las palabras. El esquema más general y reconocido de categorías gramaticales tal vez sea el de 9 clases que presentamos en la tabla 2.3

Mientras a algunas de estas categorías se les puede agregar más información, como *género* y *número* a los sustantivos, otras son categorías que no toleran modificaciones. Veremos esto con más detalle al hablar del tagset propuesto por el grupo EAGLES en breve.

Si bien en ciertos idiomas estas categorías no resultan del todo adecuadas o explicativas, dentro del scope de esta tesis podemos dejar de lado este problema. Estas clases de palabras -en general, para muchos idiomas- pueden dividirse en dos superclases conceptuales de acuerdo a su naturaleza: las clases cerradas y las abiertas. Las primeras constan de

una lista acotada y fija de palabras y está, por lo general, cristalizada. Esto es: no se incorporan, a esta altura de la historia del lenguaje, nuevas palabras a las clases gramaticales cerradas. Además, estas clases de palabras cumplen un rol funcional en la construcción de la oración y, como una nota más, suelen ser palabras cortas. Son clases cerradas, de las 9 recién enunciadas, los determinantes, los pronombres, las preposiciones, las conjunciones y las interjecciones. En contraposición, las clases abiertas son los sustantivos, los verbos, los adjetivos y los adverbios. Las clases abiertas incorporan, con naturalidad y frecuencia, nuevas palabras a su lista: se inventan nuevos objetos y en consecuencia nuevos sustantivos y nuevas acciones (nuevos verbos). Existen esquemas formales de conjugación de miembros para las clases abiertas, teniendo la mayoría sus miembros ciertas regularidades morfológicos que se repiten.

A partir de las clases de lingüística teórica y algunas otras características de la morfología de las palabras se construyen los *tagsets*. Estos son diferentes conjuntos de etiquetas que se utilizan de hecho en los algoritmos de tagging, siendo los más conocidos el tagset de 87 etiquetas usado en el Corpus Brown y algunos de sus derivados: el tagset de Penn Treebank, de 45 tags; el C5, de 61 tags, usado en el proyecto CLAWS y el C7, más grande, de 146 tags. El POS tagger de Stanford[Toutanova and Manning, 2000] (usado en esta tesis) utiliza el tagset de Penn Treebank, cuyas etiquetas y significados se listan en la Figura 2.1.

The Penn Treebank POS tagset.

1. CC	Coordinating conjunction	25. TO	*to*
2. CD	Cardinal number	26. UH	Interjection
3. DT	Determiner	27. VB	Verb, base form
4. EX	Existential *there*	28. VBD	Verb, past tense
5. FW	Foreign word	29. VBG	Verb, gerund/present participle
6. IN	Preposition/subordinating conjunction	30. VBN	Verb, past participle
7. JJ	Adjective	31. VBP	Verb, non-3rd ps. sing. present
8. JJR	Adjective, comparative	32. VBZ	Verb, 3rd ps. sing. present
9. JJS	Adjective, superlative	33. WDT	*wh*-determiner
10. LS	List item marker	34. WP	*wh*-pronoun
11. MD	Modal	35. WP$	Possessive *wh*-pronoun
12. NN	Noun, singular or mass	36. WRB	*wh*-adverb
13. NNS	Noun, plural	37. #	Pound sign
14. NNP	Proper noun, singular	38. $	Dollar sign
15. NNPS	Proper noun, plural	39. .	Sentence-final punctuation
16. PDT	Predeterminer	40. ,	Comma
17. POS	Possessive ending	41. :	Colon, semi-colon
18. PRP	Personal pronoun	42. (	Left bracket character
19. PP$	Possessive pronoun	43.)	Right bracket character
20. RB	Adverb	44. "	Straight double quote
21. RBR	Adverb, comparative	45. ‘	Left open single quote
22. RBS	Adverb, superlative	46. “	Left open double quote
23. RP	Particle	47. ’	Right close single quote
24. SYM	Symbol (mathematical or scientific)	48. ”	Right close double quote

Fig. 2.1: Tagset Penn Treebank

Por su parte, el POS-tagger español de Freeling, por su naturaleza multilingüe, utiliza el tagset propuesto por el grupo EAGLES (*Expert Advisory Group on Language Engineering Standards*)[7], una organización europea que fomenta la investigación de procesamiento

[7] `http://www.ilc.cnr.it/EAGLES96/home.html`

Nombres			
Pos.	Atributo	Valor	Código
1	Categoría	Nombre	N
2	Tipo	Común	C
		Propio	P
3	Género	Masculino	M
		Femenino	F
		Común	C
4	Número	Singular	S
		Plural	P
		Invariable	N
5-6	Clasificación Semántica	Persona	SP
		Lugar	G0
		Organización	O0
		Otros	V0
7	Grado	Aumentativo	A
		Diminutivo	D

Tab. 2.4: Valores para cada coordenada de tags para sustantivos

de lenguajes multilingüe, mientras que por una cuestión de compatibilidad, se preservan los tags de Penn Treebank para el inglés. Los tags de EAGLES tienen en consideración diferentes matices para contemplar las variaciones de diferentes idiomas. Sobre un conjunto inicial de 12 categorías -las 9 recién enunciadas más 'Signos de puntuación', 'Numerales' y 'Fechas y horas' define etiquetas mucho más especificas. En concreto, un tag consta de entre 6 y 7 atributos, cada uno de las cuales expresa una característica de la palabra dependiendo del valor especificado en el atributo anterior (salvo, claro, la primera posición, que especifica la clase general). Para algunas clases alguno de estos valores no tienen sentido, mientras que para algunas palabras algunos valores no están o no pueden definirse (en el caso de subespecificación de un atributo, esta se nota como un '0'). En las tablas 2.4 y 2.5 presentamos la especificación para el conjunto de tags relacionados con sustantivos y un ejemplo de su aplicación.

Existen varios enfoques algorítmicos al problema del POS tagging: desde los más primitivos basados en reglas escritas a mano pasando a los basados en HMMs (Hidden Markov Models), Maximum Entropy o Transformation Based Learning. Un detalle de estos métodos excede la introducción al problema que supone esta sección de la tesis. El POS tagger de Freeling está basado en el approach de HMMs y el de Stanford en el de Maximum Entropy, ambos modelos de machine learning. En A.2 Freeling y A.1 Herramientas de Stanford se encuentran algunos comentarios más técnicos de los algoritmos de POS tagging que utilizamos en este trabajo y vínculos a bibliografía pertinente y en la sección siguiente 2.3.2 Reconocimiento de Entidades Nombradas (NER) veremos una descripción más detallada de enfoques algorítmicos a ese problema que ilustrará, al menos de modo general, la estructura de la algoritmia basada en machine learning aplicada a procesamiento de lenguajes.

Forma	Lema	Etiqueta
chico	chico	NCMS000
chicas	chico	NCFP000
gatito	gato	NCMS00D
oyente	oyente	NCCS000
oyentes	oyente	NCCP000
cortapapeles	cortapapeles	NCMN000
tesis	tesis	NCFN000
Barcelona	barcelona	NP000G0
COI	coi	NP000O0
Pedro	pedro	NP000P0

Tab. 2.5: Ejemplos de tags y lemas para sustantivos

En la tabla 2.6 listamos ejemplos concretos de análisis de la oración de ejemplo del comienzo de esta sección ("El hombre bajó la escalera.") para mostrar un funcionamiento real de los algoritmos utilizados en esta tesis.

En el scope de este proyecto, utilizamos pos-tagging para filtrar clases de palabras inútiles y para seleccionar tres tipos: las qwords, los verbos y los sustantivos (Las qwords son los pronombres interrogativos: qué, quién, cómo, etc. y en inglés, who, when, where...).

2.3.2. Reconocimiento de Entidades Nombradas (NER)

El reconocimiento de entidades nombradas (NER, de Named Entity Recognition) es una subtarea de Information Extraction. Information Extraction es, brevemente, todo el dominio de problemas vinculado con la extracción de información estructurada a partir de datos no estructurados o semi estructurados. NER es, dentro de este dominio, el proceso de reconocer unidades de información (las entidades nombradas) tales como nombres de personas, organizaciones, lugares, expresiones numéricas como tiempo, fechas, dinero, porcentajes, etc. A veces se habla de NERC (Named Entity Recognition and Classification) para poner énfasis en la asignación de un tipo (por ejemplo: nombre de empresa) a la entidad nombrada reconocida.

Los primeros sistemas de NER eran algoritmos basados en reglas hardcodeadas, mientras que los modernos incorporan técnicas de machine learning y son, en general, algoritmos basados en features.

El primer sistema data de 1991 y constaba de reglas escritas a mano y heurísticas simples. Recién en 1996, con el estímulo de la MUC-6 (una conferencia reconocida en el área que dedicó una edición a NER), el área comenzó a acelerar su crecimiento.

Muchos trabajos sobre NER están basados sólo en inglés, pero también existen trabajos para otros idiomas y, más en general, que buscan la independencia del idioma (o también ser multi-idioma). En la CONLL-2003 (otra conferencia reconocida del área) se trabaja fuertemente el problema NER para el alemán, mientras que en la CONLL-2003 se estudia el español y el holandés y, en general, el estado de arte tiene avances, más o menos prometedores, para una gran variedad de idiomas.

Freeling (ES)		
Forma	Etiqueta	Descripción
El	DA0MS0	Determinante, Artículo, Masculino, Singular
hombre	NCMS000	Nombre, Común, Masculino, Singular
bajó	VMIS3S0	Verbo, Principal, Indicativo, Pasado, Tercera Persona, Singular
la	DA0FS0	Determinante, Artículo, Femenino, Singular
escalera	NCFS000	Nombre, Común, Femenino, Singular
.	Fp	Punto final

Freeling (EN)		
Forma	Etiqueta	Descripción
The	DT	Determiner
man	NN	Noun, singular or mass
came	VBD	Verb, past tense
down	RP	Particle
the	DT	Determiner
stairs	NNS	Noun, plural
.	Fp	Sentence final punctuation

Stanford (EN)		
Forma	Etiqueta	Descripción
The	DT	Determiner
man	NN	Noun, singular or mass
came	VBD	Verb, past tense
down	RP	Particle
the	DT	Determiner
stairs.	NN	Noun, plural

Tab. 2.6: Ejemplos de POS tagging

El problema del reconocimiento de entidades nombradas está acotado a lo que el filósofo del lenguaje Saúl Kripke llamó "designador rígido", dejando afuera las descripciones definidas. Por ejemplo, podemos referirnos a Saúl Kripke como "Saúl Kripke" o como "el filósofo de lenguaje que acuñó el concepto de designador rígido". El primer ejemplo es un designador rígido y un NER debería detectarlo, mientras que el segundo es una descripción y por lo tanto queda afuera de esta subtarea. Notar que ambos denotan unívocamente a un individuo. Para identificar a la compañía automotriz creada por Henry Ford, dos designadores rígidos son "Ford" y "Ford Company", etc. Más en general, los designadores rígidos incluyen nombres propios tanto como clases naturales (por ejemplo, especies de biología y nombres de sustancias, etc). Además, se incorpora al problema la detección de expresiones temporales ("26 de Agosto de 2013") y ciertas expresiones numéricas como dinero ("U$D 250") y otros tipos de unidades ("20 %", "10,25", etc). En un principio se buscaba detectar nombres propios en general, pero luego se incorporó la clasificación como un paso de esta subtarea. Las clases más utilizadas, por una cuestión completamente pragmática, son: persona, organización y lugar (location). Estas tres clases se conocen con el nombre de ENAMEX. A su vez, existen trabajos que subclasifican estas tres clases, dando como output, para un lugar (location), un subtipo como "ciudad", "estado" o "país" y para personas alguna definición más específica ("político", "farandulero", "deportista", etc). Algunos trabajos incluyen otras clases como "varios", para aquellos que no caen con un grado alto de confiabilidad en ninguna categoría, y también categorías específicas para los valores numéricos (date, time, money, percent, etc). Las categorías estándar pueden, sin embargo, adaptarse a las clases de entidades nombradas de un dominio de problemas puntual, pero por lo general este enfoque requiere del entrenamiento de módulos de machine learning, lo cual suele requerir una serie de inputs de los que no siempre se dispone (principalmente, un corpus de datos suficientemente grande para entrenar el sistema y la disponibilidad temporal para configurarlo).

Además de los ya mencionados sistemas de reconocimiento de entidades nombradas basados en reglas escritas a mano del comienzo de las investigaciones en el área, existen los basados en machine learning, en los cuales vamos a detenernos brevemente en esta sección. Los algoritmos basados en machine learning son entrenados sobre un corpus de datos con ejemplos positivos y negativos de entidades nombradas, a partir de los cuales infieren sus propias reglas de reconocimiento basadas en features. Hay tres tipos de enfoques al problema basados en machine learning: aprendizaje supervisado, aprendizaje semi supervisado y aprendizaje no supervisado. El aprendizaje supervisado es la técnica actualmente más utilizada para resolver el problema de reconocimiento de entidades. Estos métodos incluyen Hidden Markov Models (HMM), Decision Trees, Maximum Entropy Models (ME), Support Vector Machines (SVM) y Conditional Random Fields (CRF). Estos métodos son variantes de un modelo único de aprendizaje supervisado que consiste en leer un gran corpus de datos anotados, crear features de identificación y clasificación de entidades a partir de estos datos (generar un *modelo entrenado*) y finalmente identificar y clasificar un input nuevo en base a este modelo.

En cuanto al aprendizaje semi supervisado, la principal técnica aplicada a NER es conocida como "bootstraping" e involucra un grado de supervisión bajo, como por ejemplo, configurar un conjunto inicial de semillas (seeds) para el algoritmo de aprendizaje. Más genéricamente, un algoritmo de aprendizaje semi automático podría buscar a partir de los ejemplos iniciales (dados manualmente) otras entidades que cumplan el mismo rol léxico en contextos similares, para luego iterar sobre el conjunto ampliado. Otro enfoque

consiste en aplicar una serie de reglas simples basadas en patrones (por ejemplo: "New York" es una entidad de tipo location; si empieza con "Sr." es una entidad de tipo person") y luego identificar contextos de uso común sobre un corpus para generar reglas basadas en contextos. Un contexto puede incluir desde el rol semántico de las palabras en cuestión hasta ciertos patrones (como por ejemplo "empezar con 'Sr.'").

Finalmente, existen algoritmos de reconocimiento de entidades basados en aprendizaje no supervisado. El enfoque típico es el clustering, por ejemplo: agrupar entidades nombradas de diferentes clusters basados en similaridad de contexto. La forma general consiste en aplicar diferentes recursos léxicos (por ejemplo, Wordnet) sobre patrones léxicos y estadísticas tomadas de un gran corpus no anotado.

En nuestro trabajo utilizamos los NER taggers de Stanford[Finkel et al., 2005] y de Freeling, el primero implementando un algoritmo CRF, es decir, de aprendizaje supervisado (ver A.1 Herramientas de Stanford) mientras que el segundo es un algoritmo trivial basado en Autómatas Finitos (ver A.2 Freeling). Para descripciones y referencias de diferentes implementaciones de algoritmos de NERC ver [Nadeau and Sekine, 2007].

2.3.3. Clasificación de Preguntas (QC)

Question Classification es la tarea de categorizar preguntas en diferentes clases semánticas que impongan restricciones significativas a las respuestas potenciales para utilizarlas en fases posteriores del proceso de QA. La clasificación de preguntas es una subclase del problema de la clasificación. Un clasificador es una herramienta que asigna a un elemento una de k clases. La clasificación es un área bastante fecunda de PLN y, más en general, de machine learning. Los clasificadores de preguntas son herramientas que clasifican preguntas según su tipo de respuesta esperada. Por ejemplo: "¿Quién descubrió América?" espera, más allá del nombre concreto, *un nombre de persona*; "¿Cuándo se descubrió América?" espera *una fecha* (o, más en general, *un tiempo*), "¿Dónde se descubrió América?" espera, como respuesta, *un lugar*, etc. Este es un eje de clasificación conocido como tipo de respuesta esperado, aunque existen otros. Notar que el último ejemplo, en concreto confuso (¿tiene sentido la pregunta?), no lo es a nivel estructural.

El rol del módulo de QC en un sistema de QA es doble: Por un lado, impone restricciones a la respuesta final, permitiendo filtrar y verificar respuestas candidatas. Por otra lado, provee información para estructurar el flujo de código de los procesos subsiguientes, permitiendo implementar estrategias puntuales para cada tipo de respuesta esperada. Por ejemplo, para la pregunta "¿Quién fue el primer presidente constitucional de Argentina?" resulta de gran utilidad, a la hora de evaluar pasajes, saber que la respuesta esperada debe ser una *persona*: de este modo se puede evitar el procesamiento lingüístico de un gran dominio de pasajes no relevantes, por ejemplo, pasajes que no contienen ninguna entidad nombrada de tipo "persona". Estas mismas razones justifican la deseabilidad de la especificidad: saber que la respuesta final debe ser un *presidente* o un *político* es más informativo y útil para el resto del proceso que solo saber que es una *persona*.

Debido a la complejidad de análisis lingüístico intrínseca en la clasificación específica, los sistemas de QC más básicos adoptan un esquema de clases acotado y basado en reglas simples. Típicamente, las clases son: *Persona*, *Lugar*, *Organización*, *Fecha*, *Cantidad*, *Duración* y *Medida*, mientras las reglas de clasificación son parecidas a las siguientes:

- Si la pregunta empieza con *Quién* o *Quiénes*, entonces el tipo es *Persona*

- Si la pregunta empieza con *Dónde*, entonces el tipo es *Lugar*
- Si la pregunta empieza con *Cuándo*, entonces el tipo es *Fecha*
- Si la pregunta empieza con *Qué*, entonces determinar el tipo de acuerdo al sustantivo principal de la pregunta (utilizando pos-tagging)
- ...

Como nota al pie, este esquema de clasificación semántico es uno entre otros. Por ejemplo, [Lehnert, 1986] propuso un esquema *conceptual* de 13 clases en las que incluye, por ejemplo: antecedentes y consecuencias causales, habilitación, verificación, disyunción, etc.

Estas reglas básicas -triviales, si se quiere- cubren gran parte de las preguntas con una eficacia alta. En A.1 Herramientas de Stanford presentamos un enfoque más complejo que permite una clasificación más granular, basado en machine learning. El uso de algoritmos de machine learning tiene una serie de ventajas interesantes a la hora de definir un sistema de clases complejo. Como mencionamos al hablar de los otros taggers, la definición de reglas manuales es una tarea tediosa y con poca capacidad de adaptarse o modificarse, mientras que la definición de un set de features adecuado para el aprendizaje programático, en cambio, permite la fácil incorporación de nuevas clases y/o la incorporación de nuevas mejoras descubiertas. Por otro lado, un esquema de clases semánticamente rico -más granular que el modelo básico recién enunciado- requiere la consideración de una gran cantidad de dimensiones lingüísticas, tanto sintácticas como semánticas, lo que hace que la definición manual de reglas una tarea potencialmente imposible en términos de costo de tiempo humano.

Finalmente, cabe mencionar que no existen clasificadores de preguntas para el español y que a la hora de abordar este problema en nuestra implementación debimos apelar a mecanismos ad-hoc.

2.3.4. Otras tareas

Otras tareas de procesamiento de lenguajes que son parte de la 'caja de herramientas' con las que podemos encarar problemas que requieran análisis lingüísticos son: detección de idiomas, resolución de correferencias,[8] generación de árboles sintácticos y de dependencias, codificación fonética, anotaciones basadas en recursos léxicos (como Wordnet), desambiguación de sentido de palabras (Word sense disambiguation), extracción de relaciones [9] y traducción automática (o machine translation).

Finalmente, la lematización y el stemming son dos tareas de uso frecuente también. La lematización consiste en agrupar diferentes formas de una palabra bajo un ítem único. Por ejemplo, el verbo en infinitivo *caminar* puede aparecer bajo diferentes conjugaciones y formas: *caminó, caminando, caminando, caminamos*, etc. Un lematizador tomaría como input alguna de estas formas y devolvería *caminar*. La lematización es un proceso similar al stemming, con la diferencia de que el stemming opera sobre una sola palabra, sin información del contexto de ocurrencia (las palabras que la rodean en el contexto de emisión) y, en

[8] Por ejemplo: en "Nietzsche murió en 1900. Antes de morir, él estuvo aquejado de una enfermedad mental degenerativa", *él* refiere a *Nietzsche*.

[9] Por ejemplo, para "Nietzsche murió en 1900." extraer la relación *morir(Nietzsche, 1900)*.

consecuencia, pierde información útil (como el pos tag, para el que se requiere el contexto), siendo algoritmos más simples y eficientes, pero menos precisos que la lematización.

3.1. Introducción general

Como vimos brevemente en la introducción, QA es un área de investigación de ciencias de la computación que busca generar respuestas concretas a preguntas expresadas en algún lenguaje natural. Es, por esto mismo, un problema complejo que involucra herramientas y modelos de otras áreas de existencia autónoma, como information retrieval (IR), procesamiento del lenguaje natural (PLN) e information extraction (IE).

También señalamos algunos ejes de subclasificación de problemas de QA. Con respecto al dominio de hechos y conocimientos sobre los que se espera que el sistema sepa responder, los sistemas se clasifican como de dominio abierto (*open domain*) o de dominio cerrado (*closed domain*): mientras de los primeros se espera que sepan responder preguntas acerca cualquier tema o dominio, de los segundos solo se espera que sepan responder preguntas acerca de un dominio acotado particular. A su vez, los datos pueden ser estructurados, semi estructurados o no estructurados. El ejemplo típico de una base de conocimientos estructurada es una base de datos relacional, un tipo de datos semi estructurados puede ser un documento XML no normalizado, mientras que el tipo de datos no estructurado por antonomasia es un corpus de documentos en texto plano.

Las bases de conocimiento de los sistemas de QA pueden tener uno o más tipos de datos, pero cada unos de estos tipos determina un enfoque algorítmico diferente. Por ejemplo, si la base de conocimientos es una DB relacional con un lenguaje formal de consultas similar al SQL, el problema de QA típico consiste en encontrar un mapeo desde la pregunta en lenguaje natural a una consulta formal expresada en un lenguaje comprensible para la base de datos. Esto implica que el foco de trabajo está en la generación de esta consulta y no en el trabajo posterior sobre los resultados de ejecutarla. Por otro lado, un corpus no estructurado no permite una consulta formal, por lo que el enfoque usual es obtener una lista de documentos relevantes para luego aplicar distintas técnicas de procesamiento de textos para extraer la respuesta buscada.

Estos dos ejes de clasificación (grado de especificidad del dominio y grado de estructuración de los datos) suelen tener una correlación, a saber: los dominios cerrados suelen disponer (o permitir la construcción sencilla de) una base de conocimientos estructurada, mientras que los dominios abiertos suelen forzar datos no estructurados. Esta correlación no es una asociación. En realidad, muchos sistemas open domain actuales son híbridos para estas clasificaciones, ya que suelen combinar varias bases de conocimiento de distintos tipos. Un sistema híbrido podría tener un corpus no estructurado basado en la web para preguntas generales y, a su vez, varias bases estructuradas para dominios específicos, que permitan responder sólo algunas preguntas con un resultado mejor. Otro uso puede ser extraer respuestas mediante métodos de PLN sobre corpora no estructurados para luego verificar su tipo contra bases de conocimiento estructuradas generales como Freebase o dbPedia.

En cuanto a aplicaciones comerciales masivas, podemos mencionar la aplicación Siri, el asistente personal de iOS y también la incorporación de funcionalidades de question

aswering en los grandes buscadores[1]. Otros proyectos relativamente actuales que podríamos mencionar son Wolfram Alpha[2], un sistema online de question answering lanzado por la compañía Wolfram Research y también a IBM-Watson, el sistema de IBM que venció a los favoritos del show de trivia estadounidense Jeopardy!

La organización de este capítulo es como sigue: en lo que queda de esta sección (3.1 Introducción general), repasaremos la historia de la disciplina, desde sus orígenes hasta el estado actual de las competencias en torno a las cuales está hoy estructurada la investigación (3.1.1), luego reseñaremos las métricas utilizadas por estas competencias a la hora de evaluar la performance de los sistemas (3.1.2). En la siguiente sección (3.2 Literatura y sistemas) pasaremos revista de diferentes investigaciones actuales sobre question answering para definir un modelo más o menos estándar del dominio de problemas y los acercamientos típicos, discutiendo diferentes investigaciones sobre QA open-domain (3.2.1), un enfoque estructurado para dominios cerrados (3.2.2) y, finalmente el caso concreto de la implementación de IBM-Watson (3.2.3).

3.1.1. Historia y Competencias

Los primeros sistemas que podemos considerar como pertenecientes al área datan de los años sesenta. Las primeras décadas de investigación al respecto están centradas en sistemas de dominio cerrado, es decir, en sistemas cuyo objetivo consiste en responder preguntas sobre temas específicos, apoyándose en general en una base de conocimientos estructurada, específicamente creada para el sistema. Por ejemplo Baseball [Green et al., 1961], un proyecto del MIT que data de 1961, es la primera investigación académica cuyo enfoque la incorpora con certeza en este área que pudimos encontrar en Internet. Baseball respondía preguntas sobre algunos hechos vinculados al torneo de la Liga Americana de baseball de un año concreto. La información se archivó de manera estructurada y contenía: el día, el mes, el lugar, los equipos y los puntajes de cada juego por un año[3]. Otro ejemplo conocido es el chatbot-psicóloga Eliza, cuyo código original fue escrito en 1966, que se basa en reglas muy simples de matcheo de las lineas ingresadas por el usuario para simular una conversación de terapia bastante convincente[4]. Existieron otros desarrollos e investigaciones durante las siguientes décadas, con complejidad creciente, pero con un enfoque similar al de Baseball y de Eliza. Pero la investigación, la producción de software y, principalmente, la formación de una comunidad de desarrollo e investigación mínimamente articulada vinculada con el question answering estuvo impulsada fuertemente por el lanzamiento de los ejercicios de QA en la competencia TREC[5] (Text Retrieval Conference) en el año 1999, la TREC-8 [Voorhees and Tice, 1999].

TREC se convirtió en una referencia obligada y centro de nucleamiento de la investigación en QA. Esta primera competencia mono-lengua (para inglés), consistía en retornar

[1] Buscar, por ejemplo: "¿Cuándo nació Bill Gates?" en Google

[2] `www.wolframalpha.com`

[3] Como una nota de color, trascribimos las dos primeras oraciones del abstract: "Baseball is a computer program that answers questions phrased in ordinary English about stored data. The program reads the question from punched cards.". Traducido: Baseball es un programa de computadora que responde preguntas verbalizadas en inglés común sobre datos guardados. El programa lee la pregunta de tarjetas perforadas.

[4] En internet están disponibles diferentes implementaciones de Eliza. En nuestro repositorio de github `https://github.com/julian3833/eliza` puede encontrarse una implementación funcional en python

[5] `http://trec.nist.gov/`

un fragmento de texto de entre 50 y 250 bytes conteniendo la respuesta a preguntas fácticas (factoids). La evaluación, que marcó la historia futura de la evaluación de QA en competencias, consideraba el sistema de QA como un todo, es decir, no consideraba la evaluación de subcomponentes. La organización permitía dar una lista con 5 respuestas posibles por cada pregunta, y esto siguió así hasta la TREC-2002, cuando se comenzó a exigir una sola respuesta por pregunta. TREC siguió lanzando un track de QA en todas sus competencias anuales, variando la forma y la complejidad de las tareas propuestas, por ejemplo: incorporando preguntas largas, con un formato realista y corpora de mayor tamaño, también introduciendo preguntas sin respuesta (NIL answers) y preguntas de tipo Lista (por ejemplo: '¿Qué libros escribió Friedrich Nietzsche entre 1870 y 1896?'), pregunta de definiciones (por ejemplo: '¿Quién es el General Paz?'), en TREC 2003, preguntas agrupadas por un tema (target), explícito o implícito y de diferentes tipos -como personas, organizaciones, cosas, eventos, etc-, restricciones temporales (por ejemplo: antes, durante y después de una fecha, evento o período), anáforas y co-referencias entre preguntas (por ejemplo: ¿Quién es George Bush?. ¿Y quién es *su* esposa?), etc. Las competencias anuales de la TREC, por otro lado cambiaron el enfoque que existía en décadas anteriores, hacia preguntas de dominio abierto y corpora de datos textuales planos.

En lo que respecta a los sistemas multi-idioma (y mono-idioma no centrados en inglés), más allá de algunos eventos anteriores, un hito clave es el lanzamiento, en 2003, de la primer tarea de QA [Magnini et al., 2003] de CLEF (Cross Language Evaluation Forum)[6] proponiendo tareas tanto monolingües como multilingües en varios idiomas europeos. Se propusieron tareas monolingües para francés, español, alemán, italiano y neerlandés, y tareas multilingües, con corpora en inglés y preguntas en otros idiomas europeos. En este año, se exigía respuestas de hasta 50 bytes, se permitía una lista de 3 respuestas por pregunta y las tareas podían incluir preguntas sin respuesta en los corpora. En la CLEF 2004, la tarea de QA principal incorporó 9 lenguas de origen (lengua de la pregunta) y 7 de destino (lengua del corpus), habilitando 55 ejercicios mono y bilingües. La cantidad de respuestas se redujo a 1 y se incorporaron preguntas de tipo "¿Cómo...?", de definiciones, de listas y también se agregaron restricciones temporales (antes, durante y después de una fecha, evento o período). En general, las respuestas concretas deben ir acompañadas de un fragmento de texto "soporte", es decir, el fragmento de texto del cual se extrajo la respuesta. CLEF continuó presentando *tracks* de tareas de QA todos los años desde entonces, con diferentes variaciones. Otra conferencia de prestigio en el área es NTCIR [7], que es un análogo a CLEF para idiomas asiáticos, incorporando tareas monolingües y bilingües entre inglés, japonés y chino, entre otras.

3.1.2. Métricas de evaluación

En cuanto a las métricas de evaluación utilizadas por estas competencias, es importante distinguir la evaluación de las respuestas a una pregunta y la evaluación general del sistema. Con respecto a la evaluación de respuestas, la clasificación usual de evaluación consiste en 4 valores asignados a cada respuesta: Correcta (C), si responde a la pregunta y el fragmento de soporte justifica esa respuesta; No soportada (U, de Unsupported), si responde a la pregunta pero el fragmento de soporte no está o no justifica la respuesta; Inexacta (X), si sobra o falta información en la respuesta (algunas competencias solo aceptan respuestas

[6] `http://www.clef-initiative.eu/`
[7] `http://research.nii.ac.jp/ntcir/`

exactas) y, finalmente, Incorrecta (I), si ninguna de las condiciones anteriores se cumple.

La evaluación de respuestas puede ser estricta (strict) o *indulgente* (lenient) dependiendo de si considera las respuestas no soportadas como válidas o como inválidas. Por otro lado, la evaluación puede ser manual -llevada a cabo por un jurado especial- o bien automática. En el caso de la evaluación automática, siempre es indulgente, ya que no hay forma automática sencilla de decidir si un fragmento soporta o no una respuesta. Para este tipo de evaluaciones se suele utilizar un set de patrones de respuestas correctas que se deriva de un set de "respuestas conocidas" generadas por los organizadores y los mismos competidores, conocido como R-set, pero no es un proceso trivial: en principio, es difícil saber qué respuestas correctas pueden esperarse, ya que por la naturaleza del problema es probable que diferentes sistemas devuelvan respuestas textualmente distintas.

Con respecto a la métricas de evaluación para sistemas como un todo, veremos a continuación las métricas más conocidas: MRR (Mean reciprocal rank, o promedio de rankings recíprocos), CWS (Confidence Weighted Score, Puntaje ponderado por confiabilidad) y K-Measure (Medida K).

MRR - Mean Reciprocal Rank

El MRR mide la habilidad de un sistema para responder un conjunto de pregunta Q. Asume que el sistema devuelve una o más respuestas por pregunta, en orden de "confiabilidad". El puntaje de cada pregunta individual ($RR(q_i)$) se define como la inversa de la posición de la primera respuesta correcta en la lista de respuestas dada, o cero si no hay ninguna.

$$\text{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} RR(q_i). \tag{3.1}$$

$$RR(q_i) = \begin{cases} 0 & \text{si no existe ninguna respuesta correcta} \\ \frac{1}{\text{Posicion de } q_i} & \text{si } q_i \text{ es la primer respuesta correcta} \end{cases}$$

MRR puede tomar valores entre 0 y 1 inclusive, aunque el RR de cada pregunta tiene un espectro finito de valores (por ejemplo, para 5 respuestas puede valer 0, 0.2, 0.25, 0.33, 0.5 o 1). La primera vez que se usó MRR fue en la competencia TREC-8. En esa competencia se permitía una lista de 5 respuestas ordenadas para cada pregunta.

CWS - Confidence Weighted Score

CWS (de Confidence Weighted Score, Puntaje de confiabilidad ponderada, también conocido como Precisión promedio) asume que el sistema devuelve una respuesta única por pregunta pero que estas respuestas a las diferentes preguntas están ordenadas por grado de confiabilidad descendente. CWS recompensa respuestas correctas colocadas primero. CWS toma valores entre 0 y 1. Uno de los principales problemas de esta métrica es que puede juzgar no la capacidad para responder preguntas sino la de ordenarlas por confiabilidad.

$$\text{CWS} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{\text{Cantidad de rtas correctas en las primeras } i \text{ posiciones}}{i}. \tag{3.2}$$

K-Measure

K-Measure fue propuesta en [Herrera et al., 2004]. Pide a los sistemas una lista de respuestas por pregunta y un grado de confiabilidad para cada una de ellas, entre 0 y 1.

$$\mathrm{K} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{\sum_{r \in ANS_i} conf(r) \times eval(r)}{\max\{|R_i|, |ANS_i|\}}. \tag{3.3}$$

Para una pregunta i, $R(i)$ es el total de respuestas distintas conocidas, mientras que ANS_i son las respuestas dadas por el sistema a evaluar para la pregunta i, $conf(r)$ es la confiabilidad entre 0 y 1 asignada por el sistema a la respuesta r y $eval(r)$ es la evaluación hecha por la organización.

$$eval(r) = \begin{cases} 1 & \text{si } r \text{ es juzgada correcta} \\ 0 & \text{si } r \text{ está repetida} \\ -1 & \text{si } r \text{ es incorrecta} \end{cases}$$

Vale que $K \in \mathbb{R} \wedge K \in [-1, 1]$, y $K = 0$ se identifica con un sistema que asigna confiabilidad 0 a todas sus respuestas y, por lo tanto sirve como baseline.

La dificultad principal de esta métrica es determinar un R_i exhaustivo. Una propuesta alternativa que tiene esto en cuenta, propuesta también en [Herrera et al., 2004] es $K1$, que elimina la ocurrencia de R_i, pero solo sirve para evaluar sistemas que retornan una sola respuesta por pregunta.

$$\mathrm{K1} = \frac{\sum_{r \in ANS_i} conf(r) \times eval(r)}{|Q|}. \tag{3.4}$$

$$eval(r) = \begin{cases} 1 & \text{si } r \text{ es juzgada correcta} \\ -1 & \text{si } r \text{ es incorrecta} \end{cases}$$

Nuevamente, vale que $K1 \in \mathbb{R} \wedge K1 \in [-1, 1]$ y $K1 = 0$ se condice con un sistema sin conocimiento.

Con respecto de los resultados concretos de sistemas existentes, podemos mencionar aquí como ilustración los resultados de la TREC'07 (En la que participó Qanus, nuestro framework baseline para nuestro sistema con soporte multilenguaje) y la CLEF'07, de la que tomamos los ejercicios para evaluar nuestra adaptación multilingüe de dominio abierto. En ambas competencias se permitía solo una respuesta, juzgada por su exactitud por jueces humanos (con diferentes mecanismos de supervisión y control cruzado, excelentemente explicados en [Dang et al., 2008] y [Voorhees and Tice, 1999], para TREC y en [Clef, 2007b] para CLEF). La precisión se define como la cantidad de respuestas exactas sobre la cantidad de preguntas totales. En la tabla 3.1 puede observarse la precisión de los 10 mejores sistemas competidores (TREC es una buena métrica del estado para inglés), siendo 70 % de respuestas exactas la precisión del mejor, LympaBA07, 20 % la del décimo, QUANTA y de 12 % la de QA-Sys (el sistema basado en Qanus que compitió en esta competencia). Por otro lado, al respecto del español podemos ver la tabla 3.2, donde la precisión baja notablemente a 44 % para el mejor, Pribeam, llegando a 7 % para el quinto TALP, mientras que para el portugués podemos ver la tabla 3.3, con resultados de 60 % para el mejor, prib071ptpt y de 4 % para el séptimo, esfi072ptpt.

Run Tag	Submitter	Accuracy	NIL Prec	NIL Recall
LymbaPA07	Lymba Corporation	0.706	0.000	0.000
LCCFerret	Language Computer Corporation	0.494	0.000	0.000
lsv2007c	Saarland University	0.289	–	0.000
UofL	University of Lethbridge	0.258	0.052	0.500
QASCU1	Concordia University	0.256	0.000	0.000
FDUQAT16A	Fudan University	0.236	0.053	0.312
pronto07run3	Universita di Roma "La Sapienza"	0.222	0.000	0.000
ILQUA1	State University of New York (SUNY) at Albany	0.222	0.000	0.000
Ephyra3	Carnegie Mellon University and Universitaet Karlsruhe	0.208	0.048	0.062
QUANTA	Tsinghua University (State Key Lab)	0.206	0.091	0.062

Fig. 3.1: Resultados de TREC 07

Run	**% Overall Accuracy [200]**
Priberam	44,5
Inaoe	34,5
Miracle	15
UPV	11,5
TALP	7

Fig. 3.2: Resultados de CLEF'07 para tareas ES-ES

Run Name	**Overall Accuracy (%)**
diue071ptpt	40,9%
esfi071ptpt	7,4%
esfi072ptpt	4,0%
feup071ptpt	22,8%
ines071ptpt	11,4%
ines072ptpt	14,1%
prib071ptpt	61,7%

Fig. 3.3: Resultados de CLEF'07 para tareas PT-PT

3.2. Literatura y sistemas

3.2.1. Enfoques sobre open domain

Existe un consenso general a la ahora de definir el modelo de software más abstracto o de arquitectura para encarar la construcción de un sistema de question answering open domain. Esta arquitectura consiste en un pipeline con al menos tres módulos o componentes bien diferenciados:

- Módulo de procesamiento de la pregunta
- Módulo de procesamiento de documentos
- Módulo de procesamiento de la respuesta

Cada uno de estos módulos cuenta a su vez con subcomponentes sobre los cuales hay mayor o menor consenso, que determinan, en definitiva, la performance del sistema concreto implementado.

El módulo de procesamiento de la pregunta tiene como subcomponente principal un Clasificador de Preguntas (Ver 2.3.3 Clasificación de Preguntas (QC) y A.1 Herramientas de Stanford) y puede incorporar otros componentes como identificadores del 'foco' (*focus*) y del tipo esperado de la respuesta (*answer type*), y también puede incluirse en este módulo un componente encargado de expandir o reformular la pregunta para optimizar los resultados del módulo de procesamiento de documentos (este subcomponente también se suele considerar como parte del módulo de procesamiento de respuestas). Este último subcomponente tiene diferentes nombres en la literatura (*query reformulation*, *query generation*, *query expansion*, etc).

En el núcleo del módulo de procesamiento de documentos, por lo general, hay uno o más índices invertidos (por ejemplo, de Lucene o Indri) como el descripto en 2.2.2 Índice invertido de búsqueda, o bien accesos a un buscador web, o, más en general, estructuras de information retrieval. Como vimos (2.2.1), existen dos métricas fundamentales a la hora de evaluar el funcionamiento de un módulo de information retrieval: precisión y recall. La precisión es la proporción de documentos relevantes devueltos sobre el total documentos devueltos por el módulo, mientras que el recall es la proporción de documentos relevantes devueltos sobre el total de documentos relevantes existentes en la base de conocimientos. Estas dos métricas suelen comportarse una con la otra como un trade-off, es decir: suelen aparecer escenarios de diseño en los que se deberá optar por priorizar una o bien la otra. Considerando esto, existe una tercera métrica extendida, la medida F, que busca parametrizar este compromiso.

En el contexto de uso de los sistemas de QA, la métrica que se busca maximizar para el módulo de procesamiento de documentos es el recall, es decir: no es importante que el módulo sepa no dejar de lado documento no relevantes (precisión), lo esencial es que recupere todos los documentos relevantes existentes que pueda (exhaustividad). El motivo es simple: se espera que el análisis lingüístico fino del módulo de procesamiento de la respuesta sea capaz de eliminar texto irrelevante en la misma o en mejor medida que el sistema de information retrieval subyacente al módulo de documentos. Por otro lado, si al recuperar los documentos se filtra una respuesta válida como irrelevante, entonces el resto del pipeline verá sus probabilidades de éxito mermadas (sino directamente anuladas). La

expansión/reformulación de la pregunta del paso anterior apunta, justamente, a generar una query que priorice el recall sobre la precisión.

Otras tareas que se realizan en el módulo de procesamiento de documentos es la división de los documentos en parágrafos, el filtrado de documentos o parágrafos irrelevantes y el ranking de los resultados. Si bien los sistemas de information retrieval suelen rankear sus resultados, este ranking puede resultar no del todo adecuado para la tarea en cuestión y mejorable teniendo en cuenta la información disponible.

Finalmente, el módulo de procesamiento de respuestas introduce el valor agregado que distingue un sistema de IR típico y un sistema de QA propiamente dicho. Este módulo es el menos estandarizado de los tres y tiene diferentes enfoques más o menos elaborados en la literatura, pero el grado de consenso sobre la viabilidad del uso de un grupo de técnicas fijas sobre otras no ocurre como, por ejemplo, en el uso de un clasificador de preguntas en el módulo de procesamiento de preguntas o de un índice de búsquedas en el módulo de procesamiento de documentos. Típicamente, el módulo consta de tres momentos: a partir de los parágrafos o documentos rankeados generados por el módulo de documentos se busca identificar oraciones o respuestas candidatas mediante distintas técnicas. Para estas respuestas candidatas identificadas, se extrae la información concreta que responde a la pregunta utilizando heurísticas basadas en las etiquetas obtenidas para la pregunta (clase de pregunta, foco, tipo de respuesta, etc). Finalmente, se implementa algún mecanismo de corroboración o de validación de respuestas, generando una respuesta final.

A continuación nos extenderemos con más detalle sobre los tres módulos propuestos señalando algunos enfoques usuales, usando como guía los papers [Allam and Haggag, 2012], [Lampert, 2004], [Wang, 2006], [Moldovan et al., 2000] y [Tellex et al., 2003].

Módulo de procesamiento de la pregunta

Este módulo recibe como input una pregunta formulada en algún lenguaje natural y genera como output una representación de la misma que resulte útil para los módulos siguientes. Esto suele realizarse agregando labels o etiquetas (ver detalle de "Features", en 2.1 Terminología) a la pregunta completa y a sus distintos términos.

La etiqueta principal generada es el tipo de pregunta (*question type*) y para obtenerlo se utiliza un clasificador como el que describimos en (2.3.3)· Lamentablemente, la clasificación de preguntas no está desarrollada para otros idiomas que el inglés, al menos, no con el mismo grado de precisión y visibilidad que, por ejemplo, el clasificador de Stanford [Hermjakob, 2001] [Li and Roth, 2002]. Como vimos anteriormente, el principal problema a la hora de clasificar una pregunta es la ambigüedad intrínseca de ciertas clases de preguntas.

A partir del tipo de pregunta se define el tipo de respuesta esperada (*answer type*). No existe un algoritmo automático estándar difundido para generar el tipo de respuesta esperado (por ejemplo: individuo humano, grupo humano, creación humana, lugar, número en general, cantidad, fecha, cantidad de dinero) El enfoque general encontrado es la aplicación de un mapeo simple basado en reglas, desde el tipo de pregunta generado por el clasificador a una serie de tipos de respuestas predefinido para el dominio de problema. El

Otro análisis interesante pero, según nuestra investigación, no sistematizado, es el análisis del "foco" (*question focus*). En [Moldovan et al., 2000] se define el foco como una palabra o secuencia de palabras que indican qué se está preguntando. Por ejemplo, para la pregunta "¿Quién fue nombrado presidente de Argentina en 1983?" el foco sería "presidente"). El concepto aparece también en [Ferrucci et al., 2010] pero no encontramos detalles sobre

los mecanismos técnicos implicados en la extracción y todo parece señalar a una serie de heurísticas de procesamiento lingüístico escritas a mano.

Otros análisis que se realizan sobre la pregunta son el POS-tagging (Ver 2.3.1 Part-of-speech (POS) tagging), el reconocimiento de entidades nombradas (NER, ver 2.3.2 Reconocimiento de Entidades Nombradas (NER)) y la búsqueda de dígitos u otros patrones útiles para el contexto que excedan al NER.

Finalmente, se ejecuta la reformulación o expansión de la pregunta para generar un input para el módulo de information retrieval tal que maximice el recall. La idea guía es la extracción de keywords relevantes y existen diferentes enfoques, usando los POS-tags y el resultado del NER. También pueden utilizarse recursos léxicos como Wordnet para generar sinónimos de términos importantes.

En [Lampert, 2004] y [Moldovan et al., 2000] se propone una heurística basada en 8 reglas para implementar este paso. La implementación consiste en agregar, en orden, los siguiente tokens:

1. Seleccionar todas las palabras no stop words entre comillas
2. Seleccionar todas las entidades nombradas reconocidas
3. Seleccionar todas las construcciones nominales con sus adjetivos
4. Seleccionar todas las demás construcciones nominales
5. Seleccionar todos los sustantivos con sus adjetivos
6. Seleccionar todos los demás sustantivos
7. Seleccionar todos los verbos
8. Seleccionar el 'focus' de la pregunta

Módulo de procesamiento de documentos

El input del módulo de procesamiento de documentos es la pregunta reformulada según describimos en los párrafos inmediatamente anteriores y su output es una lista de documentos (o parágrafos) ordenados según la factibilidad de que contengan la respuesta a la pregunta. El módulo consta, en general, de uno o más sistemas de information retrieval (entre los cuales puede estar la web indexada) y de submódulos de filtrado, fragmentación y reordenamiento. Sobre los sistemas de information retrieval es importante notar que suelen usarse sistemas tradicionales y no aquellos como el utilizado, por ejemplo, en latent semantic analysis (LSA). Este último enfoque demostró ser útil a la hora de eliminar o reducir problemas de sinonimia y polisemia en information retrieval, pero en QA es importante que las keywords de búsqueda mismas estén presentes en los documentos relevantes y no que sean solamente "semánticamente similares". Dada la gran cantidad de documentos que se espera que retorne el subsistema de information retrieval (como señalamos, se prioriza el recall y no la precisión), existe un paso de filtrado (en [Ferrucci et al., 2010] se lo nombra como 'soft filtering') en el cual: 1º se reduce la cantidad de documentos a retornar por el módulo en general y 2º se reduce la cantidad de texto en cada documento. El principio que guía este proceso es la idea de que las palabras clave de la pregunta deben aparecer, de alguna manera, cercanas entre sí. De este modo, es posible buscar por estas keywords en los

documentos y quedarse solo con los N parágrafos que contengas las keywords, desechando el resto del texto. Finalmente, se procede a ordenar los parágrafos/documentos filtrados. Una implementación posible de este ordenamiento es utilizar radix sort, considerando features de ordenamiento específicos para el dominio de problemas o bien re generando un índice de information retrieval con la nueva información recortada.

Módulo de procesamiento de la respuesta

Finalmente, el módulo de procesamiento de respuestas es el encargado de generar una respuesta a partir del output de los otros dos módulos. Para ello debe identificar la respuesta dentro de los parágrafos, extraer la fracción de texto que responde puntual y concretamente a la pregunta y, finalmente, validar su corrección.

Para identificar la respuesta se utilizan distintas heurísticas, utilizando el tipo de respuesta esperado cuando es posible. Sin embargo, el tipo de respuesta esperado no siempre es explícito en la pregunta o en la respuesta, por lo que se utilizan diferentes reglas basadas en POS-tags, NERs, en un parser específico para el reconocimiento de las respuestas candidatas.

Para las respuestas candidatas se extrae luego el o los términos concretos de la respuesta final a partir del fragmento de texto elegido. Existe sobre este punto heurísticas más o menos avaladas basadas en distancias entre keywords, número de keywords presentes y otras métricas similares (veremos ejemplos de esto en la implementación de IBM y en el baseline de Qanus en breve). Típicamente, si no se encuentra ninguna coincidencia confiable, los sistemas de QA productivos recaen en la devolución de los primeros n parágrafos mejor rankeados por el módulo de documentos.

Como señalamos anteriormente, en las competencias TREC hasta 2001 se permitía devolver una lista de 5 respuestas, mientras que desde el 2002 se exige una única respuesta.

Finalmente, hay distintas formas de validar la corrección o de estimar un grado de confianza en la corrección de una respuesta. Un enfoque típico es utilizar diferentes corpora específicos estructurados con los cuales contrastar la información obtenida hasta el momento. Veremos este enfoque en breve en 3.2.3 IBM-Watson.

3.2.2. QA como interfaz a una base de datos

El enfoque a la hora de encarar una solución de question answering sobre dominios cerrados (específicos) difiere sustancialmente de los enfoques sobre dominios abiertos. Un dominio de conocimientos general fuerza datos en texto plano, documentos escritos en lenguaje humano, tal como vimos en la sección anterior. La razón de esto es que no existen bases de datos, o, en general, repositorios de datos estructurados acerca de... todo. En sistemas open domain, sin embargo, es posible incorporar bases de conocimiento estructuradas como complemento de otras no estructuradas. Cuando coexisten ambos tipos de datos, como vimos, hablamos de sistemas híbridos.

Los usos que puede hacer un sistema híbrido de sus bases de información estructurada son varios. Un uso típico es responder preguntas sobre dominios específicos, esto es, un sistema puede utilizar bases de conocimiento estructuradas y algoritmia específica para ciertos temas concretos (por ejemplo: Liga Americana del año 1958) y para ciertos tipos de preguntas (¿cuál fue el resultado del partido...?), mientras que para el resto de las preguntas recae en métodos de dominio abierto. Estos "subsistemas" son más costosos y difíciles de construir, pero, en general, logran mejores resultados para las preguntas que

acepta (y brindan una estimación de la confiabilidad de la respuesta bastante buena). Otro uso típico es verificar el tipo de una respuesta extraído de un corpus plano, contra ontologías como Freebase, Yago o dbPedia [Kalyanpur et al., 2012], por ejemplo, verificando que la respuesta generada para "¿Quién fue el presidente de...?" sea una persona y, mejor, un presidente.

Los sistemas de dominio cerrado puros, por otro lado, tienen interés en diferentes ámbitos en los que se dispone de información estructurada de utilidad para crear una interfaz más natural con la DB, permitiendo así el acceso a la base de datos a personas sin los conocimientos técnicos necesarios para escribir una consulta en el lenguaje formal que brindan las bases de datos como interfaz de consultas. En combinación con procesamiento del lenguaje (entendido como la traducción del sonido hablado a texto) abre muchas posibilidades interesantes de aplicaciones.

Para ambos casos de uso de los modelos de dominio cerrado (sistemas híbridos y de dominio cerrado puros) el enfoque algorítmico consiste en traducir la pregunta formulada en un lenguaje natural a una consulta del lenguaje formal (potencialmente: SQL, SPARQL o similar). Dado que la cantidad de "cuestiones" sobre las que se puede responder es siempre acotada, el modelo estándar de dominio cerrado sobre datos estructurados busca traducir la pregunta formulada en lenguaje humano a una consulta acerca de algún aspecto de la base de datos. Debe, además poder decidir si una pregunta tiene sentido en el contexto de la información de la que dispone.

En esta tesis utilizamos como marco teórico el desarrollado por Ana-Maria Popescu et. al. en sus papers [Popescu et al., 2003a] y [Popescu et al., 2003b].

En estos trabajos, Ana María Popescu et. al. definen la noción de pregunta *semánticamente tratable* en el contexto de una base de datos concreta. Proponen, así mismo, un modelo teórico para identificar este tipo de preguntas y una forma de transformarlas en consultas de SQL, así como también un sistema concreto que implementa el modelo, llamado Precise. Popescu argumenta que una interfaz a una base de datos en lenguaje natural puede no identificar una pregunta, pero que jamás debe mal interpretarla activamente, es decir, interpretar algo distinto a lo preguntado y dar una respuesta que no se condice con la necesidad de información del usuario. Una mala interpretación de una pregunta reducirá la confianza del usuario en el sistema, volviéndolo inusable. En cambio, identificando una pregunta como intratable, se puede disparar un proceso de reformulación o especificación asistida de la pregunta, lo cual no es tan costoso en términos de confianza en el sistema.

La idea principal que guía a sus trabajos es proponer una clase de preguntas específica que sea 1) suficientemente sencilla para ser interpretada por un modelo computacional y, a la vez, suficientemente abarcadora de las preguntas generalmente hechas por humanos. La intuición detrás de este enfoque es que, mientras, en el caso general, una pregunta puede ser compleja, ambigua y difícil de comprender (incluso por un humano), también hay preguntas simples, unívocas y con una interpretación sencilla incluso para una máquina (por ejemplo: "¿Qué restaurantes de comida china hay en Belgrano?"). La *tratabilidad semántica*, cualidad de una pregunta para una base de datos dada, cuya definición intentaremos reconstruir a continuación, define esta clase de preguntas simples.

En lo que resta de esta sección, describiremos con detalle el modelo teórico propuesto por Popescu en sus publicaciones, así como también el sistema concreto implementado por sus autores con ese modelo como guía.

Definiciones iniciales: elementos, compatibilidad, token, correspondencia, tipado de tokens, marcador sintáctico

Según el modelo relacional usual, una base de datos está compuesta de tres tipos de **elementos**: *valores, atributos y relaciones*. Cada elemento es distinto y único. Por relación nos referimos a lo que usualmente llamamos tabla (por ejemplo: Universidades), por atributo a una columna de una tabla (por ejemplo: Dirección), y por valor a un dato particular de una fila de una columna (Por ejemplo: Figueroa Alcorta 1234).

Popescu define la noción de **compatibilidad** entre elementos como sigue:

1. Un valor es compatible con su atributo
   - Figueroa Alcorta 1234 es compatible con Dirección
2. Un valor es compatible con la relación que contiene a su atributo
   - Figueroa Alcorta 1234 es compatible con Universidades
3. Un atributo es compatible con su relación
   - Dirección es compatible con Universidades
4. Cada atributo tiene un conjunto de qwords compatibles
   - Dirección es compatible con "Dónde"

Observemos que mientras las primeras tres reglas de compatibilidad son independientes de los contenidos de la base de datos, la cuarta, en principio, requiere una formulación explícita (humana) de la relación entre qwords y atributos.

Las **qwords**, recordemos, son los pronombres interrogativos, palabras típicas con las que comienza una pregunta: quién, qué, cómo, cuándo, dónde, etc para el castellano, why, where, who, which, etc para el inglés. En los trabajos de Popescu, se define extensivamente como una palabra dentro del conjunto {what, which, where, who, when}.

Por otro lado, se introduce una noción de **token**, diferente de la que dimos en 2.1 Terminología. En el contexto de este modelo, un token es un conjunto de lemas de palabras (Ver 2.3.4 Otras tareas) que corresponden a un elemento de la base de datos. Por ejemplo, {experiencia, requerir} y {experiencia, necesario} son tokens que corresponden al atributo "Experiencia Requerida" de una base de datos.

Otra definición inicial importante es el concepto de **correspondencia**[8], que no está definido teóricamente sino operacionalmente al describir el sistema Precise. El procedimiento utilizado es el siguiente (Utilizaremos el atributo "Experiencia Requerida" como ejemplo para ilustrar cada paso):

- Cada elemento de la base de datos se separa en palabras individuales:
  - "Experiencia Requerida" → {Experiencia, Requerida}
- Se genera un conjunto de sinónimos para cada palabra usando Wordnet:
  - experiencia → {experiencia, conocimiento, habilidad,...}

[8] Traducimos aquí *to match* por *corresponder*

  - requerida → {requerida, necesaria, indispensable,...}

- Se toman los lemas o raíces de todas las palabras
  - experiencia → {experiencia, conocimiento, habilidad,...}
  - requerida → {requerir, necesario, indispensable,...}
- Se generan tokens combinando los lemas de los sinónimos de cada elemento:
  - tokens = {(experiencia, requerir), (conocimiento, requerir), (habilidad, requerir),(experiencia, necesario), (conocimiento, necesario), (habilidad, necesario),(experiencia, indispensable), (conocimiento, indispensable), (habilidad, indispensable)}
- Este conjunto de tokens se asocia a cada elemento:
  - "Experiencia Requerida" → tokens (como está definido en el bullet anterior)

Con este procedimiento se construye un diccionario de elementos de la base de datos en listas de tokens "sinónimos", llamado **lexicon**. Finalmente, basándose en este diccionario, se construye la función **correspondencia**, que va de tokens en elementos de la base de datos. Dado un token, se obtienen todos los elementos de la base de datos a los que ese token corresponde, esto es, todos los elementos para los cuales el token en cuestión pertenece al conjunto de tokens generado según el procedimiento recién enunciado. Notar que un token puede corresponder a más de un elemento de la base de datos.

Los tipos de elementos a los que un token corresponde determinan el conjunto de **tipos posibles de un token** dado: token de relación, token de atributo, token de valor. Qué tipo tome un token dentro de sus tipos posibles dependerá de la correspondencia utilizada en el contexto dado.

Finalmente, un **marcador sintáctico** se define como un token perteneciente a una lista fija, independiente de la base de datos concreta, de tokens que no hacen ninguna contribución semántica a la interpretación de la pregunta (como "the", "are" y "on").

Tokenización completa, asociación sintáctica, traducción válida, tratabilidad semántica

El conjunto de preguntas que este trabajo rescata como semánticamente tratables es un conjunto conceptualmente simple pero prácticamente muy abarcador. Esto significa que se abordan preguntas formuladas de forma sencilla, pero que corresponden a una gran cantidad de preguntas de las que realmente se deberá ocupar un sistema de interfaz de lenguaje natural a una base de datos. En cualquier caso, como ya dijimos antes, existe siempre la posibilidad de pedirle al usuario una reformulación de la pregunta.

La definición de tratabilidad semántica se basa en -e intenta formalizar- la observación de que muchas preguntas formuladas en lenguaje natural constan de una especificación de pares atributo/valor así como también de valores sueltos, donde el atributo asociado está implícito. Un atributo puede, además, estar asociado a una qword.

Tomando el ejemplo de Popescu, consideremos la pregunta: *"What French restaurants are located downtown?"* en el contexto de una base de datos simplificada que conste de una sola relación **Restaurants** con atributos **Name**, **Cuisine** y **Location**. En este caso, la palabra "French" refiere al valor French del atributo de la base de datos *implícito* Cuisine,

mientras que las palabras "located" y "downtown" refieren al atributo *explícito* Location y a su valor Downtown. Finalmente, la palabra "restaurant" refiere a la relación Restaurants, como así también la qword "What".

Para definir la tratabilidad semántica es necesario introducir antes dos definiciones más: **tokenización completa** y **asociación sintáctica**.

Considerando las definiciones recién enunciadas, se define como una **tokenización completa de una pregunta *q*** a cualquier conjunto de tokens en los que cada término que no es un marcador sintáctico de q aparece en exactamente un token del conjunto.

Por su parte, decimos que dos tokens están **sintácticamente asociados**[9] en el contexto de una pregunta q cuando estos dos tokens cumplen ciertas restricciones en el árbol sintáctico de la pregunta.

La función *attachment*, que modela la asociación sintáctica, tiene como dominio un par de palabras o lemas (y la oración en la que aparecen) y como imagen verdadero o falso. En la implementación presentada por los investigadores (Precise), como veremos en la sección siguiente, se utiliza un Charniak Parser para modelar esta función, aunque los detalles de las condiciones implementadas no se presentan nunca.

Con estos conceptos definidos, podemos dar finalmente la definición de **traducción válida**[10]. Una traducción válida de una pregunta q en un conjunto de elementos E de una base de datos es una correspondencia de alguna tokenización completa de q en elementos de E que cumple las siguientes tres condiciones (notar que los términos *correspondencia*, *compatibilidad* y *asociación sintáctica* están usados según las definiciones técnicas recién reconstruidas):

1. Cada token se corresponde con un único elemento de E.
2. Cada token de atributo se relaciona con un único token de valor, cumpliendo las siguientes condiciones:
   - el atributo de la base de datos que corresponde al token de atributo y el valor de la base de datos que corresponde al token de valor son compatibles
   - ambos tokens están sintácticamente asociados
3. Cada token de relación está relacionado a un token de atributo o bien a un token de valor, cumpliendo las siguientes condiciones:
   - la relación de la base de datos que corresponde al token de relación y el elemento de la base de datos que corresponde al token de atributo o token de valor son compatibles
   - ambos tokens (token de relación - token de atributo o bien token de relación - token de valor) están sintácticamente asociados

Notar que no se exigen condiciones sobre los tokens de valor (esto captura la intuición de que los valores pueden aparecer con su atributo asociado implícito).

Si una pregunta tiene al menos una traducción válida en la que todos los tokens son distintos y en la que al menos unos de sus tokens de valor es una qword, entonces la pregunta es **semánticamente tratable**. Que una pregunta sea *semánticamente tratable* significa

[9] Traducimos aquí *syntactic attachment* como *asociación sintáctica*
[10] Traducimos aquí *valid mapping* por *traducción válida*

que existe una traducción válida de los tokens significativos de la pregunta a elementos de la base de datos con ciertas condiciones y, como veremos en la siguiente sección, este conjunto de elementos puede traducirse, de una manera trivial, en una consulta SQL.

En el próximo título veremos la implementación concreta de este marco teórico por sus autores, comenzando con un ejemplo que permitirá consolidar los conceptos recién definidos.

Implementación: Precise

Dada una pregunta q, Precise determina si es semánticamente tratable y si lo es, genera la consulta SQL correspondiente. Si no lo es, rechaza la pregunta pidiendo al usuario una reformulación. El problema de encontrar una correspondencia de una tokenización completa de q en un conjunto de elementos de la base de datos que cumpla las condiciones 1, 2 y 3 es traducido a un problema de matching de grafos y resuelto aplicando un algoritmo de max-flow (flujo máximo). Cada solución del algoritmo de max-flow corresponde a una interpretación semántica posible de la pregunta. Precise recolecta todas estas soluciones, descartando aquellas que no cumplen con la función *attachment* y con las restantes genera consultas SQL que corresponden a la pregunta q: si hay más de una consulta, pregunta al usuario cual de ellas formuló. Si hay una sola, retorna el resultado. Siguiendo al paper original, presentaremos aquí, en primer lugar, un ejemplo del sistema funcionando y, luego, describiremos los módulos del mismo en mayor detalle. En la implementación propia sobre la base de datos World, presentada en 4 Implementación de datos estructurados, se podrá ver nuestra implementación de este modelo.

Comencemos con el ejemplo del paper[Popescu et al., 2003a]: veamos los pasos por los cuales Precise traduce la pregunta *"What are the HP jobs on a Unix system?"* en una consulta SQL. Esta pregunta fue elegida para ilustrar la clase de ambigüedades que el modelo puede resolver automáticamente. Por brevedad y claridad, asumamos que la base de datos consta de una sola relación (**Jobs**) con atributos **Description**, **Platform** y **Company**. En primer lugar, el tokenizador produce una única tokenización completa de la pregunta: {what, HP, job, Unix, system}. Notar que el tokenizer elimina marcadores sintácticos tales como "the" y "a".

Buscando estos tokens en el lexicón, Precise obtiene la lista de los elementos de la base de datos que se corresponden con cada token (ver Figura 3.4).

El siguiente paso es el Matcher, que construye el grafo de atributos y valores presentados en la Figura 3.5. Para comprender el significado de cada nodo en el grafo es útil leerlo columna por columna de izquierda a derecha. El nodo más a la izquierda, S es el nodo fuente. La columna "Value Tokens" consiste en los tokens que corresponden con valores de la base de datos (que, a su vez, aparecen en la columna "DB Values"). Por ejemplo, el token HP es ambiguo y puede corresponder tanto con un valor del atributo **Company** como con un valor del atributo **Platform**. Se agregan aristas entre tokens de valor y valores de la base de datos para cada una de estas posibles correspondencias. Las aristas sólidas representan el flujo final obtenido por el algoritmo, mientras que las aristas punteadas representan flujos alternativos posibles.

El Matcher conecta cada valor de la base de datos con su atributo correspondiente. Cada atributo es luego conectado con el token de atributo correspondiente y también con el nodo I, que representa atributos *implícitos*. Por otro lado, todos los tokens de atributo son conectados con el nodo E, que representa a los atributos *explícitos*. Finalmente, tanto

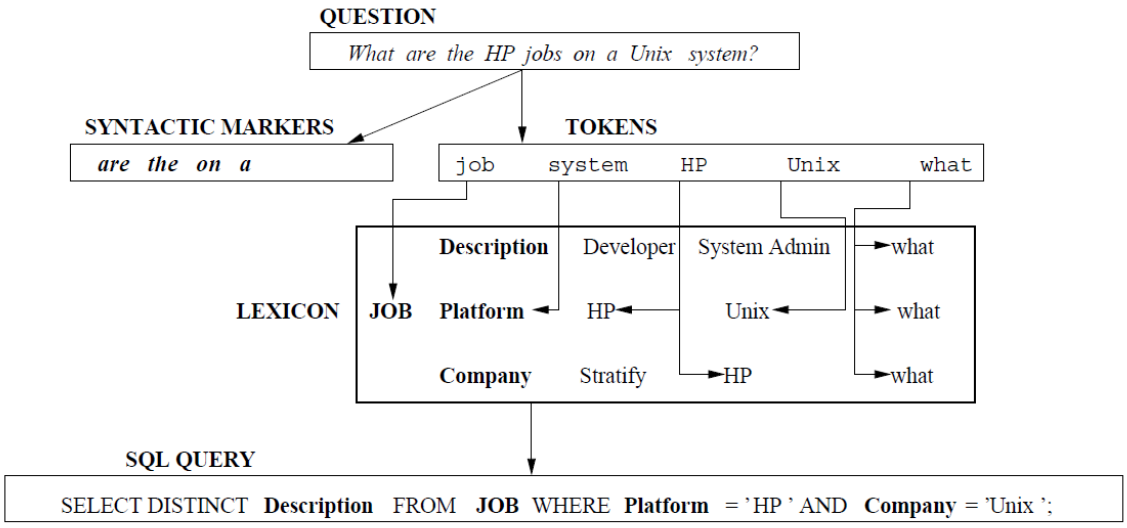


Fig. 3.4: La transformación de la pregunta "What are the HP jobs on a Unix system?" a una consulta SQL por Precise, el trabajo de Popescu et al.

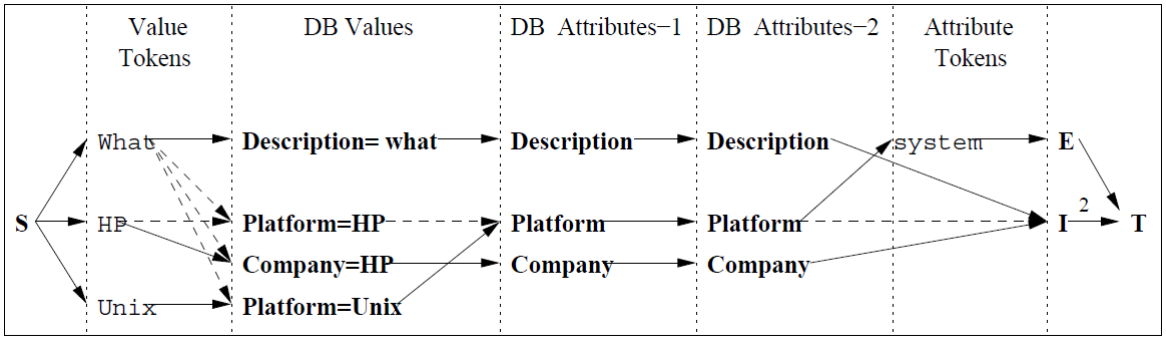


Fig. 3.5: El grafo de atributos y valores creado por Precise para la pregunta "What are the HP jobs on a Unix system?"

E como I son conectados con T, el nodo destino.

Las segunda columna de "DB attributes", donde cada atributo de la base de datos es conectado consigo mismo por una arista de flujo máximo 1, garantiza que cada atributo sea usado, a lo sumo, una vez. Estas aristas son necesarias ya que más de un valor puede ser compatible con un atributo dado, y un atributo puede corresponder con más de un token de atributo, pero la definición de traducción válida dada exige que cada atributo de la base de datos sea utilizado a lo sumo una vez.

El grafo es interpretado como una red de flujo en la que cada arista tiene capacidad 1, a menos que se indique lo contrario. La capacidad de la arista de E a T es el número de tokens de atributo (en este ejemplo, 1). La capacidad de la arista de I a T es el número de tokens de valor menos el número de tokens de atributo, en este ejemplo: 2. Estas capacidades fuerzan al algoritmo a enviar una unidad de flujo de cada token de valor hacia cada atributo explícito. El Matcher ejecuta el algoritmo de max-flow sobre este grafo buscando una solución entera (el flujo máximo que puede pasar por la red desde el nodo

S al nodo T respetando las capacidades de cada arista). En este ejemplo, el flujo máximo es 3. De hecho, el flujo máximo del grafo de atributos y valores construido por Precise es siempre el número de tokens de valor, porque cada token de valor tiene que participar en el emparejamiento construido por el algoritmo.

Las flechas sólidas representan el camino elegido por el algoritmo de max-flow. Notar que la ambigüedad sobre si HP era el nombre de una compañía o una plataforma fue automáticamente resuelto al maximizar el flujo. El algoritmo decidió que HP era una plataforma, porque de otro modo no hubiera habido una interpretación posible para el token "Unix" y el flujo final hubiera sido 2.

Después de que todos los atributos y valores fueron apareados con elementos de la base de datos, el sistema verifica que todos los tokens de relación se corresponden con un token de valor o con uno de atributo. En este caso, con un único token de relación (job), esto se reduce a verificar si alguna de las relaciones de la base de datos relacionadas con el token contienen algún atributo que se corresponda con algún token de atributo. Como en el ejemplo job solo se corresponde con la única relación existente (Job), todos los atributos están contenidos por ella y la condición se hace verdadera. Finalmente, el algoritmo encontró una correspondencia uno a uno entre los tokens de la pregunta y elementos de la base de datos que satisfacen las condiciones semánticas requeridas en la definición de tratabilidad semántica. En este punto, Precise verifica que los tokens emparejados satisfacen las condiciones sintácticas modeladas en la función *attachment* para los pares de tokens (what, job), (HP, job) y (Unix, system). Si esto es así, entonces una *traducción válida* ha sido encontrada y el sistema retorna la consulta SQL que puede verse en la base de la Figura 3.4. En el trabajo se presenta también un ejemplo sobre preguntas cuya traducción a consultas SQL implica la aparición de diferentes cláusulas JOIN posibles. Nosotros omitiremos esa parte ya que en nuestra implementación del modelo estructurado no soportamos esa funcionalidad y utilizamos una única cláusula JOIN canónica por cada apareamiento de tablas posible. Baste mencionar que la implementación consta de otra red de flujo, más simple que la recién presentada, para desambiguar tokens de relación.

Pasemos ahora revista de los diferentes módulos del sistema, que son los siguientes:

- Lexicon: encargado de generar, para cada elemento de la base de datos, el conjunto de tokens sinónimos que se utilizará para verificar correspondencia.
- Tokenizer: encargado de generar todas las tokenizaciones completas de la pregunta y, para cada token, consultar al lexicon y retornar la lista de elementos de la base de datos que le corresponden.
- Matcher: encargado de verificar que las tokenizaciones completas y los elementos correspondientes generados por el tokenizer cumplan las condiciones 1 a 3 utilizando el modelo de grafos y el algoritmo max-flow recién ilustrado.
- Parser plug-in: el módulo computa la función de asociación sintáctica, basándose en el parse tree de Charniak.
- Query generator: dado el conjunto de elementos de la base de datos traducido de la pregunta, genera una query SQL.
- Equivalence Checker: verifica si diferentes queries son la misma formulada de diferentes maneras.

El **Lexicon** es un módulo que se construye offline, como una derivación de la base de datos, siguiendo el procedimiento enunciado en la sección anterior al describir la función de *correspondencia*. La interfaz del módulo, utilizada por el Tokenizer, consta de dos operaciones:

1. Dado un lema, devolver el conjunto de tokens que lo contienen

2. Dado un token, devolver el conjunto de elementos de la base de datos que le *corresponden*, según la definición de correspondencia dada en la sección anterior de esta tesis.

Para construir el Lexicon a partir de la base de datos, se sigue el siguiente procedimiento: a cada elemento de la base de datos se le asocia un conjunto de lemas (este paso puede ser simplemente lematizando los términos o agregando, a mano, un "conjunto aumentado" de lemas). A partir de estos lemas se construye una lista de sinónimos, utilizando una ontología (como, por ejemplo, Wordnet), obteniendo los sinónimos de cada palabra y luego generando los conjuntos de pares ordenados pertinentes (Ver el algoritmo en detalle en la sección anterior).

Por su parte, el **Tokenizer** es el punto de entrada del sistema. Recibe como input una pregunta en lenguaje natural y genera como resultado un conjunto de todas las posibles tokenizaciones completas de la misma. Para ello:

- lematiza las palabras

- elimina los marcadores sintácticos,

- para cada lema, consulta al Lexicon por el conjunto de tokens que lo contienen

- para cada token potencial, verifica que los demás lemas del token estén presentes en el conjunto de lemas de la pregunta original

- para cada partición completa de los lemas en listas de tokens (es decir, para cada tokenización), busca los elementos de la base de datos que corresponden a cada uno de los tokens que la componen.

De este modo, el Tokenizer genera diferentes particiones del conjunto de lemas (tokenizaciones de q), cada una con una traducción de cada token a todos los elementos de la base de datos que le corresponden. Será responsabilidad del Matcher decidir que tokenizaciones completas son traducciones válidas.

El **Matcher** representa la innovación central de Precise, reduciendo el problema de encontrar una interpretación semántica de una pregunta formulada en un lenguaje natural ambiguo en un conjunto de elementos de la base de datos a un problema de matching de grafos. En concreto, el módulo toma como input una tokenización completa de q, con todos los posibles elementos de la DB que corresponden a cada token. Más claramente:

- Una lista de tokens en donde cada palabra de q pertenece a un y solo un token

- Para cada uno de los tokens de la lista, un conjunto de elementos E de la base de datos (los elementos que le *corresponden*, en el sentido técnico de corresponder)

Este módulo es el encargado de verificar qué conjunto de elementos E de entre los disponibles cumple con las condiciones semánticas 1 a 3 (de la definición de traducción válida). El problema es modelado como un problema de matching (apareamiento) de grafos y resuelto con un algoritmo de max-flow, tal como acabamos de presentar en el ejemplo. Cada solución dada por el algoritmo es una posible interpretación semántica de la pregunta.

Luego, el **Parser Plug-in** verifica la parte sintáctica de las condiciones, tal como mencionamos en el ejemplo anterior. Este módulo utiliza un *Charniak parser* (un tipo especifico de árbol sintáctico) para extraer relaciones entre palabras en la oración. Solo las traducciones que satisfagan las condiciones de asociación sintáctica son traducciones válidas.

El **Query Generator** toma como input los elementos de la base de datos seleccionado por el Matcher y los convierte en una consulta SQL, con las reglas presentada en la tabla 3.1. Omitimos, de nuevo aquí, la construcción de la cláusula JOIN.

Sección de la consulta	Pares de elementos
SELECT	Elementos apareados con qwords
WHERE	Pares de atributos y valores generados por el Matcher
FROM	Todas las relaciones mencionadas

Tab. 3.1: Traducción de pares de elementos de la base de datos en una consulta SQL

El **Equivalence Checker** elimina, en el caso de soluciones múltiples, soluciones repetidas generadas de manera distinta (por ejemplo, los pares atributo/valor aparecen en otro orden). Si Precise encuentra dos consultas SQL distintas, no genera una respuesta sino que pide al usuario por una desambiguación. Por ejemplo, en la pregunta "¿Qué trabajos de analista en sistemas existen en Buenos Aires?", "analista en sistemas" puede ser el nombre de un trabajo, mientras que también puede ocurrir que "analista" sea el nombre del trabajo mientras que "sistemas" sea el área. En este caso, Precise no logra desambiguar la situación y necesita que el usuario lo haga por él mismo.

Con esto terminamos con el modelo de dominio cerrado, en la próxima sección veremos un caso de éxito de un proyecto de question answering híbrido implementado por IBM y en el próximo capítulo describiremos la implementación de un modelo de question answering de dominio cerrado en el que utilizamos este marco como orientación.

3.2.3. IBM-Watson

Watson[Ferrucci et al., 2010][Kalyanpur et al., 2012] es un sistema diseñado por IBM con el objetivo de competir en tiempo real en el programa de televisión estadounidense Jeopardy, logrando resultados del nivel de los campeones humanos de este programa.

El proyecto demoró 3 años de investigación, en los cuales se logró obtener la performance esperada (nivel humano experto) en cuanto a precisión, confiabilidad y velocidad, logrando derrotar a dos de los hombres con mayores récords históricos del show en un programa en vivo[11] en febrero de 2011.

[11] En esta url está disponible el programa en el que el sistema vence a sus competidores humanos: `http://www.youtube.com/watch?v=WFR3lOm_xhE`

El objetivo del proyecto puede considerarse una extensión de lo que fue Deep Blue, el sistema que logró el nivel de los expertos humanos en el ajedrez, porque buscó superar un reto significativo y visible del campo de la Inteligencia Artificial tanto para la comunidad científica como para la sociedad en general: "¿puede un sistema computacional ser diseñado para competir con los mejores hombres en alguna tarea que requiera altos niveles de inteligencia humana y, si es el caso, qué clase de tecnología, algoritmos e ingeniería se requiere?" [Ferrucci et al., 2010].

A continuación presentamos brevemente la descripción de Watson según los trabajos [Ferrucci et al., 2010] y [Kalyanpur et al., 2012].

Watson es la implementación específica para participar en este programa de una arquitectura más genérica de question answering, Deep-QA, que da el nombre al proyecto de la corporación. Esta arquitectura es de construcción reciente y ejemplifica perfectamente la complejidad del problema de QA de dominio abierto, incorporando tecnologías de punta de distintos dominios de ciencias de la computación y de IA en particular: information retrieval, natural language processing, knowledge representation and reasoning, machine learning e interfaces humano - computadora. En el transcurso de esta tesis, IBM lanzó el programa 'Watson Ecosystem' (en noviembre de 2013) que promete la utilización de tecnología de punta para aplicaciones creadas por la comunidad[12].

Watson debió cumplir con ciertas condiciones bastante exigentes, las que determinan su éxito como un real hito de question answering:

- Confiabilidad de la respuesta:
  Jeopardy tiene tres participantes con un pulsador y el que desee responder debe pulsar antes que los demás. Además, existe una penalización por respuestas incorrectas, por lo que es esencial que el sistema pueda determinar la confiabilidad de la respuesta obtenida a fin de optar por responder o no responder.

- Tiempos de respuesta:
  La confiabilidad de la respuesta, o al menos una estimación, debe calcularse antes de que pase el tiempo para decidir responder (6 segundos) y también de que otro participante oprima su pulsador (menos de 3 segundos en promedio).

- Precisión:
  El tipo de respuestas que se dan en el show suelen ser respuestas exactas (por ejemplo: solamente un nombre, un número o una fecha, etc).

El sistema tiene una arquitectura conocida como 'híbrida', ya que cuenta con varios componentes heurísticos que estiman ciertos features y grados de confiabilidad para diferentes respuestas, los cuales son evaluados por un sistema general que sintetiza un grado de confiabilidad para una respuesta final y determina así si responder o no responder.

Jeopardy consta de un tablero con 30 pistas (o preguntas) organizadas en seis columnas, cada una de las cuales es una categoría. Las categorías van desde temas acotados como "historia" o "ciencias" hasta temas más amplios como "cualquier cosa" o "popurrí". Watson intenta respuestas sobre varias hipótesis de dominio y verifica en cual de ellos se logran respuestas de mayor confiabilidad.

[12] Announcing the IBM Watson Ecosystem Program: `http://www-03.ibm.com/innovation/us/watson/`

Por otra parte, el grueso de las preguntas de Jeopardy son del tipo *factoid*, esto es, preguntas cuya respuesta esta basada en información fáctica acerca de una o más entidades individuales.

A su vez, existen ciertos tipos de pistas que requieren un enfoque particular, por ejemplo, pistas que constan de dos subpistas muy débilmente relacionadas, o problemas matemáticos formulados en lenguaje humano, o problemas de fonética, etc, que no pueden ser simplemente dejados de lado porque, si bien tiene poca probabilidad de aparición, cuando aparecen lo hacen en bloque y pueden arruinar el juego de Watson. Se acordó con la productora del programa, sin embargo, dejar de lado preguntas audiovisuales (aquellas que presentan una imagen o un audio y requieren interpretarlo) y preguntas que requieren instrucciones verbales del presentador.

Para determinar el dominio de conocimiento, los investigadores analizaron 20000 preguntas, extrayendo su LAT (lexical answer type, o tipo léxico de respuesta). El LAT se define como una palabra en la pista que indica el tipo de la respuesta esperado. Por ejemplo, para la pista "Inventada en el Siglo XIV para agilizar el juego, este movimiento involucra dos piezas" el LAT es "movimiento". Menos del 12 % de las pistas no indicaba explícitamente ningún LAT, usando palabras como "esto" o "eso". En estos casos, el sistema debe inferir el tipo de respuesta del contexto. Del análisis de estas 20000 pistas se reconocieron 2500 tipos léxicos distintos, de los cuales los 200 más frecuentes no llegaban a cubrir el 50 % del total de pistas. Esto implica que un approach estructurado (orientado por el tipo de respuesta), si bien resulta útil para algunos tipos, no es suficiente para abordar el problema completo.

Métricas

Las métricas de resultados, además del tiempo de respuesta, son la *precisión* (preguntas contestadas correctamente / preguntas contestadas) y el *porcentaje de respuestas dadas* (preguntas contestadas / total de preguntas). Mediante la configuración de un threshold de *confiabilidad* pueden obtenerse distintas estrategias de juego: un umbral bajo repercutirá en un juego más agresivo, incrementando la proporción de respuestas contestadas, pero disminuyendo su precisión, mientras que un umbral alto determinará un juego conservador, con menos respuestas dadas pero mayor precisión en las mismas. Es un clásico escenario de trade-off entre dos atributos de calidad. Un buen sistema de estimación de confiabilidad implica una mejora general del sistema, aún cuando el módulo de generación de respuestas permanezca idéntico.

En el show, el porcentaje de respuestas dadas depende de la velocidad con la que se llega a presionar el pulsador, lo cual sólo interesa para el dominio de QA como una restricción temporal.

Mediante análisis numérico, los investigadores determinaron que los campeones de Jeopardy lograban tomar entre el 40 % y el 50 % de las preguntas y, sobre ellas, lograban una precisión de entre el 85 % y el 95 %, lo que determinaba una barrera de performance bastante exigente en lo que respecta a QA.

Baseline

El equipo de IBM intentó utilizar dos sistemas consolidados en QA y adaptarlos al problema de Jeopardy. El primero fue PIQUANT (Practical Intelligent Question Answering

Technology), un sistema desarrollado por IBM en conjunto con el programa del gobierno estadounidense AQUAINT y varias universidades, que estaba entre los mejores según la TREC. PIQUANT consta de un pipeline típico con tecnología de punta, logrando un rango del 33 % de respuestas correctas en las evaluaciones TREC-QA. Los requerimientos de la evaluación de TREC son muy distintos de los de Jeopardy: TREC ofrece un corpus de conocimiento relativamente pequeño (1M de documentos) de donde las respuestas deben ser extraídas y justificadas, el tipo de preguntas de TREC son menos complejas a nivel lingüístico que las de Jeopardy y la estimación de confiabilidad no resulta una métrica importante (dado que no hay penalización por respuestas incorrectas). Además, los sistemas tienen permitido acceder a la web y las restricciones temporales son, por mucho, más amplias (por ejemplo: una semana para responder 500 preguntas). En Jeopardy, además de las restricciones ya mencionadas, un requerimiento fue que el sistema trabaje sobre datos locales y no acceda a la web en tiempo real. El intento de adaptar PIQUANT al problema de Jeopardy dio pésimos resultados en comparación con los necesarios: 47 % de precisión sobre el 5 % de respuestas con mayor confiabilidad y 13 % de precisión en general.

Por otro lado, el equipo intentó adaptar el sistema OpenEphyra [13], un framework open-source de QA desarrollado en CMU (Carnegie Mellon University) basado en Ephyra (no libre), diseñado también para la evaluación TREC. OpenEphyra logra un 45 % de respuestas correctas sobre el set de datos de evaluación TREC 2002, usando búsqueda web. La adaptación resultó aún peor que la de PIQUANT (con menos del 15 % de respuestas correctas y una mala estimación de la confiabilidad).

Se probaron dos adaptaciones de estos sistemas, una basada en búsquedas de texto puro y otra basada en reconocimiento de entidades. En la primera, la base de conocimiento se modeló de manera no estructurada y las preguntas se interpretaron como términos de una query, mientras que en la segunda se modeló una base de conocimientos estructurada y las preguntas se analizaron semánticamente para reconocer entidades y relaciones, para luego buscarlos en la base. Comparando ambos enfoques en base al porcentaje de respuestas dadas, el primero dio mejores resultados para el 100 % de las respuestas, mientras que la confiabilidad general era baja; por otro lado, el segundo enfoque logró altos valores de confiabilidad, pero sólo en los casos en que efectivamente logra identificar entidades. De aquí se infiere que cada enfoque tiene sus ventajas, en el dominio de problemas apropiado.

La arquitectura DeepQA

Los intentos de adaptación iniciales, como vimos, no dieron resultados, así como tampoco sirvieron las adaptaciones de algoritmos de la literatura científica, los cuales son difíciles de sacar de su contexto original y de las evaluaciones sobre las cuales fueron testeados. Como conclusión de estos intentos frustrados, el equipo de IBM entendió que una arquitectura de QA no debía basarse en sus componentes concretos sino en la facilidad para incorporar nuevos componentes y para adaptarse a nuevos contextos. Así surgió DeepQA, la arquitectura de base, de la cual Watson es una instancia concreta para un contexto particular (con requerimientos de alta precisión, buena estimación de confiabilidad, lenguaje complejo, amplitud de dominio y restricciones de velocidad). DeepQA es una arquitectura de cómputo paralelo, probabilístico, basado en recopilación de evidencia y scoring. Para Jeopardy se utilizaron más de 100 técnicas diferentes para analizar lenguaje natural, identificar y adjudicar valor a fuentes de información, encontrar y generar hipótesis, encontrar

[13] `http://www.ephyra.info/`

y rankear evidencias y mergear y rankear hipótesis en función de esta evidencia. La arquitectura sirvió para ganar Jeopardy, pero también se adaptó a otros contextos como la evaluación TREC, dando resultados mucho mejores que sus predecesores.

A continuación, enumeraremos la lista de pasos que sigue el sistema para obtener la respuesta a una pregunta:

Adquisición de contenidos

El primer paso de DeepQA es la adquisición de contenidos. Este paso es el único que no se realiza en tiempo de ejecución y consiste en crear la base de conocimiento en la cual el proceso final buscará la respuesta a la pregunta, combinando subprocesos manuales y automáticos.

En principio se caracteriza el tipo de preguntas a responder y el dominio de aplicación. El análisis de tipos de preguntas es una tarea manual, mientras que la determinación del dominio puede encararse computacionalmente, por ejemplo, con la detección de LATs que señalamos antes. Dado el amplio dominio de conocimientos que requiere Jeopardy, Watson cuenta con una gran cantidad de enciclopedias, diccionarios, tesauros, artículos académicos y de literatura, etc. A partir de este corpus inicial, el sistema busca en la web documentos relevantes y los relaciona con los documentos ya presentes en el corpus. Todo esto ocurre previo a la ejecución real, por lo que el sistema no accede a la web en tiempo real, tal como era requerido por los organizadores (dado que sería una ventaja con respecto a los humanos, que no tienen permitido este acceso).

Además de este corpus de documentos no estructurados, DeepQA maneja contenidos semi-estructurados y estructurados, incorporando bases de datos, taxonomías y ontologías como dbPedia, Wordnet y las ontologías de Yago.

Análisis de la pregunta

El primer paso en run-time es el análisis de la pregunta. En este paso el sistema intenta entender qué es lo que la pregunta está formulando y realizar los primeros análisis que determinan cómo encarará el procesamiento el resto del sistema. Watson utiliza shallow parsers, deep parsers, formas lógicas, POS-tags, correferencias, detección de entidades nombradas y de relaciones, question classification, además de ciertos análisis concretos del dominio del problema.

En este proceso se clasifica el tipo de la pregunta (los tipos están determinados por el show: puzzles, matemáticos, etc). También se busca el tipo de respuesta esperada, dónde los tipos manejados por Watson son los LATs extraídos de las preguntas de ejemplo. El LAT determina el "tipo" de la respuesta, qué clase de entidad *es* la respuesta (una fecha, un hombre, una relación, etc). El equipo de IBM intentó adaptar distintos algoritmos de clasificación preexistentes, pero después de intentar entrenarlos para el dominio de tipos de Jeopardy, llegaron a la conclusión de que su eficacia era dependiente del sistema de tipos default, y que la mejor forma de adaptación era mapear su output a los tipos utilizados por Watson.

Otra anotación importante es el "foco" de la pregunta, la parte de la pregunta tal que si se la reemplaza por la respuesta, la pregunta se convierte en una afirmación cerrada. Por ejemplo, para "El hombre que escribió Romeo y Julieta", el foco es "El hombre que" (notar que esta definición de foco según los autores de IBM es diferente a la dada anteriormente

en este capítulo). Este fragmento suele contener información importante sobre la respuesta y al reemplazarlo por una respuesta candidata se obtiene una afirmación fáctica que puede servir para evaluar distintos candidatos y recolectar evidencia. Por ejemplo, reemplazando por distintos autores y verificando que la oración resultante esté presente en el corpus.

Por otro lado, muchas preguntas involucran relaciones entre entidades y, más puntualmente, tienen una forma sujeto-verbo-objeto. Por ejemplo, tomando la pista anterior, podemos extraer la relación *escribir(x, Romeo y Julieta)*. La amplitud del dominio de Jeopardy hace que la cantidad de entidades y de relaciones entre entidades sea enorme, pero esto empeora aún más al considerar las distintas formas de expresar la misma relación. Por eso, Watson sólo logra encontrar directamente una respuesta mediante reconocimiento de entidades y relaciones sobre el 2 % de las pistas. En general, este tipo de enfoque es útil sobre dominios más acotados, mientras que la detección de relaciones como approach general a un problema de question answering de dominio amplio es un área de investigación abierta.

Generación de hipótesis

El tercer paso (segundo en run-time) es la generación de hipótesis: tomando como input el resultado del paso anterior se generan respuestas candidatas a partir de la base de conocimiento offline. Cada respuesta candidata reemplazada por el foco de la pregunta es considerada una hipótesis, que el sistema luego verificará buscando evidencias y adjudicando un cierto grado de confiabilidad.

En la búsqueda primaria de respuestas candidatas, se busca generar tantos pasajes como sea posible. El resultado final obtenido revela que el 85 % de las veces, la respuesta final se encuentra entre los primeros 250 pasajes devueltos por la búsqueda primaria. La implementación utiliza una serie variada de técnicas, que incluyen diferentes motores de búsqueda de textos (como Indri y Lucene), búsqueda de documentos y de pasajes, búsquedas en bases de conocimiento estructuradas como SPARQL con triple store y la generación de múltiples queries a partir de una sola pregunta. La búsqueda estructurada de triple stores depende del reconocimiento de entidades y relaciones del paso anterior.

Para un número pequeño de LATs, se definió una suerte de conjunto de entidades fijas (por ejemplo: países, presidentes, etc). Si la respuesta final no es retornada en este paso, entonces no hay posibilidad de obtenerla en los siguientes. Por eso se prioriza el recall sobre la precisión, con el supuesto de que el resto del pipeline logrará filtrar la respuesta correcta correctamente. Watson genera varios cientos de hipótesis candidatas en este paso.

Para optimizar recursos, se realiza un filtrado liviano (soft filtering) de respuestas antes de pasar a la recopilación de evidencia y al scoring de hipótesis. Un filtrado liviano es, por ejemplo, comprobar similaridad de la respuesta candidata con el LAT esperado de la respuesta. Aquellas hipótesis que pasan el filtro pasan al siguiente proceso, que realiza un análisis más exhaustivo.

Recuperación de evidencias y scoring de pasajes

Para recuperar evidencias se utilizan varios algoritmos. Uno particularmente útil es buscar la hipótesis candidata junto con las queries generadas por la pregunta original, lo que señala el uso de la respuesta en el contexto de la pregunta. Las hipótesis con sus evidencias pasan al siguiente paso, donde se les adjudica un score.

El proceso de scoring es donde se realiza la mayor parte del análisis más fuerte a nivel computacional. DeepQA permite la incorporación de distintos Scorers, que consideran diferentes dimensiones en las cuales la hipótesis sirve como respuesta a la pregunta original. Esto se llevó a cabo definiendo una interfaz común para los scorers. Watson incorpora más de 50 componentes que producen valores y diferentes features basados en las evidencias, para los distintos tipos de datos disponibles (no estructurados, semi-estructurados y estructurados). Los scorers toman en cuenta cuestiones como el grado de similaridad entre la estructura de la respuesta y de la pregunta, relaciones geoespaciales y temporales, clasificación taxonómica, roles léxicos y semánticos que se sabe que el candidato puede cumplir, correlaciones entre términos con la pregunta, popularidad (u obscuridad) de la fuente del pasaje, aliases, etc.

Los distintos scores se combinan luego en un score único para cada dimensión.

Merge y respuesta

Recién después de este momento, Watson realiza un merge entre hipótesis idénticas. Las hipótesis idénticas son diferentes formulaciones lingüísticas de lo mismo, por ejemplo: "X nació en 1928" o "El año de nacimiento de X es 1928". Finalmente, se procede a estimar un ranking único y una confiabilidad única para las distintas hipótesis. En este paso se utilizan técnicas de machine learning que requieren entrenamiento, y modelos basados en scores para ensamblar los distintos resultados intermedios y generar una respuesta final.

Con esta exposición cerramos el capítulo 3 Estado de Arte. En el próximo veremos nuestra implementación del modelo estructurado closed domain de Popescu para inglés.

4. IMPLEMENTACIÓN DE DATOS ESTRUCTURADOS

En este capítulo, finalmente, nos toca hablar del primero de los dos sistemas implementados en este trabajo. El sistema en cuestión implementa, con algunas limitaciones que mencionaremos, el modelo de Popescu descripto en [Popescu et al., 2003a] y [Popescu et al., 2003b] y reseñados por nosotros en 3.2.2.

Nuestro código está accesible públicamente en la siguiente dirección: `http://github.com/julian3833/popescu-world`. Allí pueden encontrarse, además del código en sí mismo, los procedimientos de instalación, ejemplos de ejecución y una lista con los principales puntos técnicos que pueden mejorarse (ver, también, 4.2.4 Conclusiones, limitaciones y trabajo futuro).

Testeamos este sistema sobre la base de datos World, en inglés, ofrecida por MySQL[1] que consta de información geográfica básica sobre países, ciudades e idiomas.

Cabe mencionar que el scope original de este sistema era ser bilingüe y ejecutarse sobre una base de datos sobre universidades, empresas e investigación nacional del ámbito de la informática. Lamentablemente, por una cuestión de tiempos y de errores a la hora de estimar esfuerzos, no pudimos completar este plan original.

La estructura del capítulo es como sigue: En 4.1 pasamos revista de la base de datos que utilizamos para implementar el sistema y en 4.2 discutimos la implementación. 4.2 se divide, a su vez, en 4.2.1, donde discutimos la implementación en sí misma, 4.2.2, donde mostramos y comentamos algunos ejemplos de ejecuciones y 4.2.4 donde analizamos los alcances, los límites y el trabajo futuro.

4.1. Base de datos

La base de datos World consta de 3 tablas: Country, City y CountryLanguage (ver Figura 4.1).

La tabla Country tiene información básica acerca de varios países del mundo. La tabla City tiene información sobre las mayores ciudades del mundo y, finalmente, la tabla CountryLanguage tiene información sobre los idiomas hablados en cada país.

Hay una relación de uno a muchos entre Country y CountryLanguage: cada país puede tener más de un idioma y, además, hay una relación de uno a muchos entre Country y City: cada país puede tener más de una ciudad.

Las definiciones de las relaciones originales son:

- Country(Code, Name, Continent, Region, SurfaceArea, IndepYear, Population, LifeExpectancy, GNP, GNPOld, LocalName, GovernmentForm, HeadOfState, Capital, Code2)
- City(ID, ContryCode, Name, District, Population)
- CountryLanguage(CountryCode, Language, IsOfficial, Percentage)

[1] Ver `http://dev.mysql.com/doc/world-setup/en/index.html` y `http://dev.mysql.com/doc/index-other.html`

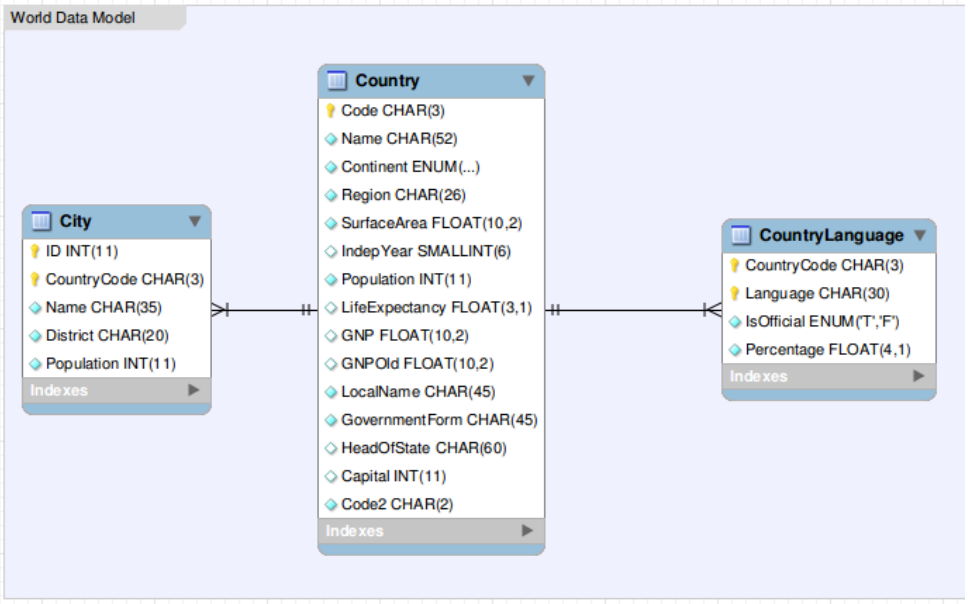


Fig. 4.1: Modelo entidad relación de la base de datos World original

Para nuestro proyecto, renombramos algunos de los elementos:

- La relación CountryLanguage fue renombrada a Language
- El atributo Language de la relación CountryLanguage fue renombrado a Name
- El atributo IndepYear de la relación Country fue renombrado a IndependenceYear

Por otro lado, dado que, por una cuestión de tiempos y de complejidad del desarrollo, no soportamos más de un join posible entre tablas, eliminamos la referencia a la tabla City en el atributo Country.Capital. Lo que hicimos fue rellenar ese campo con el nombre de la ciudad en lugar de su ID, perdiendo la posibilidad de navegar con preguntas del tipo ¿Cuál es la población de la capital del país X?. Los JOINS que manejamos, en caso de ser necesarios por estarse refiriendo más de una tabla, son City.CountryCode = Country.Code y Language.CountryCode=Country.Code.

En las tablas 4.1, 4.2, 4.3 y 4.4 pueden verse algunas filas de ejemplo de cada una de las tablas.

CountryCode	Language	IsOfficial	Percentage
ABW	Dutch	T	5.3
ABW	English	F	9.5
ABW	Papiamento	F	76.7
ABW	Spanish	F	7.4
AFG	Balochi	F	0.9
AFG	Dari	T	32.1
AFG	Pashto	T	52.4
AFG	Turkmenian	F	1.9
AFG	Uzbek	F	8.8
AGO	Ambo	F	2.4

Tab. 4.1: Algunas filas de la tabla CountryLanguage (cont)

CountryCode	Language	IsOfficial	Percentage
AGO	Chokwe	F	4.2
AGO	Kongo	F	13.2
AGO	Luchazi	F	2.4

Tab. 4.1: Algunas filas de la tabla CountryLanguage

ID	Name	CountryCode	District	Population
129	Oranjestad	ABW	–	29034
1	Kabul	AFG	Kabol	1780000
56	Luanda	AGO	Luanda	2022000
61	South Hill	AIA	–	961
34	Tirana	ALB	Tirana	270000
55	Andorra la Vella	AND	Andorra la Vella	21189
33	Willemstad	ANT	Curaçao	2345
64	Dubai	ARE	Dubai	669181
69	Buenos Aires	ARG	Distrito Federal	2982146
126	Yerevan	ARM	Yerevan	1248700
53	Tafuna	ASM	Tutuila	5200
63	Saint John's	ATG	St John	24000
130	Sydney	AUS	New South Wales	3276207

Tab. 4.2: Algunas filas de la relación City

Code	Name	Continent	Region	SurfaceArea	IndepYear
ABW	Aruba	North America	Caribbean	193.00	*NULL*
AFG	Afghanistan	Asia	Southern and Central Asia	652090.00	1919
AGO	Angola	Africa	Central Africa	1246700.00	1975
AIA	Anguilla	North America	Caribbean	96.00	*NULL*
ALB	Albania	Europe	Southern Europe	28748.00	1912
AND	Andorra	Europe	Southern Europe	468.00	1278
ANT	Netherlands Antilles	North America	Caribbean	800.00	*NULL*
ARE	United Arab Emirates	Asia	Middle East	83600.00	1971
ARG	Argentina	South America	South America	2780400.00	1816
ARM	Armenia	Asia	Middle East	29800.00	1991
ASM	American Samoa	Oceania	Polynesia	199.00	*NULL*
ATA	Antarctica	Antarctica	Antarctica	13120000.00	*NULL*
ATF	French Southern territories	Antarctica	Antarctica	7780.00	*NULL*

Tab. 4.3: Algunas filas de la tabla Country

Code	Population	Life Expectancy	GNP	GovernmentForm	HeadOfState	Capital
ABW	103000	78.4	828.00	Nonmetropolitan Territory of The Netherlands	Beatrix	129
AFG	22720000	45.9	5976.00	Islamic Emirate	Mohammad Omar	1
AGO	12878000	38.3	6648.00	Republic	José Eduardo dos Santos	56
AIA	8000	76.1	63.20	Dependent Territory of the UK	Elisabeth II	62
ALB	3401200	71.6	3205.00	Republic	Rexhep Mejdani	34
AND	78000	83.5	1630.00	Parliamentary Coprincipality		55
ANT	217000	74.7	1941.00	Nonmetropolitan Territory of The Netherlands	Beatrix	33
ARE	2441000	74.1	37966.00	Emirate Federation	Zayid bin Sultan al-Nahayan	65
ARG	37032000	75.1	340238.00	Federal Republic	Fernando de la Rúa	69
ARM	3520000	66.4	1813.00	Republic	Robert Kotšarjan	126

Tab. 4.4: Algunas filas de la tabla Country (continuación)

Incorporando estos datos al modelo teórico, hasta aquí tenemos definidos: la base de datos, sus elementos (E) y el tipo de sus elementos.

4.2. Implementación

Estructuramos esta sección como sigue. Primero (4.2.1), discutimos la implementación en sí misma, analizando los módulos más importantes del sistema, luego (4.2.2) mostramos y comentamos algunos ejemplos de ejecuciones y, finalmente (4.2.4), analizamos alcances, límites y trabajo futuro.

Este apartado es probablemente el más áspero y técnico de la tesis. Si la cantidad de conceptos presentada se vuelve demasiado difícil de retener al lector, recomendamos saltear la sección de 4.2.1 Código y comenzar por 4.2.2 Ejemplos, o bien darle una primera lectura aproximativa a 4.2.1, pasar a los ejemplos y luego sí leer en detalle 4.2.1.

4.2.1. Código

Lexicón

El lexicón, recordemos, es el módulo encargado de generar un conjunto de tokens para cada elemento de la base de datos. Una vez construido este conjunto, las responsabilidades del módulo son las siguientes:

- Dado un lema, devolver el conjunto de tokens que lo contienen
- Dado un token, devolver el conjunto de elementos de la base de datos que le *corresponden*.

El lexicón está implementado en la clase *uba.modules.Lexicon*. Debemos distinguir los momentos de construcción y de consulta del mismo. La construcción, cuyo punto de

entrada es el módulo *uba.app.CreateLexicon*, ocurre por separado, y debe ejecutarse una vez antes de poder utilizar el sistema para responder preguntas. Esta genera 4 archivos con formato json, que son luego cargados en memoria al momento de realizar consultas.

En el centro la construcción del lexicón se encuentra Wordnet, una base de datos léxica en inglés, que consta de conjuntos de sinónimos, definiciones de los mismos y relaciones semánticas entre ellos. Utilizamos esta base de datos para obtener los sinónimos de los elementos de la base de datos.

El input del algoritmo es la lista de los elementos de la base de datos (relaciones, atributos y valores) como strings. El primer paso es eliminar el camel case y separar y lematizar las palabras (la tokenizamos en el sentido dado por Popescu a token). Por ejemplo, después de este paso, el elemento GovernmentForm se convierte en el token {government, form}. En este paso eliminamos también las stopwords (por ejemplo, HeadOfState se convierte en {head, state}).

Luego, los datos pasan por el TokenAugmenter, que simplemente agrega algunos sinónimos escritos a mano a algunos de estos elementos (ver tabla 4.5). Por ejemplo, para el elemento "region" (atributo de la relación "country"), agregamos el término "location", para el elemento "surface area" agregamos los términos "total size" y "square kilometers". Este paso es llevado a cabo por la clase *uba.db.TokenAugmenter* y su intención es mejorar las chances de obtener sinónimos útiles a partir de wordnet, ampliando su input de trabajo. Al salir del TokenAugmenter tenemos, para cada elemento de la base de datos, un conjunto (que puede tener un solo string si el TokenAugmenter no tenía ningún sinónimo) de tokens (que son, a su vez, listas de lemas).

Elemento original	Sinónimos
head of state	president, leader, emperor, king
region	location
surface area	size, total size, square kilometers, km2
independence year	independent, independency

Tab. 4.5: Sinónimos introducidos por el Token Augmenter

El tercer paso es el central: en este se obtienen, para cada uno de estos tokens, tokens sinónimos. Para esto obtenemos una lista de sinónimos y palabras relacionadas para cada lema del token, luego los combinamos formando todos los sinónimos ordenados. Por ejemplo, si el token original consistía de los lemas (A, B) y para A obtuvimos los sinónimos {A1, A2} y para B los sinónimos {B1 y B2}, el resultado serán todas las combinaciones ordenadas posibles: {(A, B), (A, B1), (A, B2), (A1, B), (A1, B1), (A1, B2), (A2, B), (A2, B1), (A2, B2)}.

Finalmente, intentamos obtener sinónimos también de todas las palabras del token juntas (incluyendo las stopwords), ya que verificamos empíricamente que para algunas de ellas existía una entrada en wordnet (Por ejemplo "head of state" produce el sinónimo "chief of state" que se pierde sin este paso).

El conjunto de tokens sinónimos para cada elementos de la base de datos es luego invertido, es decir, en lugar de disponer de un mapeo de elementos de la base de datos a conjunto de tokens sinónimos, construimos un mapeo de tokens en elementos de la base de datos.

Vale mencionar que en este índice invertido agregamos también tokens para cada qword posible, mapeando a un solo elemento especial, de tipado similar a un valor (WhValue), mediante el cual los tokens de qwords acceden al espacio de elementos de la base de datos. Además, para estos WhValues definimos a mano la relación de compatibilidad con atributos de la base de datos tal como se define en los papers. Esta relación está definida en la clase *uba.db.WhGenerator* tal como la presentamos en la tabla 4.6.

El proceso del Lexicón se hace offline y con él queda construido el espacio semántico disponible para comprender palabras.

Qword	Atributos relacionados
What	**Name**, District, Population, Code, Continent, SurfaceArea, LifeExpectancy, GNP, LocalName, GovernmentForm, Capital, IsOfficial, Percentage, Region
Which	Los mismos que para what
Where	**Region**, Continent, Capital, District
Who	**HeadOfState**
When	**IndependenceYear**

Tab. 4.6: Atributos compatibles con cada Qword

A partir de este mapeo son generadas algunas estructuras utilizadas para optimizar la performance del sistema, pero sin valor teórico (por ejemplo, la lista de todos los tokens, que es también el conjunto de índices del índice invertido). Estos datos son finalmente grabados en 4 archivos, que son luego cargados a memoria en el sistema principal.

Desde la perspectiva de la lectura, las operaciones son triviales. La interfaz de servicios ofrece los métodos getTokens() y getMatchingElements(). getTokens() es la encargada de devolver, para un lema, el conjunto de tokens que lo contienen, mientras que getMatchingElements() es la encargada de devolver, dado un token, el conjunto de elementos de la base de datos que le corresponden.

Discutamos ahora la implementación del tercer paso, en el que se obtienen sinónimos y palabras relacionadas para cada lema y luego se combinan.

En el caso ideal, lo que se busca es lograr un conjunto de sinónimos para una serie de palabras. Pero lo que entendemos por conjunto de sinónimos se ve opacado por un fenómeno lingüístico conocido como polisemia que es, en algún sentido, un fenómeno inverso a la sinonimia. Al utilizar wordnet siempre existe este problema. La polisemia refiere al hecho de que una palabra puede tener más de un sentido. Es el caso de la palabra banco, que puede ser tanto un asiento en una plaza como una institución financiera. La polisemia es un fenómeno que ocurre en el espacio de relaciones entre las palabras y los conceptos, al igual que la sinonimia, que refiere al hecho de que diferentes palabras pueden referir al mismo concepto (inodoro y retrete). En el core de wordnet existe el tipo de dato synset (conjunto de sinónimos, a veces traducido como anillo de sinónimos), en el que se agrupan, bajo un sentido o concepto, todas las palabras que lo refieren.

En general, los humanos podemos distinguir qué sentido de una palabra polisémica está en uso por contexto. Entre nuestras funciones cognitivas está el reconocer que si alguien dice “voy a hacer un depósito al banco”, nosotros entendamos que banco, en este contexto,

refiere a la institución financiera y no al asiento de plaza. Pero sin contexto es imposible saber de qué sentido se está hablando.

Este problema no es tematizado en los trabajos de Popescu y, sin embargo, ellos argumentan que el sistema Precise fue construido con wordnet (sin mencionar ningún tipo de desambiguador contextual previo). En realidad, existe un atenuante para este problema que es el contexto de uso y el rol que cumplen los conjuntos de sinónimos y es que el conjunto de todos los tokens sinónimos generados funciona como un filtro sobre consultas del usuario y nunca es activo o productivo. Consideremos, por ejemplo, una base de datos sobre localización de sucursales de bancos y el elemento Banco (por ejemplo, el nombre de una relación). Buscando en sinonimos.com obtenemos los siguientes sinónimos, separados en dos líneas: 1) entidad crediticia, 2) taburete, escabel, escaño, peana, sitial, asiento. Tomando todos los sinónimos tendríamos el siguiente conjunto de sinónimos: {banco, entidad crediticia, taburete, escaño, peana, sitial, asiento} donde hay, mezclados, dos sentidos. Ahora bien: ¿qué uso hacemos de este conjunto? El usuario de una aplicación sobre localización de bancos introduce una pregunta, el sistema la separa, lematiza y chequea si los lemas pertenecen a algún conjunto de sinónimos. En este contexto, ¿es un problema que tengamos el término "asiento" en nuestro conjunto de sinónimos? Sería un problema solo si el usuario pudiese introducirlo, ya que al conjunto de sinónimos solo se accede a partir de palabras introducidas por el usuario. Si bien este problema es posible, no es probable y, quizás, siguiendo este razonamiento, quienes propusieron el modelo no hicieron ningún énfasis en él.

Nuestro sistema no introduce ningún desambiguador de sentido: utilizamos todo los sinónimos disponibles de tipo sustantivo, introduciendo potenciales errores de interpretación en este punto, con la salvedad recién mencionada.

Finalmente, agregamos también derivaciones léxicas lematizadas. Una derivación léxica es una variación de la palabra original que da otro sentido (relacionado) a la palabra original. Por ejemplo: existir, existencia, existencial, existiendo, existente son una serie de variaciones de la misma raíz. Esta opción es experimental y puede activarse o desactivarse desde el archivo de configuración ($uba.app.Config$)

En la tabla 4.7 pueden verse algunos ejemplos de los resultados.

Tokenizer

El Tokenizer en el modelo de Popescu, recordemos, es el encargado de generar todas las tokenizaciones completas de la pregunta y, para cada token, consultar al Lexicon y retornar la lista de elementos de la base de datos que le corresponden.

El Tokenizer sigue el siguiente procedimiento:

1. Separar la pregunta en palabras, eliminar puntuaciones y pasar a lower case.
2. Lematizar las palabras. Para esto usamos Freeling
3. Eliminar marcadores sintácticos.
4. Para cada lema, obtener todos los tokens que lo contienen del Lexicon (método getTokens).

Lema	Sinónimos	Derivaciones léxicas
city	metropolis, (urban, center)	citify, metropolitan
country	state, nation, land, commonwealth, res publica, body politic, rural area, area	–
language	language, linguistic communication, speech, speech communication, spoken communication, spoken language, voice communication, oral communication, lyric, words, linguistic process, terminology, nomenclature	lyricist, lyric, speak, terminological
region	region, part, area, neighborhood, realm	–
location	location, placement, locating, position, positioning, emplacement, localization, localisation, fix	locate, position, posit, emplace, localize, localise
continent	–	continental
gnp	gross national product	–
government	government, authorities, regime, governing, governance, government activity, administration, politics, political science	govern, political scientist
form	form, word form, signifier, descriptor, kind, sort, variety, shape, pattern, configuration, contour, conformation, human body, physical body, material body, soma, build, figure, physique, anatomy, bod, chassis, frame, flesh, cast, variant, strain, var., phase, class, grade, course, mannequin, manikin, mannikin, manakin	signify, sort, form, shape, pattern, contour, anatomic, anatomist, anatomical, shapely, variant

Tab. 4.7: Lemas, sinónimos y derivaciones léxicas

5. Para cada token potencial (resultado del paso anterior) verificar que todos sus lemas estén presentes en el conjunto de lemas de la pregunta original.

6. Generar el conjunto de partes de todos los tokens hasta aquí obtenidos[2].

   Probando con cada elemento del conjunto de partes en lugar de utilizar solamente el conjunto original podemos obtener subconjuntos que cumplan también la condiciones requeridas para considerarse una tokenización completa (evaluados en 7).

7. Para cada uno de estos subconjuntos, verificar 1) que sus tokens cubran completamente los lemas significativos de la pregunta original y 2) que no haya lemas repetidos entre los tokens.

El resultado será un conjunto de tokenizaciones completas. Como se ve, respetamos casi al pie de la letra la implementación sugerida por Popescu. La obtención de los elementos de la base de datos que corresponden a los elementos, por una cuestión implementativa, quedó en manos del Matcher y no del Tokenizer. La implementación, de todos modos, del servicio, está en el Lexicon. Con este conjunto de tokenizaciones completas como input, será responsabilidad del Matcher decidir cuáles de ellas son traducciones válidas.

[2] El conjunto de partes de un conjunto dado es otro conjunto formado por todos los subconjuntos del mismo. Por ejemplo, el conjunto de partes de A = {1, 2, 3} es: { ∅, {1}, {2}, {3}, {1, 2}, {2, 3}, {1, 3}, {1, 2, 3} }

Matcher

El Matcher, por su parte, toma las tokenizaciones completas generadas por el Tokenizer, construye el grafo que expusimos en la sección 3.2.2 y ejecuta el algoritmo de max flow. El problema de max-flow es un problema de grafos que consiste en "enviar" el máximo flujo posible a través de un grafo dirigido con dos nodos especiales (fuente o source y sumidero o sink) y aristas con cierta capacidad mayor o igual que cero. Este flujo debe ir desde el nodo fuente al nodo sumidero, respetando las capacidades de las aristas y respetando que, para cada nodo, el flujo saliente no puede ser mayor al flujo entrante.

Existen diferentes algoritmos para resolver este problema. En nuestra implementación, tomamos un código libre disponible en internet (aquí: `http://web.mit.edu/~ecprice/acm/acm08/MaxFlow.java`).

Las aristas implicadas en el flujo máximo posible asocian 1) los tokens de valor y de atributo y los correspondientes elementos (valores y atributos, respectivamente) de la base de datos y 2) pares de valores y atributos entre sí.

Después de esto, buscamos otras soluciones posibles de máximo flujo, dado que cualquier solución que tenga como valor para el máximo flujo la cantidad de tokens de valor en la pregunta es, potencialmente, una traducción válida. Para hacer esto retiramos, ordenadamente, las aristas del grafo que ocurren entre la columna 2 y 3 (tokens de valor y valores de la db).

Cabe notarse aquí, como ya mencionamos anteriormente, que no soportamos en nuestra implementación la desambiguación de tokens de relación. Esto significa que: asumimos que un token de relación tiene un solo elemento relación asociado y no será necesario decidir si refiere a una relación u a otra. En todos los ejemplos de bases de datos que consideramos, esta asunción era verdadera (es decir, no existía un token de relación que necesite ser desambiguado) y por ello no implementamos esta capacidad del sistema, que quedará como trabajo futuro. Así mismo, no soportamos tokens multi tipados, prefiriendo relaciones sobre atributos y valores y atributos sobre valores (el caso implementativo es simple: require un grafo extra por cada combinación de tipos distintos por token). Los casos en que esta ambigüedad ocurría eran específicos y la solución no reportaba ninguna mejora cualitativa, por lo que lo dejamos de lado, quedando como una mejora futura.

Finalmente, verificamos cuales de las soluciones con máximo flujo cumplen las condiciones requeridas para ser una traducción válida según enunciamos en 3.2.2:

1. Todos los tokens de la tokenización tienen un único elemento de la base de datos asociado y no hay elementos de la base de datos repetidos. (Mapping.meetsConditionOne())

2. Cada token de atributo se relaciona con un único token de valor respetando que: (Mapping.meetsConditionTwo())

   a) el atributo relacionado con el token de atributo y el valor relacionado con el token de valor son compatibles (esta condición está garantizada por el max-flow mismo)

   b) ambos tokens están sintácticamente asociados

3. Cada token de relación está relacionado a un token de atributo o bien a un token de valor, cumpliendo las siguientes condiciones: (Mapping.meetsConditionThree())

a) la relación de la base de datos que corresponde al token de relación y el elemento de la base de datos que corresponde al token de atributo o token de valor son compatibles

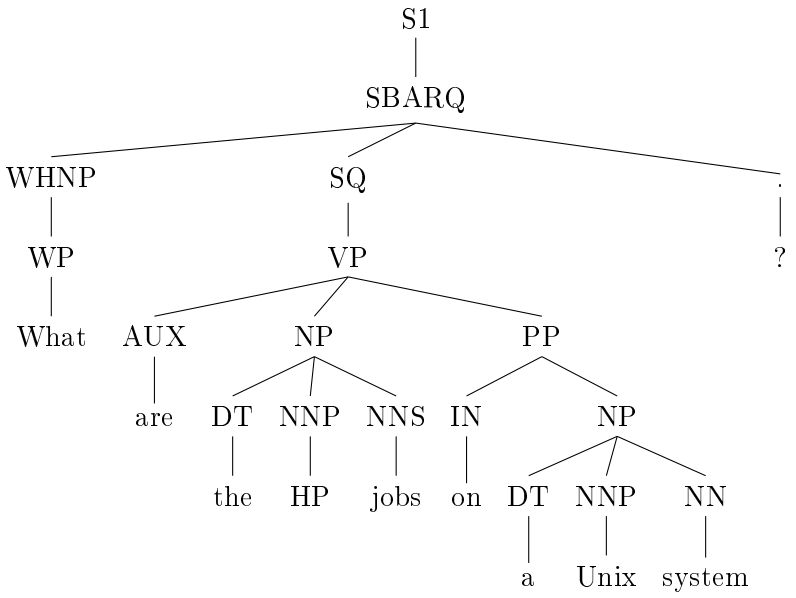
b) ambos tokens (token de relación - token de atributo o bien token de relación - token de valor) están sintácticamente asociados

4. La pregunta tiene una qword (Mapping.hasOneWhValue())

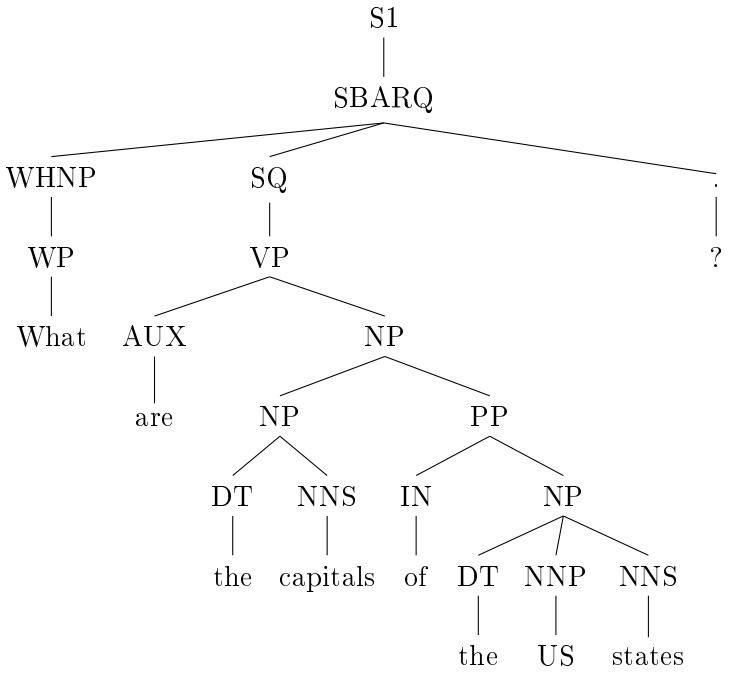
Notemos que la condición 3 implica que para cada token de relación exista algún token de atributo o de valor a) compatible y b) sintácticamente asociado. La condición a) no está verificada por el algoritmo de máximo flujo y es verificada en el método Mapping.valid()

Para verificar las condiciones de asociación sintáctica (2.b y 3.b) utilizamos la implementación oficial del árbol sintáctico de Charniak (`github.com/BLLIP/bllip-parser`), con un wrapper en java que es la clase *uba.nlp.CharniakParseTree*. Las reglas sintácticas utilizadas por Precise no están especificadas en los trabajos, por lo que creamos nuestras propias reglas tomando los ejemplos de asociaciones sintácticas dadas en los trabajos, evaluando los resultados de diferentes ejemplos y extrapolando reglas a partir de allí.

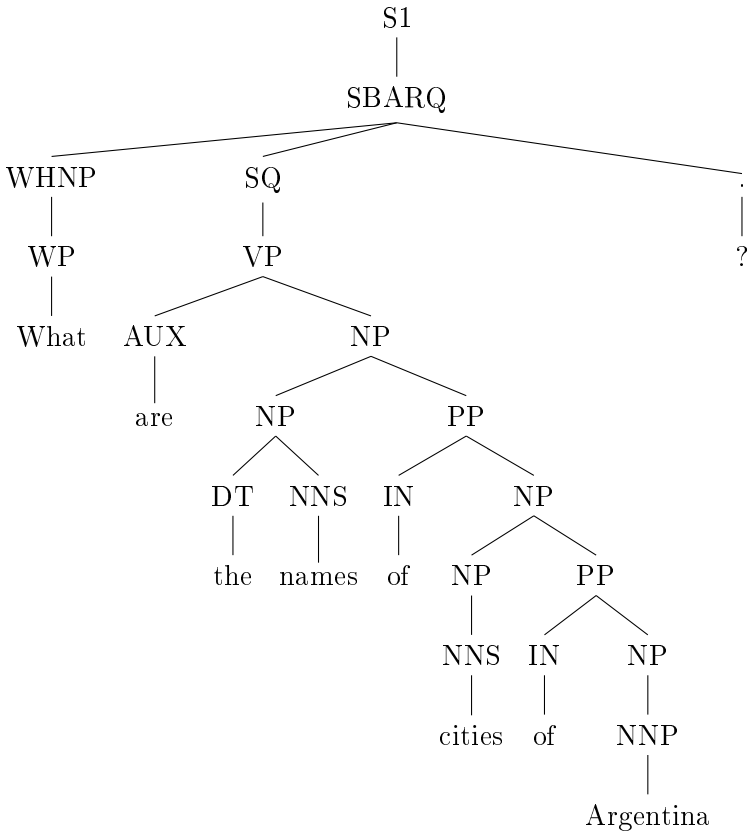
Veamos, en primer lugar, ejemplos del parse tree para tres preguntas:



"What are the HP jobs on a Unix system?"



"What are the capitals of the US states?"



"What are the names of cities of Argentina?"

En las figuras 4.2 y 4.3 pueden verse los diferentes tags para hojas y para nodos intermedios (respectivamente).

CC	Coordinating conj.	TO	infinitival *to*
CD	Cardinal number	UH	Interjection
DT	Determiner	VB	Verb, base form
EX	Existential there	VBD	Verb, past tense
FW	Foreign word	VBG	Verb, gerund/present pple
IN	Preposition	VBN	Verb, past participle
JJ	Adjective	VBP	Verb, non-3rd ps. sg. present
JJR	Adjective, comparative	VBZ	Verb, 3rd ps. sg. present
JJS	Adjective, superlative	WDT	Wh-determiner
LS	List item marker	WP	Wh-pronoun
MD	Modal	WP$	Possessive *wh*-pronoun
NN	Noun, singular or mass	WRB	Wh-adverb
NNS	Noun, plural	#	Pound sign
NNP	Proper noun, singular	$	Dollar sign
NNPS	Proper noun, plural	.	Sentence-final punctuation
PDT	Predeterminer	,	Comma
POS	Possessive ending	:	Colon, semi-colon
PRP	Personal pronoun	(	Left bracket character
PP$	Possessive pronoun	)	Right bracket character
RB	Adverb	"	Straight double quote
RBR	Adverb, comparative	`	Left open single quote
RBS	Adverb, superlative	``	Left open double quote
RP	Particle	'	Right close single quote
SYM	Symbol	''	Right close double quote

Fig. 4.2: POS tags de Penn Treebank

ADJP	Adjective phrase
ADVP	Adverb phrase
NP	Noun phrase
PP	Prepositional phrase
S	Simple declarative clause
SBAR	Subordinate clause
SBARQ	Direct question introduced by *wh*-element
SINV	Declarative sentence with subject-aux inversion
SQ	Yes/no questions and subconstituent of SBARQ excluding *wh*-element
VP	Verb phrase
WHADVP	Wh-adverb phrase
WHNP	Wh-noun phrase
WHPP	Wh-prepositional phrase
X	Constituent of unknown or uncertain category
*	"Understood" subject of infinitive or imperative
0	Zero variant of *that* in subordinate clauses
T	Trace of wh-Constituent

Fig. 4.3: Conjunto de tags para nodos intermedios del árbol sintáctico

Las reglas que construimos son las siguientes (notar que los nombres pueden denotar una direccionalidad, pero las relaciones son simétricas). Utilizamos NNx para denotar cualquier hoja sustantivo, adjetivo o adverbio (JJ, JJR, JJS, NN, NNS, NNP, NNPS, RB, RBR, RBS, CD), NN para denotar estrictamente sustantivos (NN, NNS, NNP, NNPS), Nx para denotar cualquier sintagma relacionado a estos tipos de hojas (nodos intermedios): (ADJP, ADVP, NP, S) y Wx para denotar alguna de estas hojas: WDT, WP, WP$ y WRB.

1. Hermanos:

- NNx $\Leftrightarrow$ NNx $\Leftrightarrow$ Wx si comparten el mismo el mismo padre

2. Qwords a sustantivo:
   - Una Wx dentro de un WHNP $\Rightarrow$ un NN dentro de uno o más Nx dentro de un VP dentro de un SQ con el mismo SBARQ como padre que la WHNP mencionada.
3. Sintagma nominal a sintagma preposicional:
   - NN dentro de un Nx (1) $\Rightarrow$ NNx dentro de uno o más Nx dentro de un PP con el mismo padre que el Nx marcado con el (1)
4. Sintagma nominal a sintagma verbal
   - NN dentro de un Nx (1) $\Rightarrow$ NNx dentro de uno o más Nx dentro de un VP del mismo padre que el Nx marcado con el (1)
5. Sintagma preposicional a sintagma preposicional:
   - NNx dentro de un PP (1) $\Rightarrow$ NNx dentro de uno o más Nx dentro de un PP con el mismo padre que el PP marcado con el (1)
6. Términos a tokens:
   - Dos tokens están asociados si cualquier término de uno está asociado a cualquier término de otro.

En los ejemplos presentados, los pares de términos asociados según estas reglas son los siguientes. Para el primero, (hp, jobs), (unix, system) (regla 1), (what, hp), (what, jobs) (regla 2) y, finalmente, (hp, unix), (hp, system), (job, unix), (jobs, system) (regla 3). Para el segundo, (us, states) (regla 1), (what, capitals) (regla 2) y (capitals, us) y (capitals, states) (regla 3). Para el tercero, (what, names) (regla 2), (names, cities) y (cities, argentina) (regla 3).

La definición de las reglas se hizo a prueba y error y es posible que sea simple, para alguien con mayores conocimientos lingüísticos, mejorarlas. Probablemente tenga sentido diferenciar entre sustantivos principal y modificador (Unix y system, por ejemplo) y modificar las reglas 3 y 4 para conectar solo los principales, entre otras mejoras. El sistema ofrece también la opción de no evaluar las asociaciones sintácticas en absoluto, en cuyo caso se mejora la performance pero aparecen nuevas traducciones válidas que podrían haber sido filtradas por estas condiciones, que el usuario deberá desambiguar manualmente). Además, estas reglas seguramente sean insuficientes y estén dejando afuera varias preguntas que deberían poder procesarse.

Finalmente, todos los resultados de max-flow que cumplen con las condiciones de 1 a 4 son traducciones válidas, que pasan al MappingFilter, que realiza ciertos filtrados que describiremos, de nuevo, en un título aparte.

El resultado de este módulo es una lista de Mappings (una estructura que contiene 1) una tokenización completa de la pregunta original y 2) un mapeo válido entre cada token de la misma en un elemento de la base de datos). Cada mapeo es una traducción válida de la pregunta. Si existe solo uno, entonces este mapeo se traducirá en una query que generará el resultado. Si no, corresponde al MappingFilter realizar filtrados inteligentes

de las múltiples soluciones y, en caso de que continúen existiendo múltiples soluciones, entonces se consultará al usuario qué quiso preguntar. Por otro lado, si no fue posible generar ninguna traducción válida, se retornará al usuario sin respuesta, pidiéndole que vuelva a escribir su pregunta de otro modo.

MappingFilter

La clase *uba.app.MappingFilter* es responsable, en primer lugar, de quitar las traducciones válidas repetidas y, en segundo lugar, de aplicar una serie de reglas también de filtrado. Para eliminar las traducciones repetidas, las transformamos en queries y simplemente comparamos por igualdad, ya que el generador de queries ordena las cláusulas generando, para inputs iguales pero diferentemente ordenados, la misma query. Además de este filtro, aplicamos tres reglas más, de creación propia. En primer lugar, eliminamos "semi duplicados", que consisten en atributos similares en la base de datos, en concreto, para esta base de datos, consideramos semi-duplicada a una query que consulta por Country.Name y por Country.LocalName solamente. Veremos más sobre los motivos de esto al presentar las corridas sobre las preguntas de test. La segunda y la tercera regla están basadas en la siguiente idea: en el caso en que exista más de una traducción válida en la cual la qword esté asociada con un atributo *implícito*, por como está el sistema definido, también estará asociado con todos los atributos que pueda estarlo (ver tabla 4.6 Atributos compatibles con cada Qword para las compatibilidades definidas). Esto implica que, por ejemplo, para cualquier pregunta cuya qword sea 'what' y no tenga su atributo asociado explícito, habrá 16 diferentes traducciones válidas. En los trabajos sobre los que basamos nuestro proyecto este problema no está mencionado y en los ejemplos presentados no es visible. Lo que hicimos en este caso es aplicar dos conceptos de preferencia. La segunda regla prefiere, entre los atributos implícitos posibles, aquellos cuya relación está mencionada en la pregunta (si hubiera alguna). La tercer regla señala un orden de preferencia entre los atributos mismos, una vez especificada la relación. El atributo implícito preferido está marcado en negrita en la tabla 4.6. Si tras aplicar estar reglas siguen existiendo más de una traducción válida, entonces damos al usuario el control, pidiendo por una desambiguación.

QueryGenerator

El procedimiento para generar una query a partir de una traducción válida es exactamente el mismo que se desarrollo en 3.2.2 y está implementado en el método Mapping.query().

MainProcessor

Finalmente, el punto de entrada de todo el sistema es *uba.app.MainProcessor*, que puede utilizarse especificando el parámetro -q QUESTION en cuyo caso responderá a esa pregunta y retornará el control o bien sin parámetros, en cuyo caso ingresa en un loop de preguntas-respuestas. El resultado para cada pregunta es cero, una o más queries de SQL. Agregamos la opción de desambiguación para que el usuario elija entre dos o más queries si no se pudo generar una y también una presentación de las respuestas una vez determinada la query final, pero es más bien experimental y poco sólido.

4.2.2. Ejemplos

Ejemplo 1: "Who is the head of state of Zimbabwe?"

Consideremos nuestro primer ejemplo, una corrida trivial: "Who is the head of state of Zimbabwe?".

El primer paso es el tokenizer que, según expusimos, consta de los siguientes 7 pasos:

1. Separar la pregunta en palabras, eliminar puntuaciones y pasar a lower case:
   - {who, is, the, head, of, state, of, zimbabwe}
2. Lematizar las palabras:
   - Todas las palabras ya están en su forma lematizada
3. Eliminar marcadores sintácticos:
   - {who, head, state, zimbabwe}
4. Obtener tokens para cada lema ($Lexicon.getTokens()$)
   - $getTokens(who)$ → {'who'}
   - $getTokens(head)$ → { 'head country', 'head teacher body politic', 'head body politic', 'head teacher land', 'read / write head dos', 'heading provincial', 'heading state', 'heading commonwealth', 'head teacher dos', 'head teacher country', 'read / write head body politic', 'head word land', 'head state of matter', 'read / write head provincial', 'read / write head state department', 'read / write head department of state', 'head province', 'heading body politic', 'heading res publica', 'read / write head nation', (51 más)...}
   - $getTokens(state)$ → {'heart eastern united states', 'centre eastern united states', 'central eastern united states', 'centrical eastern united states', 'midsection eastern united states', 'midriff eastern united states', 'centric eastern united states', 'center eastern united states', 'middle eastern united states', 'eye eastern united states', 'camellia state yaman', 'frederick north western united states', 'compass north western united states', 'north western united states', 'second earl of guilford western united states', 'magnetic north western united states', 'northward western united states', 'due north western united states', 'union western united states', (702 más)...}
   - $getTokens(zimbabwe)$ → {'zimbabwe', 'republic of zimbabwe', 'capital of zimbabwe'}
5. Para cada token potencial (resultado del paso anterior) verificar que todos sus lemas estén presentes en el conjunto de lemas de la pregunta original.
   - $getTokens(who)$ → {'who'}
   - $getTokens(head)$ → {'head state', 'heading state', 'head of state' }
   - $getTokens(state)$ → { 'state', 'head state', 'heading state', 'head of state'}
   - $getTokens(zimbabwe)$ → {'zimbabwe'}

6. Generar el conjunto de partes de todos los tokens hasta aquí obtenidos.
{ ∅, {'head of state', 'zimbabwe'}, {'zimbabwe', 'state'}, {'zimbabwe', 'heading state', 'state', 'head state', 'who'}, {'head of state', 'zimbabwe', 'who'}, {'head of state', 'zimbabwe', 'heading state', 'head state', 'who'}, {'head of state', 'zimbabwe', 'heading state', 'who'}, {'head of state', 'zimbabwe', 'heading state', 'state'}, {'head of state', 'zimbabwe', 'heading state', 'state', 'who'}, {'who'}, {'head of state', 'heading state', 'head state'}, {'head of state'}, {'state', 'who'}, {'head of state', 'zimbabwe', 'heading state', 'head state'}, {'heading state', 'state', 'who'}, {'heading state', 'state'}, {'heading state', 'who'}, {'head of state', 'who'}, {'head of state', 'zimbabwe', 'state', 'who'}, {'head of state', 'state', 'head state'}, {'zimbabwe', 'heading state', 'state', 'who'}, {'zimbabwe', 'head state', 'who'}, {'zimbabwe', 'heading state', 'who'}, {'head of state', 'heading state', 'who'}, {'state', 'head state', 'who'}, {'head state'}, {'heading state', 'state', 'head state'}, {'head of state', 'state', 'who'}, {'zimbabwe'}, {'state', 'head state'}, {'state'}, {'head of state', 'heading state', 'state'}, {'zimbabwe', 'state', 'who'}, {'zimbabwe', 'state', 'head state'}, {'head of state', 'zimbabwe', 'heading state', 'state', 'head state'}, {'head of state', 'zimbabwe', 'heading state', 'state', 'head state', 'who'}, {'head of state', 'zimbabwe', 'state', 'head state', 'who'}, {'heading state', 'head state', 'who'}, {'heading state'}, {'head of state', 'zimbabwe', 'state'}, {'head of state', 'zimbabwe', 'head state', 'who'}, {'head of state', 'heading state', 'state', 'who'}, {'head of state', 'zimbabwe', 'state', 'head state'}, {'head of state', 'zimbabwe', 'heading state'}, {'head of state', 'zimbabwe', 'head state'}, {'head of state', 'heading state', 'state', 'head state'}, {'head state', 'who'}, {'head of state', 'state'}, {'head of state', 'heading state', 'head state', 'who'}, {'head of state', 'heading state', 'state', 'head state', 'who'}, {'zimbabwe', 'heading state'}, {'head of state', 'head state'}, {'heading state', 'state', 'head state', 'who'}, {'zimbabwe', 'state', 'head state', 'who'}, {'heading state', 'head state'}, {'head of state', 'head state', 'who'}, {'head of state', 'heading state'}, {'zimbabwe', 'heading state', 'state'}, {'zimbabwe', 'heading state', 'head state', 'who'}, {'zimbabwe', 'who'}, {'zimbabwe', 'head state'}, {'zimbabwe', 'heading state', 'head state'}, {'zimbabwe', 'heading state', 'state', 'head state'}, {'head of state', 'state', 'head state', 'who'} }

7. Para cada uno de estos subconjuntos, verificar 1) que los lemas de sus tokens cubran completamente los lemas significativos de la pregunta original y 2) que no haya lemas repetidos entre los tokens.

   - Los lemas significativos son los obtenidos en el bullet 4: {who, head, state, zimbabwe}
   - Las tokenizaciones que los cubren sin repetir lemas son: {'who', 'head state', 'zimbabwe'}, {'who', 'heading state', 'zimbabwe'} y {'who', 'head of state', 'zimbabwe'}, con lo que queda eliminado el token 'state' y las repeticiones de los tokens que refieren tanto a 'head' como a 'state', quedando solo las diferentes formulaciones sinónimas del mismo elemento de la base de datos.

Es decir, el Tokenizer encontró tres conjuntos de tokens completos para la pregunta, Tokenizer.generateCompleteTokenizations("Who is the head of state of Zimbabwe?") → [{ 'who', 'head state', 'zimbabwe'}, {'who', 'heading state', 'zimbabwe'}, {'who', 'head of state', 'zimbabwe'}].

El siguiente paso es el Matcher, que construye el grafo de atributos y valores para cada uno de las tokenizaciones completas y decide si existe un conjunto de elementos

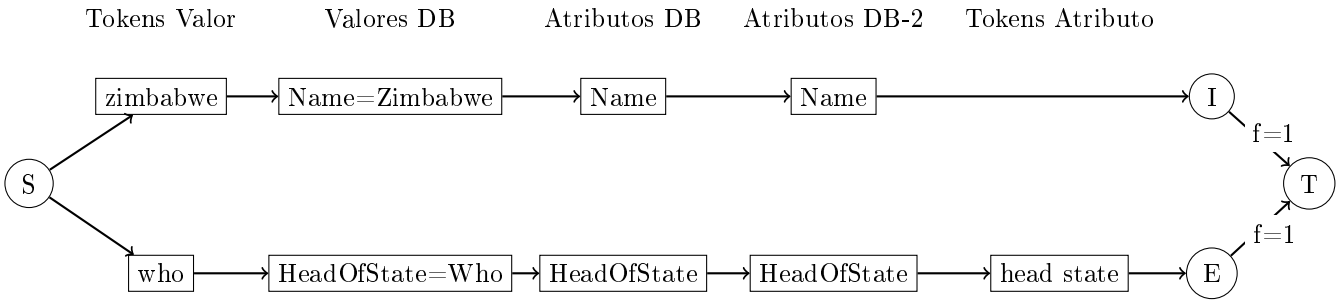


Fig. 4.4: Grafo de atributo-valor para la primer tokenización completa de "Who is the head of state of Zimbabwe?"

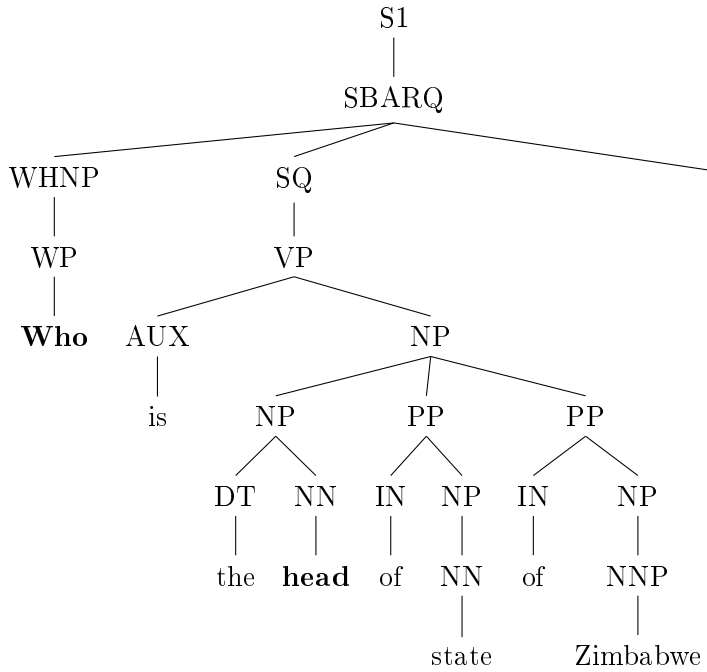
de la base de datos que permita establecer la traducción. Presentamos aquí solamente el grafo para la primer tokenización completa, que puede verse en la figura 4.4. Las otras tokenizaciones tienen un grafo análogo y generaran el mismo resultado, que será filtrado luego por el EquivalenceChecker. Notar que no es esta repetición de tokenizaciones lo que filtra el Matcher, sino la ambigüedad introducida por un token que puede referir a dos o más elementos de la base de datos (este problema no ocurre en este primer ejemplo). Para construirlo, primero, se consulta al Lexicón por el tipo y los elementos asociados de cada token mediante el servicio *getMatchingElements* como se puede ver en la tabla 4.8.

Token	Tipos	Elementos	Detalle
zimbabwe	Value	Zimbabwe	Valor del Atributo Name de la tabla Country
head state, head of state, heading state	Attribute	HeadOfState	Atributo de la tabla Country
who	WhValue/Value	Who	Q-valor compatible con: Atributo HeadOfState de la tabla Country

Tab. 4.8: Resultados de *getMatchingElements* para "Who is the head of state of Zimbabwe?"

En este primer ejemplo el grafo no desambigua ninguna relación token-elemento ya que no hay ningún token con más de un elemento asociado, por lo que el grafo de max-flow no tiene ningún atractivo especial y es construido siguiendo el procedimiento reseñado en 3.2.2 Al ejecutar el algoritmo de max-flow sobre el grafo, se obtiene que el flujo máximo es 2 y los pares generados son (Who, Country.HeadOfState), (Country.Name, Zimbabwe), sin generar ningún otro flujo alternativo como se ve en la figura 4.4.

El próximo paso es evaluar que los términos del par de tokens ('who', 'head state') estén sintácticamente asociados en el parse tree generado por el algoritmo de Charniak según nuestras reglas. El árbol sintáctico para la pregunta es el siguiente:



"Who is the head of state of Zimbabwe?"

La regla 2 (q-word a sustantivo) vincula los términos 'who' con 'head', mientras que la 6 (términos a tokens), asocia los tokens 'who' con 'head state', cumpliendo las condiciones sintácticas requeridas por la definición de traducción válida. Por su parte, el atributo Name es implícito, lo que significa que no posee ningún token asociado y, por lo tanto, no debe cumplir la regla de traducciones válidas. Además, como no hay ninguna relación mencionada explícitamente, los tokens cubren todos los términos significativos de la pregunta y no hay ningún término repetido entre los tokens, esto significa que tenemos una (única) traducción válida para esta tokenización completa, constando de los siguiente elementos de la base de datos apareados: (Atributo Country.Name, Valor Country.Name=Zimbabwe), (Atributo Country.HeadOfState, Q-Valor Who).

Las otras dos tokenizaciones generan la misma traducción y sus repeticiones son eliminadas por el *EquivalenceChecker*.

Finalmente, esta única traducción pasa al *QueryGenerator* que, aplicando las reglas descriptas en 3.2.2, genera la siguiente consulta SQL:

SELECT Country.HeadOfState **FROM** Country **WHERE** Country.Name='Zimbabwe'

Cuyo resultado es "Robert G. Mugabe".

Ejemplo 2: "What caribbean countries are also considered north american?"

Consideremos nuestro segundo ejemplo: "What caribbean countries are also considered north american?".

El primer paso es el tokenizer que, según expusimos, consta de los siguientes 7 pasos:

1. Separar la pregunta en palabras, eliminar puntuaciones y pasar a lower case:

- {what, caribbean, countries, are, also, considered, north, american}

2. Lematizar las palabras:
   - {what, caribbean, country, is, also, consider, north, american}
3. Eliminar marcadores sintácticos:
   - {what, caribbean, country, north, american}
4. Obtener tokens para cada lema (*Lexicon.getTokens*())
   - *getTokens*(*what*) → {'what'}
   - *getTokens*(*caribbean*) → {'caribbean sea', 'caribbean', 'micronesia caribbean sea', 'federated states of micronesia caribbean', 'tt caribbean sea', 'micronesia / caribbean', 'tt caribbean', 'micronesia caribbean', 'federated states of micronesia caribbean sea'}
   - *getTokens*(*country*) → {'country codification', 'country computer code', 'country codify', 'country code', 'baltic country', 'baltic language country', 'baltic sea country', 'north germanic language country', 'norse country', 'nordic country', 'scandinavian country', 'scandinavian language country', 'north germanic country', 'independent christian church country', 'free lance christian church country', 'self - employed person church building country', 'freelance church country', 'independent church service country', 'freelancer churchly country', 'free - lance churchly country', (149 más)...}
   - *getTokens*(*north*) → {'jabir aluminum ahmad aluminum jabir heart of dixie north borneo', 'jabir alabama ahmad camellia state jabir aluminum north borneo', 'jabir aluminize ahmad aluminise jabir aluminous north borneo', 'jabir aluminium ahmad camellia state jabir aluminize north borneo', 'jabir aluminize ahmad camellia state jabir aluminise north borneo', 'jabir al ahmad al jabir aluminium north borneo', 'jabir aluminous ahmad al jabir aluminise north borneo', 'jabir heart of dixie ahmad aluminous jabir camellia state north borneo', 'jabir aluminous ahmad camellia state jabir aluminous north borneo', 'jabir aluminium ahmad al jabir aluminize north borneo', 'jabir al ahmad aluminize jabir alabama north borneo', 'jabir aluminous ahmad al jabir heart of dixie north borneo', 'jabir aluminum ahmad aluminous jabir aluminum north borneo', 'jabir aluminum ahmad alabama jabir al north borneo', 'jabir heart of dixie ahmad aluminise jabir aluminous north borneo', 'jabir atomic number 13 ahmad aluminum jabir atomic number 13 north borneo', 'jabir aluminise ahmad alabama jabir aluminum north borneo', 'jabir aluminum ahmad aluminize jabir aluminize north borneo', 'jabir aluminium ahmad al jabir aluminous north borneo', 'jabir aluminous ahmad aluminium jabir camellia state north borneo', (2237 más)...}
   - *getTokens*(*american*) → { 'american virgin islands', 'southward american', 'south american', 'due south american', 's american', 'dixieland american', 'dixie american', 'confederate states of america american', 'confederacy american', 'confederate states american', 'northward american', 'due north american', 'frederick north american', 'n american', 'second earl of guilford american', 'union american', 'north american', 'compass north american', 'magnetic north american', 'american - indian language spoken language', (169 más)...}

5. Para cada token potencial (resultado del paso anterior) verificar que todos sus lemas estén presentes en el conjunto de lemas de la pregunta original.

   - $getTokens(what) \rightarrow$ {'what'}
   - $getTokens(caribbean) \rightarrow$ {'caribbean'}
   - $getTokens(country) \rightarrow$ {'country'}
   - $getTokens(north) \rightarrow$ {'north american'}
   - $getTokens(american) \rightarrow$ {'north american', 'american'}

6. Generar el conjunto de partes de todos los tokens hasta aquí obtenidos.
   { ∅, {'north american'}, {'what', 'american', 'country'}, {'north american', 'what', 'american', 'country'}, {'american', 'caribbean'}, {'north american', 'what', 'american'}, {'what', 'country'}, {'north american', 'what', 'american', 'caribbean', 'country'}, {'caribbean'}, {'north american', 'what', 'caribbean'}, {'north american', 'american', 'country'}, {'north american', 'what'}, {'american'}, {'north american', 'american', 'caribbean', 'country'}, {'north american', 'what', 'caribbean', 'country'}, {'american', 'country'}, {'what', 'american', 'caribbean', 'country'}, {'north american', 'country'}, {'what', 'american'}, {'what', 'caribbean', 'country'}, {'american', 'caribbean', 'country'}, {'north american', 'caribbean'}, {'country'}, {'north american', 'what', 'american', 'caribbean'}, {'north american', 'caribbean', 'country'}, {'north american', 'what', 'country'}, {'what', 'caribbean'}, {'north american', 'american', 'caribbean'}, {'what'}, {'north american', 'american'}, {'what', 'american', 'caribbean'}, {'caribbean', 'country'}}

7. Para cada uno de estos subconjuntos, verificar 1) que los lemas de sus tokens cubran completamente los lemas significativos de la pregunta original y 2) que no haya lemas repetidos entre los tokens.

   - Los lemas significativos son los obtenidos en el bullet 4: {what, caribbean, country, north, american}
   - Solo una tokenización los cubren sin repetir lemas: {'north american', 'what', 'caribbean', 'country'}, con lo que queda eliminado el token 'american'.

Es decir, el Tokenizer encontró un conjunto de tokens completo para la pregunta, Tokenizer.generateCompleteTokenizations("What caribbean countries are also considered north american?") → [{what, caribbean, country, north american}].

El siguiente paso es el grafo del Matcher, que en este caso sí desambigua un token de valor. Como puede verse en 4.9, al token 'caribbean' le corresponden dos posibles valores: valor del atributo Region de la tabla Country y también valor del atributo Continent de la tabla Country.

En este ejemplo el grafo decide entre dos interpretaciones posibles (Ver grafo 4.5) para el token 'caribbean', que puede ser un valor tanto del atributo Region como del atributo Continent. Teniendo presente el token 'north american', que solo puede tomar el atributo Continent, para maximizar el flujo el algoritmo asigna 'caribean' a Region, obteniendo un flujo de 3, que es el máximo, eliminando así la posibilidad de relacionar este token de valor con el atributo Continent.

Token	Tipos	Elementos	Detalle
caribbean	Value	North America	Valor de los Atributo Region y Continent de la tabla Country
north american	Value	Caribbean	Valor del Atributo Region de la tabla Country
country	Relation	Country	Relación Country
what	WhValue / Value	What	Q-valor compatible con: Atributos GovernmentForm, Population, GNP, Name, Capital, LocalName, Continent, Region, Code, LifeExpectancy y SurfaceArea de la tabla Country, District, Population y Name de la tabla City e ISOfficial y Percentage de la tabla Language

Tab. 4.9: Resultados de *getMatchingElements* para "What caribbean countries are also considered north american?"

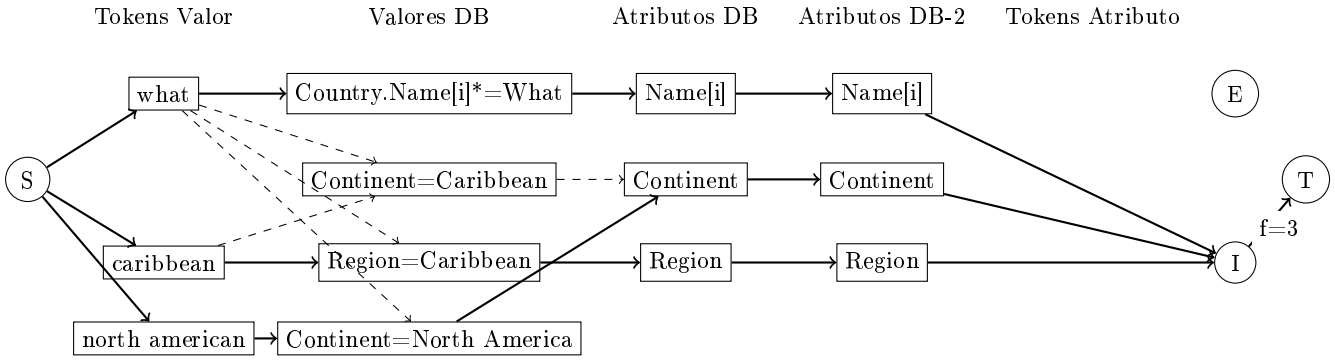
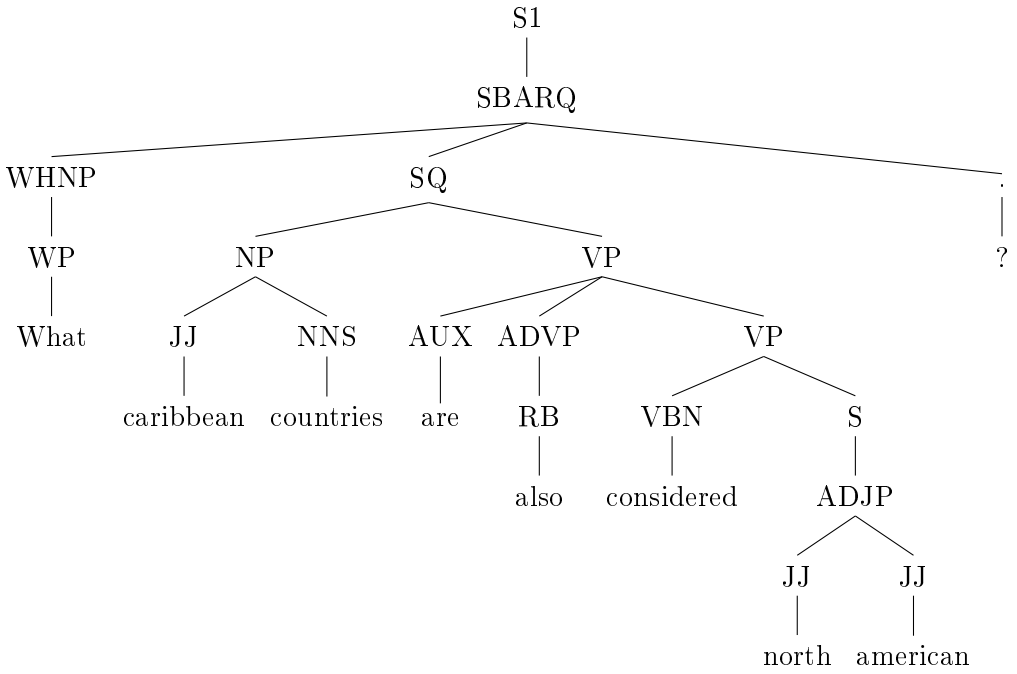


Fig. 4.5: Grafo de atributo-valor para la primer tokenización completa de "What caribbean countries are also considered north american?"

Como no hay tokens de atributo explícitos, solo queda verificar la regla relacionada con los tokens de relación: Country es compatible con todos los tokens de valor que aparecen en la pregunta y, en particular, está sintácticamente asociada a what (regla 2), por lo que las condiciones de traducción válida se cumplen.

Esto nos da una única traducción válida, pero en realidad el *Matcher* genera 16, una por cada atributo implícito posible compatible con la q-word 'What'. Notar que Country.Name no tiene ninguna particularidad especial, al menos no para el *Matcher*, sobre las demás posibilidades listadas en 4.9. Pero de esto se ocupa el *MappingFilter* con las tres reglas especiales que agregamos. Dado que la relación *country* está presente en la pregunta y no así las otras dos, aplicando las reglas de preferencia de atributos implícitos nos quedamos solo con la asociación Country.Name=What.



Charniak parse tree para "What caribbean countries are also considered north american?"

Esta única traducción pasa al *QueryGenerator* que la traduce unívocamente en una query, aplicando las reglas descriptas en 3.2.2: "**SELECT DISTINCT** Country.Name **FROM** Country **WHERE** Country.Continent = 'North America' **AND** Country.Region = 'Caribbean"' cuyo resultado es la siguiente lista:

- Aruba
- Anguilla
- Netherlands Antilles
- Antigua and Barbuda
- Bahamas
- Barbados
- Cuba
- Cayman Islands
- Dominica
- Dominican Republic
- Guadeloupe
- Grenada
- Haiti
- Jamaica

- Saint Kitts and Nevis
- Saint Lucia
- Montserrat
- Martinique
- Puerto Rico
- Turks and Caicos Islands
- Trinidad and Tobago
- Saint Vincent and the Grenadines
- Virgin Islands, British
- Virgin Islands, U.S.

4.2.3. Corridas

Si bien dada la especificidad del dominio y la especificidad del sistema es imposible comparar resultados con otros sistemas, escribimos a mano un set de preguntas de prueba de 200 preguntas que ejecutamos sobre nuestro sistema para observar resultados.

Los resultados de la primera corrida pueden observarse en la tabla 4.10. 129 dieron resultados correctos: 91 dieron una única query, 27 una selección entre queries posibles (es decir, eran ambiguas) siendo esta ambigüedad razonable, 11 no generaron ninguna query, siendo esto también razonable. Respecto a las restantes 71, debemos distinguir casos.

En primer lugar, el sistema falló en obtener respuestas de 32 preguntas por razones técnicas triviales que se corrigieron de una manera simple. La primer de estas razones es que no se pudo generar una tokenización completa por faltar alguno de los términos de la pregunta en el lexicón y en el diccionario de stopwords. Para solucionar esto agregamos algunas palabras al diccionario de stopwords a mano, aunque, claro, este procedimiento no escala. Es interesante notar en este punto que, en los trabajos sobre los que nos basamos, no están definidas las reglas para generar sinónimos a partir de Wordnet (simplemente se menciona que se usa Wordnet, pero Wordnet tiene diferentes maneras de generar sinónimos a partir de una palabra) y tampoco está definido ningún mecanismo para generar el diccionario de stopwords (solo se menciona que es independiente de la DB y que está hecho a mano). Así, una formalización teórica y una evaluación de diferentes mecanismos de generación de sinónimos y de stopwords quedará como trabajo futuro. Ambos, junto con la definición de reglas de análisis sintáctico serían las tres áreas más abiertas y disponibles para formalizar en este modelo teórico. Volviendo a los resultados: resolvimos este primer caso agregando al diccionario de stopwords algunas palabras.

En segundo lugar, tenemos que la qword What no estaba relacionada con algunos atributos (como Language e IndependenceYear). Habíamos restringido la compatibilidad de What con algunos atributos para intentar minimizar el hecho de que What parece ser compatible con todos los atributos, dependiendo el fraseo de la pregunta.

En una segunda corrida, agregamos las relaciones de compatibilidad de What con estos atributos. Con estos dos cambios, las 32 preguntas que estamos considerando en este

párrafo dieron como resultados 23 respuestas correctas exactas y las 9 restantes son 6 inherentemente ambiguas, 3 ambiguas por un problema del modelo que mencionaremos en el próximo párrafo.

Total	Categoría
91	Respuesta exacta
27	Ambiguas inherentemente + pedido de reformulación
11	Sin respuesta (razonable, fuera de scope)
32	Fallas simples en stopwords o en lexicón
10	Ambiguas por problemas de sinonimia en el modelo
4	Ambiguas. Problemas del lexicón.
6	El analizador sintáctico no encontró asociación.
19	Problemáticas. Análisis más detallado en texto.
200	Total

Tab. 4.10: Resultados de la primera corrida sobre las preguntas de test

Las 10 notadas en la tabla como 'Ambiguas por modelo' son problemas de ambigüedad que refieren a cierta redundancia entre el modelo de la DB y los sinónimos. Consideremos la pregunta 'When did Yugoslavia become independent?', que el sistema reconoce como ambigua y ofrece las dos siguientes opciones:

- SELECT DISTINCT Country.IndependenceYear FROM Country WHERE Country.LocalName = 'Jugoslavija'
- SELECT DISTINCT Country.IndependenceYear FROM Country WHERE Country.Name = 'Yugoslavia'

El problema aquí es que el lexicón generó el token Yugoslavia como correspondiente tanto al valor 'Yugoslavia' para Country.Name y para el valor 'Jugoslavija' para Country.LocalName. Este problema ocurre también con Country.Code (por ejemplo, United States es un token para USA y para United States). No vemos una solución general, más allá de establecer que, en caso de ser las diferentes opciones una variación de la misma query sobre Country.LocalName, Country.Name y Country.Code, entonces el *MappingFilter* se queda solo con aquella que menciona Country.Name (esta modificación ya fue presentada en la sección 4.2.1 Código). Con estos cambios, todas las preguntas en esta categoría se convierten en exactas (10 de la primer corrida y 3 que en la primera corrida tenían un problema con el tokenizer que al ser corregido generó este problema).

Con estas dos correcciones, realizamos la segunda corrida, cuyos resultados pueden observarse en la tabla 4.11

Las 4 distinguidas en ambas tablas como "Ambiguas. Problemas del lexicón." refieren a sinónimos demasiado amplios dados a ciertos elementos. En concreto, estas 4 son ambiguas porque el lexicón tiene una correspondencia entre el token 'republic' y el valor 'Commonwealth of the US'. Esto se solucionaría con la supervisión del lexicón o, más en general, mejorando el módulo.

Total	Categoría
126	Respuesta exacta
34	Ambiguas inherentemente + pedido de reformulación
11	Sin respuesta (razonable, fuera de scope)
4	Ambiguas. Problemas del lexicón.
6	El analizador sintáctico no encontró asociación.
19	Problemáticas. Análisis más detallado en texto.
200	Total

Tab. 4.11: Resultados de la segunda corrida sobre las preguntas de test

Las 6 que fallaron por el análisis sintáctico refieren, en efecto, a problemas de analizador sintáctico, que, como ya mencionamos, tiene enormes limitaciones ya que no fue definido por un experto sino a prueba y error. Desactivando el análisis sintáctico, estas 6 preguntas se convierten en exactas.

Sobre las 20 problemáticas pasaremos revista a continuación.

Hay seis preguntas con sentido semántico dentro del espacio de sentido del sistema, pero cuya forma no tiene ningún flujo para especificarse. Por ejemplo, es el caso de 'Who is Islam Karimov?', que como respuesta debería tener "ser el líder de X país", es decir, devolver el nombre del atributo tal que existe un valor para una fila, flujo que no existe en este modelo. Otra pregunta así es 'What countries of the British Islands are independent?' (curiosamente escrita poco tiempo antes del brexit), que no logra encontrar una respuesta porque no sabe decidir que 'Tener un valor no nulo en Independence Year' implica 'ser independiente'. 'What is the population of the capital of Germany?' también falla, debido a que no soportamos más de un join y eliminamos la navegabilidad entre capital de un país y su fila en la tabla ciudades.

'How many people live in Kabul?' genera, para 'live' diferentes tokens relacionados con LifeExpectancy (getTokens('live') stem=live–>['live anticipation', 'live expectancy', 'living expect', 'living anticipation', 'live expect', 'live expectant', 'living expectant', 'living expectancy']) pero luego la otra palabra no aparece (con razón). Lo razonable aquí sería que live sea a la vez una palabra de un token y una stopword. El modelo no soporta esto, por lo que no encuentra tokenización ni respuesta para esta pregunta.

Para 'What is the surface of Afghanistan?' no encuentra ningún token para surface, cuando debería encontrar 'surface area'. Esto podría resolverse agregando a mano al TokenAugmenter el término surface para surface area.

Para 'What languages do the frenchs speak?', el término french se asoció al lenguaje y no a 'los habitantes del país France' sin poder generar una respuesta.

Dos se corrigieron con las correcciones técnicas.

'What is the population of the district named Qandahar?' es interesante porque 'name' funciona como token y hace que el sistema no logre encontrar un máximo flujo

Para 'In which region is Rome?', 'Who is the head of state of Bielorussia?', 'What is the population of the country lead by Hugo Chavez?' el lexicón no disponía de tokens para Rome ni para Bielorussia ni para Hugo Chavez (pero si para Roma, Belarus y Hugo Chávez Frías)

Por otro lado, el sistema no soporta diferentes preguntas, por ejemplo: no soporta ORS y una pregunta era 'What cities are located in Italy or in Germany?', no soporta manejo de números (operadores mayor y menor, por ejemplo) y una pregunta era 'Which countries

have a surface area bigger than 65610.00?'.

Como conclusión tras analizar estas 20 preguntas problemáticas podemos decir que el modelo tiene limitaciones importantes en el nivel de la definición concreta del lexicón, el diccionario de stopwords y el analizador sintáctico. Sin embargo, esto no debe quitar el foco de atención de la cantidad de respuestas dadas correctamente, las inherentemente ambiguas interpretadas como tales, las imposibles de procesar que son correctamente rechazadas y, finalmente, aquellas que mostraron límites técnicos triviales y de fácil solución, que sugieren que un trabajo más exhaustivo y puntilloso en la definición y supervisión de los diferentes conjuntos de palabras disponibles pueden redundar en un incremente sustancial sencillo de la performance del sistema.

Total	Categoría
126	Respuesta exacta
38	Ambiguas
36	Sin respuesta
200	Total

Tab. 4.12: Resultados sobre las preguntas de test (con correcciones)

4.2.4. Conclusiones, limitaciones y trabajo futuro

El sistema implementado consiste en un mecanismo para traducir preguntas simples formuladas en inglés a las consultas SQL correspondientes sobre la base de datos World, permitiendo reconocer cuando una pregunta no es semánticamente tratable. Si bien es verdad que la implementación require de desarrollo humano, los tiempos de adaptación a diferentes bases de datos, con el código base existente, no son tan grandes. El potencial de los sistemas de dominio cerrado basados en traducciones a SQL tiene un interés acotado ya que su concepto es, en realidad, un agregado cosmético para una base de datos ya existente y ya consultable. Dicho de otro modo: la información sobre la que trabajamos en este caso ya está ordenada y estructuradas y es accesible, consultable, mediante un lenguaje: el SQL, que, con sus limitaciones y su tiempo de aprendizaje, es un lenguaje comprensible por humanos. Con esto en mente, el agregado que introduce un módulo de question answering sobre esta información está en proponer una interfaz más intuitiva. Este agregado, técnicamente menor, incorpora, sin embargo, un aporte clave, que es la accesibilidad de las bases de datos a un público masivo, conocedor de su idioma (inglés en este caso, pero en general, de algún idioma) pero no del SQL. Como ya mencionamos, esta posibilidad, en combinación con un procesador de voz, tiene un interés esencial.

Por otro lado, creemos que es clave destacar una hipótesis de trabajo de Popescu que nos parece fundamental a la hora de enfocar el trabajo sobre el área de question answering sobre dominios cerrados y estructurados. Esta hipótesis es: no es necesario entender todas las preguntas posibles formulables en lenguaje natural, ya que estas pueden tener una complejidad arbitrariamente grande; basta con limitarse a definir un conjunto de preguntas simples tratables que, sin embargo, permita una amplia expresividad y, más aún, que sea, en la práctica, la forma más natural y directa de formular esta pregunta. Creemos que esta hipótesis de trabajo es muy fecunda y prolífica, ya que elimina de raíz el preconcepto de que

es imposible traducir el lenguaje natural informal en un lenguaje técnico formal. Quizás sea imposible computacionalmente capturar el sentido de la frase "explicar con palabras de este mundo que partió de mí un barco llevándome" de Alejandra Pizarnik, pero definitivamente es computable traducir la pregunta "¿En qué año murió Alejandra Pizarnik?" a una consulta SQL de la forma "**SELECT** dead_year **FROM** writers **WHERE** name = 'Alejandra Pizarnik' ". Si bien el ejemplo es un poco excesivo en su contraste, consideramos que este exceso hace más patente la importancia de esta hipótesis de trabajo al señalar, digamos, *zonas* computacionales del lenguaje, en particular, del conjuntos de las preguntas.

El sistema que presentamos en nuestro trabajo incorpora todos los conceptos del marco teórico propuesto por Popescu, permitiendo distinguir preguntas semánticamente tratables de aquellas que no lo son. Con ciertas limitaciones que presentamos durante la exposición y que en breve mencionaremos resumidamente, es un sistema funcional que sirve como base de trabajo para proyectos futuros de mayor solidez y envergadura y, por qué no, para un trabajo conjunto con el área de procesamiento de voz del Departamento.

Las diferentes limitaciones y mejoras de nuestra implementación fueron mencionadas durante la exposición de la misma que realizamos a lo largo de este capítulo: a continuación las presentamos juntas para facilitar su acceso.

En un nivel técnico, habría una enorme mejora de performance implementando el lexicón sobre una base de datos relacional. Los archivos json sirven para un modelo de escala pequeña, pero no permiten escalar ya que genera un tiempo de carga innecesario, así como también uso de memoria. Existen también una serie de clases con datos dentro del código que deberían separar archivos de configuración de códigos. Estas son TokenAugmenter, WhGenerator y Config. Estas clases requieren editar el código para aplicarlo sobre una nueva base de datos, cuando en realidad solo es necesario editar definiciones.

Más allá de estas mejoras técnicas, las siguientes mejoras permitirían incrementar la usabilidad y la eficacia del sistema:

- En el Lexicón, no generar sinónimos para nombres propios. Wordnet posee sinónimos para, por ejemplo *Hugo* (Victor Hugo, etc) introduciendo ruido innecesario y fácilmente eliminable.

- En el Lexicón, agregar un desambiguador de sentidos para los sinónimos de los tokens. Este es un problema complejo de NLP e integrar una solución del mismo dentro del modelo ya definido podría redundar en un avance teórico. Desde un nivel más técnico y con menor complejidad, también sería posible implementar un desambiguador supervisado.

- Soportar diferentes paths para aparear (*joinear*) tablas.

- Soportar tokens multi tipados en el Matcher.

- Soportar desambiguación de relaciones en el Matcher.

- Interfaz con el usuario: como mencionamos, el resultado es una query y la desambiguación es entre dos o más queries. Que el resultado sea una respuesta presentada en un formato amigable y que la desambiguación sea, también, presentada en un formato amigable es una mejora de usabilidad necesaria para utilizar el sistema en la vida real.

- CharniakParseTree: evaluando con un lingüista profesional las reglas utilizadas para definir la *asociación sintáctica* seguramente se obtengan reglas más eficaces.

Con esta lista terminamos la presentación de nuestro sistema closed domain en inglés. En el próximo capítulo daremos vuelta la página y presentaremos nuestro sistema open domain multilingüe.

5. IMPLEMENTACIÓN SOBRE QANUS

En este capítulo veremos la implementación del modelo de question answering multilingüe y diferentes evaluaciones realizadas utilizando como guía dos tareas monolingües de la competencia QA@CLEF '07, una para español y otra para portugués. El sistema está basado en un framework académico de código abierto llamado Qanus. Este framework está empaquetado con un sistema de QA simple, QA-sys, que resuelve ejercicios de la competencia Trec '07 con soporte para inglés. Nosotros adaptamos este sistema para utilizar como librería de procesamiento de lenguajes a Freeling, generando un sistema baseline multilingüe. A este sistema le agregamos features reseñados en 3.2 Literatura y sistemas y realizamos diferente mediciones sobre las instancias en español y portugués sobre los ya mencionados ejercicios de CLEF '07.

La estructura de este capítulo es la siguiente: en 5.1 Ejercicio de CLEF se describe con detalle el panorama general de tareas de question answering en la competencia CLEF '07, haciendo énfasis en las tareas seleccionadas, en los corpora y las preguntas disponibles y en las decisiones tomadas a la hora de elegir estas tareas; en 5.2 Sistema reseñamos la implementación de nuestro sistema, presentando el sistema baseline de basado en Qanus (5.2.1), las adaptaciones multilingües y demás mejoras realizadas (5.2.2), las diferentes corridas que realizamos para evaluarlo (5.2.3) y, finalmente, las conclusiones, los límites y potenciales mejoras del sistema (5.2.4).

5.1. Ejercicio de CLEF

Tras investigar distintas competencias y métodos de evaluación, optamos por utilizar como guía de trabajo dos ejercicios de la tarea principal de question answering de la competencia CLEF de 2007. Principalmente por contar con un subconjunto de ejercicios bastante adecuados para nuestra tesis y disponer de un corpus de datos libre y disponible en el acto para evaluar una parte importante de las funcionalidades implementadas en nuestro proyecto.

Como mencionamos en 3.1.1 Historia y Competencias, CLEF (de *Cross-Language Evaluation Forum*) es una organización que busca fomentar la investigación en sistemas de information retrieval cross-language. En particular, una vez por año CLEF lanza una competencia de Question Answering multilingüe, con diferentes corpora y diferentes tipos de ejercicios. Estas competencia permiten obtener un patrón estándar de comparación entre distintos desarrollos y una idea general del estado de arte alcanzado en los diferentes idiomas por la diferentes actores académicos y comerciales. Por lo demás, CLEF es el principal organizador de conferencias de question answering multilingüe para idiomas europeos y la tarea principal de '07 presentaba una complejidad adecuada para el scope de esta tesis.

La competencia de 2007 es la quinta campaña de QA multilingüe de CLEF, siendo la primera en 2003.

La tarea estándar principal de question answering incorpora algunas particularidades que la diferencian de anteriores campañas: los temas o tópicos, la co-referencia y el uso de wikipedia como corpus de datos parcial. Además de la tarea principal se presentaron tres tareas más: Answer Validation Exercise (AVE) y QUAST, question answering in speech

transcription y Question Answering Real Time, con métricas de performance temporales. Como nota general, es importante notar que las tareas fueron de una complejidad elevada y en comparación con los resultados del año anterior, la mejor precisión general bajó del 49 % al 41.75 % para tareas multi-lingües, mientras que, más significativamente, bajó de 68 % a 54 % en tareas monolingües.

5.1.1. Tareas

Utilizamos como guía dos ejercicios de la tarea principal de question answering de '07 por varias razones (Ver [Clef, 2007a] y [Clef, 2007b] para un detalle exhaustivo de la conferencia en cuestión). Para esta tarea estaban disponibles en acto corpora, inputs de prueba y resultados esperados para una experimentación multilingüe que abarque tanto inglés como español y portugués. Además, estos ejercicios incorporaban, por única vez, diferentes wikipedias como parte del corpus de datos, lo cual resulta de por sí atractivo.

La tarea principal de QA de CLEF '07 ofrece dos grandes grupos de ejercicios:

- Mono-lingual: donde el idioma de la pregunta y el idioma de la fuente de información son el mismo
- Cross-lingual: donde el idioma de la pregunta y el idioma de la fuente de información difieren

En ambos grupos, los sistemas reciben un set de 200 preguntas - que pueden ser de hechos o eventos (factoid), definiciones de personas, cosas u organizaciones (definition), listas de personas, objetos o datos (list) - y se exige devolver una respuesta exacta, donde exacta significa sin más información de la mínima necesaria. Siguiendo el ejemplo de Trec, los ejercicios de este año contienen topics, i.e. clusters de preguntas relacionadas con un mismo tema y con posibilidad de co-referencia entre las preguntas del grupo. Ni el tipo de la pregunta ni el tópico es dado a los participantes.

La respuesta tiene que estar soportada por el id del documento del cual se extrajo y por la porción de texto que proveyó contexto suficiente para soportar la corrección de la respuesta. Los "textos de soporte" pueden venir de diferentes partes de documentos relevantes y deben sumar un máximo de 700 bytes. No hubo restricciones sobre la longitud del string de respuesta, pero fragmentos de información innecesaria fueron penalizados marcando la respuesta como ineXacta.

Otra particularidad de los ejercicios es que hay preguntas sin respuesta en el corpus que esperan como respuesta 'Nil'.

Se consideraron 10 idiomas para preguntas: búlgaro, holandés, inglés, francés, alemán, indonesio, italiano, portugués, romano y español. Todos estos idiomas se usaron como idiomas del corpus, a excepción del indonesio para el que no se disponía de una colección de noticias.

Se propusieron, en total, 37 tareas concretas: 8 monolingües y 29 bilingües. Dada la complejidad de la construcción del ejercicio mismo, la organización solo implementó concretamente las tareas tomadas por algún equipo competidor.

El corpus de datos propuesto consiste en dos corpora distintos: una serie de de recopilados específicamente por los organizadores de la competencia y utilizado con frecuencia en competencias anteriores y, por otro lado, snapshots de las wikipedias en diferentes idiomas con límite en Diciembre de 2006.

Table 1: Tasks activated in 2007 (in green)

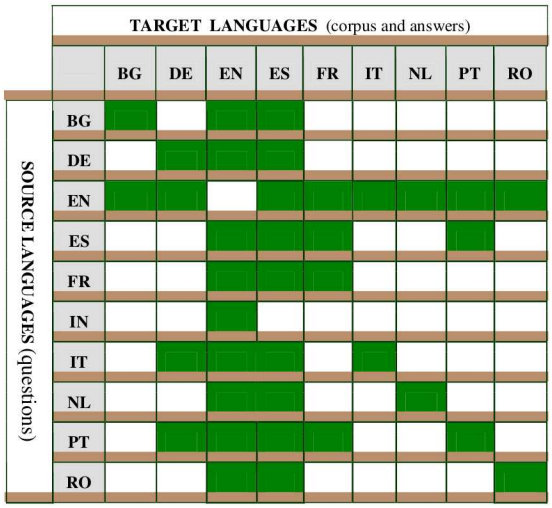


Fig. 5.1: Tareas activadas en la competencia CLEF '07

Para nuestro proyecto utilizamos las preguntas y los datos de evaluación de los ejercicios monolingües es-es (español-español) y pt-pt (portugués-portugués) implementando como corpus de datos solo las wikipedias.

Los datos de los ejercicios constan de dos archivos: un archivo con 200 preguntas agrupadas por temas y otro archivo con diferentes traducciones de la pregunta, una respuesta goldstandard con soporte textual y el id del documento de soporte para el idioma target (del corpus).

Por otro lado, en la guía de ingreso a la competencia[Clef, 2007a] se proveen links para descargar los snapshots de wikipedia sugeridos. Se ofrece una versión de wikipedia en inglés preprocesada de Noviembre de 2006 [1], dos opciones para descargar wikipedia en español: una imagen estática en html de Noviembre de 2006 [2] y un dump xml de Diciembre de 2006 [3] y también 2 opciones de portugués [4]

La guía aclara que, bajo responsabilidad de los participantes, se puede usar cualquier otra versión de Wikipedia, siempre y cuando sea anterior a noviembre / diciembre de 2006. Además, se pide que las respuestas sean tomadas de "entradas reales" o artículos de wikipedia y no de otros documentos (imágenes, discusión, categoría, template, histórico de revisiones, datos de usuario y páginas con meta información).

Los links a páginas de wikipedia en español y en portugués no estaban más disponibles (ambos respondieron 404), mientras que el formato preprocesado pedido para el inglés resultaba realmente complejo de instalar y parecía una linea muerta. Por estos motivos, seguimos la sugerencia sobre el uso responsable de otras wikipedias y utilizamos las diferentes snapshots de Wikidumps[5] que se pueden ver en la tabla 5.1.

[1] `http://ilps.science.uva.nl/WikiXML/`

[2] `http://static.wikipedia.org/downloads/November_2006/es/`

[3] `http://download.wikimedia.org/images/archive/eswiki/20061202/pages-articles.xml.bz2`

[4] `http://static.wikipedia.org/downloads/November_2006/pt/` y `http://download.wikimedia.org/images/archive/ptwiki/20061207/pages-articles.xml.bz2`

[5] Ver `http://dumps.wikimedia.org/` y `http://en.wikipedia.org/wiki/Wikipedia:Database_download#Other_languages`

Idioma	Fecha	Tamaño
Español	5 de Julio de 2006	558M
Portugués	1 de Febrero de 2007	776M
Inglés Simple	4 de Julio de 2006	26M
Inglés	4 de Noviembre de 2006	7,6G
Español	26 de Enero de 2007	902M
Español	14 de Junio de 2013	7,3G
Inglés Simple	24 de Julio de 2013	406M

Tab. 5.1: Snapshots procesados con fechas y tamaños

Idioma	Wiki	News	NIL	Total
es	155	37	8	200
pt	130	59	11	200

Tab. 5.2: Total de preguntas con respuesta en Wikipedia, en otras fuentes y sin respuesta

Utilizando la información disponible en los archivos de respuestas esperadas, discriminamos las preguntas con respuestas con soporte de wikipedia, las soportadas por el corpora de noticias de la organización y las que no tienen respuesta (NIL) (Ver tabla 5.2).

5.1.2. Análisis de las preguntas

Las 200 preguntas del test set están agrupadas por tema, y cada tema tiene de una a cuatro preguntas. Los temas pueden ser entidades nombradas, evento o también categorías como objetos, fenómenos naturales, etc (por ejemplo: Bush, Juegos Olímpicos, notebooks, huracanes, etc). Los temas no están dados en el set de test, pero pueden ser inferidos del primer par pregunta respuesta. La relación del tema con el grupo de preguntas respetaba las siguiente condiciones:

- El tema es nombrado en la primer pregunta o bien en la primera respuesta.
- Las siguientes preguntas pueden contener co-referencias al tema expresado en el primer par pregunta/respuesta

Para ilustrar el punto consideremos el primer grupo de preguntas para el ejercicio de español:

- ¿En qué colegio estudia Harry Potter?
- ¿Cuál es el lema del colegio?
- ¿En qué casas está dividido?
- ¿Quién es el director del colegio?

Tipo	Descripción
FACTOID	preguntas fácticas, preguntando el nombre de una persona, un lugar, la medida de algo, el día en que algo ocurrió, etc.
DEFINITION	preguntas como Qué/ Quién es X?
CLOSED LIST	questions that require one answer containing a determined number of items, e.g

Tab. 5.3: Definición de los tipos de pregunta

Subset	Idioma	Factoid	Definition	List	Total
Todo	es	158	32	10	200
	pt	159	31	10	200
Wiki	es	130	24	9	163
	pt	104	18	8	130

Tab. 5.4: Totales por tipo de pregunta

Como ya mencionamos, algunas preguntas podían no tener respuesta en la colección de documentos, y en ese caso la respuesta exacta era "NIL" y la justificación y el docid vacíos. La organización definió como criterio de inexistencia de la respuesta que ni los asesores humanos ni los demás sistemas participantes pudieran encontrar alguna.

En lo que respecta a los tipos de pregunta, el ejercicio considerara tres categorías: Factoid, Definition y Closed List.

Todos los tipos de pregunta podían contener restricciones temporales, i. e: una especificación temporal proveyendo importante información para devolver una respuesta correcta. Por ejemplo:
Q: Who was the Chancellor of Germany from 1974 to 1982?
A: Helmut Schmidt.
Q: Which book was published by George Orwell in 1945?
A: Animal Farm.
Q: Which organization did Shimon Perez chair after Isaac Rabin's death?
A: Labour Party Central Committee.

Los tipos de preguntas disponen a su vez de subtipos. Las preguntas factoid disponen de 8 subtipos (ver tabla 5.5), las de definiciones y listas tienen 4.

Las preguntas de tipo LIST consisten en enumeraciones de 4 subtipos posibles: PERSON, LOCATION, ORGANIZATION y OTHER. Por ejemplo:
Q: Name all the airports in London, England.
A: Gatwick, Stansted, Heathrow, Luton and City.

Tipo	Ejemplo
PERSON	Q: Who was called the "Iron-Chancellor"? A: Otto von Bismarck.
LOCATION	Q: Which town was Wolfgang Amadeus Mozart born in? A: Salzburg.
ORGANIZATION	Q: What party does Tony Blair belong to? A: Labour Party.
MEASURE	Q: How high is Kanchenjunga? A: 8598m.
COUNT	Q: How many people died during the Terror of Pol Pot? A: 1 million.
OBJECT	Q What does magma consist of? A: Molten rock.
TIME	Q: What year was Martin Luther King murdered? A: 1968.
OTHER	i.e. todo lo que no encaja en las categorías anteriores. Q: Which treaty was signed in 1979? A: Israel-Egyptian peace treaty.

Tab. 5.5: Tipos de respuesta esperada para preguntas Factoid

Como solo una fuente estaba permitida, los items debían aparecer en secuencia, uno después del otro, en un solo documento de la colección. Las preguntas de tipo DEFINITION tienen también estos mismos cuatro subtipos, pero con una semántica ligeramente diferente. En la tabla 5.6 presentamos ejemplos y la definición inferida sobre estos tipos.

Por su parte, en la tabla 5.7 puede verse una clasificación general de preguntas por tipo y por subtipo para los ejercicios de español y de portugués.

5.2. Sistema

Nuestra implementación está basada en Qanus como framework o arquitectura de diseño de QA y en Freeling como proveedor de servicios de procesamiento de lenguaje multilingüe (es decir, no habilitando ejercicios cross-lingual sino mono-lingual en varios idiomas). Qanus (Question-Answering @ National University of Singapore) es un framework de question answering basado en information retrieval y también un sistema de QA funcional simple construido sobre este framework. El proyecto se actualizó por última vez en noviembre de 2012 y contiene herramientas actuales de nlp para inglés (el POS-tagger, el NER-tagger y el Question Classifier de Stanford) y también de information retrieval (índice de búsquedas lucene) bajo licencia de código abierto. El paquete cuenta con un framework (qanus), que cumple un rol equivalente a la arquitectura DeepQA en la organización del proyecto de IBM, y un sistema montado sobre este framework -QA-sys-, equivalente a Watson (Ver Figura 5.2).

Tipo	Descripción y ejemplo
PERSON	i.e. preguntas sobre el rol, el trabajo o información importante de alguien Q: Who is Robert Altmann? A: Film maker.
ORGANIZATION	i.e. preguntas sobre la misión, el nombre completo o información relevante sobre una organización Q: What is the Knesset? A: Parliament of Israel.
OBJECT	i.e. preguntas por la descripción o función de objetos Q: What is Atlantis? A: Space Shuttle.
OTHER	i.e. preguntas por descripciones de fenómenos naturales, tecnologías, procedimientos legales, etc. Q: What is Eurovision? A: Song contest.

Tab. 5.6: Tipos de respuesta esperada para preguntas Definition

Tipo	pt	es	pt-factoid	es-factoid	pt-def	es-def	pt-list	es-list
COUNT	21	22	21	22	0	0	0	0
OBJECT	11	27	6	18	5	9	0	0
MEASURE	15	20	15	20	0	0	0	0
PERSON	33	35	21	24	9	8	3	3
TIME	19	16	18	16	0	0	1	0
LOCATION	33	18	30	18	0	0	3	0
ORGANIZATION	31	32	24	22	6	8	1	2
OTHER	37	30	24	18	11	7	2	5
Total	200	200	159	158	31	32	10	10

Tab. 5.7: Tipos de respuesta esperada en función de tipo de pregunta

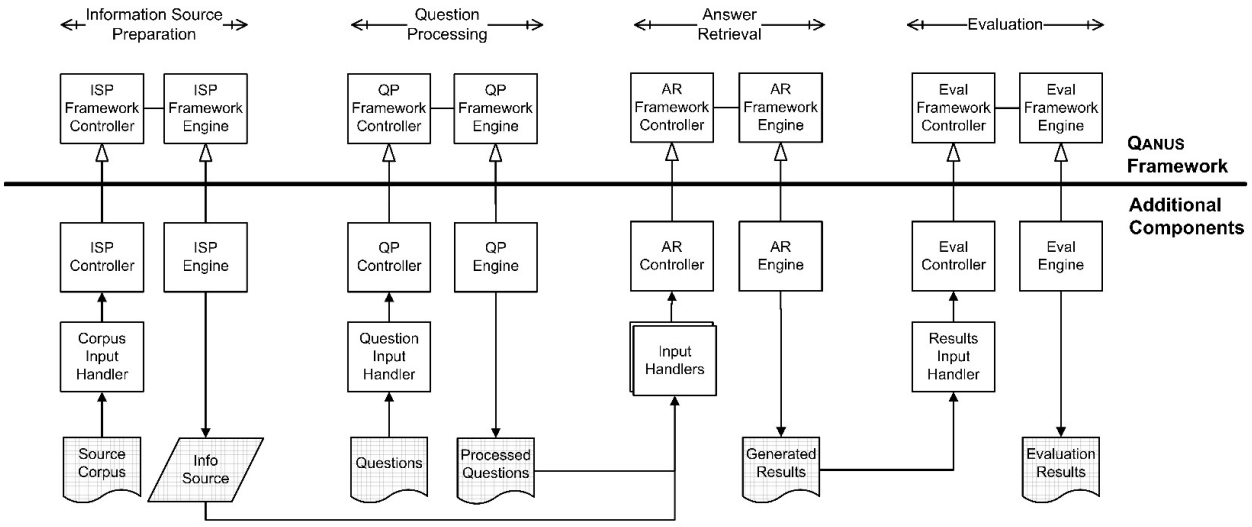


Fig. 5.2: El framework Quanus y la implementación QA-sys

La arquitectura de Qanus mantiene la estructura de pipeline típica similar a las reseñadas en la sección 3 Estado de Arte. Consta de los siguientes pasos:

Preparación de la fuente de información: El primer step del pipeline es preprocesar las fuentes de información para un acceso optimizado en pasos posteriores del proceso. La implementación QA-sys incorpora una base de conocimiento en formato XML de aquaint a un índice de búsquedas Lucene. En este paso se incorpora todo el conocimiento offline; bases de conocimiento dinámicas como la web se modelan en el pasos posteriores.

Análisis de la pregunta: Este paso, igualmente genérico, permite la incorporación de distintos componentes para anotar los tokens de la pregunta con datos útiles para ser consumidos por el paso 3. Otro procesamiento a realizar en este paso podría ser la generación de queries entendibles por los distintos motores de almacenamientos de información del paso 1. En particular, la implementación trae un pos-tagger, un ner-tagger y un question classifier, todos de Stanford. Hablaremos más de estos componentes más adelante.

Generación de respuestas: En este paso se utiliza la información generada en la preparación de la base de información y en el procesamiento de la pregunta para generar una respuesta. También puede incorporarse accesos a la web y validaciones de las respuestas candidatas. La implementación concreta evalúa cada pasaje de los primeros n documentos retornados por Lucene para la pregunta original con una serie de componentes ad-hoc de distancia para adjudicar diferentes grados de confiabilidad a los distintos pasajes.

Además, se provee de un cuarto paso opcional (el sistema de QA está completo con los tres pasos anteriores), para la fase de desarrollo y de evaluación de la performance del sistema:

Evaluación: Este paso está pensado para evaluar las respuestas generadas y presentar-

las de un modo conciso en la fase de desarrollo. Básicamente, cruza las respuestas obtenidas contra unas respuestas esperadas escritas a mano y presenta el total de respuestas dadas correctamente.

En esta sección de la tesis adoptamos el framework Qanus y adaptamos el sistema QA-sys para 1) incorporar Freeling como librería de herramientas de procesamiento del lenguaje permitiendo un funcionamiento con soporte multilenguaje, 2) trabajar sobre el corpus de datos y preguntas de los ejercicios de CLEF '07 de español y portugués descriptos en 5.1 Ejercicio de CLEF y, finalmente, 3) incorporamos y evaluamos algunas mejoras tomadas de la reseña a la literatura científica del apartado anterior 3.2 Literatura y sistemas.

5.2.1. Implementación baseline

El código está escrito en java y mantiene una interfaz común a todos los pasos: un controller cuyas responsabilidades son cargar los componentes y un engine que utiliza los componentes para leer el input, procesarlo y grabar el resultado. La adaptabilidad del framework está dada en la posibilidad de incorporar componentes respetando la interfaz especificada para los mismos o bien, en modificar esta misma interfaz.

La implementación QA-sys está desarrollada para correr sobre el tipo de datos de las evaluaciones Trec 2007. Brevemente, la implementación realiza las siguientes tareas: En el primer paso, incorpora los XML en el formato XML Aquaint propuesto por la competencia a un índice Lucene, en el segundo paso anota la pregunta con POS tags, NERs y clasifica el tipo de respuesta con un clasificador entrenado y luego, en el tercer paso se busca la pregunta sobre el índice lucene y se retorna una lista con n documentos rankeados. Estos documentos se subdividen en pasajes. Luego se aplican diferente algoritmos ad-hoc dependiendo del tipo de respuesta esperada. Por ejemplo, si la respuesta es un nombre de persona, se ejecuta NER sobre los diferentes pasajes buscando nombres candidatos, si el tipo esperado es una fecha, se utilizan expresiones regulares escritas a mano, etc. Finalmente, los pasajes candidatos se evalúan utilizando heurísticas de proximidad de los candidatos a la pregunta inicial. Para esto se utilizan diferentes Scorers que rankean los pasajes según diferentes características (features) y luego se selecciona alguna priorizando algunas características sobre otras, dependiendo también del tipo de respuesta esperada. Por último, el evaluador de resultados mide la exactitud (*accuracy*): total de respuestas correctas sobre total de preguntas.

QA-sys funciona sólo sobre preguntas del tipo factoid y, a modo de comparación, el mejor sistema según la Trec 2007, el LymbaPA07 obtuvo un grado de precisión del 0.706 y el décimo (Quanta) obtuvo 0.206, mientras que QA-sys logra el 0.119 disponiendo de una implementación sumamente simple. La métrica de precisión, en esta competencia, consistió en la evaluación por parte de jueces humanos de una respuesta exacta única [Dang et al., 2008].

En las siguientes secciones de la tesis describiremos la implementación de nuestro sistema de question answering tomando como base los tres pasos principales del pipeline de QA-sys.

5.2.1.1. Base de conocimiento

El módulo de procesamiento de la base de información de la implementación QA-sys está desarrollada para correr sobre el tipo de datos de las evaluaciones Trec 2007, un

Wikipedia	# Entradas	Nulas	Redirects	Filtradas	Válidas
simple-06	18273	22	3452	5241	9558
es-06	233750	52	62805	34947	135946
pt-07	498039	80	210983	43390	243586
simple-13	180067	8	35600	41902	102557

Tab. 5.8: Wikipedias: cantidad de entradas válidas e inválidas

formato XML conocido como “Aquaint". Por otro lado, el mismo framework esta atado al procesador de XMLs SAX, una entre varias implementaciones disponibles en el entorno de java. En nuestra adaptación, incorporamos la librería gwtwiki[6] para procesar los xml dumps de wikipedia. Como esta librería no utiliza SAX, nos vimos obligados a modificar el framework.

En este proceso descartamos artículos mal formados y entradas representado imágenes o discusiones, tal como se sugiere en la guía. Los artículos se indexan en índices Lucene como documentos con los siguiente campos: *id, title, body y all.* La tabla 5.8 muestra algunos datos del proceso de generación de índices.

Las categorías filtradas contienen artículos con alguno de los siguientes prefijos en el título: WP, wp, Biografías, Wikipedia, Wikiproyecto, Imagen, Plantilla, MediaWiki, Template, Category, Help, Image, Ayuda, Portal, Ajuda, Categoria, Categoría, Imagem, Predefinição.

5.2.1.2. Anotado de la pregunta

El módulo original de QA-sys de procesamiento de la pregunta espera preguntas en el formato Aquaint definido por Trec, al igual que el tipo de datos esperado sobre el corpus en el paso anterior. Durante el procesamiento clasifica la pregunta, y ejecuta un POS tagger y un NER tagger, utilizando como librerías de procesamiento de lenguaje el POS-Tagger, el NER-Tagger y el Question Classifier de la Universidad de Stanford, disponibles solo para inglés, descriptas en A.1 Herramientas de Stanford respectivamente.

En nuestra adaptación incorporamos los módulos de POS y NERC multilingües de freeling. Con respecto a la clasificación de preguntas, ya mencionamos que no existen clasificadores para español y tampoco para el portugués. Frente a esta situación podríamos haber implementado clasificadores heurísticos basados en reglas escritas a mano. Esta implementación nos desviaba de nuestro proyecto, por lo que utilizando la disponibilidad de una traducción confiable al inglés de las preguntas en archivos de datos (debido a la activación de los ejercicios cross-lingual en-es y en-pt), decidimos utilizar el clasificador de Stanford sobre la traducción inglesa de la pregunta para clasificarlas en el idioma en cuestión. La tabla 5.9 presenta la distribución de las 200 preguntas para español y para portugués según las seis clases principales del esquema de Li & Roth obtenida al clasificar las preguntas con el clasificador de Stanford.

Al finalizar el paso del anotado de la pregunta, las preguntas tienen asociado su tipo, las distintas entidades nombradas, el análisis gramatical y las qwords. Dejamos la generación de queries para el paso posterior ya que así es como estaba implementado en el sistema original.

[6] Java Wikipedia API (Bliki engine): `http://code.google.com/p/gwtwiki/`

Clase	# Preguntas (es)	# Preguntas (pt)
HUM	62	53
NUM	53	52
ENTY	34	28
DESC	25	30
LOC	24	36
ABBR	2	1

Tab. 5.9: Distribución por clases de las preguntas para español y portugués

5.2.1.3. Generación de Respuestas

Como baseline de desarrollo y pruebas, adaptamos los algoritmos de Qanus. En esta subsección vamos a describir esta implementación con detalle, mencionando los cambios más importantes que realizamos para adaptarla a la tarea elegida. En las siguientes comentaremos diferentes modificaciones y algoritmia agregada. El proceso de Qanus para la generación de respuestas puede estructurarse así:

1. Filtrado de preguntas
2. Generación de Queries
3. Information Retrieval
4. Extracción y rankeo de oraciones
5. POS tagging de las mejores oraciones
6. Heurísticas de AR basadas en el QC

En el primer paso, se descartan todas las preguntas cuyo tipo no sea *factoid*. Esto, en principio, deja afuera a 32 preguntas de tipo *definition* y 10 de tipo *list* para el ejercicio en español (24 y 9, respectivamente, considerando solamente las que como goldstandard answer tienen datos de wikipedia) y 31 de tipo *definition* y 10 de tipo *list* para el portugués (18 y 8, respectivamente, considerando las de wikipedia).

En en segundo paso, la generación de queries, el sistema baseline utiliza 2 algoritmos diferentes según el qtype inferido por el clasificador de preguntas. El primer caso es especifico para la clase fina 'HUM:ind' y consta de tres tipos de expresiones regulares escritas a mano (en inglés) que matchean contra preguntas de la forma: "Who (is|was) (the) NN ... NN?" (NN significa, acá, sustantivo). Por ejemplo: "Who is the CEO of Microsoft?". También se considera aquí preguntas con la forma "Who VERBO ...?". Para este tipo de expresiones, el algoritmo específico extrae los sustantivos y los verbos que siguen al patrón descripto y estos son utilizados como la query (estos términos se llaman *subject* de la pregunta y el mismo procedimiento se utiliza en un caso posterior de generación de respuestas). Eliminamos este caso ya que su dependencia del idioma volvía el procedimiento obsoleto para un sistema con soporte multilenguaje.

Por su parte, si el tipo de la pregunta no es 'HUM:ind', el algoritmo general de generación de queries espera, además de la pregunta propiamente dicha, un campo asociado a la pregunta, el 'target', aparentemente definido en la competencia TREC para la que está

Abreviatura	Nombre	Descripción
Freq	Frecuencia	Computa la cantidad de veces que los tokens de la query ocurren en la oración. Esta suma se divide por la cantidad de tokens de la oración, dando un valor entre 0 y 1.
Coverage	Cobertura	Computa cuantos tokens de la query aparecen al menos una vez en la oración, y divide esta suma por el total de tokens de la *query*.
Prox	Proximidad	Computa la distancia entre dos strings en un tercero. Ver aparte.
Span	Distancia entre tokens	Computa la distancia media entre términos de la query en la oración. Ver aparte.

Tab. 5.10: Scorers de Qanus

escrito. El campo 'target' es utilizado por mucha de la algoritmia que vendrá. Dejaremos en suspenso con qué poblamos ese dato hasta más adelante. Un uso posible es asociarlo el 'tópico' del grupo de preguntas de los ejercicios de la competencia. Más adelante veremos qué contenido le cargamos.

El método default FormQuery genera una query que contiene, sin repetir, todas las palabras no-stopwords del campo target y todos los sustantivos, verbos y adjetivos de la pregunta completa según el pos tagger de stanford. Nosotros adaptamos este método para funcionar con las etiquetas de Freeling y utilizamos este mecanismo de generación de queries como baseline.

En el tercer paso, a partir de la query generada se hace un pedido al índice lucene y se obtiene una lista de documentos. En el cuarto paso, los documentos son divididos en sus oraciones de manera indiscriminada, es decir: las oraciones no heredan posición de acuerdo al ranking de los documentos a los que pertenecían.

Las oraciones se evalúan mediante el peso ponderado de diferentes features que consideran la query generada y la oración, generando un nuevo orden entre ellas, basado en su *score*. El score es un valor real entre 0 y 1, que sigue la siguiente fórmula:

$$PassageScore = (0{,}9) * CoverageScore + (0{,}05) * FreqScore + (0{,}05) * ProxScore;$$

Los scorers *coverage*, *freq* y *prox* son diferentes métricas de evaluación de oraciones, cuya imagen es una valor real entre 0 y 1. En la tabla 5.10 se hace una presentación general de los Scorers. A continuación explicamos los algoritmos de *Prox* y *Span* ya que no son triviales (*Span* aún no ha sido utilizado pero lo será en breve).

Prox toma dos strings a buscar en un tercero. Busca ambos en el tercero y computa la distancia en tokens entre ellos. Esta distancia se calcula como la distancia entre el centro de ambos strings. Por ejemplo, para los strings de búsqueda "Argentina es un país americano" y "independizado en 1810" sobre el texto "Argentina es un país americano, originalmente una colonia española, independizado en 1810" se considera la distancia entre 'un' y 'en' (por ser los tokens 'intermedios' de ambos strings de búsqueda. La distancia entre ambos, en el tercer string, es 7 (la cantidad de tokens entre ellos). Esta distancia se divide por

la longitud en tokens del string en el que se buscan (12), dando un resultado de 0.58. Un score cercano a 1 denota que los dos string están cercanos uno al otro en el tercer string.

Por su parte, *Span* tiene un concepto similar, pero funciona sobre un solo string de búsqueda, considerando sus tokens. Los distintos tokens buscados ocurren en ciertas posiciones. *Span* considera la distancia entre las posiciones de los tokens más distantes, dividiendo el total de tokens encontrados por este valor. Un score cercano a 1 significa que los términos del string buscado están cerca en el string en el que se buscan. Por ejemplo, suponiendo los siguientes matchs de tokens (denotados por una X):
..... X ..... X ..... X .....
.....a ..... b ..... c .....

El score de *Span* está dado por #total de tokens encontrados / |posición de c- posición de a|.

En el quinto paso se seleccionan las 40 oraciones mejor rankeadas según este score y se les aplica pos tagging.

Finalmente, el sexto paso ejecuta diferentes heurísticas de extracción de respuestas en base al tipo de pregunta generado por el clasificador en el paso anterior. Define diferentes heurísticas para los siguientes casos: ABBR:exp, ABBR:abb, HUM:gr y ENTY:cremat, HUM:ind, HUM general, LOC general, NUM:date, NUM:period, NUM:count, NUM general, ENTY general, que veremos a continuación.

Caso ABBR:exp

Si el tipo de respuesta esperada es ABBR:exp (expansión de una abreviación), entonces el algoritmo de extracción es el siguiente: primero, se extraen las entidades nombradas de tipo 'Organización' desde la pregunta. Con ellas, se generan regex de expansión de esa abreviación. Por ejemplo, para la abreviación "IGFA" se genera la regex $[I][A-Za-z0-9][G][A-Za-z0-9][F][A-Za-z0-9][A][A-Za-z0-9]$. En términos coloquiales, esta regex sirve para buscar en fragmentos de texto por expresiones con la forma I... G... F... A.... Por cada una de las 40 oraciones mejor rankeadas, el proceso busca por ocurrencias de ese patrón. Finalmente, no realiza ningún trabajo posterior y devuelve la primera encontrada.

Caso ABBR:abb

Este tipo de respuesta esperada representa el pedido de una contracción de una abreviación, la inversa de la anterior. Por ejemplo, "¿Con qué siglas se conoce a la corporación 'International Business Machines'?" (IBM). El caso no tiene código: *// Abbreviations : Contractions ->What can we do?*

Caso HUM:gr y ENTY:cremat

El tipo HUM:gr refiere a grupos humanos, como compañías u organizaciones, mientras que ENTY:cremat refiere a entidades de creación humana (como libros, inventos, discos, etc). Para este caso, el proceso consiste en extraer nombres propios (proper nouns, NNPs) de todas las oraciones rankeadas (se asume que nombres propios consecutivos representan un solo nombre propio, por ejemplo Tiger/NNP, Woods/NNP forman un nombre propio

solo). Sobre estos resultados se ejecutan filtros basados en diccionarios. En concreto, se eliminan los siguientes términos: {Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday}, {January, February, March, April, May, June, July, August, September, October, November, December}, {us, uk, france, england, cuba, japan, u.s, america}. Para cada respuesta candidata que pasó el filtro, se rankean de acuerdo a diferentes scores basados los comparadores *coverage*, $freq$ y $prox$. La fórmula de ponderación utilizada es la siguiente:

$TotalScore = (0{,}55 * CoverageScore) + (0{,}2 * SentenceScore) + (0{,}1 * ProximityScore) + (0{,}15 * RepeatedTermScore)$

Donde $CoverageScore$ es el cubrimiento del '*target*' de la pregunta en la oración de soporte de la respuesta, $ProximityScore$ mide la distancia entre el 'target' de la pregunta y la respuesta candidata en el contexto de la oración de soporte, $SentenceScore$ considera la posición de la oración en el ranking de las 40 oraciones y $RepeatedTermScore$ es una penalización (su valor es negativo) si la respuesta candidata tiene tokens en común con el 'target' de la pregunta. En la adaptación eliminamos el filtrado por diccionarios ya que depende del idioma.

Caso HUM:ind

El tipo HUM:ind son preguntas que refieren a un individuo humano. El algoritmo en este caso intenta generar, para ciertos patrones de preguntas, el *subject* de la pregunta (aplicando el mismo procedimiento descripto en el caso eliminado de generación de queries). Eliminamos este caso ya que se basa en expresiones regulares específicas por idioma. El siguiente paso (primero en nuestra implementación) es extraer, para cada oración, las entidades nombradas de tipo 'PERSON'. Para cada respuesta candidata, se evalúan distintos features para generar un score general:

$TotalScore = (0{,}5 * CoverageScoreTarget) + (0{,}25 * CoverageScoreSubject) + (0{,}35 * SentenceScore) + (0{,}25 * ProximityScore) + (0{,}1 * RepeatedTermScore) + (0{,}5 * IsPronoun)$

Donde los scores representan lo siguiente:

- CoverageScoreTarget: cuántos tokens del 'target' aparecen en la oración fuente de la respuesta candidata
- CoverageScoreSubject: cuántos tokens del 'subject' aparecen en la oración fuente de la respuesta candidata (si se pudo encontrar un subject, cero si no - en nuestra adaptación es siempre cero)
- SentenceScore: puntaje derivado de la posición de la oración-fuente en el ranking de oraciones
- ProximityScore: cuán cerca están la query utilizada como input del módulo de information retrieval (sin stop-words) y la respuesta candidata en el contexto de la oración de la que se extrajo la respuesta candidata.
- RepeatedTermScore: penalización (negativo). Coverage entre el 'target' y la respuesta candidata
- IsPronoun: Penaliza si la respuesta candidata contiene pronombres. En concreto, verifica si algún token pertenece a la lista {it, we, he, she, they, our, their}.

En base a este score, se devuelve la respuesta candidata con puntaje más alto.

Caso HUM general

Este caso contempla todos los tipos de respuestas esperadas de clase HUM (humano) que no son gr ni ind (estos son dos casos: título y descripción). En este caso, se toma el primer nombre propio de la oración mejor rankeada y se utiliza eso como respuesta.

Caso LOC general

La clase LOC (location o locación) incluye preguntas que refieren a lugares. En este caso, se extraen todas las entidades nombradas de tipo 'LOCATION' de las oraciones rankeadas y se las evalúa según el siguiente scoring:
$TotalScore = (0{,}6 * CoverageScore) + (0{,}1 * SentenceScore) + (0{,}2 * ProximityScore) + (0{,}5 * SanityScore) + (0{,}3 * RepeatedTermScore)$

Donde los scores representan lo siguiente:

- CoverageScore: cuántos tokens del 'target' aparecen en la oración fuente de la respuesta candidata
- SentenceScore: puntaje derivado de la posición de la oración-fuente en el ranking de oraciones
- ProximityScore: cuán cerca están la query utilizada como input del módulo de information retrieval (sin stop-words) y la respuesta candidata en el contexto de la oración de la que se extrajo la respuesta candidata.
- RepeatedTermScore: penalización (negativo). Coverage entre el 'target' y la respuesta candidata.
- SanityScore: es un placeholder para implementar código comentado. En el código que encontramos, vale 1 (constante).

Caso NUM:date

Este tipo de respuesta esperada representa una fecha. El algoritmo verifica la ocurrencia del término 'year' en la pregunta. Si la pregunta contiene el término, genera un patrón para buscar años (una expresión regular) y devuelve el primero que encuentra. En caso contrario, devuelve el título del primer documento encontrado (ya que el módulo está implementado para responder preguntas de TREC y los documentos son artículos cuyos títulos tienen, en general, fechas). Como oración de justificación, toma la primer oración del documento.
En nuestra adaptación, recaímos en el caso general para la clase NUM, especificándole a Freeling que solo considere fechas (ver tres títulos más abajo).

Caso NUM:period

Este tipo de respuesta esperada representa un periodo de tiempo. El código busca en la pregunta por el patrón "How old (is|was) NNP1,?" (donde "NNP1," significa uno o más nombres propios seguidos) y se queda con la secuencia de nombres propios como *subject*.

Si logra detectarse este subject, se busca en las oraciones rankeadas por números y por números seguidos de los tokens "years old". Luego se rankean estos números según la siguiente fórmula:

$$TotalScore = (0{,}8 * ProximityScore) + (0{,}2 * SentenceRankScore)$$

Donde ProximityScore mide la distancia entre el número y el subject en el contexto de la oración de la que se extrajo el número y SentenceRankScore representa el ranking de la oración dentro de las 40 oraciones.

Si la detección del subject o de los números es infructuosa, se recae en una estrategia general para la clase NUM, que consiste en buscar el primer número encontrado por el pos tagger (hardcodeado para los tags usados por Stanford como CD) y devolverlo.

En nuestra adaptación, eliminamos el caso del subject porque dependía del idioma y recaímos en el el caso general.

Caso NUM:count

Este tipo de respuesta esperada representa un número. El algoritmo busca por el patrón "How many ...?" intentado extraer el *subject*, por ejemplo, para la pregunta "How many miners...?" el subject es "miners". Con este subject, realiza exactamente los mismos pasos que el caso inmediatamente anterior y nosotros realizamos la misma adaptación, esta vez especificando que los números buscados no sean fechas.

Caso NUM general

Considera los casos numéricos no contemplados en los tres casos anteriores. El algoritmo ejecutado consiste en encontrar el primer set continuo de tokens taggeados como 'CD' (números) por el POS tagger de Stanford. En nuestra adaptación a Freeling, pudimos mejorar esto separado dos tipos de números: los referidos a fechas y los referidos a cantidades. En el caso general, se busca cualquier tipo de números. Sin embargo, como especificamos en los títulos inmediatamente anteriores, para ciertos casos especificamos o bien fechas, o bien cantidades.

Caso ENTY general

El caso de entidades general incluye todas las subclases que no son 'cremat'. El algoritmo implementado es el mismo que para el caso de 'HUM' general, este es: tomar el primer nombre propio de la oración mejor rankeada y utilizarlo como respuesta.

Otro caso

Se implementa el mismo algoritmo que el caso anterior.

5.2.2. Modificaciones realizadas

Además de los cambios realizados para convertir al baseline de qanus en un sistema multilingüe basado en la librería freeling, realizamos algunos agregados mínimos a la lógica. En esta sección describiremos diferentes modificaciones realizadas y en la siguiente veremos la evaluación comparativa entre la performance del sistema baseline y las modificaciones en las diferentes tareas que realizan los submódulos.

5.2.2.1. Inferencia del tópico del grupo de preguntas

Como vimos anteriormente, los ejercicios consisten en preguntas agrupadas en torno a tópicos con entre 1 y 4 preguntas. Este tópico está disponible en los archivos de datos, pero se esperaba que en la competencia no se utilicen. Los temas pueden ser entidades nombradas, evento o también categorías como objetos, fenómenos naturales, etc. Por ejemplo, estos son algunos de los primeros temas del test set de español: 'Colegio de Harry Potter', 'Pez Espada', 'Amintore Fanfani', 'Revolución de Terciopelo'. Según mencionamos anteriormente, el tema del grupo de preguntas respeta las siguiente condiciones:

- El tema es nombrado en la primer pregunta o bien en la primera respuesta.
- Las siguientes preguntas pueden contener co-referencias al tema expresado en el primer par pregunta/respuesta

Aprovechamos el campo 'target' del sistema baseline para incorporar este tópico al flujo de procesamiento, ya que su funcionalidad es la esperada para este input. Finalmente, evaluamos diferentes contenidos para este valor. A modo de testeo, utilizamos, aún cuando en la competencia no estaba permitido, el tópico real del grupo según el test set. Por otro lado, implementamos otros tres métodos para generar el tema, siempre basándonos en la pregunta. Hicimos esto por considerar que las respuestas, probablemente, sean erróneas. Dado que no podemos saberlo, apoyar algoritmia sobre el supuesto de que estén bien sería contraproducente para el caso general. Así, implementamos: un método en el cual se agregan solo las entidades nombradas, los números y las fechas de la pregunta, otro en el que se agregan solo los sustantivos y, finalmente, otra en la que se agregan ambos. Si el método elegido no funciona, se utiliza la pregunta sin stop words ni qwords. En todos los casos, nos referimos a la primer pregunta del grupo.

En concreto, esto se reduce a cuatro casos de código, que en la sección de experimentación llamaremos: **1) Tema Test**, cuando utilizamos el tema explícito de los datos de testing, **2) Tema NERS**, **3) Tema sustantivos** y **4) Tema híbrido**.

5.2.2.2. Generación de queries

Para la extracción de queries utilizamos, como primer método y baseline de evaluación, el implementado por qanus. La query generada contiene, sin repetir, todas las palabras no-stopwords del campo target y todos los sustantivos, verbos y adjetivos de la pregunta completa solamente contra el campo *ALL* del índice (si utilizar, por ejemplo, el campo *TITLE*). En la evaluación llamaremos a este método **baseline**. Le agregamos, además, números y fechas al final. Implementamos un segundo método, aprovechando el conocimiento de la estructura de datos del índice, que consiste en agregarle a esta misma query

un caso, si el tema no es vacío, en el que se busca por este tema en el título de los artículos, con un peso virtualmente infinito frente al resto de la query.

Es decir, la query generada por el método 2 tiene la forma siguiente: *(TITLE: target)*^{n} ***OR query baseline***. En concreto, utilizamos $n = 5$. En la evaluación llamaremos a este método **improved baseline**.

Finalmente, implementamos la heurística propuesta por los papers [Lampert, 2004] y [Moldovan et al., 2000], que, recordamos, consiste en 8 reglas para agregar, en orden, a la query:

1. Todas las palabras no stop words entre comillas
2. Todas las entidades nombradas reconocidas
3. Todas las construcciones nominales con sus adjetivos
4. Todas las demás construcciones nominales
5. Todos los sustantivos con sus adjetivos
6. Todos los demás sustantivos
7. Todos los verbos
8. El 'focus' de la pregunta

Dado que no implementamos ningún algoritmo de extracción de *focus*, el último ítem fue reemplazado por el tema de la pregunta, si existiera. En la evaluación llamaremos a este método **lasso**, por el nombre del sistema en el que fue implementado por primera vez. Además, agregamos números y fechas también, como en el baseline. En realidad, detectores de entidades distintos de freeling suelen incorporar los números como entidades nombradas, por lo que no salimos de la propuesta de lasso.

5.2.2.3. Generación de respuestas

Con respecto a la generación de respuestas, incorporamos solamente distintos scorers nuevos para rankear los pasajes y no modificamos los mecanismos baseline de extracción de respuestas a partir de pasajes más allá de lo explicado en la sección anterior. Los nuevos features incorporados son:

- LengthScore : Contempla la longitud del pasaje, priorizando oraciones cortas pero no demasiado cortas. Este score trata de evitar problemas vinculados con mala redacción en los documentos que generaban pasajes enormes y sin sentido para las librerías.
- QVerbScore: Contempla la presencia de verbos de la pregunta en el pasaje candidato.
- QNounScore: Contempla la presencia de sustantivos de la pregunta en el pasaje candidato.
- QNERScore: Contempla la presencia de las entidades nombradas de la pregunta en el pasaje candidato.

Todos están normalizados (son reales entre 0 y 1 inclusive). El objetivo de los últimos tres fue distinguir, dentro de lo que es el score de cobertura, la presencia de términos de importancia destacada dentro del pasaje.

Con estos cuatro scores nuevos, implementamos tres métodos de rankeo de pasajes. El primero es el **baseline** y su formula de rankeo es la siguiente:

$$PassageScore_{bl} = (0{,}9) * CoverageScore + (0{,}05) * FreqScore + (0{,}05) * ProxScore;$$

El segundo método y tercer método utilizan este score y se definen como sigue:

$$PassageScore_2 = PassageScore_{bl} * 0{,}4 + QNERScore * 0{,}2 + QVerbScore * 0{,}15 + QNounScore * 0{,}25$$

$$PassageScore_3 = PassageScore_{bl} * 0{,}4 + LengthScore * 0{,}15 + QNERScore * 0{,}1 + QVerbScore * 0{,}1 + QNounScore * 0{,}25$$

5.2.3. Experimentación

Hicimos corridas con todas las combinaciones recién expuestas y, además, variamos la cantidad de documentos retornados por lucene (50 en el sistema baseline) y la cantidad de pasajes extraídos de ellos (40 en el sistema baseline). Por otro lado, modificamos el sistema para que devuelva una lista de respuestas en lugar de una sola, considerando que los resultados serían bajos. Sobre esta lista, utilizamos la métrica MRR (Mean Reciprocal Rank), única adaptada al caso, sobre distintas longitudes de listas de respuestas. Para evaluar consideramos solamente la respuesta y no el pasaje de justificación, es decir, realizamos una evaluación lenient (indulgente). Por otro lado, utilizamos una evaluación automática. Como $R-set$ utilizamos solamente la respuesta goldstandard disponible en los archivos de test y, considerando su tamaño acotado, evaluamos diferentes métricas. Utilizamos, primero, un match exacto, es decir, chequeamos que la respuesta dada y la respuesta esperada coincidan exactamente. Luego usamos cubrimientos de la respuesta esperada por la respuesta dada, con diferentes porcentajes de cubrimiento. Esta decisión se justifica, por ejemplo, para el caso de la primer pregunta del ejercicio para el español, en la que la respuesta esperada es "Colegio Hogwarts de Magia y Hechicería" y nuestro sistema encuentra, en 5to lugar, la respuesta "Hogwarts".

A continuación listamos sintéticamente todas las variables libres y las instanciaciones que elegimos cuyas permutaciones generan una corrida distinta:

- Idioma: español, portugués (2 opciones)
- Cantidad de Documentos retornados por el módulo de IR: 50, 100, 200 (3 opciones)
- Cantidad de Pasajes extraídos de esos documentos: 20, 40, 70 (3 opciones)
- Método de Generación de Queries: baseline (1), improved baseline (2), lasso (3) (3 opciones)
- Método de Inferencia de Temas: Test (1), NERs (2), sustantivos (3), híbrido (4) (4 opciones)

- Método de Ranking de Pasajes: baseline (1), 2, 3 (3 opciones)

Por otro lado, las siguientes opciones no determinan corridas pero sí determinan resultados distintos:

- Cantidad de respuestas dadas por el sistema: 1, 5, 10, 25 (4 opciones, MMR_1 o exacto, MRR_5, MRR_10 y MRR_25, respectivamente, las filas de las tablas de resultados)
- Forma de evaluación automática: exacto, cubrimiento del 100 %, 75 % y 50 %, 15 % y 1 % (6 opciones) (Exacto, Covr 1, Covr .75, Covr .5, Covr .15 y Covr .1, respectivamente, columnas en las tablas de los resultados de las corridas)

La cantidad de permutaciones totales es inmensa: 2 idiomas $\times$ 3 métodos de generación de queries $\times$ 4 métodos de generación de tópicos $\times$ 3 métodos de rankeo de pasajes $\times$ n distintas cantidades de documentos retornados por lucene $\times$ m cantidad de pasajes extraídos de los documentos, con $n = 3$ y $m = 3$ es, 648. Por eso, realizamos el siguiente procedimiento experimental: en primer lugar, evaluamos, para el sistema baseline, variaciones sobre la cantidad de documentos y de pasajes. Como método 'baseline' de inferencia de tópicos usamos el método 2), que agrega solo NERs como target, ya que el método baseline de generación de queries contiene sustantivos, adjetivos y verbos, pero no entidades nombradas. Sobre la combinación de totales que dieron mejores resultados para esta combinación, evaluamos los diferentes métodos. Como veremos en la corrida 1, muchas de las variaciones planteadas parecen no generar grandes cambios en los resultados, por lo que redujimos la cantidad de parámetros en las corridas posteriores.

Por otro lado, consideramos diferentes conjuntos de preguntas, discriminando por tipo de pregunta y por base de conocimientos en la que se esperaba encontrarla. Con respecto al tipo de pregunta, el sistema no considera preguntas que no sean factoids y nosotros disponemos de esa información en el archivo de datos. Con respecto a la base de conocimientos en la que se espera encontrarla podemos discriminar entre preguntas cuya respuesta goldstandard surge de wikipedias y aquellas que surgen del corpus que no utilizamos. Limitamos la presentación de resultados a preguntas factoids cuya respuesta goldstandard está en wikipedia. Los resultados para los demás casos, salvo alguna escasas excepciones, son inferiores. Como vimos anteriormente, las preguntas que se responden con wikipedia son 163 para el español y 130 para el portugués, las factoid son 158 y 159 respectivamente y las que cumplen ambas condiciones son 122 y 104.

Las corridas que vienen a continuación se estructuran de la siguiente manera: en la **Corrida 1**, variamos la cantidad de documentos retornados y la cantidad de pasajes extraídos de los mismos, dejando fijos los métodos de algoritmia y el idioma. En las siguientes tres corridas (**Corrida 2.1**, **Corrida 2.2** y **Corrida 2.3**) variamos de a uno los métodos de generación de queries, inferencia de temas y ranking de pasajes, respectivamente, dejando fijos en el método baseline los otros dos. Finalmente, en las (**Corrida 3.1** y **Corrida 3.2** utilizamos la combinación de los mejores métodos obtenidos en las anteriores corridas sobre español y sobre portugués.

Corrida 1: instanciación de cantidades

Medida	Exacto	Covr 1	Covr .75	Covr .5	Covr .15	Covr .01
Exacto	7.69	7.69	8.46	10.00	10.00	10.00
MRR_5	8.87	9.18	9.95	**12.51**	**13.47**	**13.54**
MRR_{10}	**9.37**	**9.44**	**10.21**	**12.77**	**14.25**	**14.31**
MRR_{25}	**9.56**	**9.63**	**10.40**	**13.07**	**14.58**	**14.65**

Medida	Exacto	Covr 1	Covr .75	Covr .5	Covr .15	Covr .01
Exacto	**7.75**	**7.75**	**8.53**	**10.08**	**10.08**	**10.08**
MRR_5	**8.94**	**9.25**	**10.03**	12.42	13.39	13.45
MRR_{10}	9.31	9.38	10.16	12.55	14.03	14.10
MRR_{25}	9.54	9.61	10.39	12.89	14.41	14.48

Medida	Exacto	Covr 1	Covr .75	Covr .5	Covr .15	Covr .01
Exacto	6.98	6.98	7.75	8.53	8.53	8.53
MRR_5	8.58	8.89	9.66	11.99	13.02	13.02
MRR_{10}	8.92	8.99	9.76	12.09	13.64	13.64
MRR_{25}	9.12	9.19	9.96	12.43	14.02	14.02

Tab. 5.11: Corrida 1: con 50 documentos de Lucene y 20, 40 y 70 pasajes extraídos respectivamente

En la primer corrida variamos la cantidad de documentos retornados por el índice entre $\{50, 100, 200\}$, y variamos también la cantidad de pasajes extraídos de esos documentos entre $\{20, 40, 70\}$, generando un total de 9 ejecuciones y dejando fijos los siguiente parámetros:

- Idioma: Español
- Método de Generación de Queries: 1
- Método de Temas: 2
- Método de Ranking: 1

Las tablas 5.11, 5.12 y 5.13 presentan los resultados de esta primera corrida. Notar que los valores expresados están multiplicados por 100 (MRR toma valores entre 0 y 1, aquí los presentamos como valores entre 0 y 100).

En primer lugar, observamos que el incremento de documentos retornados por el módulo de recuperación de información empobrece los resultados en todos los casos. En segundo lugar, se observa que extraer 70 pasajes empobrece los resultados, también en todos los casos. Las opciones de 20 y 40 pasajes no son tan lineales. En la tabla 5.11 marcamos en negritas los casos en los que uno u otro dieron mejores resultados. Como se puede apreciar, el cuadrante más exacto (para Exacto y MRR_5) y para comparaciones exactas, 40 dio mejores resultados. Considerando esto, utilizamos la configuración original del sistema baseline de qanus (50 documentos $\times$ 40 pasajes) para el resto de las corridas.

Corrida 2.1: Generación de Queries

Medida	Exacto	Covr 1	Covr .75	Covr .5	Covr .15	Covr .01
Exacto	6.15	6.15	6.92	7.69	7.69	7.69
MRR_5	7.24	7.40	8.17	10.09	11.12	11.12
MRR_{10}	7.37	7.40	8.28	10.33	11.76	11.76
MRR_{25}	7.67	7.73	8.61	10.84	12.27	12.27

Medida	Exacto	Covr 1	Covr .75	Covr .5	Covr .15	Covr .01
Exacto	6.25	6.25	7.03	7.81	7.81	7.81
MRR_5	7.42	7.58	8.36	10.27	11.32	11.32
MRR_{10}	7.64	7.66	8.54	10.70	12.17	12.17
MRR_{25}	7.95	8.02	8.90	11.19	12.66	12.66

Medida	Exacto	Covr 1	Covr .75	Covr .5	Covr .15	Covr .01
Exacto	6.20	6.20	6.98	7.75	7.75	7.75
MRR_5	7.82	7.97	8.75	10.68	11.72	11.72
MRR_{10}	7.95	7.97	8.84	11.12	12.58	12.58
MRR_{25}	8.24	8.31	9.18	11.60	13.05	13.05

Tab. 5.12: Corrida 1 con 100 documentos de Lucene y 20, 40, 70 pasajes extraídos respectivamente

Medida	Exacto	Covr 1	Covr .75	Covr .5	Covr .15	Covr .01
Exacto	4.69	4.69	4.69	5.47	5.47	5.47
MRR_5	6.64	6.80	6.80	8.49	9.53	9.53
MRR_{10}	6.85	6.87	7.08	8.91	10.36	10.36
MRR_{25}	7.03	7.06	7.27	9.23	10.68	10.68

Medida	Exacto	Covr 1	Covr .75	Covr .5	Covr .15	Covr .01
Exacto	4.62	4.62	4.62	5.38	5.38	5.38
MRR_5	6.35	6.50	6.50	8.23	9.26	9.26
MRR_{10}	6.66	6.68	6.87	8.86	10.31	10.31
MRR_{25}	6.81	6.83	7.02	9.15	10.59	10.59

Medida	Exacto	Covr 1	Covr .75	Covr .5	Covr .15	Covr .01
Exacto	4.69	4.69	4.69	5.47	5.47	5.47
MRR_5	6.64	6.80	6.80	8.49	9.53	9.53
MRR_{10}	6.77	6.80	6.98	8.90	10.37	10.37
MRR_{25}	7.00	7.02	7.21	9.31	10.78	10.78

Tab. 5.13: Corrida 1 con 200 documentos de Lucene y 20, 40 y 70 pasajes extraídos respectivamente

1) Baseline						
Medida	Exacto	Covr 1	Covr .75	Covr .5	Covr .15	Covr .01
Exacto	7.75	7.75	8.53	10.08	10.08	10.08
MRR_5	8.94	9.25	10.03	12.42	13.39	13.45
MRR_{10}	9.31	9.38	10.16	12.55	14.03	14.10
MRR_{25}	9.54	9.61	10.39	12.89	14.41	14.48

2) Improved Baseline						
Medida	Exacto	Covr 1	Covr .75	Covr .5	Covr .15	Covr .01
Exacto	3.91	3.91	3.91	3.91	3.91	3.91
MRR_5	5.36	5.91	5.91	8.50	9.31	9.31
MRR_{10}	5.96	6.43	6.43	9.02	10.18	10.18
MRR_{25}	5.96	6.43	6.49	9.20	10.46	10.46

3) Lasso						
Medida	Exacto	Covr 1	Covr .75	Covr .5	Covr .15	Covr .01
Exacto	7.81	7.81	8.59	10.16	10.94	10.94
MRR_5	9.27	9.58	10.76	13.52	14.49	14.56
MRR_{10}	9.64	9.71	10.89	13.65	15.15	15.21
MRR_{25}	9.94	10.01	11.18	14.05	15.59	15.65

Tab. 5.14: Corrida 2.1: Generación de Queries 1, 2 y 3 respectivamente

En esta corrida variamos únicamente el método de generación de queries, generando un total de 3 ejecuciones y dejando fijos los siguiente parámetros:

- Idioma: Español
- Cantidad de documentos retornados: 50
- Cantidad de pasajes extraídos: 40
- Método de Temas: 2
- Método de Ranking: 1

Los resultados pueden observarse en la tabla 5.14. En ella salta a la vista que las diferencias son análogas para todas las celdas de las matrices: el método 3) es ligeramente superior al 1) y el 2) queda rezagado por lejos. El método de generación de queries 3, como muestra la tabla, es el método Lasso agregado.

Corrida 2.2: Inferencia de Temas

En esta corrida variamos únicamente el método de inferencia de temas, generando un total de 4 ejecuciones y dejando fijos los siguiente parámetros:

- Idioma: Español

1) Tema Test						
Medida	Exacto	Covr 1	Covr .75	Covr .5	Covr .15	Covr .01
Exacto	6.15	6.15	6.15	6.92	6.92	6.92
MRR_5	7.87	8.41	8.79	10.21	10.85	10.85
MRR_{10}	8.08	8.49	8.87	10.53	11.77	11.77
MRR_{25}	8.29	8.72	9.16	10.91	12.18	12.18

2) Tema NERs						
Medida	Exacto	Covr 1	Covr .75	Covr .5	Covr .15	Covr .01
Exacto	7.75	7.75	8.53	10.08	10.08	10.08
MRR_5	8.94	9.25	10.03	12.42	13.39	13.45
MRR_{10}	9.31	9.38	10.16	12.55	14.03	14.10
MRR_{25}	9.54	9.61	10.39	12.89	14.41	14.48

3) Tema Sustantivos						
Medida	Exacto	Covr 1	Covr .75	Covr .5	Covr .15	Covr .01
Exacto	3.85	3.85	3.85	4.62	4.62	4.62
MRR_5	4.19	4.19	4.45	5.88	6.14	6.14
MRR_{10}	4.19	4.30	4.56	5.99	6.43	6.43
MRR_{25}	4.47	4.58	4.83	6.46	7.08	7.08

4) Tema NERs + Sustantivos						
Medida	Exacto	Covr 1	Covr .75	Covr .5	Covr .15	Covr .01
Exacto	5.47	5.47	6.25	7.03	7.03	7.03
MRR_5	7.12	7.43	8.22	9.77	10.51	10.57
MRR_{10}	7.36	7.43	8.22	9.77	11.03	11.10
MRR_{25}	7.64	7.71	8.49	10.15	11.45	11.52

Tab. 5.15: Corrida 2.2: Métodos 1, 2, 3 y 4 respectivamente

- Cantidad de documentos retornados: 50
- Cantidad de pasajes extraídos: 40
- Método de Generación de Queries: 1
- Método de Ranking de Pasajes: 1

Los resultados pueden observarse en la tabla 5.15 y resulta que el método utilizado (NERs + números de la primer pregunta como target) es el más efectivo en todos los casos.

Corrida 2.3: Ranking de Pasajes

En esta corrida variamos únicamente el método de ranking de pasajes, generando un total de 3 ejecuciones y dejando fijos los siguiente parámetros:

- Idioma: Español
- Cantidad de documentos retornados: 50

- Cantidad de pasajes extraídos: 40
- Método de Generación de Queries: 1
- Método de Temas: 2

Los resultados pueden observarse en la tabla 5.16. Las fórmulas nuevas propuestas no resultaron de utilidad en ningún caso. Viendo la diferencia de resultados entre los métodos 2 y 3, se hace claro que el *LengthScore* aporta, ya que es el único agregado sustantivo en el método 3 y este tiene una mejoría significativa en comparación con el 2. De allí surgió la idea de aplicarlo a la fórmula baseline. Hicimos dos modificaciones: Una ponderando 90 / 10 y otra ponderando 80 / 20 y 70 / 30 dando buenos resultados las tres (Ver tabla 5.17). La fórmula del score es la siguiente:

$$LengthScore = \begin{cases} 1{,}0 & 4 < \#tokens < 100 \\ 0{,}5 & 100 \leq \#tokens < 200 \\ 0{,}0 & \text{En cualquier otro caso} \end{cases}$$

Como se puede ver en 5.17, agregar este valor en el score general del pasaje mejora los resultados. Las causas más evidentes son que el score funciona como un filtro para pasajes mal formados (es decir, que por alguna razón el splitter no logró 'separar' bien) o bien pasajes bien formados pero excesivamente largos. En ambos casos, las herramientas de procesamiento de lenguajes pierden su performance, por lo que pasajes bien rankeados, en el momento de la extracción de la respuesta no son de utilidad. Evaluamos tres ponderaciones -9/1, 8/2 y 7/3-, dando mejores resultados cuanto más se ponderó el score, salvo para Exacto (o MRR_1) que se redujo en la tercer corrida. Dado que esta métrica es la preferida, decidimos quedarnos con esta fórmula (ponderando 8/2 y no 7/3). Eventualmente podría evaluarse otra métricas y descubrir por qué se dio que esta funcionó mejor para MMR_1 y 7/3 funcionó mejor para el resto. Todo parece indicar que por este camino pueden encontrarse mejoras sustantivas con bajo costo de implementación, sea mejorando la ponderación, sea complejizando la formulación del score, que, como puede verse, es realmente simple.

Corrida 3: Combinación de Óptimos

En esta última corrida, utilizamos los métodos que mejores resultados dieron, por sección, todos juntos, primero para español (5.18) y luego para portugués (5.19). Hay dos observaciones importantes. Primero, con respecto a la corrida en español, es notorio como el Exacto y, en general, todos los valores salvo el MRR_5 con matching exacto, tienen una performance peor que en la corrida 2.3, lo cual se debe, seguramente, a una interacción compleja entre los distintos pasos que conduce a este comportamiento.

En segundo lugar, con respecto a la corrida en portugués, es notoria su baja performance en relación con los resultados en español, obteniendo apenas un 3.16 considerando la métrica MRR_5. Existen varios factores para explicar esto. En primer lugar, que las herramientas para portugués no son tan maduras como las herramientas para español; en segundo lugar, que la evaluación de los métodos óptimos y, en general, todo el enfoque al sistema, estuvieron marcados por el español, siendo las adaptaciones para portugués un momento secundario que intentó mostrar, simplemente, que este soporte estaba accesible;

$PassageScore_{bl}$						
Medida	Exacto	Covr 1	Covr .75	Covr .5	Covr .15	Covr .01
Exacto	7.75	7.75	8.53	10.08	10.08	10.08
MRR_5	8.94	9.25	10.03	12.42	13.39	13.45
MRR_{10}	9.31	9.38	10.16	12.55	14.03	14.10
MRR_{25}	9.54	9.61	10.39	12.89	14.41	14.48

$PassageScore_2$						
Medida	Exacto	Covr 1	Covr .75	Covr .5	Covr .15	Covr .01
Exacto	3.10	3.10	3.10	4.65	4.65	4.65
MRR_5	5.06	5.22	5.22	7.51	8.35	8.41
MRR_{10}	5.35	5.50	5.50	7.89	9.18	9.24
MRR_{25}	5.39	5.55	5.55	8.10	9.60	9.67

$PassageScore_3$						
Medida	Exacto	Covr 1	Covr .75	Covr .5	Covr .15	Covr .01
Exacto	5.38	5.38	5.38	6.15	6.15	6.15
MRR_5	6.69	6.85	7.10	9.35	10.69	10.88
MRR_{10}	7.15	7.31	7.56	9.98	11.66	11.85
MRR_{25}	7.19	7.35	7.60	10.21	11.99	12.19

Tab. 5.16: Corrida 2.3: Fórmulas 1, 2 y 3 respectivamente

$PassageScore_{bl} \times 0{,}9 + LengthScore \times 0{,}1$						
Medida	Exacto	Covr 1	Covr .75	Covr .5	Covr .15	Covr .01
Exacto	7.81	7.81	8.59	10.16	10.16	10.16
MRR_5	9.47	9.78	10.56	12.97	13.95	14.14
MRR_{10}	9.71	9.78	10.56	12.97	14.47	14.66
MRR_{25}	10.01	10.08	10.86	13.38	14.92	15.11

$PassageScore_{bl} \times 0{,}8 + LengthScore \times 0{,}2$						
Medida	Exacto	Covr 1	Covr .75	Covr .5	Covr .15	Covr .01
Exacto	**10.08**	**10.08**	**10.85**	**12.40**	**13.18**	**13.18**
MRR_5	11.68	11.99	13.15	15.80	16.58	16.77
MRR_{10}	11.92	11.99	13.15	15.80	16.99	17.18
MRR_{25}	12.27	12.34	13.51	16.26	17.55	17.74

$PassageScore_{bl} \times 0{,}7 + LengthScore \times 0{,}3$						
Medida	Exacto	Covr 1	Covr .75	Covr .5	Covr .15	Covr .01
Exacto	10.00	10.00	10.77	12.31	13.08	13.08
MRR_5	11.94	12.50	13.65	17.04	17.81	18.00
MRR_{10}	12.35	12.67	13.83	17.12	18.32	18.51
MRR_{25}	12.58	12.91	14.06	17.46	18.70	18.90

Tab. 5.17: Corrida 2.3: Combinación métodos 1 y 3

Medida	Exacto	Covr 1	Covr .75	Covr .5	Covr .15	Covr .01
Exacto	10.00	10.00	10.77	12.31	13.08	13.08
MRR_5	**12.00**	12.31	13.46	16.18	16.95	17.01
MRR_{10}	12.24	12.31	13.46	16.26	17.44	17.50
MRR_{25}	12.55	12.62	13.78	16.72	18.00	18.06

Tab. 5.18: Corrida 3.1: Combinación de óptimos para español

Medida	Exacto	Covr 1	Covr .75	Covr .5	Covr .15	Covr .01
Exacto	1.92	2.88	2.88	3.85	3.85	3.85
MRR_5	3.16	5.03	5.03	6.91	7.15	7.15
MRR_{10}	3.45	5.33	5.33	7.33	7.71	7.86
MRR_{25}	4.08	6.00	6.00	8.21	8.84	8.94

Tab. 5.19: Corrida 3.2: Combinación de óptimos sobre portugués

en tercer lugar, también es posible que el sistema per se y el enfoque, aún sin otro inconveniente, se desempeñe peor para el set de preguntas para el portugués. A fin de cuentas, no hay que perder de vista que los sets de test usables por nuestro sistema son de 130 preguntas para español y 104 para portugués, a partir de los cuales no es posible extrapolar un comportamiento general, sino presentar resultados concretos.

Datos de la corrida 3.1, para español:

- Idioma: Español
- Cantidad de documentos retornados: 50
- Cantidad de pasajes extraídos: 40
- Método de Generación de Queries: 3
- Método de Temas: 2
- Fórmula de Ranking de Pasajes: $PassageScore_{bl} \times 0{,}8 + LengthScore \times 0{,}2$

Datos de la corrida 3.2, para portugués:

- Idioma: Portugués
- Cantidad de documentos retornados: 50
- Cantidad de pasajes extraídos: 40
- Método de Generación de Queries: 3
- Método de Temas: 2
- Fórmula de Ranking de Pasajes: $PassageScore_{bl} \times 0{,}8 + LengthScore \times 0{,}2$

5.2.4. Conclusiones, limitaciones y trabajo futuro

El sistema implementado en este capítulo es un sistema de question answering de dominio abierto con soporte para diferentes idiomas. El mismo utiliza como base el código de Qanus y Qasys - monolingüe, con soporte solo para el idioma inglés - y agrega sobre el mismo las adaptaciones necesarias para trabajar con diferentes idiomas - con los límites mencionados-, utilizando la librería de procesamiento de lenguajes Freeling y, en consecuencia, soportando tantos idiomas como esta soporta. Agregamos, asimismo, mejoras simples en los módulos de recuperación de documentos (la incorporación del algoritmo Lasso de generación de queries) y también en el de generación de respuestas (la introducción de nuevos scores generales). Para ejecutar y poder evaluar la performance de nuestro sistema con soporte multilingüe baseline, así como de estas diferentes mejoras y diferentes configuraciones del sistema, tomamos una tarea monolingüe de la competencia CLEF '07 y evaluamos nuestro modelo sobre una parte limitada de la misma, para los idiomas español y portugués (lo hicimos, en realidad, en un principio, para español y para inglés, para darnos cuenta finalmente que las respuestas esperadas para la tarea en inglés no estaban disponibles). Las partes que evaluamos consisten, en concreto, en preguntas de tipo factoid con respuesta válida en entradas de wikipedia.

Para darnos una idea de la escala de valores manejados, debemos señalar que, en primer lugar, en el documento retrospectivo de CLEF '07 se señala que "las tareas fueron de una complejidad elevada y en comparación con los resultados del año anterior, la mejor precisión general bajó del 49 % al 41.75 % para tareas multi-lingües, mientras que, más significativamente, bajó de 68 % a 54 % en tareas monolingües." En nuestro trabajo, tomamos tareas monolingües - con soporte para varias lenguas, pero monolingües - y nos restringimos a una subsección sencilla (eliminando preguntas de tipo DEFINITION y LIST), por lo que no es del todo válido comparar esta performance con la nuestra. Por otro lado, de Qasys, el sistema baseline que adaptamos para soportar diferentes idiomas, sabemos que en la TREC 07 (en inglés), competencia en la que Qasys participó, el mejor sistema, LumbaPA07, obtuvo una precisión del 70,06 % considerando, como ya mencionamos, solo respuestas exactas, mientas que el décimo (Quanta) obtuvo 20,6 % y Qasys logró un 11,9 %.

Nuestro sistema adaptado al español, con todas las mejoras en su óptimo, logra un nada despreciable 10,08 % de precisión con resultados exactos y un 12,00 % para MRR_5.

Es importante destacar, con respecto a este modelo de dominio abierto, un punto clave que saltó a la vista tanto en la reseña bibliográfica como en el código del sistema baseline Qanus, que es la limitación de las mejoras a zonas acotadas y sin enfoques sistémicos. Una mejora argumentada y con cierta presencia en la literatura del área fue el mecanismo de generación de queries que llamamos Lasso (por el sistema en el que fue implementado por primera vez) y nosotros lo incorporamos aquí, logrando mejoras mínimas en la performance, pasando de 7.75 a 7.81 en Exacto - MRR_1 - (+0.26), de 8.94 a 9.27 en MRR_5 (+0.31). Más allá de esta mejora, los puntos en los que podríamos haber enfocado nuestras mejoras eran un incremento en la eficacia de las estrategias de generación de respuestas con enfoques ad-hoc basados en nuestra intuición y en la experimentación. Esto podría ser una falla en nuestra investigación, pero más bien parece que no están estudiados ni sistematizados procedimientos sistémicos a la hora de enfrentar, en concreto, el paso de la generación de respuestas. Por su parte, un subsistema muy interesante que no pudimos encontrar y que representaría un gran avance para el área es un módulo open source que permita extraer el *focus* de las preguntas. Este módulo sí representa un problema acotado y bien definido.

El sistema que hacemos público en este trabajo (y que puede encontrarse, como dijimos,

en `http://github.com/julian3833/multilingual-qanus`, con sus límites, es un buen baseline para emprender trabajos de question answering de dominio abierto sobre bases de conocimiento no estructuradas con soporte para diferentes idiomas.

Con respecto a las limitaciones y al trabajo futuro, la lista de posibles mejoras tiene virtualmente una longitud arbitraria, pero los siguientes son los puntos, a nuestro entender, claves:

- El primer límite a mencionar es la falta de un módulo multilingüe de Question Classification. Como ya mencionamos, utilizamos la existencia de una traducción de las preguntas al inglés para utilizar el módulo en inglés de Stanford, pero nuestro sistema no dispone de un módulo multilingüe de QC. Como ya explicamos, esto es así porque no existen tales implementaciones, al menos, al momento de escribir esta tesis. Una forma sencilla de enfrentar esta situación es utilizar el QC de Stanford en inglés, anteponiendo una traducción del idioma de origen al inglés. Otra posibilidad es una implementación manual por idioma de un módulo basado en reglas escritas a mano con un sistema de clases simple. Este acercamiento tiene un problema que es que todo el módulo de generación de respuestas está basado en casos relacionados con las clases del QC de Stanford, así que haría falta afectar ese módulo también.

- Otro punto débil, aunque restringido a esta tarea particular, es la inferencia del tópico a partir del primer par de pregunta-respuesta: este procedimiento podría mejorarse enormemente.

- La ya mencionada generación del *focus*, que definitivamente tiene la entidad de un trabajo aparte y no una mera mejora para este sistema.

- El soporte de preguntas de tipo definition y list y el soporte de cláusulas de temporales.

- Mejorar concretas para los casos de generación de respuestas, que son realmente muy básicos. Seguramente con un poco de inventiva y experimentación la performance del sistema puede mejorarse sustancialmente agregando algunas condiciones y expresiones regulares a mano en los casos de generación de respuestas, así como agregando más casos basados en las clases finas de clasificador de preguntas.

- Una mejora no necesaria pero interesante sería agregar la web como base de conocimiento.

Con esta lista terminamos la presentación de nuestro sistema open domain con soporte multilingüe y, también de nuestra tesis.

Apéndices

Apéndice A

HERRAMIENTAS

A.1. Herramientas de Stanford

En este apéndice nos detendremos principalmente en el Question Clasiffier ya que es este el más interesante para nuestro trabajo, y los otros dos módulos fueron suficientemente tratados en el capítulo 2, aunque agregamos aquí algunas referencias a documentos importantes para ahondar en ellos.

Baste mencionar que el POS tagger de Stanford es un algortimo basado en entropía máxima (*maximum entropy*), implementado en java originalmente en [Toutanova and Manning, 2000], con algunos agregados y mejoras técnicas realizadas en [Toutanova et al., 2003]. Al descargar el paquete, también está disponible uno más complejo con soporte para los idiomas árabe, chino y alemán. En este trabajo utilizamos el paquete sencillo, que consta de dos modelos entrenados para inglés, usando, como señalamos en 2.3.1 Part-of-speech (POS) tagging, el tagset de Penn Treebank. Por su parte, el NER tagger de Stanford, es un algoritmo conocido como CRF Classifier (Conditional Random Field) y permite el entrenamiento con modelos propios, mientras que los modelos incluidos por defecto incorporan tres clases (PERSON, ORGANIZATION, LOCATION) y hay disponibles otros modelos para otros idiomas. Para más detalles, el clasificador implementa -con algunas variaciones- el modelo definido en [Finkel et al., 2005].

Finalmente, el Question Classifier de Stanford [Li and Roth, 2002] es un sistema de clasificación de preguntas basado en machine learning que utiliza la arquitectura de aprendizaje SNoW. Es un sistema jerárquico guiado por una semántica de dos niveles que permite la clasificación en categorías granulares. Según los tests de los autores, el proceso logra una efectividad del 90 % sobre 50 diferentes clases finales, utilizando features sintácticos y semánticos.

El clasificador dispone de una taxonomía de dos capas basada en los tipos de respuesta típicos de la TREC. Las clases superiores son seis: abreviatura, entidad, descripción, humano, lugar y numérico; divididas a su vez en 50 subclases más finas no solapadas (50 en total). La motivación de la existencia de clases superiores es doble: por un lado, ellas preservan compatibilidad y coherencia con las clases usadas en trabajos previos de clasificación de preguntas. Por otro, se esperaba obtener mejoras de performance aplicando dos fases de clasificación, pero esta intención no se corroboró en la experiencia

La tabla siguiente muestra la taxonomía de clasificación dividida en clases y subclases [1]

Clase y subclase	Definición
ABBREVIATION	abbreviation
abb	abbreviation

[1] Puede encontrarse en esta url: `http://cogcomp.cs.illinois.edu/Data/QA/QC/definition.html` y está explicada en [Li and Roth, 2002] y [Manning and Klein, 2003]

exp	expression abbreviated
ENTITY	entities
animal	animals
body	organs of body
color	colors
creative	inventions, books and other creative pieces
currency	currency names
dis.med.	diseases and medicine
event	events
food	food
instrument	musical instrument
lang	languages
letter	letters like a-z
other	other entities
plant	plants
product	products
religion	religions
sport	sports
substance	elements and substances
symbol	symbols and signs
technique	techniques and methods
term	equivalent terms
vehicle	vehicles
word	words with a special property
DESCRIPTION	description and abstract concepts
definition	definition of sth.
description	description of sth.
manner	manner of an action
reason	reasons
HUMAN	human beings
group	a group or organization of persons
ind	an individual
title	title of a person
description	description of a person
LOCATION	locations
city	cities
country	countries
mountain	mountains
other	other locations
state	states
NUMERIC	numeric values
code	postcodes or other codes
count	number of sth.
date	dates
distance	linear measures

money	prices
order	ranks
other	other numbers
period	the lasting time of sth.
percent	fractions
speed	speed
temp	temperature
size	size, area and volume
weight	weight

Uno de los principales problemas que tuvieron los autores al enfrentarse a clases tan específicas fue la ambigüedad instrínseca de ciertas preguntas. Los siguientes ejemplos de este problema están tomados de [Li and Roth, 2002]:

- What is bipolar disorder? (¿Qué es el desorden bipolar?)
- What do bats eat? (¿Qué comen los murciélagos?)

La primer pregunta puede pertenecer a la clase *definición* o bien a la clase *enfermedad / medicina* y la segunda a *comida*, *planta* o *animal*. Para abordar este problema, el clasificador asigna diferentes clases ponderadas y no una única clase.

Los dos niveles de clasificación están implementados como dos clasificadores simples en secuencia, ambos utilizando el algoritmo Winnow de SNoW. El primero etiqueta la pregunta en función de las clases más generales y el segundo asigna las clases más finas (dentro de las suclases determinadas por la clase del primero). El modelo para el algoritmo de aprendizaje representa las preguntas como listas de características (*features*) tanto sintácticas como semánticas. Los features utilizados son, en total, más de 200.000, siendo casi todos combinaciones complejas de un set acotado de features simples basados en palabras, pos tags, chunks (componentes constitucionales de la oración), chunks principales (por ejemplo: componente nominal principal), entidades nombradas y palabras semánticamente relacionadas a ciertas clases. De estos seis tipos de features primitivos, tres son sintácticos (pos tags, chunks, chunks principales) mientras que otros son semánticos (named entities, palabras relacionadas).

El clasificador, al igual que el resto de las herramientas de nlp de Stanford, está implementado en java.

A.2. Freeling

Freeling es una librería de código abierto que provee distintas herramientas de procesamiento de lenguaje natural, desarrollada y mantenida por el Centre de Tecnologies i Aplicacions del Llenguatge i la Parla (TALP) de la Universidad Politécnica de Cataluña (UPC). Freeling está diseñado para ser usada como una librería externa y ofrece un API en distintos lenguajes de programación. Su principal virtud es ser multilingüe, esto es, los diferentes analizadores que provee funcionan para un conjunto bastante amplio de idiomas. La última versión a la fecha (3.1) soporta los siguientes idiomas:

- Asturian (as)
- Catalan (ca)
- English (en)
- French (fr)
- Galician (gl)
- Italian (it)
- Portuguese (pt)
- Russian (ru)
- Slovene (sl)
- Spanish (es)
- Welsh (cy)

Cabe destacar que no todos los módulos soportan todos los idiomas. Sin embargo, dado que el proyecto está radicado en España, los idiomas necesarios para los fines de nuestro trabajo (español, portugués e inglés), soportan todos los módulos disponibles en la librería. Freeling 3.1 ofrece los siguientes analizadores lingüísticos:

- Detección de idioma
- Tokenizer
- Sentence splitting
- Análisis morfológico
- NER y NEC (Detección y Clasificación de Entidades Nombradas)
- Reconocimiento de fechas, números, magnitudes físicas, monedas
- Codificación fonética
- POS tagging
- Shallow parsing
- Dependency parsing
- Wordnet-based sense annotation
- Word Sense Disambiguation
- Coreference resolution

El módulo de POS tagging de Freeling dispone de dos tagger. El primero, *hmm_tagger*, es un algoritmo clásico que utiliza Hidden Markov Models (HMMs) con tri-gramas. La descripción del algoritmo de tagging basado en HMM se encuentra con detalle en en [Jurafsky, 2000]. Por otro lado, el módulo incorpora un método llamado *relax_tagger* que permite la creación de un sistema híbrido que permite reglas escritas a mano y modelos estadísticos.

A.3. Apache Lucene

Lucene es una librería de information retrieval, de código abierto, escrita en Java y distribuida bajo la licencia Apache Software License por la Apache Software Foundation. No está pensada para usuarios finales sino para ser integrada dentro de proyectos informáticos, resolviendo la parte de bajo nivel y brindando servicios a través de un API en diferentes lenguajes de programación. Su core es un índice invertido como el que describimos anteriormente. La implementación de un sistema que utiliza Lucene consta de dos pasos separados:

- La **creación** del índice, es por lo general un proceso offline en el cual se incorporan distintas fuentes de información al índice
- La **búsqueda** de documentos en el índice creado en el paso anterior, a partir de una query ingresada por el usuario final. Este proceso se incorpora dentro del flujo 'online' del sistema. El resultado de esta búsqueda es una lista de documentos rankeados con un cierto puntaje.

Es importante señalar que si bien el proceso de creación del índice suele estar desacoplado del resto del sistema, las fuentes de información no tiene por que ser 'offline' en el sentido de ser documentos en un disco local. De hecho, Nutch, otro proyecto de código abierto de la Apache Software Foundation es un motor de búsqueda web basado en Lucene que incorpora un crawler para indexar sitios web. Lucene soporta cualquier fuente de información que pueda convertirse en texto mediante algoritmia.
Los conceptos fundamentales de Lucene son: índice, documento, campo, término y query.

- Un índice contiene un conjunto de documentos
- Un documento es un conjunto de campos
- Un campo es el nombre de una secuencia de términos
- Un término es un token (una palabra)
- Una query es una lista de términos conectados con distintos operados lógicos

Bibliografía

[Adolphs et al., 2011] Adolphs, P., Theobald, M., Schäfer, U., Uszkoreit, H., and Weikum, G. (2011). Yago-qa: Answering questions by structured knowledge queries. In *ICSC*, pages 158–161. IEEE.

[Allam and Haggag, 2012] Allam, A. M. N. and Haggag, M. H. (2012). The question answering systems: A survey. In *International Journal of Research and Reviews in Information Sciences (IJRRIS), Vol. 2, No. 3.* Science Academy Publisher, United Kingdom.

[Bach and Badaskar, 2007] Bach, N. and Badaskar, S. (2007). A Review of Relation Extraction.

[Bouma et al., 2011] Bouma, G., Fahmi, I., and Mur, J. (2011). Relation extraction for open and closed domain question answering. In Bosch, A. and Bouma, G., editors, *Interactive Multi-modal Question-Answering*, Theory and Applications of Natural Language Processing, pages 171–197. Springer Berlin Heidelberg.

[Chung et al., 2004] Chung, H., Song, Y., Han, K., Yoon, D., Lee, J., Rim, H., and Kim, S. (2004). A practical qa system in restricted domains. In *Proceedings of the ACL Workshop on Question Answering in Restricted Domains.*

[Clef, 2007a] Clef (2007a). Guidelines for participants qa@clef 2007. `http://celct.fbk.eu/QA4MRE/scripts/downloadFile.php?file=/websites/ResPubliQA/resources/past_campaigns/2007/Documentation/QA_2007_Track_Guidelines.pdf`.

[Clef, 2007b] Clef (2007b). Overview of the clef 2007 multilingual question answering track. `http://celct.fbk.eu/QA4MRE/scripts/downloadFile.php?file=/websites/ResPubliQA/resources/past_campaigns/2007/Documentation/QA07_Overview_Working_Notes.pdf`.

[Cooper and Rüger, 2000] Cooper, R. J. and Rüger, S. M. (2000). A simple question answering system. In *TREC.*

[Dang et al., 2008] Dang, H. T., Kelly, D., and Lin, J. (2008). Overview of the TREC 2007 question answering track. In *Proceedings TREC 2007.*

[Fader et al., 2011] Fader, A., Soderland, S., and Etzioni, O. (2011). Identifying relations for open information extraction. In *EMNLP*, pages 1535–1545. ACL.

[Ferrucci et al., 2010] Ferrucci, D. A., Brown, E. W., Chu-Carroll, J., Fan, J., Gondek, D., Kalyanpur, A., Lally, A., Murdock, J. W., Nyberg, E., Prager, J. M., Schlaefer, N., and Welty, C. A. (2010). Building watson: An overview of the deepqa project. *AI Magazine*, 31(3):59–79.

[Finkel et al., 2005] Finkel, J. R., Grenager, T., and Manning, C. (2005). Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 363–370, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Green et al., 1961] Green, Jr., B. F., Wolf, A. K., Chomsky, C., and Laughery, K. (1961). Baseball: An automatic question-answerer. In *Papers Presented at the May 9-11, 1961, Western Joint IRE-AIEE-ACM Computer Conference*, IRE-AIEE-ACM '61 (Western), pages 219–224, New York, NY, USA. ACM.

[Hermjakob, 2001] Hermjakob, U. (2001). Parsing and question classification for question answering. In *Proceedings of the Workshop on Open-domain Question Answering - Volume 12*, ODQA '01, pages 1–6, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Herrera et al., 2004] Herrera, J., Peñas, A., and Verdejo, F. (2004). Question answering pilot task at clef 2004. In Peters, C., Clough, P., Gonzalo, J., Jones, G. J. F., Kluck, M., and Magnini, B., editors, *CLEF*, volume 3491 of *Lecture Notes in Computer Science*, pages 581–590. Springer.

[Jun-Ping Ng, 2010] Jun-Ping Ng, M.-Y. K. (2010). Qanus: An open-source question-answering plataform. *CS Department National University of Singapore.*

[Jurafsky, 2000] Jurafsky, D. (2000). *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition pp. 286-314.* Pearson Prentice Hall, Upper Saddle River, N.J.

[Kalyanpur et al., 2012] Kalyanpur, A., Boguraev, B., Patwardhan, S., Murdock, J. W., Lally, A., Welty, C., Prager, J. M., Coppola, B., Fokoue-Nkoutche, A., 0007, L. Z., Pan, Y., and Qiu, Z. (2012). Structured data and inference in deepqa. *IBM Journal of Research and Development*, 56(3):10.

[Lampert, 2004] Lampert, A. (2004). A quick introduction to question answering.

[Lehnert, 1986] Lehnert, W. (1986). Readings in natural language processing. chapter A Conceptual Theory of Question Answering, pages 651–657. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Li and Roth, 2002] Li, X. and Roth, D. (2002). Learning question classifiers. In *Proceedings of the 19th International Conference on Computational Linguistics - Volume 1*, COLING '02, pages 1–7, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Lin, 2007] Lin, J. J. (2007). An exploration of the principles underlying redundancy-based factoid question answering. *ACM Trans. Inf. Syst.*, 25(2).

[Magnini et al., 2003] Magnini, B., Romagnoli, S., Vallin, A., Herrera, J., Penas, A., Peinado, V., Verdejo, F., de Rijke, M., and Vallin, R. (2003). The multiple language question answering track at clef 2003. In *CLEF 2003. CLEF 2003 Workshop*. Springer-Verlag.

[Manning and Klein, 2003] Manning, C. and Klein, D. (2003). Optimization, maxent models, and conditional estimation without magic. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology: Tutorials - Volume 5*, NAACL-Tutorials '03, pages 8–8, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Manning et al., 2008] Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.

[Moldovan et al., 2000] Moldovan, D., Harabagiu, S., Pasca, M., Mihalcea, R., Girju, R., Goodrum, R., and Rus, V. (2000). The structure and performance of an open-domain question answering system. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, ACL '00, pages 563–570, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Nadeau and Sekine, 2007] Nadeau, D. and Sekine, S. (2007). A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26.

[Padró, 2011] Padró, L. (2011). Analizadores multilingües en freeling. *Linguamatica*, 3(2):13–20.

[Padró and Stanilovsky, 2012] Padró, L. and Stanilovsky, E. (2012). Freeling 3.0: Towards wider multilinguality. In *Proceedings of the Language Resources and Evaluation Conference (LREC 2012)*, Istanbul, Turkey. ELRA.

[Pires, 2012] Pires, R. (2012). A common evaluation setting for just.ask, open ephyra and aranea qa systems. *CoRR*, abs/1205.1779.

[Popescu et al., 2003a] Popescu, A.-M., Etzioni, O., and Kautz, H. A. (2003a). Towards a theory of natural language interfaces to databases. In *IUI*, pages 149–157. ACM.

[Popescu et al., 2003b] Popescu, A.-M., Etzioni, O., and Kautz, H. A. (2003b). Towards a theory of question answering inferfaces to databases".

[Ravichandran and Hovy, 2002] Ravichandran, D. and Hovy, E. (2002). Learning surface text patterns for a question answering system. In *Proc. ACL2002*.

[Sacaleanu et al., 2008] Sacaleanu, B., Orasan, C., Spurk, C., Ou, S., Ferrández, O., Kouylekov, M., and Negri, M. (2008). Entailment-based question answering for structured data. In Ramsay, A. and Bontcheva, K., editors, *COLING (Demos)*, pages 173–176.

[Shuyo, 2010] Shuyo, N. (2010). Language detection library for java. `http://code.google.com/p/language-detection/`.

[Tellex et al., 2003] Tellex, S., Katz, B., Lin, J. J., Fernandes, A., and Marton, G. (2003). Quantitative evaluation of passage retrieval algorithms for question answering. In *SIGIR*, pages 41–47. ACM.

[Tomás et al., 2008] Tomás, D., Vicedo, J. L., Bisbal, E., and Moreno, L. (2008). Trainqa: a training corpus for corpus-based question answering systems.

[Toutanova et al., 2003] Toutanova, K., Klein, D., Manning, C. D., and Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 173–180, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Toutanova and Manning, 2000] Toutanova, K. and Manning, C. D. (2000). Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *Proceedings of the 2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora: Held in Conjunction with the 38th Annual Meeting of the Association for Computational Linguistics - Volume 13*, EMNLP '00, pages 63–70, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Varelas et al., 2005] Varelas, G., Voutsakis, E., Petrakis, E. G. M., Milios, E. E., and Raftopoulou, P. (2005). Semantic similarity methods in wordnet and their application to information retrieval on the web. In *In: 7 th ACM Intern. Workshop on Web Information and Data Management (WIDM 2005*, pages 10–16. ACM Press.

[Voorhees and Tice, 1999] Voorhees, E. M. and Tice, D. M. (1999). The trec-8 question answering track evaluation. In *In Text Retrieval Conference TREC-8*, pages 83–105.

[Wang, 2006] Wang, M. (2006). A survey of answer extraction techniques in factoid question answering. In *In Proceedings of the Human Language Technology Conference and North American Chapter of the Association for Computational Linguistics (HLT-NAACL).*