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

Prototipo de Question Answering bilingüe closed y open domain

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

Julián Peller

Director: José Castaño

Buenos Aires, 2013

PROTOTIPO DE QUESTION ANSWERING BILINGÜE PARA CLOSED Y OPEN DOMAIN

Question Answering es un área de ciencias de la computación que combina herramientas de búsqueda y recuperación de la información (*information retrieval*) y de procesamiento del lenguaje natural (*nlp*), buscando generar respuestas a preguntas formuladas en algún lenguaje humano. Por ejemplo, para el input *"¿Cuándo nació Noam Chomsky?"* un sistema de QA debe devolver "*7 de diciembre de 1928*". Existen varios sistemas de QA conocidos: quizás los más populares sean Wolfram Alpha e IBM-Watson, el sistema que venció a los campeones humanos del programa de televisión estadounidense Jeopardy! en tiempo real. Desde hace un tiempo, el buscador de Google está implementado también, lentamente procesos de QA. El campo de investigación está muy lejos de cerrarse y los resultados son aún muy básicos debido a la complejidad inherente al problema.

En esta tesis investigamos diferentes enfoques al problema de question answering y diseñamos e implementanos un modelo simple de question answering closed domain para una base de datos y otro open domain sobre wikipedia, ambos bilingües.

Palabras claves: Question Answering, Closed Domain, Open Domain, Multilenguaje, Information Retrieval, Question processing, Procesamiento del lenguaje natural

Índice general

1.1. Qué es Question Answering

Question Answering es un área de ciencias de la computación que combina herramientas de búsqueda y recuperación de la información (*information retrieval*) y de procesamiento del lenguaje natural (*nlp*), buscando generar respuestas a preguntas formuladas en algún lenguaje humano. Por ejemplo, para el input *"¿Cuándo nació Noam Chomsky?"* un sistema de QA debe devolver "*7 de diciembre de 1928*". Existen varios sistemas de QA conocidos: quizás los más populares sean Wolfram Alpha e IBM-Watson, el sistema que venció a los campeones humanos del programa de televisión estadounidense Jeopardy! en tiempo real[1]. Desde hace un tiempo, el buscador de Google está implementado también, lentamente procesos de QA. El campo de investigación está muy lejos de 'cerrarse' y los resultados son aún muy básicos debido a la complejidad inherente al problema. El dominio de problemas está vinculado con la *web semántica*, ya que en ambos lo que se busca es dotar de información semántica los datos "planos" disponibles en un corpus y también escribir sistemas capaces de manejar nociones semánticas al tratar con estos datos.

1.1.1. Information Retrieval

El problema conocido como ***information retrieval*** consiste, formalmente, en retornar *información relevante* para una *consulta* (*information need*) a partir de una *base de conocimiento*. El caso de uso típico, ilustrado por los motores de búsqueda web, consiste en devolver una lista de documentos relevantes, en orden de relevancia, para una consulta del usuario. En estos sistemas, las consultas se interpretan como una serie de tokens concatenados con distintos operadores lógicos binarios, donde la barra espaciadora funciona como or inclusivo. El core del sistema es un *índice de búsqueda* de los documentos de la base de conocimiento, que podemos pensar como un diccionario indexado de palabras en documentos (más adelante tocaremos este tema). 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 de una lista de documentos estáticos hasta el ejército de *spiders* de google indexando la web en tiempo real.

1.1.2. Natural Language Processing

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

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

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 estructura gramatical 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.

1.1.3. Distintos problemas

Los sistemas de QA 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.

Algunos de estos subproblemas tienen nombre propio en la literatura y son, por así decirlo, toda un área aparte. Por ejemplo, dependiendo de la amplitud de la base de conocimiento, un sistema puede ser *closed domain*, si la base es acotada a un dominio de la realidad específico, y *open domain*, si no lo es, es decir, si se espera que sepa responder preguntar 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: este 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 query 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.

1.2. Estructura de la tesis

En este proyectos nos proponemos investigar y evaluar herramientas de procesamiento de lenguaje y diferentes enfoques existentes al problema de Question-Answering. Para esto implementamos un modelo simple de QA estructurado de dominio cerrado sobre una base de datos (ver 1.2.1 Dominio cerrado: Mitic)y otro no estructurado sobre diferentes versiones de wikipedia (ver 1.2.2 Dominio abierto: Clef '07). El foco principal es la capacidad de incorporar nuevas herramientas de procesamiento de lenguaje, en particular,

herramientas para otros idiomas de los implementados. Así, utilizamos la librería freeling que permite la fácil adaptación del sistema a nuevos idiomas. Mientras el approach estructurado resulta imposible de evaluar debido a su dominio acotado, sí resulta posible evaluar estas herramientas al utilizarlas en el modelo no estructurado, gracias a las distintas competencias cuyo *leitmotiv* es, justamente, la creación de criterios únicos de evaluación al nivel de la comunidad de investigadores del área[2]. El resultado de la tesis es un modelo básico de interfaz qa a la base de datos del proyecto mitic, cuya implementación está disponible online[3] y un modelo no estructurado open domain que resuelve algunas subtareas de la competencia Clef '07.

1.2.1. Dominio cerrado: Mitic

Esta sección de la tesis está vinculada con un marco de trabajo común entre el Grupo de Autómatas, Lenguajes, Lingüística e Información (GALLI)[4], del Departamento de Computación de la Facultad y la Fundación Sadosky[5], relacionado con la construcción de una base de conocimiento y la recuperación de la información en el dominio de la investigación y la producción relacionada con TICs en Argentina: el proyecto Mitic. Mitic[6] es una plataforma web que permite buscar investigadores, empresas, universidades, proyectos y organismos que se encuentren trabajando en temáticas relacionadas con las TIC, y explorar además las relaciones existentes entre ellos. El grafo de relaciones generado por este proyecto es la base de datos sobre la que trabajamos en la sección closed domain de esta tesis. La base de conocimiento generada es estructurada y consta de una serie de entidades: investigadores, universidades, publicaciones, proyectos, empresas y temáticas, con diferentes tipos de relaciones. Esta base de conocimiento está definida como un grafo de relaciones con diferentes pesos y valores entre las entidades, a partir de lo que originalmente era una recopilación heterogénea de datos en xml. El grafo de Mitic tiene relaciones directas y también inferidas, con un cierto grado de confiabilidad (peso) para cada relación. Tomando como punto de partida esta base de conocimiento, este proyecto se propone construir un sistema de question answering biingüe de dominio cerrado, expandible a más idiomas. Para esto, definimos sobre la base de conocimiento una ontología rígida, que determina el conjunto de preguntas tratables, y utilizamos herramientas de procesamiento de lenguaje natural para clasificar las consultas del usuario dentro de este esquema. Para la ontología , definimos una interfaz única para las entidades que permite la integración de nuevos tipos con un esfuerzo mínimo. La base de datos está principalmente en español, solo algunos títulos y resumenes de publicaciones están en inglés, pero la ontología definida es formal y no tiene idioma. Para la traducción o el mappeo de la pregunta a esta esquema utilizamos analizadores de distintas librerías, descriptas en 2 Marco teórico y en A Herramientas. En particular, ver A.3 Freeling, librería de procesamiento de lenguaje natural multilingüe, que es la principal herramienta utilizada a la hora de traducir preguntas en distintos idiomas al mismo lenguaje formal. La base de datos con la que comenzamos era una serie de xmls que requerían un esfuerzo de modelado bastante alto, pero finalmente obtuvimos esta base de datos de mongdb dónde los modelos estaban mucho mejor definidos.

[2] Mencionar un poco de historias de competencias de procesamiento multilingüe
[3] `urldelsistema`
[4] `http://www.galli.dc.uba.ar/`
[5] `http://www.fundacionsadosky.org.ar/`
[6] `http://www.fundacionsadosky.org.ar/es/Programas-Proyectos/quien-es-quien#mitic`

1.2.2. Dominio abierto: Clef '07

Por su parte, para desarrollar y evaluar mecanismos de dominio abierto resolvimos algunos ejercicios de la competencia de QA 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 multilenguaje, con diferentes corpus 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[7].

Existen distintas conferencias de evaluación de sistemas QA o de subtareas asociadas (por ejemplo TREC - Text Retrieval Conference [8]-, TAC - Text Analysis Conference [9]) - y, a su vez, estas distintas competencias ofrecen distinos 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. Otra razón fue la disponibilidad y el atractivo de la base de conocimiento para estos ejercicios: utilizan snapshots de wikipedia. La competencia del '07 ofrece dos tipos 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

Los ejercicios consideran los siguientes idiomas: inglés, búlgaro, alemán, español, italiano, francés, holandés, rumano y portugués. Por su parte, algunos ejercicios utilizan corpus de datos privados de la competencia y otros utilizan como fuente las distintas wikipedias. De los ejercicios 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. A su vez, implementamos esta misma solución para el inglés, dado que estaban disponibles las preguntas y señalados los links a los snapshots 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 que 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?

[7] `http://celct.fbk.eu/QA4MRE/`
[8] `http://trec.nist.gov/`
[9] `http://www.nist.gov/tac/`

- ¿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.

Estructura

El trabajo se estructura de la siguiente manera. En el capítulo 2 Marco teórico se hará una breve introducción terminológica y conceptual algunas de las herramientas principales de information retrieval y procesamiento de lenguaje natural utilizadas en esta tesis, complementada por el detalle técnico más concreto expuesto en A Herramientas. En el capítulo 3 Estado de Arte pasamos revista de varias implementaciones de sistemas de question answering, haciendo especial foco en el sistema de IBM -Watson- y de su arquitectura DeepQA como ejemplo de un proyecto exitoso de question answering, mientras también comentamos otros sistemas -Qanus, OpenEphyra, Aranea y Just.Ask-, haciendo énfasis en Qanus, debido a que su uso en un modelo inicial del proyecto. Finalmente, en el capítulo 4 Implementación describiremos nuestra implementación de los modelos closed y open domain para la base de datos de Mitic y para los ejercicios de la competencia CLEF '07.

2. MARCO TEÓRICO

En este capítulo vamos a recorrer rápidamente algunos términos básicos y problemas de procesamiento de lenguaje e information retrieval que deben mencionados para poder adentrarnos en el tema de esta tesis. En la primer sección vamos a introducir terminología básica como *token*, *n-grama*, *ontología* y $feature$, mientras que en la segunda veremos algunos problemas o áreas -bien resueltos o no- de procesamiento de lenguajes y recuperación de la información en los que los sistemas de question answering se apoyan o pueden apoyarse. Primero comentaremos qué es un índice invertido de búsqueda en information retrieval, luego comentaremos problemas propios de procesamiento de lenguajes: POS tagging o etiquetado gramatical, NE tagging o reconocimiento de entidades y clasificación de preguntas. Finalmente, mencionamos algunas otras herramientas típicas que tiene sentido utilizar en un sistema de question answering, aún cuando nosotros no las aplicamos en este proyecto.

2.1. Términos básicos

Tokens

Un token es una palabra o un signo de puntuación 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" }

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.

Bajar a tierra on Google N-grams y POS de HMM

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 intrinsecas. Aplicado a ciencias informáticas, principalmente dentro áreas como inteligencia artificial y representación del conocimiento, esta noción original se sostiene, acotando la noción de realidad última 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 de problemas que se pueden plantear (y resolver) para el dominio.

Bajar a tierra con YAGO, dbPedia y Freebase

Features

En machine learning, un feature puede definirse como una propiedad medible de un fenómeno siendo observado. En las aplicaciones de procesamiento de lenguaje natural, 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:

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

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

Los features aparecen en muchos contextos, por ejemplo: en 4 Implementación 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 (2.2 Herramientas), 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, reproducimos los ejemplos que se pueden encontrar en [Nadeau and Sekine, 2007] sobre dimensiones de features típicas para el proceso de reconocimiento de entidades. En este proceso, se está intentando encontrar entidades nombradas (Buenos Aires, Juán Perez, veremos de esto en muy poco tiempo). A continuación listamos 3 sets de ejemplos de tipos de features basados en propiedades de las palabras mismas, en diccionarios conocidos y en propiedadades de los documentos del corpus.

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 apóstrofre
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 (matchs con expresiones regulares) Cantidad de tokens, cantidad de carácteres 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, aerolineas, 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, parrafo, documento
Metadatos	URI, mail headers, sección del xml imágenes asociadas
Frecuencia en corpus	Frecuencia de la palabra o la frase Co-ocurrencia Permanecia de una unidad de varias palabras

2.2. Herramientas

2.2.1. Indice 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 el nombre invertido). 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}

Mencionar otras estructuras de IR y quizas índices basados en LSA (link a LASSO?)

2.2.2. 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."

El resultado de un POS-tagger podría ser el siguiente:

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

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 categorias gramaticales tal vez sea el siguiente, de 9 clases:

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

Mientas 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 funcionaal en la construcción de la oración y, como una nota más, son 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 a 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 siguiente tabla:

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*)[1], una organización europea que fomenta la investigación 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 posiciones, cada una de las cuales expresa una caracteristica de la palabra dependiendo del valor especificado en la posición anterior. 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’). A continuación presentamos la especificación para el conjunto de tags relacionados con sustantivos y un ejemplo de su aplicación.

[1] `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
		Disminutivo	D

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

Existen varios enfoques algoritmicos 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.3.1 Módulos de Freeling y A.2 Stanford POS & NER Taggers se encuentran algunos comentarios más técnico de los algoritmos de POS tagging que utilizamos en este trabajo y vinculos a bibliografía pertinente y en la sección siguente 2.2.3 Named Entity Recognition (NER) veremos una descripción más detallada de enfoques algorítmicos al problema que ilustrarán, al menos de modo general, la estructura de la algoritmía basada en machine learning aplicada a procesamiento de lenguajes. A continuación 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.

Freeling (ES)		
Forma	Etiqueta	Descripción
El	DA0MS0	Determinante, Artículo, Másculino, Singular
hombre	NCMS000	Nombre, Común, Másculino, 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

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...).

2.2.3. Named Entity Recognition (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 más 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.

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 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 basadon 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. Un ejemplo típico de este tipo de enfoque puede verse en el algortimo KnowItAll?? explicado en la sección Relation Extraction. 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 algortimos 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.2 Stanford POS & NER Taggers) mientras que el segundo es un algoritmo trivial basado en Autómatas Finitos (ver A.3.1 Módulos de Freeling). Para descripciones y referencias de diferentes implementaciones de algoritmos de NERC ver [Nadeau and Sekine, 2007].

2.2.4. Question Classification

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 nlp 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 este 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, impoen 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. 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 intrinseca 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*, *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 Stanford Question Classifier veremos 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 ventajes 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 terminos 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. En particular, para el modelo estructurado implementamos un sistema de reglas simples como el recién enunciado y para el modelo no estructurado utilizamos el resultado de la clasificación de la misma pregunta pero formulada en inglés (pues las preguntas están disponibles en ambos idiomas). Actualmente, un tesista del grupo GALLI está trabajando en la construcción de un corpus para alimentar un clasificador de preguntas basado en machine learning para el español, pero lamentablemente por cuestiones de tiempos no pudimos hacer uso del mismo en esta tesis.

3. ESTADO DE ARTE

En este capítulo analizaremos diferentes investigaciones sobre question answering para poder ilustrarnos acerca de los distintos problemas y las distintas formas de encararlos que existen. Discutiremos primero una serie de investigaciones académicas pequeñas, presentadas en general en congresos y competencias del área para inferir un modelo más o menos general del dominio de problemas y los acercamientos típicos. Luego, comentaremos el sistema de IBM -Watson- para ilustrarnos, más allá de los enfoques usuales, un enfoque exitoso. Finalmente pasaremos revista de una serie de sistemas disponibles para facilitar la creación de modelos de question answering, haciendo foco en Qanus (un framework que de hecho pudimos utilizar durante un periodo de nuestra investigación) y mencionando algunos sistemas que evaluamos teóricamente pero que, lamentablemente, no estaban disponibles *out of the box*, principalmente debido a restricciones y modificaciones en el acceso programático que implementaron los buscadores populares en los últimos años[1]

3.1. Algunos ejemplos académicos

- Watson: [Ferrucci et al., 2010] y [Kalyanpur et al., 2012]
- Qanus: [Jun-Ping Ng, 2010]
- Ephyra: [Pires, 2012]
- Varios de QA: Yago [Adolphs et al., 2011], sobre una teoria de QA como interfaz a DBs: [Popescu et al., 2003]. Corpus: [Tomás et al., 2008], qall-me: [Sacaleanu et al., 2008], practical QA: [Chung et al., 2004], simple QA: [Cooper and Rüger, 2000] y Surface de Ravishandran: [Ravichandran and Hovy, 2002]. Introducción a QA: [Lampert, 2004] y [Wang, 2006] y [Moldovan et al., 2000]
- Aranea: [Lin, 2007] (no leido)
- Passage retrieval evaluation: [Tellex et al., 2003]
- Evaluacion de las TREC8 (metrica de [Moldovan et al., 2000] LASSO): [Voorhees and Tice, 1999]

3.2. IBM-Watson

Watson 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 hombre con mayores récords históricos del show en un programa en vivo [MÁS DATOS -LINKS]

[1] Ver por ejemplo, Google Search API, deprecado el 1ero de noviembre de 2010 aquí: https://developers.google.com/web-search/

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 que 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 hombre en alguna tarea que requiera altos niveles de inteligencia humana y, si es el caso, que clase de tecnología, algoritmos e ingenieria se requiere?"[2]

Watson es la implementación específica para participar en este programa de una arquitectura más genérica de question answering, DeepQA, que da el nombre al proyecto de la corporación. Esta arquitectura ejemplifica perfectamente la complejidad del problema de QA de dominio abierto e incorpora 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.

3.2.1. El problema

Watson debe realizar tareas como parsing, question classification, question decomposition, automatic source adquisition and evaluation, entity and relation detection, logical form generation, knowledge representation and reasoning manteniendo ciertos atributos de calidad bastante exigentes derivados de la naturaleza del show. Estas restricciones son:

- 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).

- 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 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.

El programa 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 "potpourri". Watson intenta respuestas sobre varias hipótesis de dominio y verifica en cual de ellos se logran respuestas de mayor confiabilidad.

[2] Traducción propia de Building Watson: An Overview of the DeepQA Project, p2

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.

Por ejemplo:
Categoría: Ciencia General
Pista: Cuando es impactado por electrones, un fósforo emite energía electromagnética de esta forma
Respuesta: Luz (o fotones)

A su vez, existen ciertos tipos de pistas que requieren un enfoque particular, por ejemplo, pistos 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 "Investanda en 1500's 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.

3.2.2. 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.

3.2.3. 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 (Text Retrieval Conference), una autoridad en el área. PIQUANT consta de un pipeline típico (véase QANUS) 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 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 (véase OpenEphyra), 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 busqueda 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.

3.2.4. 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 realmente difíciles de sacar de su contexto original y de las evaluaciones sobre las cuales

fueron testeados. Este problema, veremos -por ejemplo, con QANUS y Reverb- , se repitió en nuestro proyecto. 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 computo paralelo, probabilistico, 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 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. Los principios de diseño subyacentes de la arquitectura son:

- Paralelismo masivo
  Para evaluar distintas hipótesis en distintos dominios con poco acoplamiento.
- Pervasive confidence estimation:
  Ningún componente genera la respuesta final, sino que da una serie de features y grados de confiabilidad y evidencia para distintas hipótesis, que luego son sintetizados.
- Integrate shallow and deep knowledge:

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

3.2.4.1. 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.

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.

3.2.4.2. 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 preguntado y realizar los primeros análisis

que determinan cómo encarará el procesamiento el resto del sistema. Watson utiliza shallow parses, deep parses, formas lógicas, pos-tags, correferencias, detección de entidades nombradas y de relaciones, question classification, además de ciertos análisis concretos del domiento 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 son por Watson son los LATs extraídos de las preguntas de ejemplo. El LAT determina el "tipo" de la respuesta, que 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 su sistema de tipos default, y que la mejor forma de adaptación era mappear su output a los tipos utilizados por Watson (un enfoque similar fue utilizado en esta tesis con respecto al clasificador de Stanford). 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". 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 entidad 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.

Una particularidad ya señalada de las preguntas de Jeopardy son las pistas con subpistas no relacionadas. Para atacar este problema, Watson genera distintas particiones y resuelve todas en paralelo, sintetizando las respuesta de cada partición generada mediante algoritmos ad-hoc de ponderación de confiabilidad y otras características.

3.2.4.3. 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 mutiples 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 siguiente. 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.

(Soft filtering)

Para optimizar recursos, se realiza un filtrado liviano 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.

3.2.4.4. 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, dónde 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 estrurctura 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 terminos con la pregunta, popularidad (u obscuridad) de la fuente del pasaje, aliases, etc.

POR EJEMPLO: COPIAR NIXON

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

(Merge)

Recién después de este momento, Watson realiza un merge entre hipótesis idénticas. Las hipótesis idénticas son diferentes formulaciones lingüisticas 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. Se utilizan técnicas jerárquicas como mixture of experts y stacked generalization y, finalmente, un metalearner fue entrenado para ensamblar los distintos resultados intermedios.

3.2.5. Tiempos y escala

DeepQA utiliza Apache UIMA, un framework que implementa UIMA (Unestructured Information Management Architecture): todos los componentes de DeepQA son IUMA-annotators, módulos que producen anotaciones y aserciones sobre un texto.

La implementación inicial de Watson corría sobre un sólo procesador y demoraba aproximadamente 2 horas en contestar una sola pregunta. La arquitectura paralela permite, sin embargo, que al correrlo sobre 2500 núcleos -de la implementación final- los tiempos de respuesta oscilen entre 3 y 5 segundos, que es lo esperado.

Finalmente, la implementación de Watson logró alcanzar el estándar de resultados de los campeones de Jeopardy y, como ya dijimos, compitió y ganó el programa en Febrero de 2011. Además, se realizaron adaptaciones para trabajar sobre los problemas de TREC, en los cuales se demostró una amplia mejoría en comparación con PIQUANT y OpenEphyra

3.2.6. Conclusiones sobre IBM-Watson

System level approach.

3.3. La arquitectura de Qanus

QANUS (Question-Answering @ National University of Singapore) es un sistema de question answering basado en information retrieval. El proyecto se actualizó por última vez en noviembre de 2012 y contiene las herramientas más actuales de nlp (el POS-tagger, el NER-tagger y el Question Classifier de Stanford) y también de information retrieval índice de búsquedas lucene), todo de código abierto. El código cuenta con un framework (Qanus), que cumple un rol equivalente a la arquitectura DeepQA en el proyecto anterior, y un sistema montado sobre este framework QA-sys, equivalente a Watson (Ver Figura 3.1). La motivación de esto es proveer a la comunidad científica un framework para ingresar al mundo de QA de una manera más sencilla y rápida, permitiendo construir nuevos sistemas de QA sobre esta arquitectura. En efecto, la arquitectura DeepQA no está disponible para la comunidad, el ya mencionado OpenEphyra, como veremos en breve, no funciona, mientras que otros sistemas resultan igualmente inaccesibles (Aranea, Qanda) mientras que QA-sys es un sistema de QA out of the box. Mencionaremos los logros y los límites de estos objetivos cuando hablemos de nuestro intento por montar nuestro propio sistema sobre Qanus.

La arquitectura, al igual que la de DeepQA, es la de un pipeline. Este sistema consta de tres pasos principales, que se ejecutan todos por separado (off-line):

3.3.1. Preparación de la fuente de información

Este paso está pensado para preprocesar cualquier base de conocimiento y dejarla preparada para el paso 3 (retorno de la pregunta). El framework se propone tan amplio que no hay más especificaciones al respecto. La implementación puntual asume una base

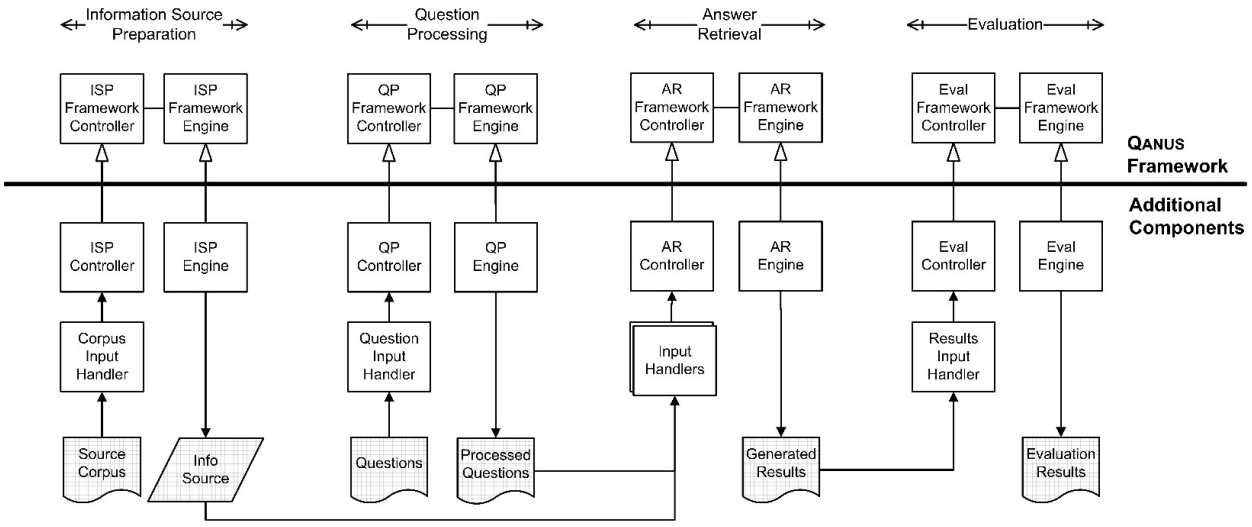

Fig. 3.1: El framework Quanus y la implementación QA-sys

de conocimiento en formato XML de AQUAINT[3] y la incorpora a un índice de búsquedas Lucene. Este paso incorpora todo el conocimiento que estará finalmente offline; accesos dinámicos a la web, por ejemplo, se modelan en el paso 3.

3.3.2. 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 classiffier, todos de Stanford. Hablaremos más de estos componentes más adelante.

3.3.3. 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

[3]

sistema:

3.3.4. Evaluación

Este paso está pensado para evaluar las respuestas generadas y presentarlas 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.

3.3.5. Implementación

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. Presuntamente, el framework es lo suficientemente abierto para permitir la implementación de sistemas basados en distintas fuentes de conocimiento (ontologías, archivos, web) y con distintos modos de funcionamiento mediante poco esfuerzo de customización.

La implementación llamada QA-sys está desarrollada para correr sobre el tipo de datos de las evaluaciones TREC 2007 (XML AQUAINT). En el primer paso, incorpora los XML en este formato a un índice Lucene, en el segundo paso utiliza 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 exactitud del 0.706 y el décimo (Quanta) obtuvo 0.206, mientras que QA-sys logra el 0.119. La implementación es realmente simple y funciona sólo a modo de ilustración de lo que puede construirse sobre el framework.

3.4. Otros sistemas de QA: OpenEphyra, Aranea y Just.Ask

El paper describe las arquitecturas de todos los sistemas, si sirve meter más info

El paper [EPHYRA1] busca crear un criterio cuantitativo para comparar la eficacia de distintos pasos de Just.Ask, Open Ephyra y Aranea basándose en la arquitectura *pipeline de tres pasos* compartida por todos. Los tres sistemas, por lo demás, están basados en la web, utilizando distintas APIs de buscadores o bien analizando los resultados de la interfaz de usuario de los mismos.

El primer ítem importante a destacar de este trabajo, es que, al momento de la experimentación **Aranea no funcionaba más y estaba discontinuado**[4]**.** El autor se comunicó con el responsable del proyecto que corroboró que las APIs de los buscadores en los que se basaba Aranea cambiaron y no había interés en readaptar el código para que vuelva a funcionar. Las comparaciones que logró entre Just.Ask y Open Ephyra son interesantes y concluyentes a favor de la performance de OpenEphyra.

(Freeling + Cambio de Base + Rigidez)

[4] (Resaltado en Sección 8, muy concluyente).

4.1. Frameworks

4.1.1. No funcionales

Para la implementación de nuestro sistema, originalmente, evaluamos la utilización de distintos frameworks disponibles. DeepQA, el producto de IBM, no es de código abierto, por lo que acerca de su implementación sólo sabemos lo que ventilaron en sus artículos técnicos. Just.ask, el sistema basado en web comparado contra OpenEphyra no está disponible en la web al momento de escribir este trabajo, mientras que OpenEphyra no funciona tal cual está diseñado originalmente (basado en web), sino que el autor sugiere unos pasos esotéricos para configurarlo para usar conocimiento local. Cabe destacar que esta falla en la funcionalidad está asociada a la que había encontrado [AUTOR DE PAPER EPHYRA1] en Aranea y está vinculado con una serie de medidas restrictivas tomadas por las compañías de buscadores, que fueron cerrando sus accesos gratuitos para la comunidad de investigación bloqueando sus APIs y el acceso automático a sus UI. Las alternativas para el uso de buscadores, actualmente, se reducen a la configuración de una serie de proxies sobre los que rotar el acceso a la UI y así engañar al detector de accesos automáticos -alternativa de legalidad cuestionable - o bien al pago por una quota de queries por mes. OpenEphyra sobrevivió a Aranea porque sus responsables escribieron una interfaz para Bing cuando Google cerró sus puertas, mientras que los responsables de Aranea no lo hicieron. Finalmente, Bing también bloqueo el acceso automático gratuito. Notar que el mismo tipo de discontinuación ocurrió con el API de traducciones de Google. La empresa declara, explícitamente, que no está dispuesta a acceder a ninguna quota de acceso gratuito para la investigación académica y que todos sus servicios son pagos.

4.1.2. Qanus

Finalmente, un sistema que *sí* estaba disponible y funcionando fue Qanus, que respetaba al pie de la letra su detalle técnico. Al comienzo del proyecto, contábamos con un corpus de datos en XML, lo cual coincidía, al menos en gran parte, con el input esperado de la implementación Qa-sys. A pesar de esto, la adaptación de los componentes no fue nada trivial y requirió un tiempo excesivo. En particular, existían dos opciones a la hora de construir un sistema sobre la arquitectura Qanus: dejar de lado la implementación Qa-sys e implementar todos los componentes de cero sobre la arquitectura, respetando las interfaces dadas por el framework, o adaptar el sistema funcionando para que trabaje sobre los nuevos datos y el nuevo entorno esperado. Frente a esta alternativa, se aparece claro que el framework en sí mismo no aporta demasiado, pues lo único que hace es atar la implementación final a una interfaz estructurada de tres procesos bastante sencillo. Además, existe un cierto grado de dependencia de la arquitectura hacia la implementación final, quizás no a nivel técnico, pero si en el modo en el que está definida la estructura. Por este motivo, encaramos una adaptación de Qa-sys a nuestro modelo de datos y a nuestros requerimientos, pero los resultados no fueron buenos en términos de resultados por

tiempo invertido. El tiempo de aprendizaje del framework mismo y el tiempo requerido para adaptar las distintas componentes propias a las interfaces esperadas por Qanus es demasiado alto para la solución que brinda. Como recién mencionamos, en realidad, el proceso de pipeline de tres pasos no tiene tantas aristas, y adaptarse a un framework es mucho menos ameno que escribirlo. Este puede ser uno de los motivos por los cuales, como acertadamente señalan los autores de Qanus, no existe ningún framework estandarizado dentro del ámbito de la investigación en QA. Después de la investigación inicial, podríamos concluir que está estandarizado, al menos a modo conceptual, la idea de que la resolución del problema se debe enfocar como un pipeline de al menos tres pasos que incluyen:

- el preprocesamiento de la base de conocimiento,
- el preprocesamiento de la pregunta,
- el retorno de la respuesta a partir de los resultados de los dos pasos anteriores.

Como último comentario al respecto, el modelo de Qanus resultaba poco atractivo a la hora de incorporar procesamiento en varios idiomas: el mejor approach utilizando esta arquitectura era implementar dos sistemas basados en Qanus paralelos y utilizar uno u otro de acuerdo con el resultado de una detección inicial.

Si bien el modelo de Qanus fue, por los motivos recién expuestos, dejado de lado, debemos destacar una serie de puntos en los que fue útil.

En primer lugar, uno de los objetivos de Qanus es facilitar el ingreso al área del QA de nuevos investigadores. Creemos que esto está logrado perfectamente: el código es muy sencillo y claro y lo mismo ocurre con la documentación, lo que hace de Qanus un proyecto muy útil desde una perspectiva pedagógica o educativa, más allá de que sea esta misma simpleza la que más adelante atente contra la usabilidad. El modelo de pipeline, que es el enfoque teórico usual al problema, y los distintos componentes y usos típicos de estos componentes en los distintos pasos del pipeline se realizan linealmente en la implementación de Qanus y Qa-sys. En particular, la similaridad entre la descripción del código de IBM (DeepQA y Watson) y el enfoque con el que Qanus ataca el mismo problema salta a la vista, considerando la diferencia de escalas.

En segundo lugar, en el plano de la investigación del estado de arte de los sistemas de QA disponibles creemos que el intento con Qanus redundó en un cierto escepticismo sobre la posibilidad de resolver nuestro problema utilizando herramientas disponibles de gran escala. La conclusión es análoga a la que tuvo el equipo de IBM al intentar usar OpenEphyra y PIQUANT: el tiempo de customización y adaptación de los framework a nuestro problema puntual es demasiado alto en comparación con el tiempo necesario para construir una nueva arquitectura que cumpla los mismos requisitos.

Por último, en un nivel técnico, Qanus nos resultó de utilidad para construir nuestro modelo final pues reutilizamos varios de los componentes de Qa-sys: En primer lugar, recuperamos el POS tagger, el NER tagger y el Question Classiffier (QC) de Stanford, que son las librerías principales con las que Qa-sys encara el procesamiento lingüístico de la pregunta y parte del proceso de generación de respuestas. Todas estas herramientas están disponibles en la web por otros medios, pero algunas -principalmente el QC- requieren un cierto tiempo de configuración inicial que los autores de Qanus ya habían resuelto. Es decir, reutilizamos, además de estos módulos externos, bastante de la configuración y las APIs de acceso a estos módulos escritos por los singapurenses. Estas herramientas

funcionan bien sólo para inputs en inglés. Las adaptaciones que hicimos las veremos más abajo. Por otro lado, incorporamos casi sin modificaciones algunas métricas de distancia entre pasajes que Qa-sys usa en el momento de la generación de la respuesta como Scorers. Estos son las clases: **FeatureSearchTermCoverage**, **FeatureSearchTermFrequency**, **FeatureSearchTermProximity**, **FeatureSearchTermSpan**. Explicaremos esta métricas en breve, dentro de nuestro modelo, bajo en nombre de "Comparadores". Este código está escrito por los autores de Qanus (es decir, no es una librería externa utilizada por ellos).

4.2. Arquitectura

4.2.1. Motivación

Después del intento con Qanus, decidimos implementar el sistema por fuera de cualquier framework y encaramos el diseño actual. En este diseño respetamos el modelo típico de pipeline de tres pasos que abunda en la literatura científica y, por lo demás, parece el indicado a la hora de encarar este tipo de problemas. Un momento no-técnico importante a destacar es la obtención de una base de datos en mongodb, resultado del trabajo del proyecto MITIC, la cual cambió sustancialmente el enfoque anterior, basado en XMLs. A partir de estos datos, fue posible delinear un esquema de entidades formal que determinó qué se puede responder y qué no. Como vimos, la estrategia de QA cuando el tipo de datos es estructurado es radicamente distinta que la estrategia cuando los datos son no estructurados. Qanus, por su parte, está orientado a un tipo de datos no estructurados: buscar documentos rankeados en un índice de búsqueda y rastrear en ellos pasajes mediante distintos métodos. Cuando la base de conocimientos consta de un tipo de datos estructurado (esto es, de entidades, relaciones, atributos de entidades) es posible delimitar una ontología más rígida que permita concentrarse en la interpretación de la pregunta hacia un lenguaje formal. El arquetipo de este enfoque puede pensarse como la traducción de un lenguaje de consulta humano a un lenguaje de consulta formal, como por ejemplo, SQL: esta estrategia de QA puede entenderse como una interfaz inteligente a una base de datos. En eje principal en este acercamiento está en el análisis lingüístico de la pregunta a fin de mapearla a un dominio conocido y, por otro lado, no es necesario hacer análisis lingüístico sobre el corpus de datos.

Por otro lado, dado que nuestro objetivo inicial incluía desarrollar métodos de QA con soporte bilingüe basados en textos (datos no estructurados) y esta base de conocimiento no lo permite, incorporamos al proyecto la resolución algunos de los ejercicios de la competencia QA4MRE (Question Answering for Machine Reading Evaluation) de la CLEF (Cross-Language Evaluation Forum) del año 2007.

Estos ejercicios fueron elegidos para poder evaluar nuestros métodos de análisis lingüsticos, ya que la naturaleza del dominio de conocimiento del proyecto mitic - por el formato de sus datos y por lo puntual de su temática- hace imposible cualquier métrica de evaluación objetiva. Tras investigar distintas competencias y métodos de evaluación, optamos por la CLEF del 2007 por contar con un subconjunto de ejercicios bastante adecuados para nuestro sistema mientras que el corpus de datos necesario para completar estos ejercicios estaba (en parte) disponible dentro de los tiempos requeridos por nuestro proyecto.

4.2.2. Modelo

Nuestro sistema resuelve dos problemas de QA similares pero distintos.

Por un lado, disponemos de una base de datos estructurada, con datos del área de la investigación y la producción en TICs en Argentina, que requiere un enfoque estructurado con métodos basados en ontologías y en formalizaciones de dominio. El enfoque aquí es traducir la pregunta formulada en lenguaje humano a un lenguaje más formal "comprensible"según el modelo de dominio.

Por otro lado, para evaluar métodos de QA para datos no estructurados, resolvimos algunos ejercicios de la competencia CLEF del '07. La base de conocimientos para estos ejercicios son algunos snapshots de Wikipedia en Español y en Inglés, anteriores al 2007. Sobre esta base de conocimiento, el enfoque no es "traducir"la pregunta a un lenguaje estructurado sino interpretarla y "compararla", mediante distintas métricas, con documentos y pasajes, buscando medidas estádisticas y otras condiciones que permitan *rankear* una respuesta candidata con un cierto grado de confianza, o bien determinar que no fue posible encontrar una respuesta para la pregunta.

Conceptualmente, el modelo consiste en los tres pasos típicos: la creación de la base de conocimientos optimizada, el análisis lingüístico de la pregunta y la generación de una respuesta desde la base de conocimientos optimizada a partir de la pregunta con sus anotaciones. Los pasos 1 y 2, la generación de la base de conocimientos optimizada y el procesamiento de la pregunta son procesos esencialmente análogos para la base de conocimientos estructurada y para la no estructurada. El tercer paso, la generación de respuestas, tiene distintos enfoques según el caso. En las siguientes secciones recorreremos ambos enfoques en conjunto, señalando las decisiones de diseño y los distintos módulos utilizados, haciendo las distinciones pertinentes cuando sea necesario.

En las secciones subsiguiente comentaremos las implementaciones de los distintos pasos, con sus decisiones de diseño y las tecnologías utilizadas.

4.3. Base de conocimiento

La creación de la base de conocimiento es el único que se ejecuta offline y consiste en la obtención del corpus original y en la generación de índices invertidos. Los índices invertidos, recordamos, son índices que permiten buscar, para una serie de términos vinculados lógicamente, un conjunto ponderado de documentos pertinentes. Por su naturaleza, este paso está separado de la ejecución del resto del código.

4.3.1. Corpus inicial

4.3.1.1. Grafo Mitic

Los datos originales del proyecto mitic constan de una serie de documentos xml y de cinco colecciones de mongodb y relaciones entre ellas (una base de datos no relacional).

Las cinco coleccciones son: universidades, investigadores, empresas, publicaciones, proyectos y temáticas.

Estas colecciones definen el dominio. Cada entidad posee sus respectivos atributos y distintas relaciones con otras entidades. Estas relaciones contienen desde vínculos explicitos

como "trabajar-en"pero también relaciones inferidas mediante distintos algoritmos durante el proyecto mitic. Las relaciones que incorporamos tienen distintos pesos según este tipo de caracteristicas y tejen un grafo de distancias más o menos .[es]peculativas.[en]tre todas las entidades.

Los atributos de cada entidad se pueden ver en el Anexo: [[HACER Y LINKEAR]].

4.3.1.2. Wikipedia

El subset de ejercicios de CLEF'07 que elegimos resolver utiliza como corpus un snapshot de wikipedia en inglés y otro en español, ambos anteriores al año 2007. El set de preguntas completo involucra una serie de documentos para los cuales hay que registrarse en la asociación: algunas preguntas se responden en base a wikipedia y otras en base a estos documentos. Identificamos las preguntas que se responden desde wikpedia chequeando los documentos fuente de las respuestas esperadas.

Wikipedia ofrece diferentes formas para obtener una propia imagen de la enciclopedia. Es posible incorporar la base de datos de wikipedia a una instalación wikimedia propia, es posible descargar una imagen estática del sitio en archivos html y también se puede descargar un archivo xml gigante con todos los artículos.

En la guía para los participantes de la competencia [Clef, 2007a] hay un link para descargar wikipedia en inglés preprocesada de Noviembre de 2006 [1] y 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]

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 (por ejemplo: "image", "discussion", "category", etc).

Los links a páginas de wikipedia en español 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 siguientes:

Idioma	Fecha	Tamaño
Inglés [4]	4 de noviembre de 2006	7,6G
Español [5]	7 de julio de 2006	558M

El mantenimiento de un mismo formato para ambos idiomas nos permitió crear un indexador único para cualquier dump en xml de una wikipedia. Durante el desarrollo y las pruebas, para evitar tiempos de carga innecesarios y también para probar, utilizamos otras imagenes disponibles de wikipedia:

Idioma	Fecha	Tamaño
Español	26 de enero de 2007	902M
Español	14 de junio de 2013	7,3G
Inglés simple	4 de julio de 2006	26M
Inglés simple	24 de julio de 2013	406M

[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`

4.3.2. Creación de Indices

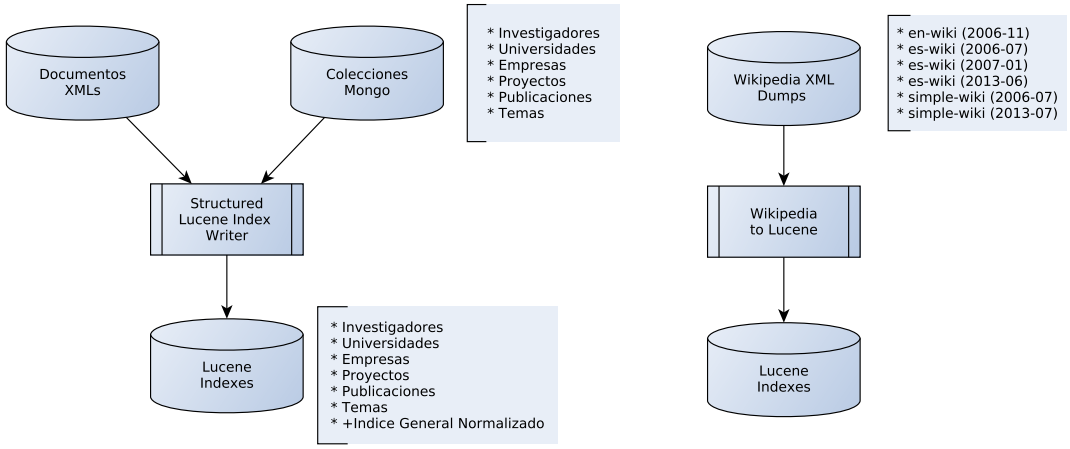

Fig. 4.1: Creación de Indices

A partir de esta base de datos de mongo y de los archivos xml fueron construidos cinco índices de búsqueda lucene y un índice de búsqueda más, general, con la información normalizada de los otros cinco. Cada uno de los cinco índices por entidad mantiene la estructura del tipo como campos de los documentos. Esto quiere decir que el índice invertido para *Investigadores* tiene los mismos campos del modelo de datos de nosql. Además, se agregó el campo "all"que resulta de la concatenación de todos los campos. Este campo resulta útil a la hora de filtrar resultados. El índice general posee un documento por cada entidad de las cinco colecciones, manteniendo también un puntero a la entidad original y su tipo.

Para la construcción de indices lucene con los dumps de wikipedia usamos la librería gwtwiki (Ver Apéndice B.2.3). Los artículos se indexan como documentos con los siguiente campos: *id, title, body y all*. En este proceso se descartan artículos mal formados y entradas representado imágenes o discusiones, tal como se sugiere en la guía. Por mera curiosidad, tomamos tiempos en la contrucción de estos índices locales sobre versiones de wikipedia. Estos son algunos tiempos de indexación para distintos dumps sobre una [[detalles de la compu]]:

Idioma	Tamaño	# Entradas	Tiempo
es	50M	# 16 millones	2.5min
es	50M	# 16 millones	2.5min
es	50M	# 16 millones	2.5min
es	50M	# 16 millones	2.5min

El proceso de creación de índices está ilustrado en la figura 4.1

4.3.3. Interfaz de servicios

La creación offline de índices lucene tiene como finalidad optimizar la base de conocimiento para responder con mayor eficiencia a búsquedas de resultados en un momento posterior.

En esta sección vamos a ver la interfaz que presenta la base de conocimientos indexada al resto de los módulos del sistema y qué dependencias existen con los módulos de análisis lingüístico.

Para la base de conocimiento estructurada, reutilizamos un modelo de datos escrito en java del grafo de entidades que obtuvimos de los investigadores del proyecto mitic (Ver B.2.4 Modelos de Morphia). A estos modelos se les agregó soporte para su representación como documento dentro de un índice. Por ejemplo, el modelo para la entidad "Universidad de Buenos Aires", además de persistirse en la colección de universidades de la base de datos de mongo, también dispone de una representación como documento en un índice lucene particular (el índice de universidades) y otra en el índice general. A nivel colecciones, cada entidad dispone de un representante que maneja el acceso a su colección en la base de datos y también a su índice. A partir de estos representantes por entidad que ofrecen acceso a una base de datos y a un índice creamos la interfaz *KnowlegdeBase*. Cada entidad tiene una interfaz de administración de sus dos motores de persistencia. Las responsabilidades de esta interfaz son las de un handler de conocimiento acerca de una cierta clase de entidades. Además, esta interfaz permite la reificación de entidades implicitas en el modelo. Estas entidades son: Ciudad, Provincia, Centro de Investigación, etc [6]. Esta reificación significa abstraer las funciones directas contra la base de datos. Mientras el *KnowledgeBase* de Investigadores habla directo contra la base mongo o contra lucene, el *KnowlegdeBase* de Ciudades habla contra Investigadores, Universidades y Empresas verificando ciertos campos y recomponiendo la forma de la entidad, de modo abstracto y sin persistencia propia.

[[Tablita con totales por entidad]]

[[Relaciones solo presentes en mongo]]

Las *KnowledgeBase* de las cinco entidad y el índice general están, a su vez, controlados un *KnowlegdeManager*, que es la interfaz del módulo que maneja la base de conocimiento.

El *KnowledgeManager* ofrece diferentes servicios de verificación de entidades. Para una cadena de tokens cualquiera, este módulo puede decidir, con un cierto grado de confianza, los siguientes problemas:

- Si la cadena de tokens es una entidad dentro del modelo de datos. Esto incluye:
  - Es una entidad del modelo de datos: una universidad, una empresa, un investigador, un proyecto, una publicacion o una tematica
  - Es una entidad inferida: una ciudad, una provincia, un centro de investigación, un lugar de trabajo
- Si la cadena es una colección del modelo de datos, es decir, si se están nombrando "Investigadores" o "Universidades" como clase de entidades.
- Si la cadena es un atributo o una relación de una clase de entidades.

La primer verificación utiliza campos de identidad de las entidades y diferentes tipos de comparadores (Ver apéndice B Comparadores). Cada entidad fue configurada con diferentes atributos de identidad y a su vez estos atributos están asociados a diferentes comparadores a con un cierto grado de peso y de confianza en el juicio general.

[6] hacer y escribir bien

La segunda y la tercera verificación utiliza estos mismos comparadores jerarquizados, pero compara las cadenas de entrada contra diccionario de sinónimos escrito a mano nombrando las diferentes clases de entidades y los atributos de cada una de ellas.

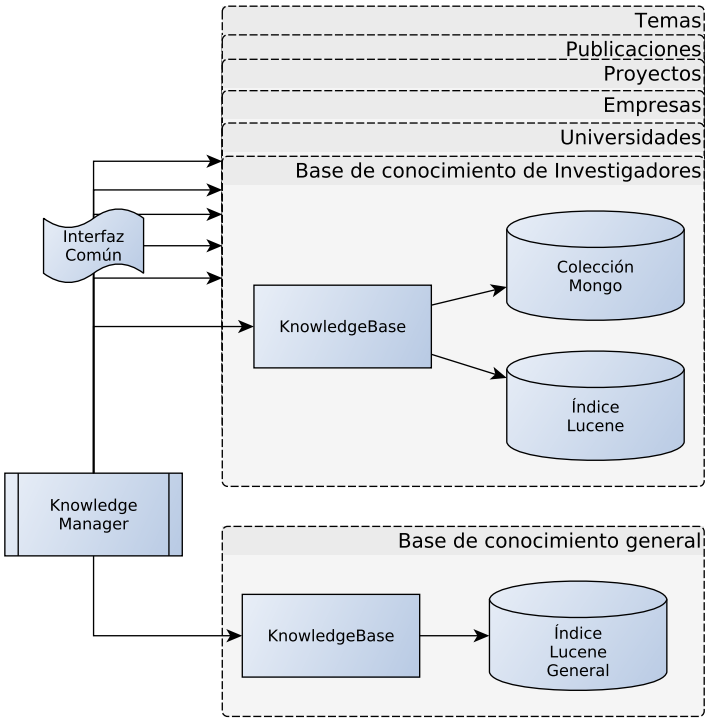

Fig. 4.2: Base de Conocimiento de grafo de TICs

La base de conocimiento de los ejercicios de Clef '07 es mucho más sencilla porque no existe ningún modelo *a priori* más allá del documento de lucene. El formato de la entidad "artículo", como señalamos antes, es: $(id, titulo, cuerpo)$. En este caso, el trabajo más fino no está en el modelado inicial del dominio sino la capacidad lingüística de extraer pasajes a partir de un artículo y recomponer información estructurada a partir de estos pasajes. Mientras que para un modelo estructurado la base de conocimiento debería permitirnos, para un cierto input, identificar univocamente una entidad y darnos pistas sobre un pedido de información acerca de esa entidad, el objetivo sobre un corpus de documentos en traer todos los documentos en los que sea posible que exista un pasaje respondiendo a la pregunta o evidencia relevante para apoyar una respuesta. Es decir, mientras una respuesta acotada es una virtud para el manejador de una base de conocimientos estructurada, el manejador de una lista de documentos de texto debería devolver una lista lo suficientemente grande para contener la respuesta dentro de los pasajes. La razón de esta política es que si por ser demasiado estrictos a la hora de retornar documentos llegasemos a descartar un pasaje candidato válido esto redundaría en una baja generar de efectividad, mientras que en pasos subsiguiente será trivial descartar toda información irrelevante sin tanto costo. Por eso, el acceso a los indices de wikpedia consta simplemente de un generador de queries similar al recién comentado accediendo y acumulando resultados (rankeados) a partir de un *LuceneIndexReader* común (Ver A.5 Apache Lucene para más información).

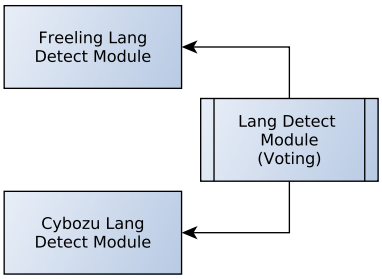

Fig. 4.3: Módulo de Detección de Idiomas

4.4. Análisis de la pregunta

En este paso se realizan diferentes análisis lingüísticos de la pregunta. El resultado son distintas características asociadas a la pregunta (anotaciones) y distintas entidades semánticas reconocidas útiles para el proceso de generación de respuestas. Las herramientas de procesamiento de lenguaje natural que utilizamos en la implementación de este paso del pipeline incluyen: detección de lenguaje, extracción y verificación de entidades nombradas (NER), de verbos, sustantivos, qwords (qué, quién, cómo) (POS), análisis de n-gramas y categorización por tipo de pregunta (QC).

El proceso de análisis de la pregunta es bastante similar para ambos approachs (estructurado y no estructurado), por lo que comentaremos ambos en simultaneo, mencionando diferencias cuando corresponda. Estructuraremos esta parte de la tesis en las siguientes secciones:

- Detección de idioma
- Detección y verificación de entidades nombradas
- Análisis gramatical
- Clasificación del tipo de pregunta

Si bien es cierto que el segundo item está basado principalmente en NER-tagging, el tercero en POS-tagging y el cuarto en Question Clasiffication, cada uno de estos pasos utiliza estas herramientas de diferentes maneras. Por ejemplo, para la detección y verificación de entidades del analisis estructurado, además del NER-tagger también utilizamos la base de conocimiento y el análisis gramatical y, para el español, la clasificación del tipo de pregunta se hace apoyandose en las qwords identificadas por el POS-tagger.

4.4.1. Idioma

El módulo de detección de idiomas de nuestro sistema utiliza dos librerías distintas. El módulo de detección de idiomas de Freeling y una librería especializada de Cybozu Labs. (Ver apéndices A.3 Freeling y A.4 Lang Detect de Cybozu Labs para más información)

Ambos permiten priorizar la detección de ciertos idiomas sobre otros desde su configuración. De esta manera podemos forzarlos a identificar sólo los idiomas esperados en

nuestro dominio. Ambos fueron configurados para detectar inglés y español para mejorar la confiabilidad, pero pueden habilitarse más idiomas de ser necesario y funcionan correctamente.

El módulo de detección simplemente evalua ambos algoritmos y decide el resultado con un cierto grado de confianza. En caso de existir un empate, se prioriza la opción de Cybozu labs que en la práctica dió resultados más exactos.

De todos modos, el problema "detección de idioma"no introduce mayores complicaciones y parece un problema bien resulto. Es decir, la mayoría de las veces ambos módulos responden lo mismo y de modo correcto. Sin embargo, para ciertos casos bordes molestos (pero lamentablemente frecuentes) el detector de Cybozu resultó funcionar mejor. Por ejemplo, está el caso de la pregunta formulada en inglés pero acerca de una entidad nombrada en español: "Where is located the Universidad de Buenos Aires?". Este problema está particularmente presente en el procesamiento de preguntas -nuestra tarea-, dado que son textos cortos en los que una construcción sustantivada en otro idioma puede desequilibrar erróneamente la balanza.

A continuación presentamos algunos ejemplos que ilustran el funcionamiento de ambas librerías y el resultado final de nuestro módulo en estos casos:

Texto	Freeling	Cybozu	Resultado
¿Dónde queda la Universidad de Buenos Aires?	es	es	es
Where is located the University of Buenos Aires?	en	en	en
Where is located the Univesidad de Buenos Aires?	en	en	en
Where is located Universidad de Buenos Aires?	es	en	en
Quién es Carolina Fernandez?	es	es	es
Who is Carolina Fernandez?	none	en	en
Quién es John McCain?	none	es	es
Who is John McCain?	en	en	en
Dónde vive John McCain y por qué vive allí?	es	es	es
Where does Carolina Fernandez live and why does she lives there?	en	en	en

Los ejercicios de Clef '07 no evaluan detección de idiomas. Los archivos de preguntas están separadas por idioma y no se espera que el idioma se infiera a partir de los textos de las preguntas, sino que es un dato dado al sistema de QA.

4.4.2. Entidades nombradas

Para la detección de entidades utilizamos la clase simple de detección (NER) y clasificación (NEC) de entidades de Freeling y el NERC de Stanford (ver A.3 Freeling y **?? ??**). Las herramientas de Stanford en general superan a las de Freeling -al igual que el detector de idiomas de Cybozu-, pero solo sirven para inglés. La clasificación utilizada por ambos es la más general de las comentadas en 2.2.3 Named Entity Recognition (NER): persona, lugar, organización y otros.

Veamos algunos ejemplos de funcionamiento de los módulos de detección de entidades.

Texto	Freeling	Stanford	Resultado
¿Dónde queda la Universidad de Buenos Aires?	es	es	es
Where is located the University of Buenos Aires?	en	en	en
Where is located the Univesidad de Buenos Aires?	en	en	en
Where is located Universidad de Buenos Aires?	es	en	en
Quién es Carolina Fernandez?	es	es	es
Who is Carolina Fernandez?	none	en	en
Quién es John McCain?	none	es	es
Who is John McCain?	en	en	en
Dónde vive John McCain y por qué vive allí?	es	es	es
Where does Carolina Fernandez live and why does she lives there?	en	en	en

Mientras la detección de entidades para los ejercicios de Clef se detiene en el reconocimiento de entidades nombradas a nivel lingüísitico, para el sistema estructurado el proceso es un poco más complejo. Esto se debe a que en este caso la detección de entidades es esencial. Si en el proceso de anotado de la pregunta no se logra identificar alguna entidad reconocida por el modelo de datos, entonces se está muy lejos de encontrar una respuesta. Por eso, además de utilizar los modulos NER recién mencionado, agregamos otros algoritmos de detección y, también, verificación de entidades.

En principio, verificamos la o las entidades nombradas reconocidas contra la base de conocimiento. El *KnowledgeManager* ofrece diferentes servicios de verificación de entidades. Para una cadena de tokens cualquiera, este módulo puede decidir, con un cierto grado de confianza, si:

- La cadena de tokens es una entidad dentro del modelo de datos. Esto incluye:
  - Es una entidad del modelo de datos: una universidad, una empresa, un investigador, un proyecto, una publicacion o una tematica
  - Es una entidad inferida: una ciudad, una provincia, un centro de investigación, un lugar de trabajo
- La cadena es una colección del modelo de datos, es decir, si se están nombrando "Investigadores" o "Universidades" como clase de entidades.
- La cadena es un atributo o una relación de una clase. (nombre de investigador)

Parte importante del trabajo para este esquema es lograr identificar este tipo de entidades lingüísticas, por lo que además de verificar los resultados del proceso de NER-tagging, también generamos otras cadenas de input. Notar además que los nombres de clase y de atributos de clase no tendrían por que ser reconocidas por el NER-tagger. Por ejemplo, para "¿Qué investigadores trabajan en Córdoba?", "investigadores" está haciendo referencia al nombre de una clase pero no es el tipo de entidades lingüísticas que detecta un NER-tagger.

Por estas razones, generamos más entidades lingüísticas posibles además de las entidades detectadas por los NER-taggers. Una vez que todas las entidades nombradas fueron

verificadas, generamos n-gramas sobre el resto de la pregunta para chequear por más tokens reconocibles. Configuramos la generación de n-gramas de 1 a 3, con ciertos filtros para no verificar construcciones que no representan entidades de manera trivial. Por ejemplo: dejamos sólo los unigramas que cumplan el rol de sustantivos, eliminamos bigramas que sean un sustantivo y una acción, salteamos NERs ya reconocidas, etre otros.

4.4.3. Análisis gramatical

De las diferentes etiquetas que generan los POS-taggers, en nuestro sistema distinguimos los verbos, los sustantivos, las qwords y las palabras triviales. Las palabras etiquetadas cumplen distintos roles a lo largo del proceso de generación de respuestas. Como señalamos recién, los n-gramas que se verifican contra la base de datos están filtrados por los roles gramaticales de sus tokens. Por otro lado, a la hora de generar el tipo de pregunta para una pregunta en español, utilizamos, como mecanismo ad-hoc, las qwords.

Clase	Ejemplos
qword	qué, quién, cómo, dónde, cuándo
verbo	trabaja, trabajar, trabajando
trivial	lo, a, de, y
sustantivos	universidad, impresora, álgebra

Los usos más intensivos de estas etiquetas son el filtrado de n-gramas que describimos en la sección anterior para el caso estructurado (4.4.2 Entidades nombradas), un algoritmo ad-hoc de etiquetado de Q-Type para el caso español (es el próximo tema a discutir en 4.4.4 Clasificación) y, para la generación de respuestas, las ponderaciones de pasajes en los scorers de los ejercicios de Clef (**?? ??**), y finalmente, sirven para desempatar por atributos o relaciones preguntadas en algunos casos del modelo estructurado.

4.4.4. Clasificación

Para clasificar la pregunta según su tipo de respuesta esperada utilizamos el Question Classiffier de Stanford, tomando la configuración de Qanus. Este clasificador arroja una clase y una subclase (ver A.1 Stanford Question Classifier para una lista detallada de las posibles clases) y un grado de confianza para esta asignación. A continuación presentamos algunos ejemplos que ilustran estos resultados.

Pregunta	Clase y Subclase	Confianza
What's his name?	HUM:ind	0.74
Where do you come from?	DESC:desc	0.62
What's your phone number?	NUM:code	0.63
How old are you?	NUM:period	0.78
When were you born?	NUM:date	0.99
What does he look like?	DESC:desc	0.82

Sin embargo, como ya señalamos oportunamente (ver 2.2.4 Question Classification), no existen herramientas de clasificación de preguntas para el idioma español. Esto nos llevó a tomar diferentes medidas para aproximar un tipo de pregunta y no tener distintos casos de código para el inglés y el español. Para las preguntas de Clef formuladas en español,

utilizamos su versión en inglés para obtener el tipo. Así, el tipo de respuesta esperada de "¿En qué colegio estudia Harry Potter?" es el mismo que el tipo de respuesta esperada de "In what school does Harry Potter study?" (ENTY:cremat con 0.22 de confianza). Además, utilizamos reglas escritas a mano sobre QWords. Las qwords son palabras clave de las preguntas que señalan el tipo de respuesta. Por ejemplo: una pregunta que comienza con 'Cuándo' tendrá como tipo de respuesta una fecha, un tiempo, etc. En el modelo estructurado, definimos una serie acotada de categorías de tipo de respuesta esperadas y unificamos los resultados del clasificador de Stanford y nuestras reglas sobre qwords para español para unificar el código. Estas categorías son: Who, Whom, Where, Which, When, What y Other. Es decir: Quién, Quiénes, Dónde, Cuál, Cuándo, Qué y otros. Las clases y subclases del clasificador de Stanford se mapearon a estas categorías que coinciden con los resultados de las reglas escritas a mano para el español.

4.4.5. Entidad de Grupo

El formato de las preguntas para los ejercicios Clef que elegimos es un xml para cada idioma. En particular, nosotros resolvimos los ejercicios en español y inglés que podían responderse en base a wikipedia. Contar del tema de los grupos. Poner algún ejemplo.

4.5. Generación de Respuestas

El proceso de generación de respuestas difiere sustancialmente entre ambos modelos de dominio. Para el sistema basado en datos estructurados, el approach es determinan en casos de código las distintas posibilidades, acotadas, de las cosas que se pueden responder. Nuestro dominio es tal que solo se pueden responder entidades, atributos de entidades o relaciones entre entidades. De ese modo, si no es posible redirigir el flujo de la pregunta hacia alguna respuesta conocida, no hay posibilidad de articular una respuesta significativa. Para el caso de los ejericicios de Clef sobre wikipedia, el enfoque es muy distinto. En primer lugar, no hay un modelo de los datos del dominio, hay textos con pasajes (u oraciones). Si bien es posible una cierta jerarquización de los datos (por ejemplo, utilizando los nombres de los articulos como verificación de la existencia de una entidad), un enfoque estructurado resulta imposible. En este contexto se utiliza la técnica de rankeo semántico de pasajes en base a features (características). Estas dimensiones de valoración de los pasajes son llamados Scorers (ver **?? ??**). Los Scorers, como veremos, pueden ser tan sencillos como preferir minimamente una cierta longitud sobre otra y también pueden incorporar dimensiones de análisis lingüístico (por ejemplo, la presencia de cierta entidad en un cierto rol semántico). El algoritmo de generación de respuestas consiste, en el caso no estructurado, en encontrar features útiles, significativos y en establecer mecanismo inteligentes de priorización de estos features.

4.5.1. Estructurado

Una vez etiquetados todos los tokens de la pregunta, se procede a marcar como procesadas las "palabras triviales". Estas son palabras que si no formaron parte de alguna otra construcción, entonces no haberlas procesado no debería considerarse un problema. Ejemplos de ellas son las proposiciones, los pronombres y algunos conectores. Si al etiquetar la pregunta no se logró identificar ninguna entidad del modelo, entonces será dificil avanzar.

Como vimos recién, la fase de procesamiento de la pregunta para el caso estructurado excede por mucho la mera anotación lingüística: al finalizar el etiquetado deberíamos disponer de alguna entidad reconocida por el modelo. Por otro lado, durante la fase de etiquetado cada palabra se marca como procesada. Como acabamos de señalar, al finalizar este proceso se marcan también como procesados ciertos tokens triviales. En este punto, el sistema debe tomar una decisión. Si no ha logrado etiquetar una cierta cantidad de tokens (más del 80 %), entonces se considera que no tiene sentido dar por "comprendida"la pregunta y se procede a un segundo análisis, computacionalmente más costoso y además más inexacto, en el que se intenta encontrar alguna entidad con otros métodos que enunciaremos en breve. En caso de encontrarse una entidad, entonces el flujo del programa retorna al curso de análisis estructurado. Si esto no ocurre, se devuelve la lista de entidades (documentos) que el índice lucene general devuelve para la pregunta original interpretada como una query normal de information retrieval. Este caso, si bien retorna información, es un caso de falla de procesamiento. Esta lista de documentos viene acompañada de un mensaje del tipo: "No se logró interpretar su pregunta" y, en un trabajo futuro, podría incorporar un sistema de recomendaciones e idas y vueltas con el usuario (¿Quizo decir...?).

Si, en cambio, el threshold de tokens es alcanzado, entonces se pasa a otro switch. En este caso el código se bifurca de acuerdo a la cantidad de entidades del modelo reconocidas. Por entidades del modelo entenderemos, aquí, entidades internas, objetos, no nombres de clase o de atributos. Distinguimos estos tres casos: 'ninguna entidad reconocida', 'una entidad reconocida', 'más de una entidad reconocida'.

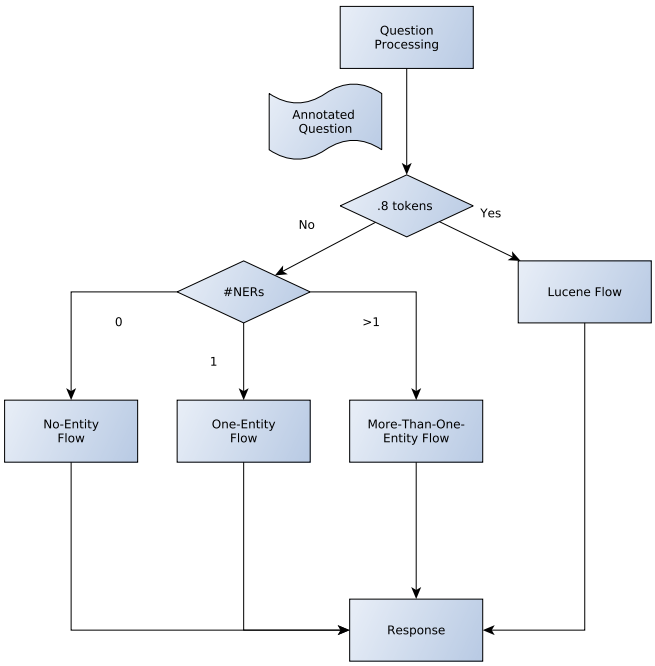

Fig. 4.4: Flow para la Generación de Respuestas - Estructurado

Ninguna o más de una entidad

En el caso en el que no se haya identificado ninguna entidad del modelo quedan diferentes posibilidades, que se verifican en orden secuencial en base a nombres de colección,

nombres de atributos, tipo de respuesta esperada y verbos. Los casos contemplados son: la pregunta por un campo de una colección (por ejemplo: direcciones de empresas en buenos aires) o por listas de entidades de una colección (investigadores de capital federal). Los diferentes casos son reglas de código escritas a mano. En todos los casos, si no se dan las condiciones para seguir especificando la dirección de la respuesta, se genera un respuesta ad-hoc con datos rankeados según el índice invertido, especificando de modo estructurado el camino recorrido hasta el momento. Por ejemplo, si se identifica que se pregunta por 'investigadores' pero no es posible decidir ninguna especificación más (de capital federal, que hayan publicado en 2008, etc) entonces se retorna una lista de investigadores rankeada según el índice invertido 'investigadores' con el resto de los datos de la pregunta.

Por otro lado, si hay más de una entidad reconocida entonces hay sólo algunos casos posibles de relaciones entre ellas que pueden ser respuestas, que también se reflejan como caso de código. Finalmente, si no es posible identificar ninguno de estos caso, se toma un camino similar al mecanismo ad-hoc basado en information retrieval.

Una entidad

El mejor caso es aquél en el que se reconocío una entidad y otros datos lingüisticos que permitan especificar qué se está preguntando. Al especificar acerca de qué/quién resulta mucho más sencillo canalizar qué se está preguntando. Los modelos que representan objetos (ver 4.3.3 Interfaz de servicios) son subclases de *NodoBase*, el cual representa una entidad en abstracto. Una entidad sabe responder preguntas acerca de ella. Para esto, utiliza las anotaciones de verbos, atributos nombrados y qwords para identificar qué se está preguntando. La pregunta puede responderse con un atributo o con una relación. Los distintos atributos de las distintas entidades se corresponden con verbos y con tipos de respuesta esperada.

[[Detalle de casos y combinaciones de verbos + tipo de respuesta esperada + atributo]] [[Ejemplos]]

4.5.2. No Estructurado

El proceso de generación de respuestas para los ejercicios de la Clef es muy distinto del anterior y puede dividirse en tres pasos principales: obtención de documentos y pasajes, ranking de pasajes y generación de respuesta. En el primer paso se accede a los índices invertido (al corpus) buscando documentos relevantes. Este paso pertenece netamente al área information retrieval. Como mencionamos al comentar Watson (en particular, ver 3.2.4 La arquitectura DeepQA), es fundamental que el resultado de este paso sea lo suficientemente amplio como para contener la respuesta pero lo suficientemente acotado como para no sobrecargar el proceso posterior de análisis lingüístico sobre los pasajes. Los documentos rankeados se dividen en pasajes. En el segundo paso, tanto los documentos como los pasajes son contrastados con distintas métricas contra los datos de la pregunta generando distintos valores para estos features. Finalmente, con esta información se procede al tercer paso, que consiste en realizar diferentes filtrados sobre los pasajes en función del tipo de respuesta esperado y en distintas formas de recopilar evidencia a favor de un pasaje o una entidad (depende el caso) para finalmente seleccionar una respuesta (o decidir que no se encontró ninguna).

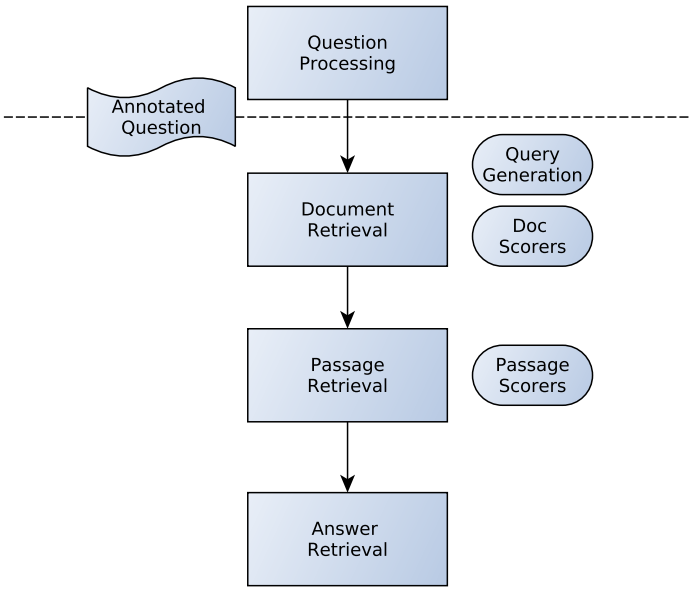

Fig. 4.5: Flow para la Generación de Respuestas - No Estructurado

4.5.2.1. Documentos

En este punto, para una pregunta dada se dispone de la entidad del grupo de preguntas y de las distintas anotaciones hechas a la pregunta en el paso anterior (4.4 Análisis de la pregunta). Por su parte, los documentos en los índices invertidos poseen los campos *Id*, *Title*, y *Text*. El mecanismo de generación de queries tiene como objetivo priorizar en el ranking los documentos relacionados con el tema asociado al grupo de preguntas. Este paso es un problema de information retrieval puro: esto es, dado un pedido de información, retornar *documentos relevantes*. El análisis semántico tiene peso en el paso posterior, a la hora de rankear pasajes. Por ejemplo, para el primer grupo de preguntas acerca de Harry Potter, solo se espera de este una lista de de documentos relacionados con ese mundo, en primer lugar. Por otro lado, es necesario que los principios de generación de queries no sean demasiado estrictos. Si en este punto quedan afuera muchas ocurrencias de una respuesta, entonces todo el resto del programa se ve afectado de manera irreparable. Es preferible generar documentos de más y luego filtrarlos mediante análisis lingüístico que ser demasiado estrictos y perder respuestas. Para lograr esto, ponderamos los documentos en los que las entidades nombradas reconocidas lingüísticamente aparecen en el título, si se dispone de más de una entidad buscamos documentos que mencionen ambos, luego priorizamos los documentos que poseen estas entidades dentro del cuerpo y también consideramos la presencia de verbos en diferentes conjugaciones y de sustantivos que ocurren en la pregunta. Finalmente, agregamos una lista de documentos enviando la pregunta misma como una query.

Dado que finalmente se realizan queries simples (masivas), cabe preguntarse cual es la razón de la generación de queries y la ponderación de documentos. Esta razón es que en el proceso de ranking de pasajes y evaluación de respuesta se utilizan features basados en el score dado por lucene a los diferentes documentos. Si una mejor posición del documento contenedor del pasaje no implica que el pasaje sea correcto, si en cambio es un indicador

de que dicho pasaje se encontró más cerca o más lejos del nucleo temático en el que se esperaba encontrarlo.

Una vez generada la lista de documentos rankeados según lucene, se procede a analizar algunos features en base a distintos scorers propios. A su vez, estos distintos valores se combinan en una evaluación general del documento, que será utilizada luego a la hora de generar una respuesta. Estas dimensiones buscan en el título y en el artículo diferente medidas sobre las entidades nombradas y sobre la pregunta completa. En concreto, se miden distancias a la entidad nombrada que identifica al grupo de preguntas (ver 4.4.5 Entidad de Grupo), las entidades nombradad en la pregunta misma, a la pregunta completa y la respuesta esperada data. Las medidas contra la respuesta esperada -dada por Clef- no pueden usarse para generar la respuesta, pero sí para evaluar la performance del sistema. En el siguiente cuadro se muestran las dimensiones que se consideran sobre los documentos.

Entidad	Campo	Comparador
Entidad de Grupo	Título	Span Covr Freq
	Texto	Span Covr Freq
Entidades de Pregunta	Título	Span Covr Freq
	Texto	Span Covr Freq
Todas las entidades	Título	Span Covr Freq
	Texto	Span Covr Freq
Pregunta	Título	Span Covr Freq
	Texto	Span Covr Freq
Respuesta	Título	Span Covr Freq
	Texto	Span Covr Freq
Score según índice	–	–

Los comparadores señalados (*Freq*, *Covr*, y *Span*) se utilizan en distintos lugares de esta tesis y su funcionamiento es explicado en el apéndice B Comparadores. Notar que

'Entidades de la pregunta' refiere tanto a aquellas reconocidas por el NER-tagger como a construcciones sustantivadas y que 'Pregunta' no es la pregunta bruta sino la priorización de verbos conjugados, sustantivos, adjetivos y entidades.

El score general del documento es un cálculo ponderado de estas diferentes dimensiones.

Lucene permite especificar cuántos documentos queremos recuperar. Para evaluar la performance de este paso, utilizamos medida distintos scores en base a la respuesta dada por dada por la conferencia para el ejercicio. Es importante notar que dado que no utilizamos las imagenes de wikipedia de la primera sugerencia, es esperable que las respuesta no estén 'tal cual'. Evaluamos distintos mecanismos de generación de documentos, con distinta cantidad total, bajo distintas métricas. Para generar documentos, probamos la query trivial ALL : $pregunta$ (1), una un poco mejorada ALL : $entidad_de_grupopregunta$ (2), secuencias concatenadas de queries tal como las describimos más arriba (3) y varios pedidos separados aplanados en un paso posterior (4). Para los cuatro métodos eliminamos los signos de puntuación. Para medir los resultados, utilizamos los comparadores de presencia exacta y diferentes grados de cobertura de términos (.8, .9 y 1). A su vez, evaluamos distintas imagenes de wikipedia para el español. Es total de preguntas del ejercicio, recordamos, es 200. Los resultados son los siguientes.

Método	# Docs	Wikipedia	Exacto	Covr 1	Covr .9	Covr . 8
1 - Trivial	100	es - 2006	132	151	152	159
		es - 2007	144	159	160	164
		en - 2006	x	x	x	x
	1000	es - 2006	144	167	167	173
		es - 2007	159	177	177	180
		en - 2006	x	x	x	x
2 - Trivial'	100	es - 2006	138	156	157	164
		es - 2007	151	167	168	171
		en - 2006	x	x	x	x
	1000	es - 2006	147	168	168	174
		es - 2007	163	178	178	181
		en - 2006	x	x	x	x
hola	ey	wiki	137	156	157	160
3 - Inteligente	100	es - 2006	127	144	144	150
		es - 2007	141	150	151	156
		en - 2006	x	x	x	x
	1000	es - 2006	143	163	163	170
		es - 2007	158	175	175	179
		en - 2006	x	x	x	x
4 - Inteligente'	100	es - 2006	142	160	161	168
		es - 2007	157	170	171	174
		en - 2006	x	x	x	x
	1000	es - 2006	147	168	168	174
		es - 2007	x	158	x	x
		en - 2006	x	x	x	x

Conclusión de esto.

4.5.2.2. Pasajes

Este paso es análogo al anterior, pero con mayor detalle y granularidad. Cada documento generado en el paso anterior, con sus diferentes puntajes para las dimensiones señaladas, se parten en pasajes u oraciones. Nuevamente, sobre estas oraciones realizamos diferentes mediciones y las combinamos generando un score final. En esta sección discutiremos las mediciones consideradas y los diferentes métodos de combinación de las mismas. Estos métodos de combinación generan distintos rankings de pasajes. Para evaluar estos rankings, nuevamente, utilizaremos la información disponible sobre las respuestas esperadas, buscando que la respuesta esperada se encuentre entre los n pasajes mejor rankeados. En primer lugar, las distintos scorers implementados son los siguientes:

Comparador	Qué	Dónde
Estadísticos		
Freq	Pregunta	Pasaje
Span	Pregunta	Pasaje
Covr	Pregunta	Pasaje
#Tokens	–	Pasaje
Basados en NLP		
Presencia	Entidad de Grupo	Pasaje
Presencia	Entidades de pregunta	Pasaje
Presencia	Verbos de pregunta	Pasaje
Presencia	Sustantivos de pregunta	Pasaje
Para evaluación		
Freq	Respuesta	Pasaje
Span	Respuesta	Pasaje
Covr	Respuesta	Pasaje

A estos Scorers se le suman los scores del documento asociado al pasaje (ver 4.5.2.1 Documentos). Sobre estas dimensiones disponibles, intentamos las siguientes combinaciones de priorización:

#	Nombre	Fórmula
1	Simple	$2 + 2 = 4$
2	Respuesta	$2 + 2 = 4$
3	Compleja	$2 + 2 = 4$

Y consideramos la ocurrencia de respuestas, de la misma manera que en el apartado anterior (Match Exacto y tres medidas de covertura de tokens: 1, .9 y .8), sobre los primeros n pasajes, con $n = 1$, 5, 10, 20, 50 y 100.

Fórmula	#Docs	Exacto	Covr 1	Covr .9	Covr . 8
1	1	x	x	x	x
	5	x	x	x	x
	10	x	x	x	x
	20	x	x	x	x
	50	x	x	x	x
	100	x	x	x	x

4.5.2.3. Respuestas

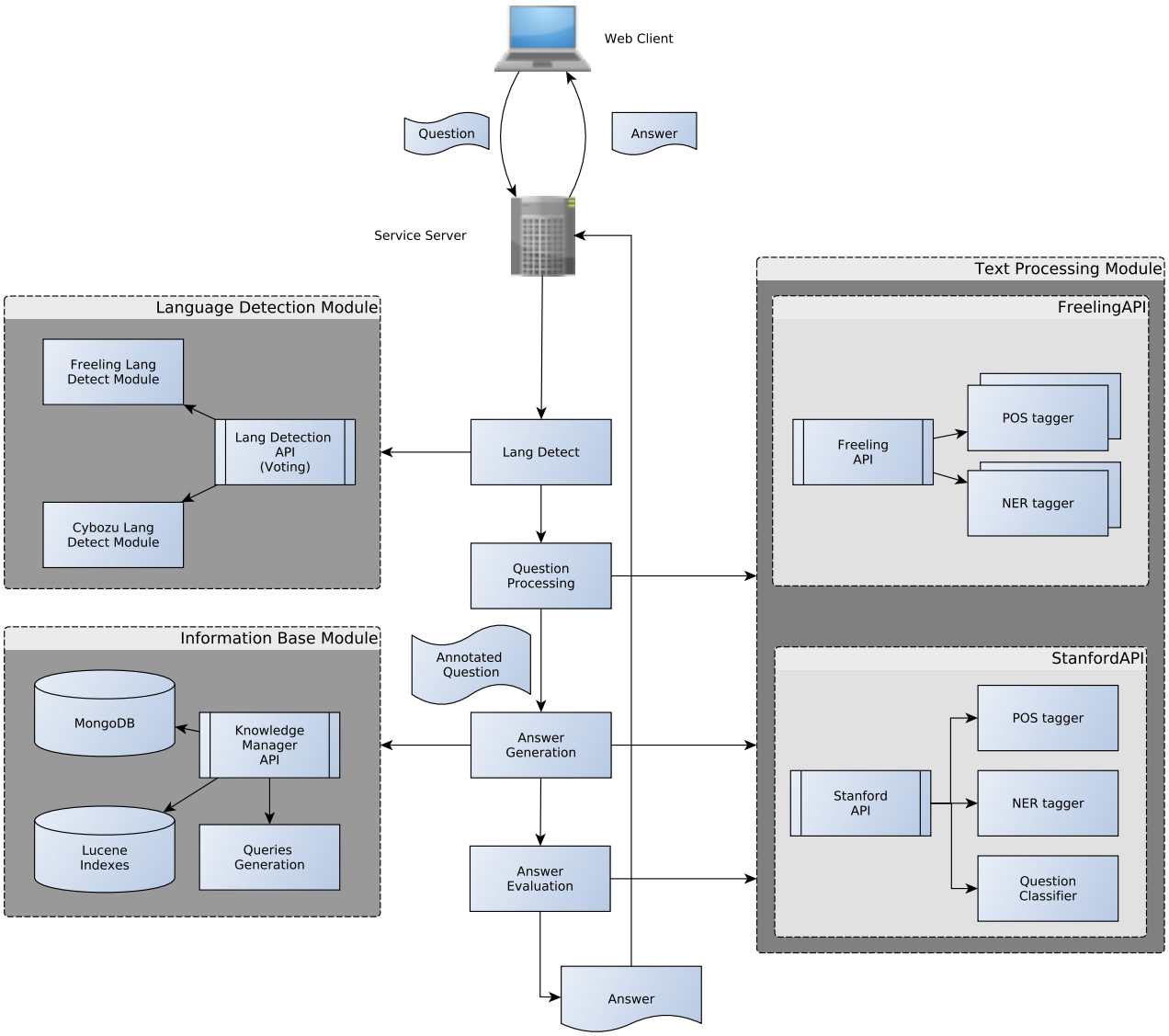

Fig. 4.6: Arquitectura

Apéndices

Apéndice A

HERRAMIENTAS

A.1. Stanford Question Classifier

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
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

[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]

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 sintácticos como semánticos. Los features utilizando 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. Stanford POS & NER Taggers

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 2.2.2 Part-of-speech (POS) tagging el tagset de Penn Treebank.

Por su parte, el NER tagger de Stanford, es una implementación general de

It comes with well-engineered feature extractors for Named Entity Recognition, and many options for defining feature extractors. Included with the download are good named entity recognizers for English, particularly for the 3 classes (PERSON, ORGANIZATION, LOCATION), and we also make available on this page various other models for different languages and circumstances, including models trained on just the CoNLL 2003 English training data. The distributional similarity features in some models improve performance but the models require considerably more memory.

Stanford NER is also known as CRFClassifier. The software provides a general implementation of (arbitrary order) linear chain Conditional Random Field (CRF) sequence models. That is, by training your own models, you can actually use this code to build sequence models for any task. (CRF models were pioneered by Lafferty, McCallum, and Pereira (2001); see Sutton and McCallum (2006) or Sutton and McCallum (2010) for more comprehensible introductions.)

The CRF code is by Jenny Finkel. The feature extractors are by Dan Klein, Christopher Manning, and Jenny Finkel. Much of the documentation and usability is due to Anna Rafferty. The CRF sequence models provided here do not precisely correspond to any published paper, but the correct paper to cite for the software is:

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating

Non-local Information into Information Extraction Systems by Gibbs Sampling. Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL 2005), pp. 363-370. http://nlp.stanford.edu/ manning/papers/gibbscrf3.pdf The software provided here is similar to the baseline local+Viterbi model in that paper, but adds new distributional similarity based features (in the -distSim classifiers). The big models were trained on a mixture of CoNLL, MUC-6, MUC-7 and ACE named entity corpora, and as a result the models are fairly robust across domains. Named Entity Recognition Stanford Named Entity Recognizer (Finkel, Grenager and Manning 2005)

Finkel, Jenny Rose, Trond Grenager, and Christopher Manning. Ïncorporating Non-local Information into Information Extraction Systems by Gibbs Sampling.roceedings of the 43nd Annual Meeting of the Association for Computational Linguistics. 2005.

A.3. 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 e inglés), soportan todos los módulos disponibles en la librería. Freeling 3.1 ofrece los siguientes analizadores lingüisticos:

- 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

A.3.1. Módulos de Freeling

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.4. Lang Detect de Cybozu Labs

Librería de Cybozu Labs - una compañía japonesa -, implementado en Java y liberado bajo Apache License 2.0. En la práctica, este paquete dio excelentes resultados. Soporta 53 idiomas con %99 de precisión para todos ellos (según sus tests). El detector se basa en perfiles de idiomas generados a partir de las distintas wikipedias y detecta el idioma de los textos usando un filtro bayesiano ingenuo (*naive bayesian*). El código está disponible, actualmente, en google-code (El link está en la sección de bibliografía [Shuyo, 2010])

A.5. 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

[[Dar ejemplos de una query]]

Apéndice B

COMPARADORES

Dada la frecuencia en la que resultaba necesario comparar dos string, decidimos reificar la operación de comparación como una familia de clases que implementan Comparadores.

Gracias a esto se hace posible cambiar las nociones de igualdad o similaridad en un módulo completo del sistema o en alguna clase simplemente configurando otro comparador como parámetro. La interfaz permite a los distintos comparadores tomar valores binarios así como también valores entre 0 y 1. A su vez, esta considerada la posibilidad de configurar un umbral (threshold) a partir del cual redondear un valor entre 0 y 1 a un valor binario. Otro factor que tuvo mucha utilidad fue la capacidad de anidar comparadores. El concepto de comparador incluye cualquier operación que tome dos strings y genere un resultado booleano o analógico. Es decir, es posible incorporar análisis lingüísticos o queries a la base de datos en ellos. También se incorpora la posibilidad de ignorar o no ignorar la diferencia de mayúsculas, y obviar o no las tildes y otros signos problemáticos. Los comparadores sirven, en general, para comparar tanto strings representando palabras como string representado listas de palabras (oraciones o textos). Algunos, en particular, sólo sirven para este segundo caso. Los comparadores que finalmente utilizamos en esta tesis son los siguientes.

Comparadores de Strings	
Nombre	Descripción
Equal	Compara por igualdad estricta
EqualNoPunct	Compara por igualdad, eliminando signos de puntuación y normalizando acentos y otras posibles diferencias que no deberían tenerse en cuenta.
Contains	Verifica si un string contiene a otro. Puede usar Equal o EqualNoPunct
EditDistance	Verifica cuan similares son dos string contando la mínima cantidad de operaciones requeridas para transformar un string en el otro

Los siguiente comparadores son algoritmos fueron adaptados a partir de los Scorers del proyecto Qanus. Todos devuelven valores reales entre 0 y 1 y sirven para comparar secuencias de tokens (y no sólo palabras). Estos comparadores, al igual que Contains, no son simétricos. Para distinguir, llamaremos primer string al buscado y segundo string a aquel en el cual se busca el primero.

Comparadores de Secuencias de Tokens		
Abreviatura	Nombre	Descripción
Freq	Frequencia	Computa la cantidad de veces que los tokens del primer string ocurren en el segundo string. Esta suma se divide por la longitud del segundo string, dando un valor entre 0 y 1.
Covr	Cobertura	Computa cuantos tokens del primer string aparecen al menos una vez en el segundo, y divide esta suma por el total de tokens del *primer* string.
Prox	Proximidad	Computa la distancia entre dos strings en un tercero. Ver abajo.
Span	Distancia entre tokens	Computa la distancia media entre términos del primer string en el segundo. Ver abajo.

Vamos a explicar los algoritmos de *Prox* y *Span* ya que no son triviales.
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. 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* estaría dado por #total de tokens encontrados / |c-a|.

B.1. Placeholders

Esta sección es un placeholder para las siguientes posibles tecnologías que podría describirse como última sección en 2 Marco teórico

- Coreference Resolution
  Ejemplos de resolución de Freeling.
- Word sense disambiguation
  Hablar algo de wordnet?
- Relation Extraction
  La detección y extracción de relaciones es una tarea de procesamiento de lenguaje

natural que consiste en extraer entidades y relaciones semánticas entre entidades a partir de un texto no estructurado.

[[Escribir sobre distintos algoritmos conocidos basado en el paper RE1]]

- Machine Translation
  Hablar de los problemas con Google y otros y de las memorias de traducción.

- Detección de idiomas
  La detección o identificación de idioma es la tarea de reconocer el idioma de un cierto contenido. Esta tarea puede pensarse como una tarea de clasificación donde las clases son los distintos idiomas que el detector puede reconocer.

- Shallow parsing (also chunking, "light parsing")
  is an analysis of a sentence which identifies the constituents (noun groups, verbs, verb groups, etc.), but does not specify their internal structure, nor their role in the main sentence. It is a technique widely used in natural language processing. It is similar to the concept of lexical analysis for computer languages. [[Expandir con detalles técnicos de wikipedia]]

B.2. Secciones del apéndice

B.2.1. MyMemory

MyMemory es la Memoria de Traducción más grande del mundo: 300 millones de segmentos a finales de 2009

Como las MT tradicionales, MyMemory almacena segmentos con sus traducciones, ofreciendo a los traductores correspondencias y concordancias. El proyecto se diferencia de las tecnologías tradicionales por sus ambiciosas dimensiones y por su arquitectura centralizada y basada en la colaboración colectiva. Todos pueden consultar MyMemory o hacer aportaciones a través de Internet; la calidad de las aportaciones será cuidadosamente ponderada. MyMemory gives you quick access to a large number of translations originating from professional translators, LSPs, customers and multilingual web content. It uses a powerful matching algorithm to provide the best translations available for your source text. MyMemory currently contains 644.377.834 professionally translated segments.

Las memorias de traducción son almacenes compuestos de textos originales en una lengua alineados con su traducción en otras. Esta definición de memorias de traducción coincide literalmente con una de las definiciones más aceptadas de corpus lingüístico de tipo paralelo (Baker, 1995). Por esto se puede decir que las memorias de traducción son corpus paralelos.

Por el momento, no estamos utilizando ningún módulo de traducciones: todo el enfoque multilingüe está dado por la detección del idioma de la pregunta y la determinación de distintas herramientas de análisis según qué idioma sea. Sin embargo, en un momento se evaluó un enfoque distinto, basado en la traducción. Por ejemplo: utilizar módulos de procesamiento sólo en inglés y "normalizar" los inputs en otros idiomas (en principio, en español), a este idioma interno, y luego lo mismo con la generación de respuestas. A pesar de que no es el enfoque actual, hubo una fase de investigación dentro del dominio de la traducción, que resultó en un módulo de traducción basado en MyMemoryAPI.

Intento con google translator y la privatización. ¿La falta de software de traducción offline? El módulo de mymemory, robado de algún lugar. El sistema de cobro.

B.2.2. Reverb

ReVerb is a program that automatically identifies and extracts binary relationships from English sentences. ReVerb is designed for Web-scale information extraction, where the target relations cannot be specified in advance and speed is important.

ReVerb takes raw text as input, and outputs (argument1, relation phrase, argument2) triples. For example, given the sentence "Bananas are an excellent source of potassium,- eVerb will extract the triple (bananas, be source of, potassium).

B.2.3. gwtwiki -Java Wikipedia API

Java Wikipedia API (Bliki engine) http://code.google.com/p/gwtwiki/ Esta librería tiene métodos útiles para trabajar con dumps de wikipedia. La usamos para testear los métodos no estructurados de la tesis.

B.2.4. Modelos de Morphia

Java Wikipedia API (Bliki engine) http://code.google.com/p/gwtwiki/ Esta librería tiene métodos útiles para trabajar con dumps de wikipedia. La usamos para testear los métodos no estructurados de la tesis.

B.3. Conclusiones

B.4. Trabajo futuro

B.5. Citas a Textos trascriptos pero no usados

- QC: [Hermjakob, 2001], [Li and Roth, 2002] y también [Manning and Klein, 2003] (y [Lehnert, 1986])
- Clef: [Clef, 2007a] y [Clef, 2007b]
- POS: El manual [Jurafsky, 2000] y los dos de Stanford: [Toutanova et al., 2003] y [Toutanova and Manning, 2000]
- LangDetect: [Shuyo, 2010]
- NER: Survey [Nadeau and Sekine, 2007] y el NER de Stanford: [Finkel et al., 2005]
- Watson: [Ferrucci et al., 2010] y [Kalyanpur et al., 2012]
- Qanus: [Jun-Ping Ng, 2010]
- RE: Survey [Bach and Badaskar, 2007], for QA [Bouma et al., 2011] y reverb: [Fader et al., 2011]
- Ephyra: [Pires, 2012]

- Freeling: [Padró, 2011] y [Padró and Stanilovsky, 2012] (este no impreso)
- Wordnet para web ir: [Varelas et al., 2005] (no leido)
- Varios de QA: Yago [Adolphs et al., 2011], sobre una teoria de QA como interfaz a DBs: [Popescu et al., 2003]. Corpus: [Tomás et al., 2008], qall-me: [Sacaleanu et al., 2008], practical QA: [Chung et al., 2004], simple QA: [Cooper and Rüger, 2000] y Surface de Ravishandran: [Ravichandran and Hovy, 2002]. Introducción a QA: [Lampert, 2004] y [Wang, 2006] y [Moldovan et al., 2000]
- Aranea: [Lin, 2007] (no leido)
- Passage retrieval evaluation: [Tellex et al., 2003]
- Evaluacion de las TREC8 (metrica de [Moldovan et al., 2000] LASSO): [Voorhees and Tice, 1999]

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.

[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*.

[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.

[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.

[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).

[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.

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

[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)*.