LUZ

Un sistema de búsqueda sobre los archivos de la última dictadura militar.

Lautaro Cardozo Ramírez lautaro.cardozo@fing.edu.uy

Lía Emilia Rivero Cor liaemilia28@gmail.com

Guillermo Zorrón Bordone guillezorron@gmail.com

Tutores:
Aiala Rosá
Fernando Carpani

Proyecto de grado para Ingeniería en Computación
Facultad de Ingeniería
Universidad de la República

Montevideo – Uruguay
Diciembre de 2021

"Quisiera saber la verdad porque para mí es la mayor de las justicias."

Luisa Cuesta

Agradecimientos

Agradecemos a Jorge Tiscornia Bazzi y Fabián Hernández Muñiz por compartir su conocimiento y ayudarnos a comprender los documentos con los que se trabaja en *Cruzar* y sus estructuras. Su aporte fue primordial en la primer etapa del proyecto. También reconocer a Elena Bing, que fue fundamental en el proceso de anotación de los textos. El intercambio con ella fue muy enriquecedor para lograr una guía de anotación más robusta que la que teníamos.

Por otra parte agradecer a Aiala Rosá y Fernando Carpani, tutores del proyecto, por invitarnos a ser parte de este gran proyecto que es *Cruzar*, confiar en nuestro trabajo y guiarnos en todo el proceso.

Por último, agradecemos especialmente a nuestras familias y amigas/os, que nos han acompañado incondicionalmente durante la carrera y para quienes este proyecto significa tanto como para nosotras/os.

RESUMEN

A partir de un golpe de Estado que ocurrió en 1973, Uruguay vivió una dictadura cívico-militar que se extendió hasta mediado de los 80's. En ese marco la población perdió derechos y ocurrieron detenciones políticas y desaparición de personas.

Durante ese periodo hubo muy poca documentación pública y no se tenía información clara sobre lo que sucedía, pero hace algunos años se liberó una gran cantidad de documentos en formato microfilm. Poder procesar la información contenida en esos documentos es de vital importancia para esclarecer los hechos ocurridos durante esta última dictadura.

Este proyecto se gestó en el marco de un proyecto más amplio, llamado *Cruzar*, que busca desarrollar estrategias para conservar, organizar y facilitar el análisis de los documentos mencionados.

Se propone constuir herramientas que se encarguen de procesar los documentos en cuestión. Para esto se cuenta con un extenso conjunto de imágenes escaneadas y la transcripción manual de los textos allí contenidos gracias al proyecto *LUISA*.

En este proyecto se busca alcanzar el objetivo de construir una herramienta que se encargue de extraer información de interés de estos textos mediante técnicas de Procesamiento de Lenguaje Natural (PLN) y almacenarla de modo que permita consultarla y recuperarla fácilmente.

Tabla de contenidos

Capítulo 1

Introducción

1.1. Motivación

En el año 1973 ocurrió un golpe de Estado en Uruguay, suceso que marcó el inicio de una dictadura cívico-militar que se extendió hasta el año 1985. Durante estos años se desconocieron los derechos individuales; hubo centenares de detenciones y se produjo la desaparición de aproximadamente 200 personas. [34]

Esta dictadura fue un periodo donde la documentación pública era muy acotada y no se tenía información clara de lo que ocurría. En el año 2007, durante la gestión de la ministra Azucena Berrutti, fueron encontrados millones de documentos en la ex Escuela de Inteligencia del Ejército y en el Ministerio de Defensa. El conjunto de estos documentos fue nombrado *archivo Berruti* en homenaje a dicha ministra. [13] Como se encuentran en formato microfilm y han perdido calidad con los años, es de gran dificultad su lectura y escaneo. Poder procesar la información contenida en el *archivo Berruti* es necesario para esclarecer los hechos del periodo en cuestión y dar respuestas a las familias que se vieron desarmadas por los mismos.

En el año 2017 comienza a funcionar el proyecto de extensión universitaria *Cruzar* [11]. Este proyecto se encarga de la sistematización, tratamiento y difusión de información del pasado reciente vinculados al terrorismo de Estado y graves violaciones a los Derechos Humanos. *Cruzar* tiene como objetivos desarrollar estrategias para conservar y organizar los documentos mencionados. Además se enfoca en construir herramientas que permitan el cruzamiento de la información contenida en pos de facilitar su procesamiento para su futura

divulgación, investigación y análisis. En este marco se gestó *LUISA* (Leyendo Unidos para Interpretar loS Archivos) [24].

LUISA es una aplicación desarrollada para recuperar archivos de texto en formato imagen de escasa legibilidad. Se trata de una plataforma online, en la cual cualquier persona puede colaborar transcribiendo pequeños bloques de distintos documentos. Es necesario particionar así los documentos para asegurar que ninguna persona tenga acceso a ningún documento entero a través de esta plataforma, ya que por el momento se considera información confidencial. Una vez que la aplicación reagrupa estos bloques ya transcriptos se obtiene una nueva versión de cada documento, esta vez en un formato más amigable para su procesamiento.

Gracias a *LUISA* se han convertido hasta el momento miles de los documentos microfilmados del archivo Berruti en material que puede ser procesado digitalmente. El siguiente paso es entonces construir herramientas que se encarguen de este procesamiento. De esta manera nace la idea de este proyecto, *LUZ: un sistema de búsqueda sobre los archivos de la última dictadura militar*.

1.2. Objetivos

Una vez que se obtienen versiones en texto plano de los documentos microfilmados resta extraer la información allí contenida y almacenarla de forma tal que facilite su análisis.

Por tratarse de una cantidad de documentos muy extensa como para ser analizada únicamente a esfuerzo humano, se hace necesario automatizar la extracción de información. En principio, las entidades de interés serían las de tipo fecha, lugar, organización y persona.

Siendo uno de los objetivos facilitar el análisis de los documentos para poder reconstruir el camino de las personas detenidas y desaparecidas, surge la necesidad de resolver las correferencias entre las entidades reconocidas para agrupar la información correctamente según lo que representa cada entidad en el mundo real. Además es necesario conocer algunas relaciones entre estas entidades. Por ejemplo, dónde se domiciliaba una persona, en qué fecha fue detenida y liberada y si pertenecía a alguna organización social.

Es importante que la información extraída sea almacenada de una forma que permita consultarla y recuperarla fácilmente por los usuarios de este

sistema, por lo que se requiere crear una base de conocimiento con ella.

A partir de lo explicado se desprenden los dos objetivos generales de *LUZ*.

- Realizar tareas de extracción de información de los documentos transcriptos en *LUISA*.
- Almacenar la información extraída de forma tal que luego sea posible crear una base de conocimiento navegable, que será el componente principal.

Para alcanzar los objetivos generales se fueron planteando a lo largo del proyecto los distintos objetivos específicos que se listan a continuación.

1. Estudio de algunos de los documentos transcriptos y definición del preprocesamiento necesario.

2. Investigación y comparación de distintos Reconocedores de Entidades Nombradas (NERs por su sigla en inglés) para poder evaluar cuál se ajusta más a la realidad del proyecto.

3. Investigación sobre resolución de correferencias.

4. Investigación sobre herramientas y algoritmos de extracción de relaciones entre entidades nombradas.

5. Evaluación y definición de formato de salida del sistema para que permita la creación de una base de conocimiento a partir de ella.

6. Creación de un sistema configurable a partir de los objetivos anteriores que tenga por entrada un archivo de texto plano y le aplique todas o algunas de las siguientes etapas:

   a) Preprocesamiento del texto

   b) Reconocimiento y extracción de entidades nombradas

   c) Resolución de correferencias

   d) Reconocimiento y extracción de relaciones entre entidades nombradas

   e) Creación de un archivo de salida con toda la información extraída en un formato que facilite la creación de una base de conocimiento

1.3. Organización del documento

El resto del informe se estructura de la siguiente manera:

- En el capítulo dos se presenta el marco teórico de la extracción de información, profundizando tanto en el reconocimiento de entidades nombradas como de relaciones. Además se presenta el problema de la resolución de correferencias mostrando algunos trabajos previos. Finalmente se hace un abordaje teórico sobre la generación de bases de conocimiento, profundizando en la utilización de *RDF* y desarrollo de ontologías.
- En el tercer capítulo se brinda una descripción de la solución propuesta, detallando cada una de las etapas del proceso diseñado.
- En el capítulo cuatro se describe la arquitectura del sistema implementado y se detalla la implementación de cada una de las etapas presentadas en el tercer capítulo.
- En el quinto capítulo se presenta el conjunto *ground truth* utilizado para las evaluaciones, su construcción y la procedencia de los datos con los que se trabajó.
- En el capítulo seis se muestran las métricas obtenidas para cada uno de los reconocedores de entidades nombradas investigados y cuáles fueron los mecanismos de comparación y evaluación utilizados. Además se muestran algunos resultados de reconocimiento de relaciones y resolución de correferencias.
- En el séptimo capítulo se presentan las conclusiones.
- En el octavo y último capítulo se presentan propuestas de trabajo futuro.

Ejemplos utilizados

Los ejemplos manejados a lo largo de este informe son o bien tomados de la prensa, o bien inventados por el equipo encargado de este proyecto manteniendo el estilo de los documentos del *archivo Berruti*. De esta manera se respetan las normas de confidencialidad establecidas por el proyecto *Cruzar*.

Capítulo 2

Fundamentos teóricos

A lo largo de este capítulo se presentan los conceptos teóricos más importantes para lograr una profunda comprensión del trabajo realizado durante el proyecto y de este informe. Estos abarcan aspectos de todas las disciplinas puestas en práctica:

- Para la extracción de información se introducen conceptos de reconocimiento de entidades nombradas y extracción de relaciones.
- Para la resolución de correferencias se estudiaron distintas técnicas para satisfacer este objetivo.
- Para generar bases de conocimiento se utilizan distintos conceptos de web semántica que se ampliarán en esta sección.

2.1. Extracción de información

La extracción de información es la identificación, clasificación y estructuración en clases semánticas, de información específica que se encuentra en fuentes de datos no estructuradas, como por ejemplo texto en lenguaje natural [28].

Dentro de esta sección se encuentran 2 grandes temáticas abordadas a lo largo del proyecto como son el reconocimiento de entidades nombradas y de relaciones.

2.1.1. Reconocimiento de Entidades Nombradas (NER)

Una entidad nombrada, según Jurafsky y Martin [20], es básicamente algo que puede ser referido con un nombre propio como una persona, un lugar o una organización, aunque este término suele extenderse para incluir por ejemplo fechas, que no son entidades nombradas en sí.

El proceso de reconocerlas se puede dividir en 2 etapas: detectar qué entidades se encuentran en el texto y clasificarlas en la categoría que corresponda. Ambas etapas tienen su complejidad implícita, la primera tiene la dificultad de discernir cuándo termina una entidad y qué palabras están ligadas entre sí, en cambio la segunda tiene la dificultad de diferenciar a qué clase de entidad nombrada corresponde.

Ejemplo de NER:

$$Se\ encontró\ a\ José\ Pérez\ en\ la\ plaza\ Varela. \tag{2.1}$$

En el ejemplo anterior las entidades nombradas que se deberían identificar son *José Pérez* y *plaza Varela*, clasificando a la primera como persona y a la segunda como lugar. En cuanto al proceso para reconocer las entidades nombradas, la primera etapa consistiría en identificar *José Pérez* y *plaza Varela*, la dificultad en este caso consiste en identificar que *José* y *Pérez* corresponden a la misma entidad e identificar que *plaza* y *Varela* corresponden a la misma entidad. La segunda etapa consistiría en clasificar *José Pérez* como un nombre de persona y a *plaza Varela* como lugar.

Se puede notar que ambas etapas están relacionadas. Por ejemplo, si la primera etapa hubiese reconocido las entidades *José Pérez* y $Varela$, lo cual no es muy preciso, la segunda etapa podría haber clasificado a las entidades *José Pérez* como persona y a $Varela$ como persona también.

Si bien existen distintas técnicas para resolver este problema, el algoritmo estándar se basa en un etiquetado palabra por palabra en el cual se asignan etiquetas a las entidades nombradas con su tipo. A los algoritmos de este tipo se les llama **clasificadores secuenciales** y para resolver este problema se los puede entrenar para etiquetar los *tokens* en un texto con rótulos que indican la presencia de tipos particulares de entidades nombradas.

Una de las formas para anotar una secuencia de etiquetas para el problema

de reconocer fragmentos de texto, como por ejemplo NER, es el etiquetado BIO, explicado en la sección 2.1.2.

Existen también algoritmos basados en características como pueden ser las formas de las palabras. Por ejemplo, dos *tokens* consecutivos que comienzan con mayúscula, se pueden identificar como un nombre propio, entre otros patrones que se pueden generalizar basados en el contexto de los textos.

2.1.1.1. Herramientas investigadas

A continuación se presentan las herramientas de procesamiento de lenguaje natural que fueron investigadas en este proyecto para resolver el reconocimiento de entidades nombradas.

SpaCy

SpaCy [19] es una biblioteca de código abierto para el procesamiento del lenguaje natural en *Python* y *Cython*. Cuenta con modelos de redes neuronales para etiquetado, análisis, reconocimiento de entidades nombradas y clasificación de texto, así como un sistema de entrenamiento listo para producción.

Si bien algunas de las funcionalidades de *SpaCy* se pueden utilizar de manera independiente, otras requieren que se carguen modelos entrenados que permiten la predicción de anotaciones lingüísticas. Esta biblioteca ofrece modelos para una variedad de lenguajes, incluyendo español, constituidos por varios componentes que utilizan un modelo estadístico entrenado con datos etiquetados.

FreeLing

FreeLing [6] es una biblioteca de código abierto escrita en *C++* y es una de las más utilizadas para el procesamiento de lenguaje natural no solo porque provee un amplio abanico de herramientas sino que además tiene soporte para varios idiomas.

Una de las ventajas que esta biblioteca tiene frente a los demás reconocedores de código abierto que fueron investigados es que tiene la capacidad de reconocer fechas en el idioma español.

Stanza

Stanza [32] es un kit de herramientas de análisis de lenguaje natural de *Python* desarrollado por el Grupo de Procesamiento del Lenguaje Natural de la Universidad de Stanford (Stanford NLP). Cuenta con herramientas de tokenización, etiquetado POS, lematización y reconocimiento de entidades nombradas, entre otras, que pueden ser usadas como un pipeline.

Este paquete está construido con componentes de redes neuronales de alta precisión que pueden ser entrenados y también cuenta con modelos ya entrenados para más de 60 idiomas. En particular, cuenta con dos modelos entrenados de NER para idioma español, uno a partir del corpus *CoNLL2002* y otro a partir de *AnCora*. Ambos modelos admiten los mismos cuatro tipos de entidades nombradas: persona (PER o PERS[1]), lugar (LOC), organización (ORG) y miscelánea (MISC).

NLTK

NLTK (Natural Language Toolkit) [1] es un kit de herramientas con el objetivo de crear programas *Python* enfocados en el procesamiento del lenguaje natural. Proporciona bibliotecas de procesamiento de texto para clasificación, tokenización, derivación y etiquetado, entre otras.

Esta librería, con respecto a los otros reconocedores investigados, es la que más tipos reconoce. Estos son *ORGANIZATION*, *PERSON*, *LOCATION*, *GPE*, *DURATION*, *DATE*, *CARDINAL*, *PERCENT*, *MONEY*, *MEASURE* y *FACILITY*. Sin embargo, para este proyecto solo los primeros cuatro tipos serán utilizados considerando *LOCATION* y *GPE* [2] como tipos de lugares.

Es importante destacar que el reconocimiento de entidades de *NLTK* está orientado al idioma inglés, por lo tanto no tiene un buen comportamiento con textos en español.

Dandelion

Como parte de este proyecto se investigó también *Dandelion API* [12]. *Dandelion* es un servicio de análisis semántico de texto el cual incluye

[1] El modelo entrenado con CoNLL2002 utiliza la etiqueta PER mientras que el entrenado con AnCora utiliza la etiqueta PERS.

[2] Entidad geo-política, estas suelen ser ciudeades, países, continentes, etc.

extracción y clasificación de entidades nombradas y análisis de sentimientos, entre otras herramientas.

En particular, la investigación fue enfocada en la extracción y clasificación de entidades nombradas. La API consiste en darle un texto de entrada devolviendo así una lista de spans que fueron reconocidos como entidades nombradas junto a sus etiquetas.

Finalmente, la opción de usar este servicio fue descartada ya que el procesamiento de los textos ocurre enteramente en el servidor de *Dandelion* y por la natuaraleza y confidencialidad de los textos que *LUZ* procesa, esto no es aceptable.

2.1.2. Etiquetado BIO

Este método, también conocido como IOB, permite tratar el reconocimiento de entidades nombradas como una secuencia de etiquetado palabra por palabra capturando tanto las entidades como su tipo [20].

Para este método de etiquetado existen 3 tipos de etiqueta: B significa comienzo, I dentro y O fuera (el significado de las letras corresponde al idioma inglés).

En el caso de las etiquetas B e I, se agrega el tipo de elemento que corresponde, indicando si el *token* está al comienzo o es interno a él. Por ejemplo, en el caso de NER, se indica si está al comienzo o es interno a la entidad nombrada.

Para el ejemplo 2.1 el etiquetado BIO de entidades nombradas sería el siguiente:

Palabra	Etiqueta BIO
Se	O
encontró	O
a	O
José	B-PER
Pérez	I-PER
en	O
la	O
plaza	B-LOC
Varela	I-LOC

Tabla 2.1: Tabla de etiquetado BIO

Existen otras variantes del etiquetado BIO como lo son IO, donde la primera I que aparece se toma como una B, y BIOES, donde la E indica el final de una entidad nombrada y la S representa una de una sola palabra.

2.1.3. Reconocimiento de relaciones

Según Jurafsky y Martin [20], este proceso se puede definir como encontrar un conjunto de tuplas ordenadas sobre elementos de un dominio. En general para las aplicaciones de extracción de información, el dominio son las entidades nombradas que ocurren en el texto, las entidades nombradas que resultan de la resolución de correferencias o las entidades seleccionadas de una ontología de dominio.

A su vez, estos autores afirman que en la actualidad existen 5 clases principales de algoritmos para extraer relaciones, estos son: patrones manuales, aprendizaje automático supervisado, aprendizaje automático semi-supervisado (utilizando *bootstrapping* o mediante supervisión distante) y no supervisado, los cuales se introducirán a continuación.

2.1.3.1. Extracción de relaciones con patrones

Uno de los primeros, y aún comúnmente usado según Jurafsky y Martin, es el algoritmo para extraer relaciones mediante patrones léxico-sintácticos desarrollado por Hearst [16].

Este algoritmo consiste en extraer, de forma manual, patrones comunes a los textos para poder construir un reconocedor capaz de predecir nuevas relaciones que tengan una estructura similar a alguno de estos.

En general, este tipo de algoritmos tienen una alta precisión y pueden ser muy útiles para dominios específicos, aunque también tienen baja recuperación[1] y es muy trabajoso crear todos los patrones para un determinado dominio.

2.1.3.2. Extracción de relaciones con algoritmos de aprendizaje supervisado

La utilización de aprendizaje supervisado para la extracción de relaciones sigue un esquema en donde se selecciona un conjunto fijo de relaciones y

[1]La **recuperación** es una métrica que expresa la proporción de elementos positivos reales que son recuperados por el sistema.

entidades para formar un corpus anotado manualmente [20]. La forma de selección de estos conjuntos es anotando manualmente textos del dominio para luego entrenar clasificadores que anoten automáticamente otros textos no vistos anteriormente.

En general, si el conjunto de entrenamiento es similar al de evaluación y si hay suficientes textos anotados manualmente, los sistemas de extracción de relaciones supervisados pueden tener una alta precisión. Pero anotar un amplio conjunto de entrenamiento es muy costoso y los modelos supervisados son frágiles, ya que no generalizan bien para distintos tipos de textos.

Por estos motivos gran parte de la investigación de extracción de relaciones se concentra en utilizar algoritmos semi-supervisados o no supervisados.

2.1.3.3. Extracción de relaciones con algoritmos semi-supervisados mediante bootstrapping

El aprendizaje automático supervisado asume que se tiene mucha información etiquetada, lo cual es costoso [20]. Pero si se tienen algoritmos con alta precisión se pueden utilizar como base para tener una gran cantidad de anotaciones. Por ejemplo, se podrían utilizar algoritmos de la sección 2.1.3.1.

Luego de lograr tener mucha información base se la puede utilizar para crear un clasificador de *bootstrapping*, el cual toma las entidades emparejadas en las anotaciones mencionadas para encontrar oraciones (ya sea en la web o en cualquier otro conjunto de datos) que contienen las mismas entidades. De estas oraciones se extrae y se generaliza el contexto para aprender nuevos patrones.

Los sistemas de *bootstrapping* también asignan valores de confianza a nuevas tuplas para evitar *cambios semánticos* que puedan llegar a producir patrones problemáticos. Los valores de confianza se basan en la eficacia del patrón con respecto al actual conjunto de tuplas y en la productividad del patrón, que es la cantidad de tuplas que genera ese patrón en los textos.

Cambio semántico se le llama al fenómeno por el cual la correspondencia entre una palabra y la entidad o proceso al que está conectada tiende a transformarse gradualmente. Por ejemplo, la palabra *bizarro* actualmente es utilizada para describir algo que es extraño o fuera de lo común. Sin embargo, según el diccionario de la RAE, significa “generoso, lucido, espléndido”.

2.1.3.4. Extracción de relaciones con algoritmos de supervisión distante

Según Jurafsky y Martin, generar textos anotados manualmente con relaciones es muy costoso, pero hay formas para encontrar fuentes de información indirectas. Por ejemplo el método de supervisión distante de Mintz [27] combina las ventajas de *bootstrapping* con aprendizaje supervisado. Los algoritmos de este tipo toman como entrada una base de datos muy extensa para contar con un gran número de ejemplos, aprendiendo muchos patrones que generan ruido y luego combinándolos para construir un clasificador supervisado.

Los algoritmos de supervisión distante comparten ventajas de cada uno de los métodos que se examinaron anteriormente. Como por ejemplo de aprendizaje supervisado que toma muchas funciones a partir de la información detallada creada manualmente. A su vez pueden hacer uso de información con alta precisión, como en los algoritmos basados en patrones, para crear un clasificador para relaciones entre entidades.

Este tipo de algoritmos no usa corpus de entrenamiento etiquetados por lo que no son permeables a cuestiones inherentes al tipo de textos del conjunto de entrenamiento y además están diseñados para contar con una gran cantidad de información sin anotar. Estos métodos de supervisión distante solo pueden ayudar a extraer relaciones con una amplia base de datos. Pero para extraer nuevas relaciones sin esta última, o relaciones para nuevos dominios, se deben usar métodos no supervisados.

2.1.3.5. Extracción de relaciones con algoritmos no supervisados

Según Jurafsky y Martin, el objetivo de la extracción de información no supervisada es extraer relaciones de la web cuando no se tiene un conjunto de entrenamiento o siquiera una lista de tipos de relaciones.

La gran ventaja de los algoritmos no supervisados de extracción de relaciones es la habilidad de manejar un gran número de relaciones sin tener que especificarlas anteriormente, pero la desventaja de ellos es tener que estandarizar las relaciones para otras fuentes de conocimiento.

2.2. Resolución de correferencias

La resolución de correferencias es un proceso lingüístico que busca determinar si dos menciones en un texto corresponden a una misma entidad [33].

La correferencia ocurre entre dos unidades lingüísticas que se relacionan debido a que existe un concepto en la realidad al cual hacen referencia.

Para explicar el problema que implica la resolución de correferencias se presenta un ejemplo traducido[1] propuesto por Hirst en 1979 [18].

La chinchilla se comió mi retrato de Richard Nixon anoche. Lo devoró tan rápido que ni siquiera tuve oportunidad de salvar el marco.

De alguna manera es posible determinar que por *"Richard Nixon"* el narrador se refiere a Richard Milhous Nixon, ex-presidente de los Estados Unidos de América. Por otro lado, se puede entender que la chinchilla involucrada es una en particular, una que es conocida por el narrador y el oyente. Se entiende también que *"lo devoró"* refiere al ya mencionado acto de la mencionada chinchilla de comerse el mencionado retrato. Por último, se entiende que *"el marco"* es el marco del retrato mencionado.

Todas las derivaciones explicadas previamente son hechas por el receptor del mensaje al momento que lo recibe. Una persona lo puede hacer fácilmente cuando conoce el contexto en el que el mensaje se emite y a su vez tiene conocimiento histórico y de actualidad del mundo.

Mientras el receptor procesa el mensaje, este toma decisiones basadas en la información que ya fue mencionada en el discurso identificando así conceptos que fueron introducidos inicialmente. Este tipo de correferencias se conoce como anáfora.

La **anáfora** es una estrategia para referir de manera abreviada a una (o varias) entidades que ya fueron mencionadas en el discurso con la suposición de que el receptor sea capaz de descifrar la referencia e identificar la entidad. La referencia es llamada anáfora, mientras que la entidad a la cual refiere es su referente o antecedente.

A continuación se puede ver un ejemplo de anáfora:

La mujer compró un libro de física. Ella se lo prestó a Mateo.

[1] originalmente en inglés

Los pronombres *"ella"* y *"lo"* son anáforas con referentes *"la mujer"* y *"un libro de física"*, respectivamente.

2.2.1. Algunas investigaciones anteriores

Se han publicado varios trabajos que abordan la resolución de correferencias. Si bien la mayoría de estos estudian la resolución de correferencias en el idioma inglés, se pueden encontrar algunos que lo hacen para el español.

En primer lugar, Emili Sapena, Lluís Padró y Jordi Turmo presentaron en 2010 *RelaxCor: Un sistema de resolución de correferencias de código abierto* [35] [36].

Este sistema representa el problema de resolución de correferencias como un grafo con menciones como vértices. Las menciones están conectadas entre sí por aristas donde cada arista está etiquetada por la suma de los pesos de las restricciones que aplican al par de menciones.

Luego, el sistema utiliza *relaxation labelling*[1] para resolver el proceso de partición de grafos satisfaciendo tantas restricciones como sea posible, obteniendo así valores de hasta *81.55* para la medida *F1*[2].

Por otro lado, Cantamutto et al., presentan, en 2015 [3], la aplicación del *modelo Multi Sieve Pass* (propuesto por Raghunathan et al. 2010 [33]) en textos en español.

Este modelo utiliza una sucesión de tamices que van enriqueciéndose en las etapas subsiguientes con información de las anteriores para ir perfeccionando la agrupación de menciones.

El funcionamiento de este algoritmo consiste en que a partir de una determinada mención, el filtro resuelve si es incapaz de determinar el antecedente, o si la identificación queda pendiente para pasos posteriores, o de lo contrario, reconoce la entidad nombrada.

[1] *Relaxation labelling* es una técnica para asignar etiquetas o valores globalmente consistentes a nodos en una red sujeta a restricciones locales, propagando iterativamente los efectos de las restricciones a través de la red. [Catalogue of Artificial Intelligence Tools. Alan Bundy]

[2] *F1* es calculada como

$$F1 = 2 \cdot \frac{P \cdot R}{P + R} \tag{2.2}$$

donde P es *precisión* y R es *recuperación*.

A continuación se menciona brevemente cada uno de los 7 filtros que conforman el algoritmo:

1. Cotejo exacto: dos construcciones iguales se resuelven como correferentes.

2. Construcciones previsibles: para encadenar dos menciones se requiere que se satisfagan alguna de las siguientes condiciones.

    a. Una de las menciones está en una construcción de aposiciones. Es decir, una construcción con dos elementos gramaticales unidos, uno de los cuales especifica al otro. Ejemplo: al gobierno del [primer emperador], [César Augusto].

    b. Las menciones están en una oración copulativa. Es decir, una oración que posee un verbo que ejerce como nexo de unión entre significados (copulativo). Ejemplo: [Tiberio] era [hijo de Livia].

    c. El candidato tiene un sustantivo que actúa como modificador.

    d. La mención es un pronombre relativo que modifica el núcleo de la frase nominal antecedente.

    e. Cuando una de las menciones es un acrónimo de la otra.

    f. Una de las menciones es un gentilicio de la otra.

3. Cotejo estricto de núcleos: añade restricciones a núcleos idénticos a partir de cotejar si efectivamente coincide en el sintagma. Ejemplo: *el Imperio Romano* y *el Imperio* son correferencias pero *el auto blanco* y *el auto negro* no lo son.

4. Cotejo laxo de núcleos: Utiliza conjuntos de candidatos a antecedente y solo se aplica a entidades nombradas.

5. Pronombres: en los seis pasos previos, el modelo ignoró la correferencia pronominal. En esta etapa, al igual que la anterior, la herramienta está preparada por los pasos previos que han ido creando listas de candidatos para la resolución de las correferencias pronominales. Se realiza un cotejo de concordancia: género, número, persona, animacidad, etiqueta NER.

Además, entre los puntos 3 y 4 se encuentran dos filtros más que actúan como relajaciones a algunas restricciones del tercer filtro.

Pasos	MUC			B^3			Pairwise		
	P	R	F1	P	R	F1	P	R	F1
{1}	95.9	31.8	47.8	99.1	53.4	69.4	96.9	15.4	26.6
{1,2}	95.4	43.7	59.9	98.5	58.4	73.3	95.7	20.6	33.8
{1,2,3}	92.1	51.3	65.9	96.7	62.9	76.3	91.5	26.8	41.5
{1,2,3,4}	91.7	51.9	66.3	96.5	63.5	76.6	91.4	27.8	42.7
{1,2,3,4,5}	91.1	52.6	66.7	96.1	63.9	76.7	90.3	28.4	43.2
{1,2,3,4,5,6}	89.5	53.6	67.1	95.3	64.5	76.9	88.8	29.2	43.9
{1,2,3,4,5,6,7}	83.7	74.1	78.6	88.1	74.2	80.5	80.1	51.0	62.3

Tabla 2.2: Performance del modelo Multi Sieve Pass

La tabla 2.2, extraída de [33], muestra los resultados en cada fase del modelo para el corpus *ACE2004-ROTH-DEV* el cual es un subconjunto de desarrollo por Bengston y Roth (2008), tomado del corpus usado en 2004 Automatic Content Extraction (ACE). Este contiene 68 documentos y 4,536 menciones en inglés.

Cabe destacar que los resultados presentados no corresponden a un corpus en español ya que estos no fueron encontrados.

2.3. Bases de conocimiento

Según M. Nickel, K. Murphy, V. Tresp y E. Gabrilovich en su artículo A Review of Relational Machine Learning for Knowledge Graph, un grafo de conocimiento es "[...] un grafo estructurado que representa bases de conocimientos que almacenan información factual en forma de relación entre entidades"[25]. A partir de esto se puede ver un grafo de conocimiento como un grafo de datos [1] que acumula y transmite conocimientos del mundo real, cuyos nodos representan entidades y cuyas aristas representan relaciones entre las mismas.

Los conocimientos a los que hace referencia un grafo de conocimiento pueden acumularse a partir de fuentes externas o del mismo grafo. El conocimiento puede componerse por declaraciones simples como *"El ornitorrinco es un mamífero"*, o por declaraciones cuantificadas como *"Todos los mamíferos son de sangre caliente"*. Mientras las declaraciones simples

[1] Conjunto de vértices y aristas que permite representar datos interconectados, así como las relaciones entre ellas, de forma comprensible y como un único y más amplio conjunto de datos

pueden acumularse como aristas en el grafo, es necesaria una base de conocimientos para que pueda acumular declaraciones cuantificadas.

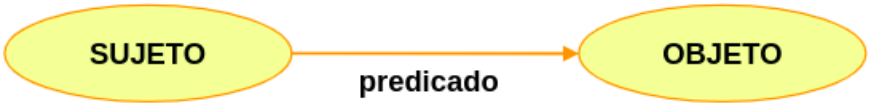
La presencia de una base de conocimientos implica que se utilizan métodos deductivos para derivar y acumular más conocimientos. Por ejemplo, a partir de *"El ornitorrinco es un mamífero"* y *"Todos los mamíferos son de sangre caliente"* se deriva que *"El ornitorrinco es de sangre caliente"*. Por lo tanto, un grafo de conocimiento tiene un grafo de datos y además un mecanismo que le da la capacidad de inferir. La utilización de mecanismos de inferencias a partir de un conjunto de hechos o afirmaciones permite hacer explícito lo implícito.

2.3.1. RDF

RDF (del inglés *Resource Description Framework*) es un modelo de datos basado en grafos que se utiliza para compartir información a través de la Web, el cual es estandarizado por *World Wide Web Consortium* (*W3C*). Este modelo de datos pone especial énfasis en la identificación global de todos los elementos del modelo utilizando *Uniform Resource Identifiers* (*URIs*) [2].

Según Rachel Heery [17], una característica que hace destacar a este modelo de otros es que fue diseñado teniendo en cuenta las características de la Web. Por lo que posibilita la combinación de distintas fuentes de datos, aunque los esquemas subyacentes sean distintos.

RDF está diseñado para representar información de una manera suficientemente estructurada como para ser utilizada por las aplicaciones pero mínimamente restrictiva. El modelo de datos en RDF se basa en la idea general de que las cosas que se quieren describir tienen ciertas propiedades, que a su vez tienen ciertos valores. Por lo tanto, las declaraciones siempre tendrán la forma *sujeto-predicado-objeto* denominada *tripleta* [22]. Este modelo suele representarse gráficamente mediante nodos y aristas de la siguiente manera:



2.3.2. Ontologías

Según Tom Gruber, una ontología es un conjunto de primitivas representacionales con las que modelar un domino de conocimiento o discurso. Las primitivas representacionales son típicamente clases, atributos, y relaciones entre los miembros de una clase [15].

Partiendo de Noy y McGuinness [29], se puede decir que los principales objetivos de las ontologías son los siguientes:

a) Compartir la comprensión común de la estructura de información entre personas o agentes de software.

b) Permitir la reutilización del conocimiento perteneciente a un dominio.

c) Permitir hacer explícitos los supuestos de un dominio.

d) Separar el conocimiento de un dominio del conocimiento que se puede denominar operacional.

e) Analizar el conocimiento de un campo, por ejemplo en lo que se refiere al estudio de los términos y relaciones que lo configuran ya sea formalmente o no.

Visto lo anterior, se puede concluir que la meta final al momento de desarrollar ontologías es mejorar la representación de la información y los sistemas de recuperación de información.

Capítulo 3

Diseño de la solución

En este capítulo se describe la solución propuesta para resolver los objetivos planteados. A continuación, se detallan cada una de las etapas del proceso diseñado, el cual tiene como entrada un conjunto de archivos en texto plano y como salida una base de conocimiento de alcance local a cada archivo de texto.

3.1. Preprocesamiento del texto

Este proyecto requiere procesar textos muy particulares. En primer lugar, estos responden a normas de otra época y muchos además son del ámbito militar, por lo tanto es esperable que se repitan ciertas estructuras a lo largo de los documentos, lo cual es una ventaja para hacer un procesamiento de los textos mediante patrones. Por otro lado, cada texto que procesa este sistema proviene de la plataforma LUISA [1]. Esto implica que:

- Cada texto es un puzzle de transcripciones de los distintos bloques de imagen que lo componen.
- Dado que los microfilms se han deteriorado hay palabras que no tienen transcripción.
- Las transcripciones de algunos sellos quedan en medio de frases o incluso palabras.
- Como las transcripciones fueron hechas a mano por distintas personas
    - se encuentran errores de tipeo,

[1] A futuro se utilizarán textos obtenidos de sistemas automáticos de digitalización de textos a partir de imágenes, como las herramientas de OCR. Esto puede agregar nuevas reglas de preprocesamiento o dejar otras obsoletas.

- palabras faltantes o sin sentido,
- y uso incoherente de letras mayúsculas, minúsculas, espacios y otros signos de puntuación.

Lo anterior dificulta el reconocimiento de entidades nombradas y su posterior procesamiento, ya que algunos nombres de personas o lugares aparecen escritos de distinta manera a lo largo de un mismo texto o no respetan la inicial mayúscula, característica que suelen usar los reconocedores para identificar estos tipos de entidades.

Debido a estas razones, se tomó la decisión de implementar una etapa de preprocesamiento de los textos que toma ciertas reglas y se las aplica con el objetivo de mitigar algunos de los problemas mencionados. A continuación se mencionan los tipos de reglas propuestos hasta el momento:

- Reemplazar, que sustituye una secuencia de caracteres por otra (que podría ser vacía). Este tipo de preprocesamiento permitió quitar la transcripción de los sellos de los documentos.
- Remover espacios extra, que evita que la tokenización devuelva *tokens* vacíos que estorben el posterior procesamiento del texto.

3.2. Reconocimiento de entidades nombradas

Para resolver el problema del reconocimiento de entidades nombradas se estudiaron varias herramientas y se integraron las que correspondían con *LUZ*, lo que será explicado con mayor detalle a lo largo de este capítulo. Además se decidió implementar un reconocedor de entidades a partir de patrones identificados en los textos y otro a partir de los diccionarios provistos por Jorge Tiscornia Bazzi.

Estos módulos fueron implementados para ejecutar los reconocedores en cuestión desde un mismo motor que estandarice su entrada y salida con el objetivo de evaluar y comparar su desempeño según los distintos tipos de entidades y posteriormente generar, a partir de los mejores, un motor de NER que optimice el rendimiento de esta etapa de *LUZ*. Por ejemplo, si *reconocedor*1 tiene el mejor desempeño reconociendo entidades de tipo persona pero *reconocedor*2 es el que da mejores valores para el reconocimiento de fechas, se podría implementar un motor de NER que utilice *reconocedor*1 para reconocer las personas y *reconocedor*2 para las fechas.

3.2.1. Patrones identificados

Como se mencionó en la sección anterior (3.1), un gran porcentaje de los textos que van a ser procesados por el sistema implementado provienen de un ámbito formal militar, lo que implica que se repitan ciertas estructuras a lo largo de los documentos. Esto facilita el uso de patrones tanto para el reconocimiento de entidades nombradas como para la extracción de relaciones tal como se verá en la sección 3.4.

Se implementaron entonces varios patrones para distintos tipos de entidades nombradas que fueron modelados a partir de la lectura de 100 documentos y del conocimiento compartido por Fabián Hernández Muñiz y Jorge Tiscornia Bazzi.

Algunos de estos patrones corresponden con las distintas maneras en las que aparecen escritos los nombres de las personas. Además de la forma convencional, donde los nombres van seguidos por los apellidos y tienen todos mayúscula al inicio, en muchos de los documentos militares aparecen los apellidos escritos enteramente en letras mayúsculas tanto antes como después de los nombres (que siempre mantienen su mayúscula inicial). Por ejemplo *"MORALES PEREZ Juana"* o *"José Enrique DOMÍNGUEZ GONZALEZ"*.

Otras entidades expresadas de forma particular en estos textos son las fechas. En algunos textos los años son representados por sus últimas 2 o 3 cifras, por lo que es común encontrar expresiones como *"año 74"* o *"año 974"*. Además, las fechas de los documentos militares suelen aparecer con el formato $DDHHMM < MES > AAA$, donde por ejemplo, $040910JUN975$ refiere al día 4 de junio del año 1975 a la hora 9 y 10 minutos.

Como en una gran cantidad de textos se hace mención a las personas por su número de requeridas, se implementó también un patrón que modela estas entidades.

Por último, para reconocer lugares se implementaron patrones a partir de las palabras *escuela*, *liceo* y *finca*, que se vio que eran tipos de lugares que aparecen con gran frecuencia en los textos a procesar. También se implementó una regla que reconoce lugares a partir de las expresiones *"domiciliado en"* y *"domiciliada en"*.

En total se implementaron 12 patrones para el reconocimiento de entidades nombradas: 1 para números de requeridos, 2 para personas, 4 para fechas y 5 para lugares. Se pueden ver con detalle en .1.

3.2.2. Herramientas investigadas

En esta sección se presentan ejemplos de la utilización de las herramientas de procesamiento de lenguaje natural que se decidieron integrar al proyecto para realizar el reconocimiento de entidades nombradas.

SpaCy

Como se menciona en el capítulo anterior, algunas funcionalidades de esta biblioteca necesitan que se carguen previamente modelos entrenados. Para este proyecto se utiliza el modelo *es_core_news_md*, el cual cuenta con un componente de NER que clasifica las entidades en las categorías persona, organización, lugar y miscelánea usando las etiquetas PER, ORG, LOC y MISC respectivamente.

Tomando como entrada el texto

La Secretaria General de Estado, Melina Carreras, se trasladó a la Suprema Corte de Justicia en la mañana del 5 *de mayo para prestar declaración.* (3.1)

SpaCy devuelve el siguiente resultado:

La	O
Secretaria	B-ORG
General	I-ORG
de	I-ORG
Estado	I-ORG
,	O
Melina	B-PER
Carreras	I-PER
,	O
se	O
trasladó	O
a	O
la	O
Suprema	B-LOC

Corte	I-LOC
de	I-LOC
Justicia	I-LOC
en	O
la	O
mañana	O
del	O
5	O
de	O
mayo	O
para	O
prestar	O
declaración	O
.	O

FreeLing

A continuación se muestra un ejemplo particular de la información que se puede obtener de un *token* utilizando *FreeLing*:

id	t2.4
begin	156
end	171
form	Melina_Carreras
lemma	melina_carreras
tag	NP00SP0
ctag	NP
pos	noun
type	proper
neclass	person
nec	PER

Tabla 3.2: Datos *FreeLing* para la entidad *Melina Carreras*

Del ejemplo anterior se pueden señalar varios puntos. Primeramente, este *token* corresponde a la entidad nombrada *Melina Carreras*. Ahora, esta entidad está conformada por dos *tokens*, no uno. Esto se trata de una particularidad que *FreeLing* tiene. Cuando se detecta una entidad nombrada, los *tokens* de esta son concatenados por guiones bajos (_) agrupándolos en un solo *token*. Es por esto que en el ejemplo planteado la entidad nombrada *Melina Carreras* es reemplazada por *Melina_Carreras*. Por último, el dato *nec* establece que esta entidad nombrada es clasificada como persona.

Para personas, lugares y organizaciones la clasificación está dada por el dato *nec*, en cambio para el caso en que la entidad nombrada es una fecha, por ejemplo *5 de mayo*, el analizador devuelve un solo *token* de la forma *5_de_mayo* pero con el dato *pos="date"*. Este dato es el que permite conocer que el *token* corresponde a una fecha.

Dado el ejemplo 3.1 *FreeLing* devuelve el siguiente resultado:

La	O
Secretaria	B-PER
General	I-PER
de	I-PER
Estado	I-PER
,	O
Melina	B-PER
Carreras	I-PER
,	O
se	O
trasladó	O
a	O
la	O
Suprema	B-ORG

Corte	I-ORG
de	I-ORG
Justicia	I-ORG
en	O
la	O
mañana	O
del	O
5	B-DATE
de	I-DATE
mayo	I-DATE
para	O
prestar	O
declaración	O
.	O

Stanza

Al ejecutar el reconocedor de entidades de *Stanza*, tanto con el corpus *CoNLL2002* como con *AnCora*, para el ejemplo 3.1 se obtiene el siguiente resultado:

La	O
Secretaria	B-PER
General	I-PER
de	I-PER
Estado	E-PER
,	O
Melina	B-PER
Carreras	E-PER
,	O
se	O
trasladó	O
a	O
la	O
Suprema	B-ORG

Corte	I-ORG
de	I-ORG
Justicia	E-ORG
en	O
la	O
mañana	O
del	O
5	O
de	O
mayo	O
para	O
prestar	O
declaración	O
.	O

Como se puede ver, *Stanza* devuelve la anotación BIOES del texto de entrada, por lo que es necesario traducir esta notación a la notación BIO para seguir el estándar de los demas reconocedores de *LUZ*.

NLTK

Otro de los reconocedores de entidades nombradas soportados por *LUZ* hasta el momento se basa en la librería *NLTK*. El resultado de ejecutar este reconocedor para el ejemplo 3.1 es el siguiente:

La	B-PER
Secretaria	O
General	O
de	O
Estado	O
,	O
Melina	B-PER
Carreras	I-PER
,	O
se	O
trasladó	O
a	O
la	O
Suprema	B-PER

Corte	I-PER
de	O
Justicia	B-ORG
en	O
la	O
mañana	O
del	O
5	O
de	O
mayo	O
para	O
prestar	O
declaración	O
.	O

Como se mencionó en el capítulo anterior, el reconocimiento de entidades de *NLTK* no tiene un buen rendimiento con textos en español y en este ejemplo se nota claramente ya que la única entidad nombrada detectada y clasificada correctamente es el nombre de la persona *Melina Carreras*.

Cabe destacar para todos los reconocedores que si bien *Secretaria General de Estado* y *Melina Carreras* conforman una aposición, aludiendo así a la misma entidad nombrada y en particular a la misma persona; *LUZ* tratará estas menciones como dos entidades nombradas diferentes.

Sin embargo como trabajo futuro, esta estructura lingüística podría procesarse en la etapa de resolución de correferencias así como fue propuesto

en el *modelo Multi Sieve Pass* por Raghunathan et al. 2010 [33]. De esta manera, estas dos entidades nombradas podrían ser guardadas en la base de conocimiento con una relación de correferencia entre ellas.

3.2.3. Entidades nombradas a partir de diccionarios

Siguiendo con la motivación de introducir reconocedores que se ajusten más al dominio específico de los textos, se propone un reconocedor basado en diccionarios, los cuales contienen entidades nombradas ya conocidas, buscando las ocurrencias exactas para cada una de estas.

Un diccionario es un archivo que contiene una lista de expresiones que se asocia a una clase de entidad nombrada. Las porciones de texto en los diccionarios son las entidadas nombradas conocidas que se quieren extraer de los textos.

Es de importancia notar que esta propuesta busca obtener un alto grado de precisión tanto en la detección de entidades nombradas como en su clasificación.

Para plantear un ejemplo concreto, se supone que el texto

$$\textit{El jefe de la Seccional 9a se trasladó a la Suprema Corte de Justicia en la mañana del 5 de mayo para prestar declaración.} \tag{3.2}$$

es procesado por el reconocedor que dispone de un solo diccionario asociado a la clase ORG que contiene las porciones de texto:

- *Seccional 9a*
- *Suprema Corte de Justicia*

Entonces etiquetaría *Seccional 9a* y *Suprema Corte de Justicia* como organizaciones (ORG).

Ahora, como en la etapa de reconocimiento de entidades nombradas de este proyecto se definió como postcondición que el resultado no contenga entidades nombradas superpuestas, es necesario atender algunos casos borde que suceden para este reconocedor cuando hay coincidencias superpuestas.

Por ejemplo, se supone que un diccionario que contiene las porciones de texto

- *Suprema Corte de Justicia*
- *Corte de Justicia*

es utilizado por el reconocedor para procesar el texto 3.2.

El reconocedor consume el texto priorizando las coincidencias con mayor largo, por lo que la salida del mismo solo consta de la entidad nombrada *Suprema Corte de Justicia*.

Por otro lado, cuando las coincidencias superpuestas tienen el mismo largo, el caso se resuelve tomando la entidad nombrada que comienza antes.

Por ejemplo, suponiendo un texto que contiene *"Camino Arenas Claras"*, y un diccionario que contiene las expresiones:

- *Camino Arenas*
- *Arenas Claras*

el resultado del reconocedor solo consta de la entidad nombrada *Camino Arenas* por encontrarse más a la izquierda en el texto.

Por último, se supone que el texto a analizar incluye la porción de texto *"Julio Herrera y Reissig"*. Además, existen dos diccionarios que también contienen esa expresión. El primero está asociado a la clase PER y el segundo a LOC. Como los reconocedores de este proyecto no están diseñados para devolver entidades nombradas con múltiples etiquetas, se resuelve dando prioridad a los diccionarios que están configurados primeramente. En la sección 4.3.2 se explica en detalle la configuración de los diccionarios.

3.2.4. Tokenización

Para hacer posible la evaluación de rendimiento del reconocimiento de entidades nombradas fue necesario tokenizar los textos y anotar cada uno de los *tokens* utilizando la notación BIO. En la sección 6.1.1 se detalla la necesidad de utilizar esta notación.

Para lograr esta comparación fue imprescindible utilizar el mismo algoritmo de tokenización para cada uno de los reconocedores, de lo contrario la lista de *tokens* podría diferir en longitud dependiendo del algoritmo utilizado.

Por ejemplo, la tokenización realizada por la librería *SpaCy* para la porción de texto *"Covid-19 en Uruguay: 164 nuevos casos"* es:

Covid-19 , en , Uruguay , : , 164 , nuevos , casos

mientras que la tokenización realizada por *Stanza* es:

Covid-19 , en , Uruguay: , 164 , nuevos , casos

Se puede notar que *SpaCy* separa la palabra *Uruguay* y el signo de puntuación *:* a diferencia de *Stanza*.

Finalmente, se decidió utilizar el tokenizador provisto por la libería *NLTK* independientemente del reconocedor elegido para realizar el reconocimiento de entidades nombradas.

3.3. Resolución de correferencias

Si bien esta área tiene mucho potencial para continuar investigando (incluso aplicando algunas de las técnicas que se mencionaron en el marco teórico siendo adaptadas a los textos del dominio), para el alcance de este proyecto se plantea acotar el problema a la resolución de correferencias que se dan exclusivamente entre entidades nombradas.

3.3.1. Algoritmo de resolución de correferencias entre entidades nombradas

El algoritmo de resolución propuesto busca desambiguar entidades nombradas encontradas en el mismo texto. Este es un proceso simple que no utiliza información sintáctica o semántica ni conocimiento del mundo o de cualquier otro externo al texto, sino que utiliza como entrada el resultado de la etapa de reconocimiento de entidades nombradas. Por otro lado, el resultado del algoritmo es el conjunto de cadenas de correferencias entre las entidades.

La idea principal del procedimiento es procesar el texto de izquierda a derecha decidiendo si cada una de las entidades nombradas que se encuentran corresponden a una primera mención[1] o a una referencia de una entidad nombrada que ya fue mencionada anteriormente.

[1]Se dice que una entidad nombrada es *primera mención* cuando esta no hace referencia a otra anteriormente nombrada.

Como ejemplo, se plantea el siguiente texto:

$$\begin{aligned} &\textit{El ex jugador de fútbol, Diego Forlán, nació en} \\ &\textit{Uruguay el año } 1979. \textit{ Diego fue reconocido como el} \\ &\textit{mejor jugador del mundo en la Copa Mundial de Fútbol} \\ &\textit{del año } 2010. \textit{ Peñarol fue el último equipo en el que} \\ &\textit{Forlán jugó profesionalmente.} \end{aligned} \tag{3.3}$$

Se puede observar que en este texto aparecen seis entidades nombradas donde la entidad nombrada *Diego Forlán* es mencionada tres veces.

El algoritmo de resolución de correferencias resuelve este texto en un conjunto de siete cadenas de correferencias. Las más simples son cadenas de largo 1, lo que significa que en el texto sólo aparecen primeras menciones para estas entidades. Estas son *Uruguay*, *año 1979*, *Copa Mundial de Fútbol*, *año 2010* y *Peñarol*.

Por otro lado, se encuentran otras dos cadenas de correferencias: *Diego Forlán* ← *Diego* y *Diego Forlán* ← *Forlán* que tienen la particularidad de compartir la primera mención.

Cuando dos (o más) cadenas de correferencias comparten la primera mención, el sistema considerará todas las menciones de cada una de las cadenas como referencias de una única entidad nombrada (la cual es definida por la primera mención).

3.3.2. Existencia de correferencia entre dos entidades nombradas

Esta sección explica el mecanismo utilizado para decidir si dos entidades nombradas son correferentes.

Dos entidades nombradas serán correferentes entre sí cuando cumplan las siguientes condiciones:

- Ambas entidades comparten el tipo de entidad nombrada (por ejemplo: ambas son organizaciones)
- y cada uno de los *tokens* de una de las entidades nombradas cumplen lo siguiente:
    - Si el *token* es una abreviación de la forma *A.* (una sola letra seguida

de un punto), entonces debe existir un *token* que inicie con la letra de la abreviación en la otra entidad nombrada.

- De lo contrario, el *token* debe existir en la otra entidad nombrada.

Es importante señalar que las comparaciones realizadas por el mecanismo anterior no son sensibles a mayúsculas, y que aunque pueda notarse que el mecanismo no es simétrico entre las dos entidades en cuestión, el algoritmo de resolución ejecuta estas verificaciones con ambas permutaciones de estas entidades, evitando así un sesgo.

A continuación se muestra un ejemplo para explicar lo anteriormente descripto:

$$\begin{gathered} \textit{Camila Santurión estudia Ingeniería en Computación.} \\ \textit{El hermano de Camila, Carlos Santurión, estudia bachillerato.} \\ \textit{C. SANTURIÓN planea estudiar arquitectura el año entrante.} \end{gathered} \tag{3.4}$$

Este caso intenta centrarse en las entidades nombradas *Camila Santurión*, *Carlos Santurión* y *C. SANTURIÓN*, las cuales son etiquetadas como personas.

Se puede notar que las entidades *Camila Santurión* y *Carlos Santurión* no son correferentes ya que no cumplen las condiciones planteadas. En cambio, *C. SANTURIÓN* cumple las condiciones para con cualquiera de las otras dos entidades, por lo tanto el algoritmo tiene que desambiguar esta correferencia para ser capaz de agregar la entidad nombrada a la cadena correcta de correferencias.

Para desambiguar este caso, el algoritmo simplemente se queda con la última entidad nombrada (*Carlos Santurión*, en este caso), suponiendo que en el discurso la entidad nombrada antecedente es probablemente la más cercana a la referencia.

3.4. Extracción de relaciones

Para llevar a cabo esta labor se utiliza un enfoque basado en patrones, como se explica en el marco teórico 2.1.3.1. Por lo tanto, se procedió a analizar los textos del dominio para identificar las distintas formas en que las relaciones se presentan. En este proyecto se identifican únicamente las relaciones que están

compuestas por 2 entidades nombradas consecutivas dentro del texto, las cuales cumplen con una expresión regular y están separadas a una distancia menor que una ventana de tolerancia. Esta se define como la máxima cantidad de *tokens* que puede haber entre la entidad sujeto y la entidad objeto, los cuales deben satisfacer la expresión regular establecida también en el patrón.

Además de lo explicado anteriormente, fue de interés del proyecto aplicar lematización para favorecer la extracción de relaciones, reemplazando cada *token* del texto por su correspondiente lema. El objetivo de esta variante es evitar que las conjugaciones de los verbos, el género y el número de los sustantivos y adjetivos afecten los resultados o eleve sustancialmente la complejidad de las expresiones regulares.

Las herramientas de lematización que se investigaron fueron *Stanza* y *FreeLing*, las cuales se pueden seleccionar al ejecutar el algoritmo, permitiendo luego comparar resultados entre una y otra.

3.4.1. Algoritmo de extracción de relaciones

Este algoritmo, como se mencionó anteriormente, está basado en la aplicación de patrones, por lo que para cada patrón se busca extraer todas las relaciones que cumplan con este.

La solución planteada utiliza la librería *NLTK* [1] para extraer las relaciones a partir de un patrón, el tipo de etiqueta de la primera y segunda entidad nombrada de la relación, el valor de la ventana y una lista con todos los *tokens* del texto. Como se puede notar, esta librería facilita significativamente la implementación del algoritmo de extracción de relaciones, por lo que en este proyecto se procedió a crear una configuración de patrones para tener centralizada toda la información de cada patrón y brindarle a esta herramienta los parámetros que necesita.

Básicamente el procedimiento consiste en aplicar cada uno de los patrones configurados a los *tokens* de un texto utilizando *NLTK* para extraer las relaciones y por último modificar el formato de salida de esta herramienta, simplificando así la generación del grafo de conocimiento.

Por otro lado, en el caso que el texto provisto al reconocedor de relaciones esté lematizado, se utilizarán patrones lematizados.

Dado el siguiente texto de ejemplo:

Diego Forlán, ex integrante de la selección uruguaya de fútbol, asumió el cargo de entrenador en Peñarol luego de retirarse. (3.5)

Se supone que se tiene una configuración de un patrón donde se busca satisfacer dentro de una ventana de tolerancia una entidad nombrada de tipo persona, luego un fragmento de texto que aplique al patrón *integrante de* y luego una segunda entidad nombrada de tipo organización. Además, se tiene un reconocedor de entidades nombradas que reconoce a *Diego Forlán* como persona, y como organizaciones a *selección uruguaya de fútbol* y a *Peñarol*.

Bajo estos supuestos, el algoritmo de extracción de relaciones ejecuta de la siguiente manera:

1. Se toma la configuración del patrón mencionado anteriormente y la lista de *tokens* del texto.

2. La librería *NLTK* recorre los *tokens* del texto de izquierda a derecha buscando satisfacer el patrón [1]. Por lo tanto, encuentra una relación del tipo *integrante de* entre *Diego Forlán* y *selección uruguaya de fútbol*.

3. Se procesa la salida del reconocedor facilitando así etapas posteriores.

Cabe destacar en el ejemplo anterior que *Peñarol* es una organización y aunque estuviera dentro de la ventana de tolerancia para *Diego Forlán*, el algoritmo identificaría a la primera entidad del tipo organización luego de encontrar la expresión regular.

[1] Esto significa que busca dentro de la ventana de tolerancia la primera entidad nombrada de tipo persona que es *Diego Forlán*, luego el texto *integrante de* y a continuación una organización.

3.5. Generación del grafo de conocimiento

Para facilitar la consulta y navegabilidad de los resultados obtenidos en las etapas anteriores se genera un grafo de conocimiento para cada texto.

3.5.1. Ontología LUZ [4]

Se definió una ontología propia para representar el conocimiento obtenido del sistema que propone este proyecto.

A grandes rasgos esta ontología permite representar algunas relaciones predefinidas que se dan entre personas, lugares, organizaciones y fechas. Además contempla la representación de correferencias que se puedan dar en los textos entre pares de entidades nombradas o entre entidades nombradas y otras unidades lingüísticas (como pueden ser los pronombres). Por último, soporta el conocimiento dado por la *provenance*[1] de cada ocurrencia en el texto, ya sean entidades nombradas o relaciones entre estas.

Entidades nombradas

La ontología define cuatro clases para representar cada uno de los tipos de entidades nombradas que *LUZ* maneja hasta el momento:

- La clase *Person* fue definida para representar personas, las cuales son reconocidas mayormente por su nombre. Además, es una subclase de la clase *Person* provista por *schema.org* [9].
- La clase *Place* fue definida como subclase de la clase *Place* provista por *schema.org* [10] para representar lugares obtenidos de los textos.
- La clase *Organization* fue definida para representar organizaciones y es subclase de la clase *Organization* provista por *schema.org* [8].
- Por último, la clase *Date* fue definida para representar fechas heredando las propiedades y restricciones de la clase *DateTimeDescription* provista por *OWL Time* [7].

La herencia definida para las clases propias de la ontología se debe a que *schema.org* y *OWL Time* ya definen propiedades para estos conceptos. Se utiliza la propiedad *name* de *schema.org* para representar el nombre de

[1] La **provenance** de algo es el lugar del cual proviene o del cual es originario.

los individuos de las clases *Person*, *Place* y *Organization*; y se utilizan las propiedades de la clase *DateTimeDescription* de *OWL Time* para representar un desglose de las fechas reconocidas en los textos.

Por último, la clase *NamedEntity* es definida como la unión de las cuatro clases anteriormente descriptas.

Relaciones

Como se explicó anteriormente, el algoritmo de detección de relaciones presentado en este proyecto está diseñado para reconocer relaciones binarias. Esto significa que al momento de representar la información de la instancia de una relación se dispone de dos entidades nombradas y el tipo de relación obtenido.

Sin embargo, la *reificación*[1] propuesta para modelar relaciones entre entidades nombradas en la base de conocimiento permite representar relaciones no binarias facilitando la adaptación de la generación del grafo en el futuro cuando sea necesario agregar algoritmos de detección de relaciones más sofisticados.

Esto implica modelar cada tipo de relación como una clase OWL diferente donde cada una de estas clases define sus propiedades específicas pudiendo representar apropiadamente cada uno de sus roles.

Primeramente, se define *Relation* como la clase de todos los tipos de relaciones que se hayan predefinido en el sistema o sean halladas en los textos.

Para cada tipo de relación, como por ejemplo una persona dada que tiene domicilio en un lugar dado, se definió una clase específica con propiedades específicas que permiten representar el conocimiento de la relación. Estas clases se listan a continuación junto a sus propiedades:

- *Arrest* → Representa la relación de una persona con el lugar y fecha de su detención. Esta clase es un caso claro de relación no binaria ya que cuenta con tres roles. Las propiedades utilizadas para registrar este conocimiento son *hasArrested*, *where* y *when*, respectivamente. Las tres tienen la clase *Arrest* como dominio y como rango las clases *Person*, *Place* y *Date*, respectivamente.

[1] *Reificación* es el proceso en el cual una idea abstracta es convertida en un modelo de datos explícito.

- *HomeLocation* → Representa la relación de una persona con su lugar de residencia. Las propiedades utilizadas para registrar este conocimiento son *hasDomiciled* y *where*, respectivamente. Ambas tienen la clase *HomeLocation* como dominio y como rango las clases *Person* y *Place*, respectivamente.
- *Membership* → Representa la relación de una persona con una organización de la que es o era miembro. Las propiedades utilizadas para registrar este conocimiento son *hasMember* y *membershipHasOrganization*, respectivamente. Ambas tienen la clase *Membership* como dominio y como rango las clases *Person* y *Organization*, respectivamente.
- *Release* → Representa la relación de una persona que estuvo detenida con la fecha de su liberación. Las propiedades utilizadas para registrar este conocimiento son *hasReleased* y *when*, respectivamente. Ambas tienen la clase *Release* como dominio y como rango las clases *Person* y *Date*, respectivamente.
- *Disposal* → Representa la relación de una persona con una organización judicial que dispone de ella, ya que dada la naturaleza de los textos a procesar aparecen varias expresiones como *"Raúl FERNANDEZ, indagado por distribuir material ofensivo al gobierno, a disposición de la Justicia Militar de Inst. de 6to. Turno"*. Las propiedades utilizadas para registrar este conocimiento son *hasPersonAtTheDisposal* y *theDisposalOf*. Ambas tienen la clase *Disposal* como dominio y como rango las clases *Person* y *Organization*, respectivamente.
- *Requisition* → Representa la relación de una persona con su número de requerida. Las propiedades utilizadas para registrar este conocimiento son *hasRequiredPerson* y *hasRequisitionNumber*, respectivamente. Ambas tienen la clase *Requisition* como dominio, mientras que la primera tiene a la clase *Person* como rango y la segunda a un *string*.

Cabe destacar que todas las propiedades que representan los roles de las relaciones tienen cardinalidad N:1. Esto implica que por un lado, los individuos implicados en la relación pueden participar en el mismo rol en otras instancias de relación. Por otro lado, si en la realidad existe un caso donde, por ejemplo, dos personas están domiciliadas en el mismo lugar será necesario instanciar

dos relaciones distintas en vez de solo una con dos instancias de la propiedad *hasDomiciled*.

Ocurrencias de entidades nombradas

La motivación para agregar el concepto de ocurrencia se da por la necesidad de conservar el conocimiento de la *provenance* de las entidades nombradas en cada uno de los textos.

En primer lugar, es de especial importancia conocer de qué texto proviene cada entidad nombrada. Pero el conservar la *provenance* de una entidad nombrada como tal no es suficiente, sino que es más útil la de una ocurrencia de entidad nombrada. Finalmente es esta la razón para incluir en el diseño este concepto de ocurrencia.

La clase *Occurrence* es definida junto a un conjunto de propiedades para las cuales su dominio es *Occurrence* y permiten representar el conocimiento buscado. Además, se define una clase *Sheet* que representa una hoja de texto.

En cuanto a las propiedades, *hasOccurredIn* es la responsable de relacionar una ocurrencia con la hoja de texto donde ocurrió. Es decir, su dominio es la clase *Occurrence* y su rango es *Sheet*. Esta relación es la que permite resolver la necesidad del origen de la ocurrencia.

Una vez definido lo anterior, también se considera el caso en el que una entidad nombrada aparezca más de una vez dentro de una sola hoja de texto. El diseño planteado para la ontología contempla y resuelve el caso agregando las propiedades *occurrenceBegin* y *occurrenceEnd* las cuales permiten indicar, para cada ocurrencia, el índice de inicio y final de la misma, respectivamente, en el texto de entrada.

Por último, es posible notar que este conjunto de clases y propiedades han sido definidas para mantener el conocimiento de la *provenance*, pero una ocurrencia aún necesita ser relacionada con un individuo de tipo entidad nombrada, es por esto que se introduce la propiedad *hasOccurrence* la cual es definida con el domino *NamedEntity* y con el rango *Occurrence*.

Ocurrencias de relaciones

La motivación para representar este conocimiento se da por la misma razón que para el caso de las entidades nombradas. Para representarlo es necesario apoyarse en el diseño descripto para las ocurrencias de entidades nombradas.

Este diseño define la clase *RelationOccurrence* junto a las siguientes propiedades:

- La propiedad *hasEntityOccurrence* es utilizada para representar las ocurrencias de las entidades nombradas que participan en la instancia de la relación, por lo que tiene a *RelationOccurrence* como dominio y *Occurrence* como rango.
- La propiedad *hasRelationOccurrence* es utilizada para enlazar una instancia de relación con la su ocurrencia reconocida del texto, por lo que en este caso el dominio es *Relation* y el rango es *RelationOccurrence*.

Por último, *hasOccurredIn* permite también relacionar una ocurrencia de relación con la hoja de texto donde ocurrió.

Finalmente, estas clases y propiedades en conjunto conforman un marco de trabajo que hace posible un sistema de consulta, basado en el lenguaje *SPARQL*, sobre la base de conocimientos resultante.

Correferencias

El diseño para soportar el conocimiento de las correferencias es muy simple después de las definiciones anteriores. Las correferencias entre entidades nombradas se pueden representar fácilmente relacionando dos ocurrencias de entidades nombradas.

Sin embargo, también es posible representar correferencias que no sean entidades nombradas, por ejemplo, los pronombres que sí refieren a entidades nombradas. Para esto el sistema crea ocurrencias que no pertenecen a una entidad nombrada, pero sí corresponden a un *span*[1] específico con sus correspondientes índices de inicio y final.

Por lo tanto, se introduce la propiedad *hasCoreference* entre ocurrencias (es decir, su dominio y rango es la clase *Occurrence*) que justamente permite representar que un ocurrencia en el texto tiene otra que la correfiere.

[1] Porción de texto.

3.5.2. Ejemplo de un grafo generado a partir de un texto

Se presenta un simple texto el cual será de utilidad para dar un ejemplo de salida de grafo de conocimiento:

$$\begin{aligned} &\textit{Amelia RAMA CASTILLO es estudiante de Ingeniería.} \\ &\textit{El 24 de agosto del año 975 fue detenida Amelia RAMA.} \end{aligned} \tag{3.6}$$

Además, se supone que el texto es procesado por las siguientes etapas:

- Reconocimiento de entidades nombradas a partir de expresiones regulares.
- Resolución de correferencias utilizando el algoritmo de resolución entre entidades nombradas.
- Extracción de relaciones utilizando la librería *NLTK*.

Del reconocimiento de entidades nombradas se etiqueta a *Amelia RAMA CASTILLO* y *Amelia RAMA* como *PER* y a *24 de agosto del año 975* como *DATE*.

De la resolución de correferencias se obtiene la cadena de correferencias *Amelia RAMA CASTILLO* ← *Amelia RAMA*.

De la extracción de relaciones se obtiene la relación *(24 de agosto del año 975, "es detenida", Amelia RAMA)*.

De las entradas descriptas, se obtiene el siguiente grafo (serializado en formato *Turtle*) y además se presenta un diagrama visual del mismo.

```
@base <http://mh.uy/luz/resource/> .
@prefix luz: <http://mh.uy/luz#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix schema: <https://schema.org/> .
@prefix time: <http://www.w3.org/2006/time#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<person/r3_i3_Amelia_RAMA_1> a luz:Person ;
    rdfs:label "Amelia RAMA" ;
    luz:hasOccurrence <occurrence/r3_i3_OCC3> ;
    schema:name "Amelia RAMA" .

<person/r3_i3_Amelia_RAMA_CASTILLO_1> a luz:Person ;
```

```
    rdfs:label "Amelia RAMA CASTILLO" ;
    luz:hasOccurrence <occurrence/r3_i3_OCC1> ;
    schema:name "Amelia RAMA CASTILLO" .

<date/r3_i3_24_de_agosto_del_a%C3%B1o_975_1> a luz:Date ;
    rdfs:label "24 de agosto del año 975" ;
    luz:hasOccurrence <occurrence/r3_i3_OCC2> ;
    time:day "---24"^^xsd:gDay ;
    time:month "--08"^^xsd:gMonth ;
    time:monthOfYear <http://www.w3.org/ns/time/gregorian/August> ;
    time:unitType time:unitDay ;
    time:year "1975"^^xsd:gYear .

<relation/r3_i3_REL1> a luz:Arrest ;
    luz:hasArrested <person/r3_i3_Amelia_RAMA_1> ;
    luz:hasRelationOccurrence <occurrence/r3_i3_OCC4> ;
    luz:when <date/r3_i3_24_de_agosto_del_a%C3%B1o_975_1> .

<occurrence/r3_i3_OCC1> a luz:Occurrence ;
    luz:hasCoreference <occurrence/r3_i3_OCC3> ;
    luz:hasOccurredIn <sheet/3_3> ;
    luz:occurrenceBegin 0 ;
    luz:occurrenceEnd 20 .

<occurrence/r3_i3_OCC2> a luz:Occurrence ;
    luz:hasOccurredIn <sheet/3_3> ;
    luz:occurrenceBegin 53 ;
    luz:occurrenceEnd 76 .

<occurrence/r3_i3_OCC3> a luz:Occurrence ;
    luz:hasOccurredIn <sheet/3_3> ;
    luz:occurrenceBegin 90 ;
    luz:occurrenceEnd 101 .

<occurrence/r3_i3_OCC4> a luz:RelationOccurrence ;
    luz:hasEntityOccurrence <occurrence/r3_i3_OCC2>,
        <occurrence/r3_i3_OCC3> ;
    luz:hasOccurredIn <sheet/3_3> .

<sheet/3_3> a luz:Sheet ;
```

```
    luz:roll "3" ;
    luz:sheet "3" .
```

Figura 3.1: RDF resultante del texto 3.6

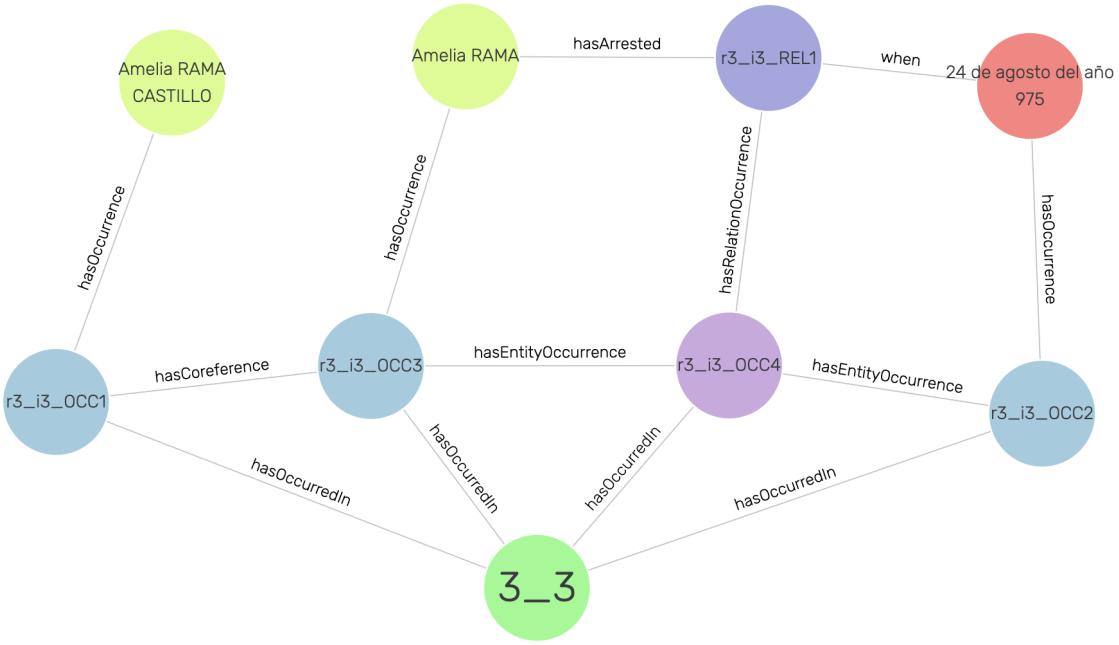


Figura 3.2: Grafo visual del RDF 3.1

En el RDF anterior (3.1) y más cómodamente en el grafo visual (3.2) se pueden notar las personas *Amelia RAMA CASTILLO* y *Amelia RAMA* que tienen las ocurrencias *OCC1* y *OCC3* respectivamente. A su vez, la ocurrencia de *Amelia RAMA* es tomada como correferencia de la ocurrencia de *Amelia RAMA CASTILLO*.

Por otro lado, el nodo rojo representa la fecha *24 de agosto del año 975* la cual tiene una ocurrencia *OCC2*. Esta fecha y la persona *Amelia RAMA* participan de una instancia de relación *Arrest* cuyo nodo es *REL1*. Además, con *OCC4* se representa la ocurrencia de esta instancia de relación entre estas dos entidades nombradas.

Para terminar, todas las ocurrencias en el grafo han ocurrido en la hoja 3_3, la cual es la número 3 del rollo 3. Esto último tampoco está

explícitamente representado en el grafo porque estas propiedades son de tipo *DatatypeProperty*, por lo que tienen un literal como objeto. Sin embargo, en la sección 4.6.2 se explicará cómo el sufijo *3_3* del IRI de un individuo de la clase *Sheet* es construido a partir de los números de rollo y hoja.

3.6. Entrenamiento de modelos

Dado que el dominio de los textos es muy específico y el lenguaje utilizado de muchos de estos proviene de un origen militar, se pensó en entrenar modelos como parte del proyecto para utilizar en el reconocimiento de entidades nombradas.

A partir de eso se comenzó a trabajar en el tema, logrando como resultado un mecanismo para generar modelos entrenados de manera iterativa partiendo de un conjunto de entrenamiento. Este conjunto se construye curando manualmente el resultado de alguno de los reconocedores de entidades nombradas que forman parte de este proyecto.

Tomando el resultado de lo mencionado en el párrafo anterior y utilizando la librería *SpaCy* [19] se ejecuta una iteración para entrenar un modelo. Si se vuelve a ejecutar el algoritmo con un nuevo conjunto de datos de entrenamiento, se entrena el modelo anterior con la nueva información, por lo tanto permite afirmar que el procedimiento de entrenar modelos es iterativo.

Esta solución fue llevada a cabo para su uso futuro ya que para poder utilizar todo su potencial se debería tener una gran cantidad de textos anotados.

Una vez que se tenga un modelo entrenado, éste puede ser utilizado en el módulo de *SpaCy* para reconocer entidades nombradas como se explica en la sección 4.3.3.

Capítulo 4

Implementación de la Solución

Este capítulo describe la arquitectura del sistema implementado para alcanzar los objetivos propuestos para el proyecto siguiendo los lineamientos planteados en el diseño de la solución (3).

Este sistema consta de una aplicación con arquitectura REST escrita en *Python* con el *framework Flask* como servidor web. Por otro lado, la aplicación está diseñada como un pipeline configurable con las siguientes etapas:

- Preprocesamiento del texto.
- Reconocimiento de entidades nombradas.
- Resolución de correferencias.
- Extracción de relaciones.
- Generación del grafo de conocimiento.

4.1. Motivación del pipeline

Antes de diseñar la arquitectura de pipeline para este proyecto, las etapas enumeradas anteriormente se habían implementado como servicios diferentes del sistema con la finalidad de facilitar la etapa de desarrollo y depuración del trabajo. Todos estos tomaban como entrada el número de rollo y número de hoja de un texto en particular.

El preprocesamiento del texto se ejecutaba mediante la llamada *POST /api/v1/preprocess* y guardaba el texto preprocesado en un archivo de texto plano en el disco.

El reconocimiento de entidades nombradas se ejecutaba mediante la llamada *GET /api/v1/ner_predict* con un parámetro opcional para indicar

si el texto a procesar requería preprocesamiento o no. A diferencia del servicio mencionado anteriormente, los resultados de este no son almacenados en el disco automáticamente sino que son devueltos en el cuerpo de la respuesta *HTTP*.

Por otra parte, si el texto a procesar disponía de anotaciones reales, es decir que fueron anotados con notación BIO manualmente o por lo menos una anotación considerada como *gold standard*, entonces el servicio permitía configurarse por parámetros para extraer métricas de evaluación contra estas anotaciones. Las métricas de evaluaciones serán explicadas en la sección 6.1.1.

La resolución de correferencias y la extracción de relaciones se ejecutaban mediante las llamadas *GET /api/v1/coreferences* y *GET /api/v1/relations* respectivamente, que devolvían únicamente los resultados del procesamiento en el cuerpo de la respuesta *HTTP*. Además para ambas se necesitaba ejecutar el procesamiento de reconocimiento de entidades nombradas ya que es una entrada requerida para los algoritmos.

A continuación se presenta el flujo general de ejecución utilizado antes de implementar la arquitectura de pipeline, en el cual se preprocesa un texto, se le aplica reconocimiento de entidades nombradas, se extraen relaciones y por último se detectan correferencias.

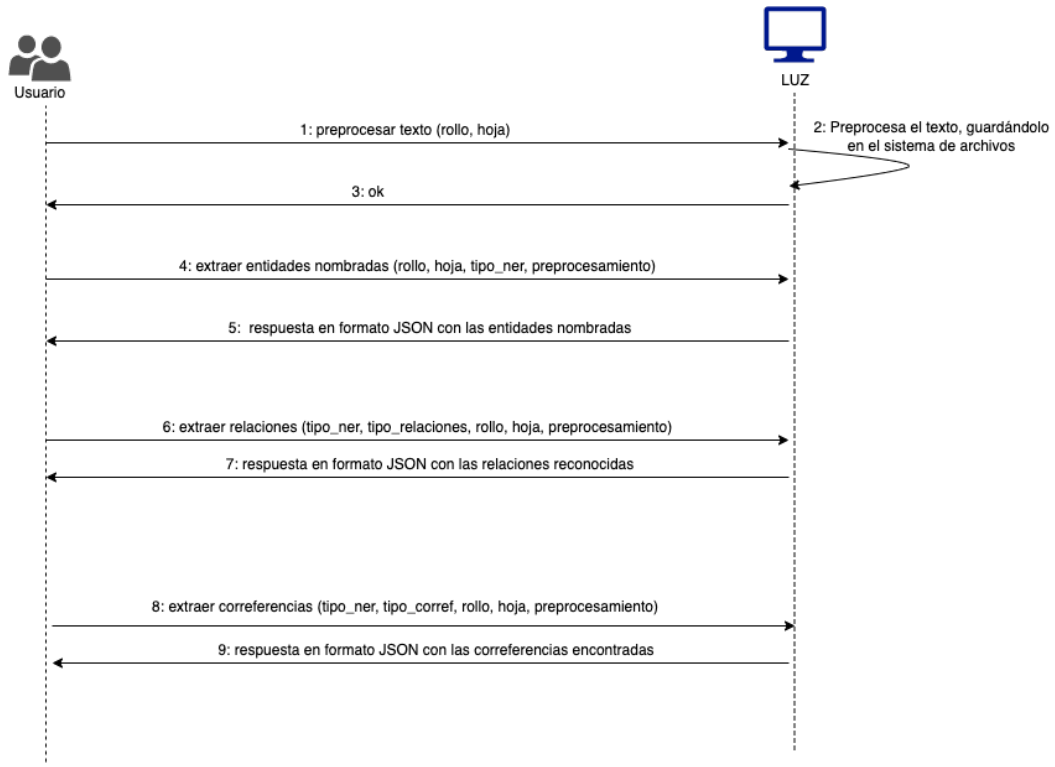


Figura 4.1: Ejemplo flujo del sistema antes de implementar el pipeline

Cabe destacar que en el ejemplo que se presenta en la figura anterior 4.1, se ejecuta nuevamente el reconocimiento de entidades nombradas para extraer relaciones y encontrar correferencias, sin aprovechar la ejecución anterior de NER.

Ante el crecimiento del proyecto y la necesidad de seguir agregando funcionalidades, se notó que realizando una sola ejecución del sistema y aprovechando la salida de las distintas etapas intermedias, se podrían optimizar las ejecuciones de las etapas posteriores. Para esto se planteó un arquitectura basada en un pipeline de etapas[1] donde cada una se presenta en un orden fijo, pero es configurable el ejecutar o no de cada una mediante parámetros definidos por el usuario de la API.

Este diseño permite utilizar un único servicio que procesa todas las etapas necesarias, incluyendo esta vez la generación del grafo de conocimiento.

[1]Este diseño fue motivado por el mecanismo que *SpaCy* utiliza para el procesamiento de documentos de texto.

Además, se agregaron varias mejoras en la nueva arquitectura. En primer lugar, como se mencionó anteriormente se guardan los resultados intermedios, esta característica del pipeline es abordada con detalle en la sección 4.7. En segundo lugar, la arquitectura fue diseñada de tal forma que es posible desarrollar e inyectar en el pipeline nuevos algoritmos (o reconocedores) para las etapas de reconocimiento de entidades nombradas, resolución de correferencias y extracción de relaciones. Por último, se agrega un mecanismo de traza el cual se encarga de registrar los principales procesos que el pipeline ejecuta.

A continuación se muestra para el mismo caso de uso planteado en la figura 4.1 un diagrama de flujo de datos utilizando la arquitectura del pipeline.

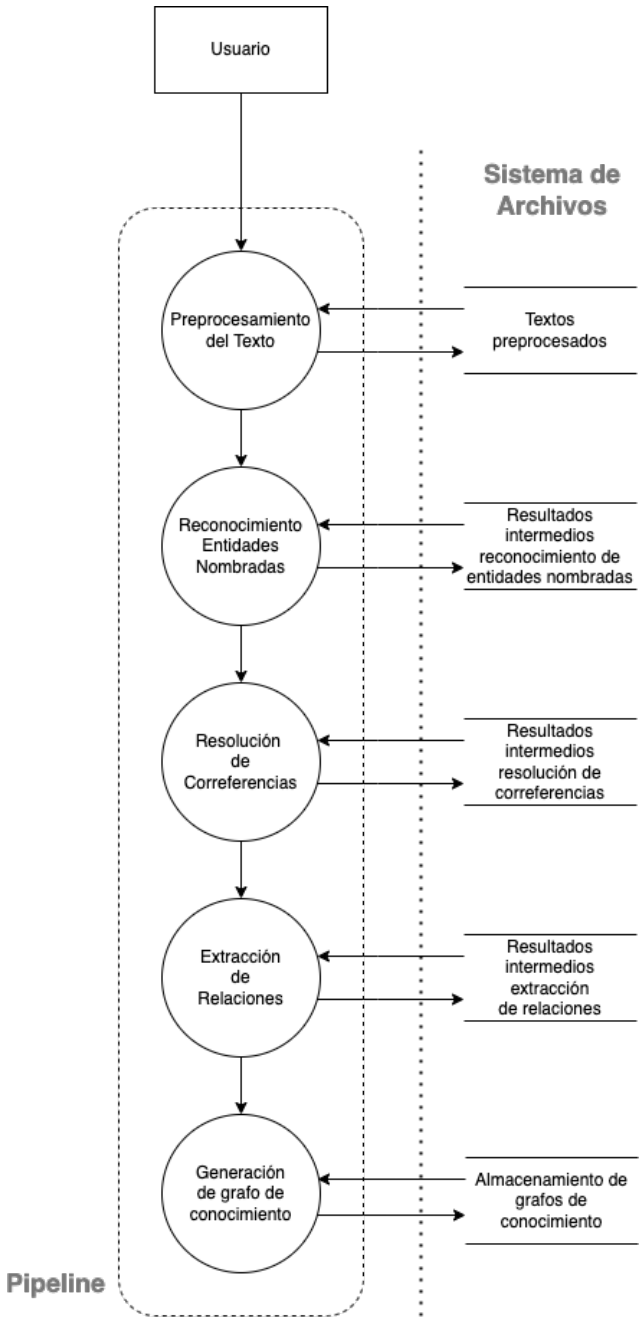


Figura 4.2: Ejemplo flujo de datos del sistema luego de implementar el pipeline

Cabe destacar que para el diagrama anterior se realiza una sola petición al servidor de *LUZ* en vez de 4 como se hacía anteriormente al pipeline. Además, genera un grafo de conocimiento en el sistema de archivos y arroja como respuesta los resultados intermedios de la ejecución en formato *JSON*.

Como se mencionó anteriormente, el pipeline es configurable mediante parámetros, los cuales se listan a continuación:

- **roll:** este parámetro es requerido e indica el rollo sobre el cual se desea ejecutar el pipeline.
- **sheet:** este parámetro es opcional e indica la hoja sobre la cual se desea ejecutar el pipeline. Si este no está presente, se procede a ejecutar el pipeline para el rollo entero.
- **preprocessed:** este parámetro indica si el o los textos a ejecutar en el pipeline son preprocesados. Cuando el valor de éste es *true*, se busca una versión preprocesada del texto. Si ésta se encuentra, será usada para la ejecución de las etapas posteriores, de lo contrario se ejecuta el preprocesamiento sobre el texto. Si el parámetro es *false* o no está presente, se utilizan los textos sin preprocesar.
- **only:** este parámetro indica si solo se desean ejecutar algunas etapas específicas en lugar de todo el pipeline. Por defecto se ejecutan todas las etapas. Para este parámetro se espera una lista de elementos separados por *coma* (,) donde los valores posibles son "preprocessing", "ner", "coreferences", "relations" y "graph".
- **except:** este parámetro indica qué etapas no se desean ejecutar para el pipeline. Si el parámetro no está presente, se toma la lista vacía por defecto. El valor esperado se comporta de la misma manera que el parámetro *only*.
- **force_exec:** este indica si se desea volver a ejecutar las etapas ignorando los resultados intermedios. Este parámetro es booleano.
- **ner_type:** este parámetro es requerido, indica el reconocedor de entidades nombradas que se desea utilizar para todas las etapas. Los valores posibles son: "freeling", "spacy", "stanza-ancora", "stanza-conll02", "nltk", "regexp" o "dictionaries".
- **trained:** este toma valores booleanos, es opcional y su valor por defecto es *false*. Este parámetro sólo puede ser *true* si *ner_type* es *SpaCy* e indica si se quieren utilizar modelos entrenados por el proyecto *LUZ*. Si bien no se tienen modelos entrenados por el momento, la implementación contempla el uso de ellos a futuro.
- **coref_types:** este parámetro es requerido si se ejecuta la etapa de resolución de correferencias, y debe ser una lista donde cada elemento

indica los algoritmos de resolución de correferencias a utilizar. Si bien en *LUZ* se tiene hasta el momento uno solo implementado, el pipeline contempla que se puedan agregar nuevos a futuro. El único valor posible es: "named_entity_coreference".

- **rels_types:** este parámetro es requerido si se ejecuta la etapa de extracción de relaciones, y es una lista donde cada elemento indica los algoritmos de extracción de relaciones a utilizar. Si bien en *LUZ* se tiene hasta el momento uno solo implementado, el pipeline contempla que se puedan agregar nuevos a futuro. El único valor posible es: "nltk".
- **lemmatized:** este parámetro indica si los algoritmos de extracción de relaciones utilizarán los textos lematizados. Este es opcional y por defecto su valor es *false*.

4.2. Preprocesamiento del texto

El motor de preprocesamiento tiene como entrada un texto y una lista de reglas de preprocesamiento y como salida otro texto. Se implementaron los tipos de reglas explicados en 3.1, nombrados como *replace* y *collapseSpaces* respectivamente.

La lista de reglas a aplicar es tomada desde el archivo *util/preprocessing_operations.json*. A continuación se muestra un ejemplo:

```
1  {
2    "operations": [
3    {
4      "ref": "Colapsa todos los espacios y
           tabulaciones consecutivos por un espacio",
5      "operation": "collapseSpaces"
6    },
7    {
8      "operation": "replace",
9      "oldValue": "PROCESADO",
10     "newValue": ""
11     "disable": true
12   },
13   {
```

```
14        "operation": "replace",
15        "oldValue": "DEP[ \\.]",
16        "newValue": ""
17      }
18      ]
19    }
```

Figura 4.3: Ejemplo de reglas de preprocesamiento

Cada regla puede ser deshabilitada mediante el campo *disable*. Por su parte, la regla *replace* utiliza dos parámetros: $oldValue$, que es la expresión regular que modela la secuencia de caracteres a ser reemplazada y $newValue$, que es la secuencia de caracteres que se quiere poner en su lugar. Es importante aclarar que las reglas se aplicarán al texto en el mismo orden que se definen en el archivo. Esto puede cambiar completamente el resultado como se verá en la siguiente sección (4.2.1).

Además, este motor guarda en su archivo de log *log/preprocessing_trace.log* cada regla aplicada al texto, su orden y si el texto resultó modificado o no, tal como se muestra a continuación. Esto permite tener un mayor control sobre el desempeño de esta etapa.

```
1 2021-04-14 18:30:09.863991 | /home/ProyectoLuz/
    Textos/texto1.txt cambia con collapseSpaces
2 2021-04-14 18:30:09.864003 | /home/ProyectoLuz/
    Textos/texto1.txt no cambia con replace 'PROCESADO' by
    ''
3 2021-04-14 18:30:09.864010 | /home/ProyectoLuz/
    Textos/texto1.txt cambia con replace 'DEP[ \.]' by ''
4 2021-04-14 18:30:09.864016 | /home/ProyectoLuz/
    Textos/texto1.txt cambia con collapseSpaces
```

4.2.1. Problemas encontrados y decisiones tomadas

Al probar este motor fueron encontrados algunos problemas. Como se explicó anteriormente en la sección 3.1, los textos que se procesan en este sistema son bastante inconsistentes. Particularmente las transcripciones de los sellos no aparecen siempre de la misma manera, lo que dificulta el

preprocesamiento. Tomando como ejemplo el sello con la inscripción “DEP .”, su transcripción es encontrada con o sin el punto, con varios espacios antes o después, con varios espacios entre las letras y el punto, pegado a una palabra o incluso en el medio. Estos diferentes tipos de apariciones para un mismo sello dificulta su completa remoción de cada documento.

Para resolver este problema se evitó implementar varias reglas del tipo *replace* que se ajustaran a cada una de las formas en la que cada uno de los sellos aparecen en los textos, ya que implicaría un gran esfuerzo de lectura de los textos e implementación y aún así podrían quedar por fuera algunas ocurrencias. Por el mismo motivo tampoco se eligió definir una única regla por sello compuesta por una expresión regular de alta complejidad que aplique a todas esas ocurrencias. En contraparte se implementó una solución que combina dos reglas simples. Como el motor de preprocesamiento aplica las reglas en el mismo orden que se definen en el archivo *.json* sin importar si las reglas se encuentran repetidas, se decidió duplicar la regla *collapseSpaces* para que se ejecute antes y después de las reglas *replace* que remueven las transcripciones de los sellos. Al ejecutarse antes de hacer las sustituciones se asegura que la cantidad de espacios dentro de la secuencia de caracteres a reemplazar sea la esperada. Esta es una buena estrategia siempre que se utilicen reglas del tipo *replace*, ya sea para remover caracteres o no. Por otro lado, la segunda ejecución de *collapseSpaces* solo es necesaria en caso de hacer remoción, como es el caso de los sellos, ya que al hacerlo es probable que se generen dobles espacios.

A partir de esto el ejemplo 4.3 quedaría así:

```
1  {
2    "operations": [
3    {
4      "ref": "Colapsa todos los espacios y
         tabulaciones consecutivos por un espacio",
5      "operation": "collapseSpaces"
6    },
7    {
8      "operation": "replace",
9      "oldValue": "PROCESADO",
10     "newValue": "",
11     "disable": true
```

```
12      },
13      {
14        "operation": "replace",
15        "oldValue": "DEP[ \\.]",
16        "newValue": ""
17      },
18      {
19        "ref": "Colapsa todos los espacios y
            tabulaciones consecutivos por un espacio",
20        "operation": "collapseSpaces"
21      }
22      ]
23    }
```

Figura 4.4: Ejemplo de reglas de preprocesamiento

Es importante aclarar que este problema de preprocesamiento de los sellos está siendo abordado con más detalle por otra línea de trabajo de *Cruzar*. Por lo que la solución planteada anteriormente es temporal.

4.3. Reconocimiento de entidades nombradas

Esta etapa del pipeline cuenta con un motor que toma como entrada un texto y un tipo de reconocedor. Por otro lado, esta etapa tiene como salida una lista compuesta por cada *token* del texto con su correspondiente etiqueta BIO.

Suponiendo que se toma como entrada el texto a continuación:

Luis Lacalle Pou se reunió en la Torre Ejecutiva con la intendenta del Frente Amplio, Carolina Cosse, para conversar sobre la gestión de la pandemia. (4.1)

La lista obtenida como salida del reconocedor de entidades nombradas

tendría la siguiente forma:

$$\{(B-PER,\ Luis),\ (I-PER,\ Lacalle),\ (I-PER,\ Pou),\ (O,\ se),\ (O,\ reunió),\ (O,\ en),\ (O,\ la),\ (B-LOC, Torre),\ (I-LOC, Ejecutiva),\ (O,\ con),\ (O,\ la),\ (O,\ intendenta),\ (O,\ del),\ (B-ORG,\ Frente),\ (I-ORG,\ Amplio),\ (O,\ ,),\ (B-PER,\ Carolina),\ (I-PER,\ Cosse),\ (O,\ ,),\ (O,\ para),\ (O,\ conversar),\ (O,\ sobre),\ (O,\ la),\ (O,\ gestión),\ (O,\ de),\ (O,\ la),\ (O,\ pandemia),\ (O,\ .)\} \tag{4.2}$$

El etiquetado de las entidades y su corrección depende de qué tipo de reconocedor se elija utilizar para la ejecución de esta etapa del pipeline. La implementación de los distintos módulos para cada tipo de reconocedor que se puede utilizar hasta el momento serán presentados en detalle a continuación.

4.3.1. Expresiones regulares

Se implementó un reconocedor de entidades nombradas basado en expresiones regulares. La entrada de este reconocedor es un texto, la lista de los índices de comienzo y fin de cada *token* de ese texto y una lista de reglas a aplicar.

La lista de reglas es tomada desde el archivo *util/regexps_rules.json*. A continuación se muestra un ejemplo:

```
{
  "rules": [
    {
      "ref": "Nombres APELLIDOS",
      "label": "PER",
      "regexp": " (?:|[^A-Za-z])((?:[A-Z][a-z]+ )+
              (?:[A-Z]{2,} ?)+)[^A-Za-z]"
    },
    {
      "ref": "Apellidos, Nombres",
      "label": "PER",
      "regexp": " (?:|[^A-Za-z])((?:[A-Z][a-z]+ )+,
              (?:[A-Za-z]{2,} ?)+)[^A-Za-z]"
    },
```

```
15        ]
16    }
```

Figura 4.5: Ejemplo de archivo de reglas a aplicar.

Como se observa en el ejemplo, cada una de las reglas definidas tiene 3 componentes.

- *ref* es simplemente una referencia para facilitar la lectura de quien se encargue de mantener esta configuración.
- *label* es el tipo de entidad con el que corresponde etiquetar a los *tokens* que satisfacen la expresión regular del patrón.
- *regexp* es la expresión regular que modela propiamente al patrón en cuestión.

En este caso, la primer regla reconocería entidades nombradas de tipo *persona* que tengan una forma similar a *María Elena EGUREN PEREZ* y la segunda reconocería entidades del mismo tipo pero que tengan una forma similar a *Ramirez Lopez, Juan Alberto*.

Para el reconocimiento de las entidades primero se utiliza el módulo de expresiones regulares *re* de *Python*, que retorna la porción de texto que satisface la expresión regular que toma de entrada. Luego se utiliza la función *string.find* de *Python* para obtener el índice de inicio y final, con respecto al texto tomado como entrada, de cada una de estas porciones de texto encontradas. Finalmente se itera sobre la lista de los índices de comienzo y fin de cada *token* en el texto y se compara con la salida de *string.find* para etiquetar cada *token* con el formato BIO y generar una salida con el formato descripto en la sección 4.3.

Para evitar que el resultado de esta etapa contenga entidades superpuestas, las reglas se aplican al texto en el mismo orden que se definen en el archivo *util/regexps_rules.json*. Esto puede cambiar totalmente el resultado, ya que si algún *token* es parte de dos porciones de texto que satisfacen distintas reglas, al final tendrá la etiqueta de la primera regla aplicada y la segunda coincidencia será ignorada. Por ejemplo, si se tiene como entrada el siguiente texto:

Luis LACALLE POU se reunió en la Torre Ejecutiva con la intendenta del Frente Amplio, Carolina COSSE, para conversar sobre la gestión de la pandemia. (4.3)

y la lista de reglas definida en el ejemplo 4.5

La salida obtenida será:

$$\{(B-PER,\ Luis),\ (I-PER,\ LACALLE),\ (I-PER,\ POU),\ (O,\ se),\ (O,\ reunió),\ (O,\ en),\ (O,\ la),\ (O, Torre),\ (O, Ejecutiva),\ (O,\ con),\ (O,\ la),\ (O,\ intendenta),\ (O,\ del),\ (O,\ Frente),\ (O,\ Amplio),\ (O,\ ,),\ (B-PER,\ Carolina),\ (I-PER,\ COSSE),\ (O,\ ,),\ (O,\ para),\ (O,\ conversar),\ (O,\ sobre),\ (O,\ la),\ (O,\ gestión),\ (O,\ de),\ (O,\ la),\ (O,\ pandemia),\ (O,\ .)\} \tag{4.4}$$

Si bien *Carolina COSSE* satisface la regla *Nombres APELLIDOS* y *Amplio, Carolina* satisface la regla *Apellidos, Nombres*, al ser porciones de texto que se superponen, la coincidencia con la segunda regla se descarta. Por otro lado, si las reglas estuvieran en el orden inverso, la lista obtenida como salida sería:

$$\{(B-PER,\ Luis),\ (I-PER,\ LACALLE),\ (I-PER,\ POU),\ (O,\ se),\ (O,\ reunió),\ (O,\ en),\ (O,\ la),\ (O, Torre),\ (O, Ejecutiva),\ (O,\ con),\ (O,\ la),\ (O,\ intendenta),\ (O,\ del),\ (O,\ Frente),\ (B-PER,\ Amplio),\ (I-PER,\ ,),\ (I-PER,\ Carolina),\ (O,\ COSSE),\ (O,\ ,),\ (O,\ para),\ (O,\ conversar),\ (O,\ sobre),\ (O,\ la),\ (O,\ gestión),\ (O,\ de),\ (O,\ la),\ (O,\ pandemia),\ (O,\ .)\} \tag{4.5}$$

descartando esta vez la coincidencia entre *Carolina COSSE* y la regla *Nombres APELLIDOS*, que se aplicaría en segundo lugar. Esta estrategia evita además que se devuelva una misma entidad nombrada con más de una etiqueta.

4.3.2. Entidades nombradas a partir de diccionarios

Los diccionarios son archivos en formato *JSON* que se encuentran almacenados en el disco y contienen fragmentos de texto. Todos estos fragmentos dentro de un mismo diccionario serán reconocidos como entidades nombradas del mismo tipo.

El reconocedor, basado en una configuración, lee cada diccionario por separado y construye una expresión regular que modela un *OR* entre todos los fragmentos de texto en el diccionario. Luego, aplica la expresión construida en el texto recogiendo los índices de comienzo y final de cada fragmento de texto que coincide con la expresión regular.

La configuración de este reconocedor está ubicada en el archivo *config/dictionaries_config.json* permitiendo así que el agregar o quitar diccionarios a la ejecución solo dependa de modificar este archivo. Esta configuración se trata de una lista de hashes donde cada uno corresponde a un diccionario utilizado. Los hashes contienen tres posibles claves:

- *filename*: indica la ruta, relativa a la raíz del proyecto, del archivo que contiene el diccionario.
- *label*: indica el tipo de entidad nombrada que corresponde con las porciones de texto contenidas en el diccionario.
- *disabled*: indica si la aplicación de este diccionario está deshabilitada

También se pueden encontrar ejemplos de configuración de diccionario en el archivo *config/dictionaries_config.json.example*.

En cuanto a los archivos de diccionario, el *JSON* contenido debe ser una lista de strings donde cada uno de estos corresponde a una porción de texto.

El siguiente ejemplo define dos diccionarios donde *dic2* está deshabilitado por la configuración, lo que implica que no es utilizado en la ejecución.

Figura 4.6: Ejemplo de configuración de diccionarios

```
[
  {
    "filename": "util/dictionaries/dic1.json",
    "label": "ORG"
  },
  {
    "filename": "util/dictionaries/dic2.json",
    "label": "PER",
    "disabled": true
  }
]
```

Por otro lado, la prioridad entre diccionarios se da de arriba hacia abajo, así que los diccionarios que son configurados al principio de la lista, son prioritarios

sobre los siguientes. Esto es así para resolver el último caso borde planteado en 3.2.3.

4.3.3. Módulos implementados a partir de las herramientas investigadas

A continuación se presentan detalles técnicos y de implementación relacionados con las herramientas de reconocimiento de entidades nombradas que fueron introducidas en 3.2.2.

SpaCy

Los modelos pre entrenados por *SpaCy* pueden diferir en tamaño, velocidad, uso de memoria, precisión y los datos que incluyen. Como se menciona en la sección 3.2.2, *LUZ* utiliza por defecto el modelo *es_core_news_md* el cual es cargado en memoria mediante el método *spacy.load()*. Si bien no se cuenta con otros modelos entrenados con los textos del dominio, esta implementación soporta la utilización de aquellos que puedan ser entrenados con *SpaCy* en el futuro. Para esto se establece el parámetro de consulta *trained=true* al momento de ejecutar el pipeline.

Al aplicarle a un texto el procesamiento que ofrece este modelo, se obtiene un objeto del tipo *spacy.Doc*, que es una secuencia de objetos tipo *spacy.Token*. Los objetos de este último tipo cuentan, entre otros, con los atributos *ent_type_* y *ent_iob_* que indican el tipo de entidad de la que es parte el *token* y el código BIO que le corresponde, respectivamente. Con ambos atributos se forma la salida esperada en esta etapa del pipeline.

FreeLing

Para este proyecto se utilizó el programa *analyze*[31] que permite utilizar todas las herramientas que *FreeLing* provee mediante línea de comandos el cual se instala en el servidor que corre el sistema *LUZ*.

Para detectar y clasificar entidades nombradas se ejecuta *analyze* de la siguiente manera:

$$analyze \quad -f \quad es.cfg \quad --ner \quad --nec \quad --output \quad xml \tag{4.6}$$

donde *-f es.cfg* indica que el analizador debe ser ejecutado para el idioma español, y *–ner* y *–nec* establecen que se quiere detectar y clasificar entidades nombradas. Por último, *–output xml* indica que la salida sea en formato *xml*.

La salida que *FreeLing* provee está dividida por enunciados donde cada uno de estos contiene una lista de *tokens*. A su vez cada *token* está definido por un conjunto de datos representados como clave-valor los cuales forman toda la información sintáctica y semántica del *token*.

Stanza

Para ejecutar este reconocedor en el pipeline de *LUZ* es necesario, además, indicar el corpus a utilizar para el reconocimeinto. Para usar los modelos que *LUZ* soporta es necesario instalarlos de antemano mediante la ejecución del script *lib/stanza_setup.py*.

El módulo implementado para ejecutar la librería *Stanza* es muy simple. Primero, se crea un objeto de tipo *stanza.pipeline.core.Pipeline* al cual se le configura el procesador[1] de NER y luego se lo ejecuta con la lista de *tokens* del texto como entrada. Por último, se realiza el parseo de los resultados en notación BIOES a notación BIO.

NLTK

El módulo implementado toma como entrada la lista de *tokens* de un texto y utiliza un clasificador ya entrenado, provisto por *NLTK*, para el reconocimiento de entidades nombradas. A este clasificador se accede mediante la función *nltk.ne_chunk* que, como su nombre lo indica, utiliza la técnica de fragmentación. Dicha función toma como entrada una lista de *tokens* con etiquetado POS y devuelve una estructura de *nltk_tree* anidados que mantiene el etiquetado de la entrada y agrega la clasificación de entidades nombradas con las categorías *PERSON*, *ORGANIZATION*, *LOCATION* y *GPE*.

Para conseguir las etiquetas POS a partir de la lista de *tokens* se ejecuta la función *nltk.pos_tag* previo a la función explicada anteriormente. Luego para armar la salida deseada se utiliza la función *nltk.tree2conlltags* que convierte el árbol (que fue obtenido de *nltk.ne_chunk*) en una lista de 3-uplas con la forma

[1] Son las unidades del pipeline neuronal de *Stanza*. Cada uno realiza funciones específicas de PLN y crean diferentes anotaciones de un texto.

$(token,\ POS-tag,\ BIO-tag)$.

4.4. Resolución de correferencias

Esta etapa del pipeline cuenta con un motor que recibe como entrada el texto y la lista de etiquetas BIO de las entidades nombradas a lo largo del mismo.

El texto recibido en esta etapa es el mismo que el recibido en la etapa de reconocimiento de entidades nombradas, es decir, el texto no es preprocesado de manera diferente a como lo fue en la etapa anterior.

Como se mencionó anteriormente, el pipeline está diseñado de tal forma que se pueden inyectar algoritmos de resolución de correferencias.

La salida de este motor es una lista que representa cada una de las cadenas de correferencias pertenecientes al texto.

4.4.1. Salida del motor de resolución de correferencias

La lista esperada como salida tiene tantos elementos como cantidad de entidades nombradas más cada uno de los *tokens* que no pertenecen a ninguna entidad nombrada.

Como ejemplo, se supone la siguiente porción de texto:

$$\begin{aligned} &Diego\ Forlán\ anotó\ tres\ goles\ en\ el\ partido\ pasado.\\ &Hoy,\ Diego,\ solo\ logró\ anotar\ uno. \end{aligned} \tag{4.7}$$

Suponiendo que la tokenización fue hecha por un tokenizador que no deja los signos de puntuación como *tokens* separados, la lista devuelta tendrá un largo de 14, pues hay 2 entidades nombradas y 12 *tokens* que no pertenecen a ninguna de ellas.

La lista resultado contiene números enteros (o valores nulos):

- Si en la posición i de la lista se encuentra un número entero $j >= 0$, significa que el *token* (o la entidad nombrada) que corresponde a la posición i de la lista hace referencia al *token* (o entidad nombrada) correspondiente a la posición j de la lista.

- Si en la posición i de la lista se encuentra el número -1, entonces en la posición i del texto se encuentra una entidad nombrada que no hace referencia a otra anteriormente nombrada. Se dice que esta entidad nombrada es *primera mención*.
- Por último, si en la posición i de la lista se encuentra un valor nulo, entonces en la posición i del texto hay un *token* no perteneciente a una entidad nombrada que no es correferencia.

El algoritmo presentado para este proyecto solamente detecta correferencias entre entidades nombradas. Sin embargo, el diseño propuesto para la lista de cadenas de correferencias permite que en el futuro se puedan representar cadenas entre *tokens* que no pertenezcan a una entidad nombrada, como es el caso de los pronombres. Es decir, si se quisiera inyectar al pipeline un algoritmo de resolución de correferencias entre entidades nombradas y pronombres, entonces este diseño no debería sufrir cambios, ya que en la posición correspondiente al pronombre en la lista, se podría referenciar la posición de una entidad nombrada. De igual forma se pueden representar correferencias entre dos *tokens* que no son entidades nombradas.

4.4.2. Algoritmo de resolución de correferencias entre entidades nombradas

El algoritmo trabaja de forma local al texto recorriendo la lista de *tokens* con etiquetas BIO de izquierda a derecha agregando a la lista resultado valores nulos cuando el *token* está etiquetado con O o números enteros cuando los *tokens* forman una entidad nombrada siguiendo el criterio descripto en la sección 4.4.1.

Volviendo al ejemplo 3.3, la salida del algoritmo presentado es la lista con los valores indicados en la siguiente figura:

$$\overbrace{El}^{null}\ \overbrace{ex}^{null}\ \overbrace{jugador}^{null}\ \overbrace{de}^{null}\ \overbrace{fútbol,}^{null}\ \overbrace{Diego\ Forlán,}^{-1}\ \overbrace{nació}^{null}\ \overbrace{en}^{null}$$
$$\overbrace{Uruguay}^{-1}\ \overbrace{el}^{null}\ \overbrace{año\ 1979.}^{-1}\ \overbrace{Diego}^{5}\ \overbrace{fue}^{null}\ \overbrace{reconocido}^{null}\ \overbrace{como}^{null}\ \overbrace{el}^{null}$$
$$\overbrace{mejor}^{null}\ \overbrace{jugador}^{null}\ \overbrace{del}^{null}\ \overbrace{mundo}^{null}\ \overbrace{en}^{null}\ \overbrace{la}^{null}\ \overbrace{Copa\ Mundial\ de\ Fútbol}^{-1} \tag{4.8}$$
$$\overbrace{del}^{null}\ \overbrace{año\ 2010.}^{-1}\ \overbrace{Peñarol}^{-1}\ \overbrace{fue}^{null}\ \overbrace{el}^{null}\ \overbrace{último}^{null}\ \overbrace{equipo}^{null}\ \overbrace{en}^{null}\ \overbrace{el}^{null}\ \overbrace{que}^{null}$$
$$\overbrace{Forlán}^{5}\ \overbrace{jugó}^{null}\ \overbrace{profesionalmente.}^{null}$$

Se pueden observar las primeras menciones *Diego Forlán*, *Uruguay*, *año 1979*, *Copa Mundial de Fútbol*, *año 2010* y *Peñarol*.

Por otro lado, se encuentran otras dos menciones *Diego* y *Forlán*, en las posiciones 11 y 33, respectivamente. Dado que ambas tienen el valor 5 en la salida del algoritmo, cada una forma una cadena de correferencias diferente compartiendo la misma primera mención (*Diego Forlán* en posición 5).

Entrada: *etiq_tokens* ← lista de *tokens* con etiquetas BIO
Salida: cadena de correferencias
inicio
  entidades_mencionadas ← lista vacía
  mientras *hay tokens para procesar en etiq_tokens* **hacer**
    Sea *t*1 el *token* a procesar
    si *t1 está etiquetado con O* **entonces**
      Agregar valor nulo a la ***cadena de correferencias***
    si no
      Se continúa recorriendo la lista *etiq_tokens* hasta consumir toda la entidad nombrada
      entidad_actual ← lista de *tokens* consumidos en el paso anterior
      Busca de derecha a izquierda una entidad antecedente para *entidad_actual* en *entidades_mencionadas*
      si *encontró antecedente* **entonces**
        Agrega a la ***cadena de correferencias*** el índice que el antecedente encontrado tiene en la lista
      si no
        Agrega −1 a la ***cadena de correferencias***
      fin
      Agrega *entidad_actual* a *entidades_mencionadas*
    fin
  fin
fin

Algoritmo 1: Resolución de correferencias entre entidades nombradas

El algoritmo 1 muestra a alto nivel el proceso ejecutado para la resolución de correferencias entre entidades nombradas.

Es importante señalar que si en la posición i de la lista resultado se encuentra un número entero $j >= 0$, entonces $j < i$ ya que este algoritmo supone que la entidad nombrada en i hace referencia a una entidad nombrada ya mencionada utilizando así solo la información que ha sido consumida en pasos anteriores del proceso.

4.5. Extracción de relaciones

Esta etapa del pipeline se conforma por un motor que toma como entrada el texto tokenizado, la configuración de cada uno de los patrones, la información sobre las entidades nombradas detectadas en la etapa anterior. Esta última se explicará con más detalle en la sección 4.5.2.

La salida de este motor es una lista de tuplas cuyos elementos representan un par de entidades nombradas y qué relación las vincula.

4.5.1. Configuración de patrones

Para cada patrón se tiene una configuración que será utilizada a la hora de aplicar el algoritmo.

A continuación se presenta un ejemplo de patrón:

```
1   {
2     "integrante de": {
3       "ref": "integrante de",
4       "ent1": "PER",
5       "ent2": "ORG",
6       "regexp": ".*(integrante de).*",
7       "lemmatized_regexp": {
8         "stanza": ".*(integrante de).*",
9         "freeling": ".*(integrante de).*"
10      },
11      "window": 15,
12      "owlLuzRelation": {
13        "name": "Membership",
14        "ent1Property": "hasMember",
15        "ent2Property": "membershipHasOrganization"
16      }
17    }
18  }
```

Figura 4.7: Ejemplo configuración de un patrón

Donde el significado de cada una de las claves es el siguiente:

- *ref*: identifica el patrón.
- *ent1*: indica el tipo de la entidad nombrada que se encuentra primero.
- *ent2*: indica el tipo de la entidad nombrada que se encuentra última.

- *regexp*: es la expresión regular utilizada sin lematización.
- *lemmatized_regexp*: es un objeto con las expresiones regulares utilizadas con lematización. Debe haber una clave dentro de este objeto para cada una de las herramientas de lematización utilizadas.
- *window*: representa el tamaño de la ventana de tolerancia.
- *owlLuzRelation*: indica cómo mapear la relación a la ontología de *LUZ*.

4.5.2. Lista de entidades nombradas

Una vez obtenida la lista de etiquetas BIO en la etapa de reconocimiento de entidades nombradas (3.2), se genera una nueva lista que agrupa todos los *tokens* con etiqueta B e I. De este modo, los elementos de esta nueva lista son, o bien entidades nombradas etiquetadas según su tipo (PER, LOC, etc), o bien *tokens* que no corresponden a ninguna entidad nombrada manteniendo su etiqueta O.

Suponiendo que se parte de la siguiente lista de *tokens* con etiquetado BIO:

$$\{(B-PER,\ Ana),\ (I-PER,\ Carmona),\ (O,\ vive),\ (O,\ en),\\ (B-LOC,\ Punta),\ (I-LOC,\ del),\ (I-LOC,\ Este)\} \tag{4.9}$$

La lista de entidades nombradas resultante sería:

$$\{(PER,\ Ana\ Carmona),\ (O,\ vive),\ (O,\ en),\ (LOC,\ Punta\ del\ Este)\} \tag{4.10}$$

4.5.3. Salida del motor de extracción de relaciones

La lista que se obtiene como salida cuenta con tantos elementos como relaciones se hayan encontrado en el texto.

Se toma el siguiente segmento de texto como ejemplo:

Ana Carmona vive en Punta del Este pero su amigo Juan vive en Solymar. (4.11)

Suponiendo que es de interés extraer la relación *vive en* que vincula una entidad nombrada de tipo persona con una de tipo lugar, la lista obtenida tendría 2 elementos. Cada uno de estos elementos sería una 3-upla compuesta por lo siguiente:

1. El índice que corresponde a la entidad nombrada de tipo *persona* en la lista de entidades nombradas del texto.

2. El nombre que identifique al tipo de relación encontrada.

3. El índice que corresponde a la entidad nombrada de tipo *lugar* en la lista de entidades nombradas del texto.

Es importante que la salida cuente con el índice de las entidades nombradas vinculadas por cada relación ya que en un mismo texto pueden aparecer entidades nombradas idénticas pero de distinto tipo. Por ejemplo, *Emilio Frugoni* puede referirse tanto a una persona como a una calle. Si bien la solución propuesta implica que eventualmente una misma entidad sea representada más de una vez en el grafo de conocimiento generado, facilita el trabajo futuro de desambiguación de entidades en casos como el del ejemplo citado.

4.5.4. Algoritmo de extracción de relaciones

Esta área tiene mucho potencial para continuar aplicando algunas de las técnicas mencionadas en el marco teórico adaptándolas a los textos del dominio.

Para este proyecto se propone utilizar el módulo *sem.relextract* provisto por *NLTK* (Natural Language Toolkit) para realizar la extracción de relaciones de interés a partir de patrones identificados en el dominio de los textos. Las funciones del módulo mencionado obtienen de forma simple las posibles relaciones de un texto a partir de una estructura *nltk_tree*[1]. Luego hacen un filtrado según cada patrón provisto y los tipos de entidades nombradas que espera.

Los patrones de relaciones utilizados son de la forma:

$$< nombre\_de\_la\_relación >, < tipo\_de\_entidad\_del\_sujeto >, \\ < expresión\_regular >, < tipo\_de\_entidad\_del\_objeto >, < window > \tag{4.12}$$

Donde *window* es la ventana de tolerancia asociada al patrón, anteriormente explicada en la sección 3.4. El nombre de la relación es el que se utilizará para

[1] Agrupación jerárquica de hojas y subárboles creada a partir del etiquetado POS y BIO de un texto.

generar la lista de salida, que luego servirá al momento de generar el grafo de conocimiento en la siguiente etapa (4.6).

A partir de lo anterior y continuando con el ejemplo 4.11, la relación *vive en* podría estar definida de la siguiente forma:

$$vive\ en,\ PER,\ .*(vive\ en).*,\ LOC,\ 5 \tag{4.13}$$

Esta representación indica que el núcleo de la relación es "vivir en" y vincula un sujeto de tipo persona con un objeto de tipo lugar que no distan a más de 5 *tokens*.

Asimismo, la salida del motor para ejemplo 4.11 sería:

$$\{(1,\ vive\ en,\ 4), (8,\ vive\ en,\ 11)\} \tag{4.14}$$

Donde 1, 4, 11 y 14 corresponden respectivamente a los índices de *Ana Carmona*, *Punta del Este*, *Juan* y *Solymar* en la lista de entidades nombradas de ese texto.

El pseudocódigo a continuación (2) muestra a alto nivel el proceso que se ejecuta para la extracción de relaciones. En el primer bloque de código se puede ver el uso de las funciones de *NLTK* y de los patrones tal como se mencionó anteriormente. Como dichas funciones utilizan las entidades pero no su ubicación en el texto, en el segundo bloque se busca el índice en la lista de entidades nombradas del texto de cada una de las entidades nombradas involucradas en las relaciones encontradas. De esta manera se llega a construir la salida requerida por el motor explicada en la sección anterior.

Entrada: *etiq_tokens* ← lista de *tokens* con etiquetas BIO
entidades_nom ← lista de entidades nombradas
Salida: lista de relaciones
inicio
  pos_tags ← lista de etiquetas POS del texto a procesar
  nltk_tree ← árbol sintáctico del texto a procesar
  patrones ← lista de patrones que representan las relaciones que se quieren extraer de la forma mencionada en 4.12
  rels_encontradas ← lista vacía
  resultado ← lista vacía
  mientras *hay patrones para procesar* **hacer**
    $p1$ ← patrón a procesar
    dic_relaciones ← resultado de ejecutar *nltk.relextract* con el patrón $p1$
    mientras *hay diccionarios de relaciones para procesar en dic_relaciones* **hacer**
      *dic_r*1 ← diccionario a procesar
      *parseo_dic_r*1 ← parseo de *dic_r*1 a la forma
      $((< tipo\_entidad1 >, < entidad1 >) , < nombre\_relación > , (< tipo\_entidad2 >, < entidad2 >))$
      $rels\_encontradas \leftarrow rels\_encontradas + parseo\_dic\_r1$
    fin
  fin
  mientras *hay entidades para procesar en entidades_nom y relaciones para procesar en rels_encontradas* **hacer**
    $e1$ ← entidad a procesar
    $r1$ ← relación a procesar
    si *r1[0] es igual a e1* **entonces**
      mientras *la siguiente entidad en entidades_nom tenga etiqueta O* **hacer**
        Se continúa recorriendo la lista *entidades_nom*
      fin
      si *r1[2] es igual a la siguiente entidad nombrada en entidades_nom* **entonces**
        elemento ← un elemento con la forma mencionada en 4.5.3
        $resultado \leftarrow resultado + elemento$
      fin
    fin
  fin
fin

Algoritmo 2: Extracción de relaciones entre entidades nombradas

4.6. Generación del grafo de conocimiento

Esta etapa es la última del pipeline y tiene como objetivo reunir en un grafo de conocimiento toda la información recabada en las etapas anteriores del proceso.

La salida de esta etapa, que a su vez es la salida del pipeline, es el grafo de conocimiento que se serializa[1] en formato *Turtle* (*.ttl*) para poder ser guardado en el disco.

Se decidió utilizar la biblioteca *RDFLib* [23] para la generación y serialización del grafo.

4.6.1. Entrada utilizada para la generación del grafo

El generador del grafo utiliza la información que tiene disponible de etapas anteriores. Esto significa que si alguna de las etapas no se ha ejecutado porque la etapa no fue pedida, entonces la información no estará en el grafo, aún si existieran resultados intermedios ya calculados en ejecuciones anteriores (ver la sección 4.7).

A continuación se describe la entrada obtenida de cada una de las etapas del pipeline:

- Reconocimiento de entidades nombradas: De esta etapa se utiliza la lista de entidades nombradas (explicada en 4.5.2) junto al índice de inicio y final de cada *token* (o entidad nombrada) de manera de poder registrar la *provenance* de las mismas.
- Resolución de correferencias: De esta etapa se toma la cadena de correferencias descrita en 4.4.1 y se la transforma en una lista de pares de las posiciones que los elementos de la correferencia tienen en la lista mencionada en el punto anterior.
- Extracción de relaciones: Se utiliza la lista de tripletas de las posiciones de las entidades nombradas y la referencia del tipo de relación en cuestión. Esta estructura fue presentada en la sección 4.5.3.

[1] La **serialización** consiste en un proceso de codificación de un objeto en un medio de almacenamiento (como puede ser un archivo) con el fin de transmitirlo a través de una conexión en red como una serie de bytes o en un formato humanamente más legible como XML o JSON, entre otros.

4.6.2. Generación de IRIs

Se definió un esquema para la generación de los IRIs[1] utilizados para cada tipo de recurso definido en la ontología *LUZ*.

Primeramente cabe destacar que todos los recursos definidos por el sistema están bajo una ubicación configurable mediante variables de entorno. Para los ejemplos próximos esta ubicación es *https://mh.uy/luz/resource/*.

A continuación se describe brevemente la generación de los sufijos de los identificadores para cada uno de los tipos.

Entidades nombradas

Para entidades nombradas se definió la siguiente estructura

$$\{tipo\}/r\{rollo\}\_i\{hoja\}\_\{nombre\}\_\{ocurrencia\} \tag{4.15}$$

donde:

- *tipo* representa el tipo de entidad nombrada teniendo como posibles valores *person*, *location*, *organization* o *date*.
- *rollo* representa el número de rollo.
- *hoja* representa el número de hoja en el rollo.
- *nombre* es el span que representa la entidad nombrada en el texto. Cada espacio o *slash* (/) contenido en el span es reemplazado con un guión bajo.
- *ocurrencia* es un valor implementado mediante un contador numérico autoincrementado. Asegura la unicidad del identificador si sucediera que en el mismo texto dos spans diferentes, con el mismo contenido de texto, fueran reconocidos como entidades nombradas del mismo tipo.

Por ejemplo, si en la hoja 234 del rollo 3, se encontraran dos ocurrencias distintas de *14 de agosto* y ambas son reconocidas como fechas, entonces ambas tendrían identificadores diferentes y serían los siguientes:

https://mh.uy/luz/resource/date/r3_i234_14_de_agosto_1
https://mh.uy/luz/resource/date/r3_i234_14_de_agosto_2

[1] El Identificador de Recursos Internacionalizado es un estándar de protocolo de Internet que se basa en el protocolo Identificador de Recursos Uniforme al expandir en gran medida el conjunto de caracteres permitidos.

Relaciones

Para las instancias de relaciones se definió la siguiente estructura

$$relation/r\{rollo\}\_i\{hoja\}\_REL\{identificador\} \tag{4.16}$$

donde *identificador* es un valor implementado mediante un contador numérico autoincrementado que asegura la unicidad de cada relación en el texto.

Por ejemplo, una relación en la hoja 234 del rollo 3 sería identificada como *https://mh.uy/luz/resource/relation/r3_i234_REL12*.

Ocurrencias

Para ocurrencias de entidades nombradas y de relaciones se definió la siguiente estructura

$$occurrence/r\{rollo\}\_i\{hoja\}\_OCC\{identificador\} \tag{4.17}$$

donde *identificador* es un valor implementado mediante un contador numérico autoincrementado que asegura la unicidad de cada ocurrencia en el texto, ya sea de una entidad nombrada o instancia de relación.

Por ejemplo, una ocurrencia en la hoja 234 del rollo 3 sería identificada como *https://mh.uy/luz/resource/occurrence/r3_i234_OCC12*.

Hojas

Para las hojas se definió la siguiente estructura

$$sheet/\{rollo\}\_\{hoja\} \tag{4.18}$$

Por ejemplo, el identificador para la hoja 234 del rollo 3 sería *https://mh.uy/luz/resource/sheet/3_234*.

4.7. Resultados intermedios

Algunas de las etapas del pipeline podrían ser muy costosas dependiendo de los algoritmos ejecutados y es por esto que se propone implementar un sistema

de almacenamiento de los resultados que proporcionan cada una de ellas.

El pipeline tiene la capacidad de almacenar los resultados de todas y cada una de las etapas que lo conforman de modo que el cálculo que realizan no necesite volver a hacerse nuevamente en posteriores ejecuciones, sino que cada una obtenga los resultados que almacenó previamente.

Sin embargo, ya que el pipeline es configurable de forma remota, se puede elegir cuáles etapas ejecutar y con qué configuración, por lo tanto es necesario guardar varios tipos de resultados intermedios diferentes para la misma etapa.

Por ejemplo, recordando que el orden de las etapas es inmutable pero algunas de ellas no obligatorias, sea *C1* una posible configuración:

- Reconocimiento de entidades nombradas desde diccionarios.
- Extracción de relaciones con NLTK.
- Construcción del grafo de conocimiento.

Con esta configuración *C1* el resultado obtenido de la extracción de relaciones dependerá directamente del resultado obtenido del reconocedor de diccionarios. También el grafo resultado depende directamente de los resultados de las dos etapas anteriores.

Ahora se supone la siguiente configuración *C2*:

- Reconocimiento de entidades nombradas utilizando expresiones regulares.
- Extracción de relaciones con NLTK.
- Construcción del grafo de conocimiento.

En este caso el resultado obtenido de la primera etapa de *C2* podría ser totalmente diferente al de la primera etapa de *C1*. Mientras que la segunda etapa de *C2* es igual a la de *C1*, el resultado obtenido de ambas igual podría ser diferente porque tienen entradas diferentes.

En general, el resultado de cualquier etapa dependerá totalmente de su configuración y el resultado de sus etapas anteriores.

4.7.1. Almacenamiento de los resultados

Los resultados intermedios son almacenados en el disco duro del servidor que ejecuta el pipeline de modo que pueda leerlos cuando lo necesite.

Los resultados son guardados en formato *JSON* junto a la configuración del pipeline a partir de la que fueron obtenidos en un objeto *JSON* con la siguiente estructura:

```
{
  "metadata": {
    "preprocessed": <boolean>,
    "ner_type": <string>,
    "coref_types": <array>,
    "rels_types": <array>,
  },
  "ner": <array>,
  "coreferences": <array>,
  "relations": <array>
}
```

El objeto *metadata* es el que determina la configuración del pipeline y contiene los siguientes metadatos:

- *preprocessed*: indica si el texto fue preprocesado o no.
- *ner_type*: indica el reconocedor utilizado para obtener entidades nombradas desde el texto. Los valores posibles hasta el momento son *freeling*, *spacy*, *stanza-ancora*, *stanza-conll02*, *nltk*, *regexp* o *dictionaries*.
- *coref_types*: indica el nombre de los algoritmos de resolución de correferencias utilizados en la ejecución. El valor posible hasta el momento es *named_entity_anaphora*.
- *rels_types*: indica el nombre de los algoritmos de extracción de relaciones utilizados en la ejecución. El valor posible hasta el momento es *nltk*.

Algunas de estas claves podrían no estar presentes, y eso significaría que la etapa no fue ejecutada.

Por último, las claves *ner*, *coreferences* y *relations* contienen las salidas de de las respectivas etapas.

La estructura de este contenido está principalmente diseñada para ser manejada fácilmente por la implementación. Sin embargo, se puede notar que es amigable al ojo del desarrollador lo que permite que esta arquitectura de resultados intermedios sirva, también, como una herramienta de depuración al desarrollar nuevas características y/o algoritmos para el pipeline.

4.7.2. Búsqueda de resultados intermedios

El sistema guarda cada ejecución en un archivo diferente para facilitar la implementación y la distribución de los resultados.

Sin embargo, cuando el pipeline tiene que ejecutar una etapa para un texto dado con una configuración determinada, primero tiene que recolectar los resultados correspondientes a la ejecución de la etapa con la misma configuración desde el disco duro (si existen) para utilizarlos y así ahorrarse la ejecución.

Frente a la necesidad anterior surgió el siguiente problema: el sistema debía leer cada uno de los archivos en el fichero de resultados intermedios buscando el que tenga la configuración buscada para un texto en particular. El tiempo de ejecución en el peor caso para este procesamiento es lineal a la cantidad de resultados intermedios que ya se hayan almacenado, donde el peor escenario se da cuando no ha habido una ejecución para el mismo texto con la misma configuración anteriormente.

La solución planteada a continuación permite realizar la búsqueda de los resultados en un tiempo constante. Esta consiste en mantener el número de hoja y rollo (los cuales identifican el texto) en el nombre del archivo que guarda los resultados. Además, se genera un código hash a partir del objeto *metadata* que registra la configuración de la ejecución. Este código de diez dígitos es generado con el algoritmo *blake2b*[1] y es agregado al nombre del archivo también.

En conclusión, todas las variables que determinan la unicidad del resultado de una etapa están contenidas en el nombre del archivo de resultados intermedios.

4.7.3. Ejecución forzada

Pueden presentarse algunos casos de uso donde el usuario del pipeline no desea utilizar los resultados que fueron almacenados anteriormente sino que desea ejecutar los algoritmos del pipeline.

El pipeline ignorará los resultados intermedios almacenados cuando el parámetro de consulta *force_exec*, proporcionado por la API, tenga el valor booleano *true*.

[1] https://docs.python.org/3/library/hashlib.html#blake2

Igualmente, el sistema almacenará los resultados en el disco después de la ejecución de cada etapa en caso de que se quiera disponer de estos datos en ejecuciones futuras.

4.8. Entrenamiento de modelos

Se implementó un módulo que se puede ejecutar por medio de *LUZ*, pero a diferencia de la gran mayoría de funcionalidades que presenta el proyecto, esta no se ejecuta utilizando el servidor web y tampoco es parte del pipeline.

El módulo implementado se encarga de entrenar un modelo a partir de un proyecto exportado de *INCEpTION* [21]. Para esto se debe tener como variable de entorno la dirección del proyecto exportado.

La ejecución del entrenamiento desde la línea de comandos se realiza de la siguiente manera:

python run.py train

El proceso de entrenamiento que ejecuta el módulo implementado es el siguiente:

1. Obtiene las anotaciones del proyecto de *INCEpTION* exportado.
2. Selecciona el modelo a entrenar. Para esto se busca en disco uno ya entrenado, de lo contrario se utiliza *es_core_news_md*[37], el mismo que en la implementación de *SpaCy* para *LUZ*.
3. Se entrena el modelo seleccionado en el punto anterior.
4. Se guarda en disco el modelo resultado luego de entrenar.

Cabe destacar que el lugar en donde se busca el modelo a entrenar en el punto 2 es el mismo en el que se guarda en el punto 4. Esto implica que el modelo es entrenado de forma iterativa.

4.9. Traza

Por cada ejecución del pipeline, este registra un conjunto de datos que puede permitir trazar qué etapas fueron ejecutadas sobre cada texto. En primer

lugar, se registra la configuración con la que se realizó la ejecución. Esto básicamente es el conjunto de parámetros que el usuario del servicio ingresó al ejecutarlo.

Por otro lado, para cada etapa del pipeline, se registra también si esta realiza efectivamente el procesamiento o carga los resultados intermedios guardados en el disco. En el caso de que se haga el procesamiento, el registro indica el nombre del archivo de resultados intermedios generado y en el caso de que se hayan cargado resultados anteriores entonces el registro indica el nombre del archivo cargado.

El esquema de estos tipos de registro es el siguiente:

```
fecha_y_hora | [nom_arch_texto] [etapa] información
```

Figura 4.8: Esquema de traza del pipeline

donde los valores posibles para *etapa* son *ner*, *coreferences*, *relations* y *graph*; los valores posibles para *información* son:

- *etapa ejecutada*, cuando la etapa realiza efectivamente el procesamiento;
- *nom_arch_res_int generado*, cuando se guardan los resultados intermedios en el archivo *nom_arch_res_int*; y
- *nom_arch_res_int cargado*, cuando se utilizan los resultados intermedios leídos de *nom_arch_res_int*.

A continuación se puede ver un ejemplo de los registros descriptos anteriormente donde la etapa de reconocimiento de entidades nombradas fue ejecutada y los resultados intermedios fueron guardados en el disco:

```
2021-06-19 22:19:59.473509 | Pipeline
   configurado - {'preprocessing': None, 'ner': {'
   recognizer': 'regexp', 'trained': False}, '
   coreferences': None, 'relations': None, 'force
   ': True}
2021-06-19 22:19:59.492696 | [tmp/texts/Rollo3_3
   .txt] [ner] etapa ejecutada
2021-06-19 22:19:59.494556 | [tmp/texts/Rollo3_3
   .txt] [ner] results3_3_b'\x1a\x03\xe4\xc7\xb4\
   xd7\xfc:v\xa3'.json generado
```

Figura 4.9: Ejemplo de traza del pipeline

Cabe notar que todos y cada uno de los registros contienen el instante de tiempo permitiendo trazar una historia de ejecuciones del pipeline.

Capítulo 5

Ground truth

Ground truth es un término que se refiere a datos del mundo real o verdad obtenida en la práctica. Se utiliza para poder comparar con algo que se considera correcto y de esta manera evaluar una propuesta de solución.

En el caso de este proyecto se construyó un conjunto *ground truth* para medir el reconocimiento de entidades nombradas, que es la base del procesamiento que hace *LUZ*.

5.1. Origen de los datos

Como se ha mencionado anteriormente, los textos con los que se trabaja en *LUZ* provienen de *LUISA*, que a partir de la participación de la comunidad genera transcripciones manuales de cada palabra o frase de los documentos microfilmados.

El grupo encargado de este proyecto tiene acceso a algunas imágenes de estos documentos y sus transcripciones realizadas mediante *LUISA*. En particular se cuenta con los datos de las primeras 2 tandas de documentos, que se llaman "tanda0" y "tanda1" y cuentan con 163 y 1000 documentos respectivamente, de varios rollos distintos.

Los datos fueron compartidos como un *dump* de base de datos que contiene varias tablas con información de las hojas, bloques y textos. Por el funcionamiento de *LUISA* las transcripciones de los documentos son proporcionadas por bloques, que son los recuadros que se presentan a los usuarios de la plataforma para que los transcriban. Estos bloques no corresponden necesariamente con una palabra, algunos son conjuntos de pocas

palabras o palabras cortadas. Además, para cada bloque existen muchas transcripciones que no son idénticas, algunas incluso son vacías. Por esta razón, para la reconstrucción de los textos se toma la transcripción más repetida de cada bloque. Los documentos de texto que resultan de esa reconstrucción son los utilizados como entrada de *LUZ*.

5.2. Construcción

El conjunto *ground truth* de este proyecto está constituido por 150 textos anotados, 75 del rollo número 643 y 75 del rollo número 677, que se obtuvieron a partir de anotaciones automáticas hechas con los patrones identificados en los textos 3.2.1 y posteriores curaciones manuales hechas por el equipo encargado del proyecto y Elena Bing.

La decisión de no hacer las anotaciones totalmente manuales se tomó debido al poco plazo que se tuvo para llevar a cabo esta tarea y también a que las anotaciones automáticas con patrones tienen un alto nivel de precisión, lo que reduce el trabajo manual posterior. Además, para el proceso de curación se utilizó *INCEpTION*, que es una plataforma de anotación que tiene como objetivo facilitar tareas de anotación interactiva y semántica [21].

Esta herramienta ya había sido estudiada por el equipo encargado de este proyecto en una investigación previa en el marco del curso *Recuperación de Información y Recomendaciones en la Web* [5], donde se reconocieron las siguientes fortalezas para llevar a cabo las tareas manuales de anotación y curación necesarias para *LUZ*:

- Puede ser ejecutada en un servidor remoto, permitiendo que los usuarios puedan acceder desde cualquier parte.
- Cuenta con una gestión de roles que permite limitar los permisos de acceso a distintas funcionalidades para cada usuario.
- Tiene la funcionalidad de importar y exportar proyectos y además se pueden cargar archivos con textos ya anotados. Esto es de vital importancia para este proyecto ya que se necesita cargar en la plataforma los textos anotados automáticamente en base a patrones y luego exportar los textos ya curados para hacer las evaluaciones deseadas.
- Permite editar y crear los conjuntos de etiquetas que se quieren utilizar para anotar los textos. De esta manera se pudo configurar un conjunto

de etiquetas para la anotación de entidades nombradas que coincidiera con la implementada en *LUZ*.

En el anexo se encuentran los manuales desarrollados para la instalación de *INCEpTION* .2 y para las anotaciones y curaciones manuales .3.

5.2.1. Curación de los textos

En esta sección se presentarán las distintas cuestiones que surgieron durante la etapa de curación de las anotaciones automáticas que se fueron atendiendo y resolviendo a medida que se avanzaba.

Como la granularidad de las anotaciones es a nivel de *token* y no de caracter cuando la tokenización no tiene el resultado esperado, quedan algunos *tokens* que incluyen palabras que deberían ser anotadas con etiquetas distintas. Por ejemplo *"8/mayo/1975-Art.20"* o *"Alberto,19"*. Esto deriva en que el etiquetado no tenga la mejor calidad por tener que elegir entre una de las dos categorías para ese tipo de *tokens*. Si bien estos casos se fueron viendo uno a uno, la medida general tomada fue que si una parte del *token* no pertenece a ninguna entidad, o sea que le corresponde una etiqueta O, y otra sí, se le asigna la etiqueta correspondiente a la entidad en cuestión. Por otra parte, cuando ambas partes pertenecen a distintas entidades, se le dio prioridad a la entidad que solo estuviera compuesta por ese *token*. Por ejemplo, si se tiene la siguiente expresión tokenizada

$$\overbrace{Montevideo, 20}\ \overbrace{de}\ \overbrace{setiembre}\ \overbrace{de}\ \overbrace{1974} \tag{5.1}$$

el primer *token* se etiquetaría como LOC ya que la entidad DATE a la que pertenece "20" puede ser representada por los demás *tokens* que la componen. De esta manera se busca perder la menor cantidad de información posible.

La anotación de las entidades de tipo organización implicó varios intercambios de criterios entre quienes anotaron y finalmente se decidió basar estas anotaciones en la siguiente definición: "conjunto de personas que se relacionan entre sí y tienen un objetivo en común". De esta manera se apuntó a anotar todos los organismos del Estado, grupos políticos, centros de enseñanza, clubes sociales y sindicatos, entre otros, ya que para este proyecto aporta valor reconocer la mayor cantidad de entidades de este tipo sin importar su importancia y magnitud. Otra dificultad que presenta el reconocimiento de

organizaciones es que hay varias entidades de este tipo que pueden ser también etiquetadas como lugares. Esto depende del contexto en el que se encuentre. Por ejemplo, si en un texto aparece la expresión *"se lo detuvo en frente a la Escuela Industrial de Treinta y Tres"* se interpreta la entidad *Escuela Industrial de Treinta y Tres* como un lugar. No obstante, si se tiene la expresión *"los docentes de la Escuela Industrial de Treinta y Tres"*, se interpreta como organización.

En el caso de entidades compuestas, es decir, que puedan ser desglosadas en más de una, como *"Escuela Industrial de Treinta y Tres"* que podría anotarse como

$$\overbrace{Escuela\ Industrial}^{ORG}\ de\ \overbrace{Treinta\ y\ Tres}^{LOC} \tag{5.2}$$

se definió anotarlas como una entidad única y clasificarlas según la primera de las entidades internas. Esto implica que el ejemplo anterior se clasifique como organización. La decisión se debe en parte a que por el momento este proyecto no incluye la detección de sub entidades y al anotarlas por separado sin relación alguna se perdería información.

Aspirando a reconocer la mayor cantidad de entidades posibles en pos de aproximarse a extraer tanta información como podría hacerlo una persona, se tomó la decisión de anotar todas las entidades que sean humanamente reconocibles al momento de hacer las curaciones manuales. Esto incluye anotar palabras con errores tipográficos y/o con arrobas (@)[1]. Esto evita sesgar las anotaciones, y por lo tanto las evaluaciones, para obtener métricas que sean lo más reales posibles.

Cabe reconocer que en todas las decisiones tomadas para esta etapa se asumió el riesgo de que podrían existir ambigüedades al momento de anotar y que habrían diferencias de criterio según la persona que anote cada texto. Para mitigar este riesgo hubiera sido deseable hacer iteraciones sobre las curaciones manuales para que cada texto fuera revisado por varias personas distintas y poder calcular el nivel de acuerdo entre los anotadores del corpus[2]. Sin embargo, el poco tiempo con el que se contaba para esta tarea no lo permitió.

[1] En las instrucciones de transcripción de *LUISA* se pide agregar el caracter @ previo a una palabra ilegible.

[2] Inter-Annotator Agreement (IAA) mide qué tan bien dos (o más) anotadores pueden tomar la misma decisión de anotación para cierta categoría. https://towardsdatascience.com/inter-annotator-agreement-2f46c6d37bf3

5.3. Estadísticas del corpus anotado

El corpus conformado por los 150 textos anteriomente mencionados cuenta con un total de 3994 entidades nombradas anotadas manualmente con un promedio de 26.63 entidades nombradas por texto.

La siguiente tabla muestra la cantidad de entidades por clase y el promedio de entidades por texto.

Clase	**Total**	**Promedio**
Persona	1671	11.14
Organización	837	5.58
Lugar	788	5.25
Fecha	698	4.65

Tabla 5.1: Cantidad de entidades nombradas por clase y promedio por texto

Capítulo 6

Resultados

6.1. Evaluación de NER

Para la evaluación de la etapa de reconocimiento de entidades nombradas se tomó el conjunto *ground truth* detallado en el capítulo 5 como conjunto de evaluación y se calculó *precisión*, *recuperación* y *medida F* para cada reconocedor tanto para el reconocimiento de entidades como para la clasificación de las mismas, como se explicará a continuación.

6.1.1. Mecanismos y medidas de evaluación y comparación

Al investigar el uso de varios reconocedores de entidades nombradas surgió la necesidad de compararlos con la intención de conocer las debilidades y fortalezas de cada uno en particular. Para esto, como se adelantó en la sección 4.1, se implementó un sistema de métricas que está disponible desde el servicio *GET /api/v1/ner_predict*.

Para acceder a las métricas de un reconocedor, en primer lugar, el texto para el cual se ejecuta el servicio debe contar con anotaciones reales que sirvan como *ground truth*. El sistema puede comparar la salida del reconocedor con estas anotaciones permitiendo así evaluar su desempeño.

En segundo lugar, se debe llamar al servicio con el parámetro de consulta *with_metrics* en *true* indicando así que se desea obtener métricas de los reconocedores.

6.1.1.1. Mecanismos de comparación y evaluación

La comparación de reconocedores tuvo varias iteraciones al principio del proyecto debido a que las primeras propuestas no eran lo suficientemente robustas para una evaluación adecuada a partir del *ground truth*.

Sin embargo, luego de probar algunos mecanismos que se basaban en comparar igualdad entre dos entidades nombradas o solapamiento de *tokens* entre las mismas, se decidió evaluar los resultados utilizando la notación BIO del texto.

A partir de la estructura con este sistema de etiquetado, la evaluación del resultado contra el etiquetado manual se realiza comparando las etiquetas de los *tokens* uno a uno verificando que estas sean iguales.

6.1.1.2. Medidas de evaluación

Para evaluar la salida obtenida de un reconocedor, se calcularon tres medidas: *precisión*, *recuperación* y F_1. *Precisión* mide el porcentaje de elementos positivos reales que el sistema detecta como positivos [20].

$$precisión = \frac{verdaderos\ positivos}{verdaderos\ positivos + falsos\ positivos} \tag{6.1}$$

Recuperación mide el porcentaje de elementos positivos reales que son detectados como positivos por el sistema.

$$recuperación = \frac{verdaderos\ positivos}{verdaderos\ positivos + falsos\ negativos} \tag{6.2}$$

La medida F_1 es una forma de combinar las dos medidas anteriores de forma balanceada.

$$F_1 = \frac{2 * precisión * recuperación}{precisión + recuperación} \tag{6.3}$$

El cálculo de estas medidas se realizó utilizando el módulo *sklearn.metrics.precision_recall_fscore_support* de la biblioteca *scikit-learn* [26] que provee varias herramientas utilizadas para el aprendizaje automático.

A su vez, estas medidas fueron calculadas para las siguientes dos tareas:

- **Reconocimiento de entidades nombradas**: esta evaluación se realiza *token* a *token*, lo que implica que no se evalúa si las entidades son reconocidas completamente. El propósito en este caso es evaluar si el reconocedor logró detectar para cada uno si es parte de una entidad nombrada o no, sin importar su tipo. Por lo que el cálculo de las medidas propuestas se realiza a partir de listas con valores binarios que indican si el *token* pertenece a una entidad nombrada o no.
- **Clasificación de entidades nombradas**: para esta tarea se realizan los siguientes dos tipos de evaluación.
  - *A nivel de token*: verifica que la categoría de cada *token* coincida con la categoría del *token* correspondiente en el etiquetado real, sin dar importancia al etiquetado BIO.
  - *A nivel de entidad nombrada*: esta estrategia, utilizada en la conferencia *CoNLL 2003* [38], calcula el rendimiento del sistema utilizando la medida F_1 donde *precisión* es el porcentaje de entidades nombradas detectadas por el sistema que son correctas y *recuperación* es el porcentaje de entidades nombradas reales que son detectadas por el sistema. Una entidad nombrada es correcta si y sólo si es una coincidencia exacta de la entidad nombrada real (todos sus *tokens* coinciden y tienen la misma clasificación). Para este cálculo se utilizó el módulo *metrics.classification_report* de la librería *seqeval* [30].

6.1.2. Evaluación de reconocimiento de entidades nombradas

En la siguiente tabla se muestran los valores obtenidos para cada uno de los reconocedores integrados al pipeline en cuanto al reconocimiento de entidades.

Reconocedor	P	R	F1
FreeLing	0.519	0.664	0,583
SpaCy	0.485	0.667	0,562
Stanza-AnCora	0.518	**0.705**	0.597
Stanza-CoNLL	0.642	0.644	**0.643**
NLTK	0.618	0.423	0.502
Patrones	0.810	0.384	0.521
Diccionarios	**0.833**	0.028	0.054

Tabla 6.1: Métricas de reconocimiento de entidades nombradas

Como se puede observar, el reconocedor más preciso, es el de diccionarios. Esto era esperable por la naturaleza de su funcionamiento. Sin embargo tiene un valor de *recuperación* muy bajo, ya que, como se menciona más adelante, solo cubre algunas menciones de entidades de tipo lugar y organización. Para obtener un mejor valor se podrían agregar nuevos diccionarios que abarquen otras categorías de entidades.

Algo similar ocurre con el reconocedor de patrones, que alcanza un valor de *precisión* apenas más bajo que el de diccionarios. Su alta *precisión* tiene sentido teniendo en cuenta que las expresiones regulares fueron construidas específicamente para este dominio de textos. No obstante, como no abarcan todas las menciones de entidades existentes en los documentos, ni todas las categorías, este reconocedor tiene un bajo nivel de *recuperación*. Este valor se podría mejorar agregando nuevas expresiones regulares que abarquen más menciones de entidades.

Por otro lado, el reconocedor que más armonía alcanza entre *precisión* y *recuperación* es el de *Stanza* con el corpus *CoNLL*.

A partir de estos resultados se desprende que una buena estrategia para aprovechar los puntos fuertes de los mejores reconocedores sería aplicar en un principio los reconocedores de patrones y de diccionarios y luego el de *Stanza* con el corpus *CoNLL*.

6.1.3. Evaluación de clasificación de entidades nombradas a nivel de *token*

En esta sección se mostrará el desempeño de los reconocedores utilizados para cada una de las categorías de entidades nombradas que se busca identificar

con *LUZ*.

Lugares

A continuación se observarán los valores alcanzados por los reconocedores para el reconocimiento de entidades de tipo lugar.

Reconocedor	P	R	F1
FreeLing	0.427	0.164	0.237
SpaCy	0.334	0.279	0.304
Stanza-AnCora	0.727	0.176	0.283
Stanza-CoNLL	0.492	**0.358**	**0.414**
NLTK	0.218	0.029	0.051
Patrones	0.675	0.198	0.306
Diccionarios	**0.824**	0.091	0.164

Tabla 6.2: Métricas para la categoría LOC

En esta tabla se ve un comportamiento similar al de la evaluación del reconocimiento de entidades, donde el reconocedor de diccionarios alcanza altos valores de *precisión* pero de los peores valores de *recuperación*. Esto último se debe a que únicamente reconoce países y departamentos de Uruguay como entidades de tipo lugar. Además, como muchas calles tienen nombre de países o departamentos, muchos *tokens* que deberían ser etiquetados como *LOC* no son reconocidos. Por ejemplo, si en un texto aparece la dirección *"Bulevar Artigas 1234"* se esperaría que el reconocedor la anote completamente como una entidad de tipo lugar. Sin embargo, en este caso solo anotaría *Artigas* con la etiqueta *LOC* ya que coincide con el nombre de un departamento de Uruguay.

El reconocedor de patrones tiene un comportamiento idéntico, donde la baja *recuperación* se debe a que las expresiones regulares construidas hasta el momento para este tipo de entidades cubren solo algunos tipos de lugares específicos como por ejemplo direcciones de domicilio, escuelas y liceos y deja por fuera otros como plazas, para las cuales se podría agregar un nuevo patrón. Que su *precisión* no destaque tanto en este caso se relaciona con que en varios textos las expresiones regulares toman más *tokens* de los que corresponde debido a que las condiciones de parada no aparecen como se espera en el texto, en general por malas transcripciones o baja calidad de los documentos.

Por ejemplo, si se tomara el siguiente fragmento de texto

"*finca de la calle Tristán Narvaja No.*1234 , *domicilio de la persona Ana María RODRIGUEZ PEREZ* , *soltera de* 26 *años*"

(6.4)

se esperaría que el reconocedor anote *"finca de la calle Tristán Narvaja No.1234"* como un lugar. Sin embargo, como la expresión regular que busca reconocer las fincas espera que el número de la dirección esté precedido por un espacio, en realidad anota *"finca de la calle Tristán Narvaja No.1234 , domicilio de la persona Ana María RODRIGUEZ PEREZ , soltera de 26"*.

Nuevamente el reconocedor que alcanza la mejor *medida F* es el de *Stanza* con el corpus *CoNLL*.

Organizaciones

En la tabla a continuación se muestran las medidas obtenidas para el reconocimiento de las entidades nombradas de tipo organización.

Reconocedor	P	R	F1
FreeLing	0.194	0.356	0.251
SpaCy	0.281	0.248	0.263
Stanza-AnCora	0.347	**0.393**	**0.369**
Stanza-CoNLL	0.379	0.323	0.349
NLTK	0.174	0.193	0.183
Patrones	0.0	0.0	-
Diccionarios	**0.585**	0.091	0.137

Tabla 6.3: Métricas para la categoría ORG

En este caso los valores son bastante bajos para todos los reconocedores pero el que obtiene el mejor valor de *medida F* es *Stanza* con el corpus *AnCora*.

Cabe destacar que el valor de *precisión* y *recuperación* es 0 para el reconocedor basado en patrones debido a que no se tienen expresiones regulares para reconocer organizaciones.

Al observar el comportamiento del reconocedor de diccionarios se notó que mientras en los diccionarios se tienen organizaciones como "SERVICIO DE INFORMACION DE DEFENSA" en los documentos aparecen muchas expresiones como "Departamento II del SERVICIO DE INFORMACION DE

DEFENSA", que debería clasificarse como una organización en sí. Por lo tanto, la anotación obtenida por el reconocedor de diccionarios para esta expresión sería

$$\overbrace{Departamento}^{O}\ \overbrace{II}^{O}\ \overbrace{del}^{O}\ \overbrace{SERVICIO}^{ORG}\ \overbrace{DE}^{ORG}\ \overbrace{INFORMACION}^{ORG}\ \overbrace{DE}^{ORG}\ \overbrace{DEFENSA}^{ORG} \tag{6.5}$$

y la anotación manual

$$\overbrace{Departamento}^{ORG}\ \overbrace{II}^{ORG}\ \overbrace{del}^{ORG}\ \overbrace{SERVICIO}^{ORG}\ \overbrace{DE}^{ORG}\ \overbrace{INFORMACION}^{ORG}\ \overbrace{DE}^{ORG}\ \overbrace{DEFENSA}^{ORG} \tag{6.6}$$

De esta manera, varios *tokens* que deberían ser etiquetados como *ORG* no son reconocidos, lo que afecta negativamente a su *recuperación*. Además, los diccionarios no contienen todas las organizaciones presentes en los textos, lo cual sería imposible, por lo tanto se pierden varias menciones de entidades de este tipo.

Por la naturaleza del reconocedor de diccionarios su *precisión* debería tender a 1 pero eso no sucedió ya que en las anotaciones manuales no se tuvieron en cuenta varias de las organizaciones que aparecen listadas en los diccionarios. Esa lista debería haber sido parte de la guía de anotación manual para evitar este error.

Personas

A continuación se presentarán los valores alcanzados por los distintos reconocedores para el reconocimiento de entidades de tipo persona.

Reconocedor	P	R	F1
FreeLing	0.612	0.467	0.530
SpaCy	0.714	0.583	0.642
Stanza-AnCora	**0.867**	0.688	**0.767**
Stanza-CoNLL	0.813	**0.707**	0.756
NLTK	0.483	0.451	0.466
Patrones	0.725	0.599	0.656
Diccionarios	0.0	0.0	-

Tabla 6.4: Métricas para la categoría PER

Se vuelve a obtener una alta *precisión* del reconocedor de patrones y, a diferencia de la evaluación para las entidades de tipo lugar, sí alcanza buenos valores de *recuperación*. Esto se debe a que las expresiones regulares construidas para las entidades de tipo persona son más abarcativas. Por otro lado, el reconocedor más preciso y también más armónico entre *precisión* y *recuperación* es el de *Stanza* con el corpus *AnCora*.

Fechas

Para el caso de las entidades nombradas de tipo fecha se tienen solo dos reconocedores para comparar: el de patrones, implementado por el equipo a cargo del proyecto, y *FreeLing*, que de las herramientas investigadas para *NER* es la única que puede reconocer fechas en español. A continuación se muestran las medidas obtenidas para el reconocimiento de entidades de este tipo.

Reconocedor	P	R	F1
FreeLing	0.832	**0.549**	**0.662**
Patrones	**0.985**	0.468	0.635

Tabla 6.5: Métricas para la categoría DATE

Como se muestra en la tabla, el reconocedor más preciso es el de patrones, alcanzando casi el 100 %. Sin embargo, *FreeLing* presenta mejores valores de *recuperación* y *medida F*.

La baja *recuperación* del reconocedor de patrones se debe a que existen varias menciones de entidades de tipo fecha que no satisfacen las expresiones regulares implementadas hasta el momento. Por ejemplo, no están contempladas las fechas constituídas solo por el día y el mes, sin mención del año, como “18 de setiembre”. Es una oportunidad de mejora agregar patrones

que modelen este tipo de fechas "incompletas" a este reconocedor para mejorar su desempeño.

6.1.4. Evaluación de clasificación de entidades nombradas a nivel de entidad nombrada

A continuación se presentan los resultados de la evaluación de clasificación a nivel de entidad nombrada, la cual, a grandes rasgos, mide cuántas entidades nombradas fueron detectadas correctamente.

Reconocedor	P	R	F1
Freeling	0.206	0.150	0.173
SpaCy	0.165	0.234	0.193
Stanza-AnCora	0.419	0.214	**0.284**
Stanza-CoNLL	0.238	**0.327**	0.276
NLTK	0.119	0.051	0.071
Patrones	**0.584**	0.102	0.173
Diccionarios	0.526	0.180	0.268

Tabla 6.6: Métricas a nivel de entidad nombrada para LOC

Reconocedor	P	R	F1
Freeling	0.088	0.202	0.123
SpaCy	0.153	0.176	0.164
Stanza-AnCora	0.175	**0.296**	0.220
Stanza-CoNLL	0.231	**0.296**	**0.260**
NLTK	0.037	0.104	0.054
Patrones	0.0	0.0	-
Diccionarios	**0.303**	0.012	0.023

Tabla 6.7: Métricas a nivel de entidad nombrada para ORG

Reconocedor	P	R	F1
Freeling	0.399	0.370	0.384
SpaCy	0.435	0.414	0.424
Stanza-AnCora	0.579	**0.536**	**0.556**
Stanza-CoNLL	0.372	0.442	0.404
NLTK	0.149	0.242	0.184
Patrones	**0.633**	0.496	0.556
Diccionarios	0.0	0.0	-

Tabla 6.8: Métricas a nivel de entidad nombrada para PER

Reconocedor	P	R	F1
Freeling	0.674	0.460	0.547
Patrones	**0.970**	**0.509**	**0.667**

Tabla 6.9: Métricas a nivel de entidad nombrada para DATE

A partir de estos números y los presentados en la evaluación de clasificación a nivel de *token*, se pueden observar varios puntos. En primer lugar, se nota que los reconocedores basados en patrones y diccionarios son los más precisos a la hora de detectar entidades nombradas correctas. Sin embargo, el reconocedor de la librería *Stanza* sigue siendo el más armónico al clasificar personas, organizaciones y lugares.

En cuanto a las entidades de tipo fecha, el reconocedor basado en patrones supera a *FreeLing* en todas las medidas.

Por último, se nota un detalle en la recuperación de organizaciones donde *Stanza* obtiene el mismo valor usando cualquiera de los dos corpus.

Es importante aclarar que si bien al trabajar con un reconocedor en base a patrones pueden haber ambigüedades que eviten que su *precisión* se acerque a 1, en el caso de este proyecto la calidad de los documentos y sus transcripciones aumentan este riesgo. Como se explicó en la sección 3.1, los textos obtenidos de las transcripciones hechas en *LUISA* tienen algunas desventajas para su procesamiento. Por ejemplo, como las expresiones regulares que modelan las entidades de tipo persona se basan en ciertas combinaciones de palabras en mayúscula y minúscula, el no respetar el uso de las mayúsculas en las transcripciones puede implicar que una expresión regular reconozca alguna expresión que no es una entidad del tipo deseado o que ni siquiera es una entidad. En el caso de los lugares ocurre algo similar con los signos de puntuación, que son algunos de los caracteres utilizados como condición de parada de las expresiones regulares, lo que hace que en algunos casos se tomen más *tokens* de los que corresponden como se mencionó anteriormente.

6.2. Evaluación de Resolución de correferencias

En esta sección se mostrarán los resultados de la etapa de resolución de correferencias. Más específicamente, para el algoritmo detallado en 3.3.1 se analizarán algunos de los casos donde los resultados son los esperados. Pero también se enumerarán aquellos donde no lo son debido a casos borde que el algoritmo no contempla o incluso problemas previos como la transcripción de los textos.

En primer lugar, cabe destacar que al no contar con un corpus de evaluación con correferencias anotadas manualmente, no es posible plantear un estudio con métricas exactas como las presentadas para el reconocimiento de entidades nombradas. Sin embargo, para el análisis se utilizará el *ground truth* definido en 5 contando así con entidades nombradas anotadas manualmente, en lugar de utilizar las reconocidas por alguna de las herramientas de NER.

En total se detectaron 248 cadenas de correferencias en el conjunto *ground truth*. A continuación se muestra una tabla con el total de cadenas según su largo.

Largo	2	3	4	5	6	7	8	9
Total	179	42	10	6	6	3	1	1

Tabla 6.10: Cantidad de cadenas según su largo

Por otro lado, a causa de la sensibilidad del contenido de los textos de entrada, los resultados no pueden ser presentados en su totalidad, por lo que se mostrarán únicamente entidades nombradas sin contexto alguno.

6.2.1. Correferencias detectadas correctas

Como se explicó anteriormente, las correferencias son representadas en formato de cadena donde una entidad nombrada A puede hacer referencia a otra entidad B la cual podría correferir a una tercera entidad C. Esto forma una cadena de correferencias de la forma:

$$Entidad\ C \leftarrow Entidad\ B \leftarrow Entidad\ A \tag{6.7}$$

Este método permite, entre otras cosas, representar cadenas no lineales de correferencias que compartan al menos una entidad nombrada.

Por ejemplo, las tres fechas siguientes resultan en dos cadenas de correferencias solapadas donde la entidad en común es *1975*.

$$\begin{aligned} 1975 &\leftarrow 26 \textit{ de mayo de } 1975 \\ 1975 &\leftarrow 3 \textit{ de enero de } 1975 \end{aligned} \tag{6.8}$$

En el conjunto utilizado se detectaron 7 casos donde dos cadenas de correferencias se encuentran solapadas. En particular, en uno de ellos, las cadenas solapadas comparten las primeras dos entidades nombradas.

Por otro lado, se encuentran varios casos exitosos, como los dos que siguen, en el que se detectan correferencias entre entidades nombradas donde una es abreviación de la otra.

$$\begin{aligned} D.\ E.\ II &\leftarrow Departamento\ II\ (E-2) \\ R.O.E. &\leftarrow Resistencia\ Obrero\ Estudianti\ l \end{aligned} \tag{6.9}$$

Se puede notar que la palabra *Estudiantil* tiene un espacio en medio formando así dos *tokens* en realidad. Esto no afecta la resolución de la correferencia ya que en este caso cada una de las letras de la abreviación son iniciales de alguna palabra en la otra entidad.

6.2.2. Correferencias detectadas incorrectas

La lógica que el algoritmo utiliza para resolver abreviaciones puede resultar en casos erróneos como se ve en el siguiente ejemplo.

$$R.\ C.\ E. \leftarrow Es-\ cuela\ Compte\ y\ Rique\ t \tag{6.10}$$

Así mismo, también se encuentran algunos casos de detección de correferencias sin abreviaciones que no son correctas. En un ejemplo concreto, el algoritmo detecta que *Felicia Esther BERROSFE de RODRIGUEZ* es correferencia de *Esther RODRIGUEZ RODRIGUEZ* y esto se debe a que todos los *tokens* de esta última entidad ocurren en la primera. Esta es una detección incorrecta porque las entidades se refieren a diferentes personas en la realidad.

Otro caso aún más complejo es el de las correferencias transitivas erróneas.

Este fenómeno se da con cadenas de correferencias con al menos tres entidades nombradas como la planteada en 6.7 donde *Entidad A* es correferencia de *Entidad B* que a su vez es correferencia de *Entidad C*, pero la primera no es correferencia de esta última.

Se presenta un ejemplo claro de este problema.

$$\begin{aligned} \textit{Liceo ``Zorrilla''} &\leftarrow \textit{ZORRILLA} \\ \textit{ZORRILLA} &\leftarrow \textit{Agrupacion Estudiantes Zorrilla} \end{aligned} \tag{6.11}$$

En este caso la correferencia correcta es la primera, ya que *ZORRILLA* hace mención al *Liceo Zorrilla* y no a la agrupación de estudiantes.

6.2.3. Correferencias no detectadas

Por otro lado, se notaron ocasiones en los que aparecen otros tipos de abreviaciones que no son de la forma *A.* (una sola letra seguida de un punto) sino que involucran más letras en ella, por lo que la correferencia no es resuelta por el algoritmo propuesto.

En particular, se encuentran dos casos claros donde en el primero *DEPARTAMENTO II (EXTERIOR)* no se ha podido resolver como correferencia de *DPTO. II (EXTERIOR)* ya que *DEPARTAMENTO* tendría que haber sido abreviada como *D.* en vez de *DPTO.*. En segundo lugar, *Fuerzas Armadas* no se reconoce como correferencia de *FF.AA* ya que esta abreviación no es de la forma *F.A.*

6.2.4. Casos que fueron correctamente no detectados como correferencias

En primer lugar, se muestra un caso en el que la correferencia no es detectada gracias a la verificación de las clases de las entidades nombradas.

Como se explicó en 3.3.2, para que dos entidades nombradas sean correferentes, deben haber sido clasificadas de la misma manera.

En particular, se encuentra un caso donde aparecen las entidades *JEFATURA DE POLICIA DE MONTEVIDEO* y *Montevideo* las cuales tiene potencial para ser correferentes, en cuanto al contenido textual. Sin embargo, como estas corresponden a clases diferentes (organización y lugar, respectivamente), el algoritmo resuelve que no están relacionadas.

A modo estadístico, esta verificación filtró 83 casos de correferencias en el conjunto *ground truth* donde las entidades no eran de la misma clase.

En contraparte, existen casos donde las dos entidades nombradas son clasificadas de la misma manera, pero difieren en algún *token*. Por ejemplo, las entidades *Hector Arturo GARBARINO FREIRE* y *Hector Arturo GARBARINO COUTO* de tipo persona coinciden en todos los *tokens* excepto el último, por lo que el algoritmo resuelve que estas no son ocurrencias de la misma persona.

6.2.5. Problemas previos a la resolución

Se han abordado los casos más notorios de detección y filtrado de correferencias, pero aún resta enumerar otros en los que el algoritmo no pudo detectar correctamente. Esto se debe a errores cometidos por los reconocedores de entidades nombradas o incluso por la calidad actual de los textos de entrada. Esto último se está trabajando en proyectos paralelos que buscan mejorar la calidad de las transcripciones.

En primer lugar, se pueden notar varios casos en los que los espacios entre palabras no están colocados correctamente. Por ejemplo, *Urug uay* no ha podido ser reconocido como correferencia de *Uruguay* debido al espacio agregado en la reconstrucción de los textos. En contraparte, hay ocasiones en las que hay espacios faltantes. Uno de los casos encontrados se trata de *JESSIMACHIE* el cual no se reconoció como correferencia de *JESSI MACHIE*.

Si bien este problema se podría resolver eliminando todos los espacios de las entidades a la hora de resolver las correferencias, esto afectaría otros aspectos lingüísticos cayendo así, por ejemplo, en casos de *calambur*[1]

Por otra parte, es muy común que las oraciones en este tipo de textos terminen con los caracteres *punto* y *guión* seguidos (.-) evitando así que el tokenizador separe la última palabra del *punto*. Esto implica que el algoritmo de resolución no reconozca correferencias potenciales, por ejemplo para las entidades *PIRIAPOLIS.-* y *PIRIAPOLIS*. Este problema podría haber sido atacado en la etapa de preprocesamiento del texto reemplazando esta secuencia por un *punto*, permitiendo una adecuada tokenización.

[1]Figura retórica que se caracteriza por alterar la unión de las palabras para luego modificar el significado de la oración. Ejemplo: *Entreno en coche de carreras. / En tren o en coche de carreras.*
Fuente: https://www.ejemplos.co/calambur/#ixzz7BvOpJgpX

Otros casos recurrentes son los de errores tipográficos como, por ejemplo, en las entidades *Instituto "Osimani y Lloro na"* e *Instituto "Osimani y Llere na"* donde el cambio de una letra no hace posible la adecuada resolución de la correferencia.

Por último, se encuentran varios casos de error debido a fallas cometidas por el reconocedor de entidades nombradas, como pueden ser clasificaciones incorrectas. Una ocurrencia concreta de esto son las entidades *Asamblea general del Claustro Universitario.-* y *Asamblea General*, clasificadas como organización y persona, respectivamente.

6.3. Evaluación de Extracción de relaciones

A lo largo de esta sección se observarán los resultados arrojados por la extracción de relaciones, detallando los casos de éxito y error del algoritmo presentado en la sección 3.4. En 6.3.1 y 6.3.2 se presentará la evaluación sin utilizar lematización y en 6.3.3 se abordará cómo afecta a los resultados esta configuración.

Se comenzará evaluando la extracción de relaciones sin utilizar lematización, dejando esta evaluación para la última subsección dentro de esta sección.

Es importante resaltar en este capítulo que el reconocimiento de relaciones se basa en expresiones regulares para 2 entidades nombradas seguidas en el texto a procesar, por lo que toda relación que no se ajuste a los patrones definidos, no será identificada por *LUZ*. Cabe aclarar que se denominarán como entidades nombradas seguidas a dos entidades nombradas que aparezcan en el texto o bien pegadas, o bien separadas por *tokens* que no pertenezcan a otras entidades. También se debe notar que no se cuenta con relaciones anotadas manualmente, por lo que no se podrán extraer métricas exactas como se hizo para las entidades nombradas. Igualmente en está sección se utilizará el *ground truth* definido en el capítulo 5, tomando las entidades nombradas anotadas que se utilizarán para analizar las relaciones.

Por otro lado a lo largo de esta sección se cambiarán los nombres de las personas y las direcciones de los ejemplos a modo de asegurar la confidencialidad de la información.

6.3.1. Relaciones detectadas

Ejecutando el algoritmo de relaciones para las anotaciones pertenecientes al *ground truth* se logró extraer un total de 40 relaciones donde 25 son del rollo 643 y 15 del 677. En la siguiente gráfica se muestra cuántas instancias fueron encontradas para cada tipo de relación.

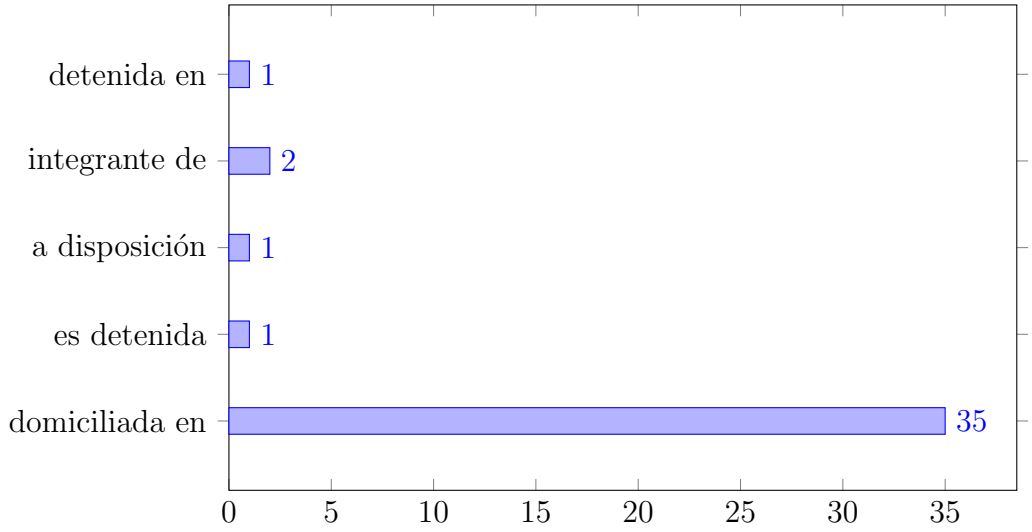


Figura 6.1: Relaciones reconocidas

Esta gráfica denota claramente una gran cantidad de relaciones detectadas del tipo *domiciliada en*, lo cual tiene sentido ya que a la hora de recabar patrones se observó que era una expresión muy utilizada en los textos. En la siguiente sección se explicará por qué no se identificaron los otros tipos de relaciones.

A continuación se muestra un ejemplo de cada una de las relaciones detectadas en forma de 3-upla donde el primer valor corresponde a la primera entidad nombrada, el segundo al identificador de la relación y el último valor a la segunda entidad nombrada.

{ 29 *de mayo de* 1975, *es detenida*, *José María MOLINARES PEREZ* }

{ *DARIO JORGE ROMARIO LIMA*, *domiciliada en*, *Canelones* 2779 }

{ *Pedro GONZALEZ TUALA*, *a disposición*, *Justicia Milit r de Instrucción de* 6*to. Turno* }

{ *FAGUNDEZ MARTINEZ*, *detenida en*, *Facultad de Humanidades* }

{ Enrique Y@NTAS OB@@TI, integrante de, U.J.C }

Se puede notar en la última relación de los ejemplos que la entidad nombrada contiene arrobas lo cual es un claro error de las transcripciones de los textos.

Al ser este tipo de algoritmo basado en expresiones regulares y en el reconocimiento de entidades nombradas, los resultados obtenidos son sensibles a la salida de NER, por lo que si estos presentan errores es muy probable que el resultado del reconocedor de relaciones no sea del todo correcto.

Un ejemplo de esto se puede ver con la relación *integraba* cuando el nombre de la persona es etiquetado erróneamente. En particular, esto ocurre cuando se utiliza el reconocedor *NLTK* en uno de los textos de *LUZ* en el cual reconoce *Peligrosidad Posterior* como una persona y a *Movimiento Marxista* como una organización, por lo que claramente la primer etiqueta es errónea. Luego, como se encuentra la palabra *integraba* dentro de la ventana de tolerancia entre estas entidades nombradas, el algoritmo extrae la relación:

$$\{\ Peligrosidad\ Posterior,\ integraba,\ Movimiento\ Marxista\ \}$$

Esto es, claramente, un error arrastrado desde el reconocimiento de entidades nombradas.

6.3.2. Relaciones no detectadas

A continuación se procede a analizar patrones para los cuales no se han encontrado relaciones en los textos.

La relación *integraba* no fue reconocida en ningún texto, esto se debe a que no se encuentran en el *ground truth* ocurrencias de la palabra "integraba" que es lo que busca el patrón.

Otra relación que no fue extraída de los textos analizados es *liberada*. Sin embargo, se notó que uno de los problemas era que las entidades pueden aparecer en distinto orden a como se esperan. Un ejemplo de este caso se detalla a continuación:

José Pedro VARELA ARTIGAS. – liberado el día Servicio de Inform
DEPARTAMENTO 26 *de Julio de* 1.975.–

El patrón implementado busca una fecha, luego la expresión regular que coincide con la palabra "liberado" y por último la persona. En este ejemplo se encuentra la persona antes que la fecha por lo que no es reconocida. Esto se podría solucionar agregando más patrones para las relaciones.

Es interesante analizar el siguiente fragmento de texto donde no se detecta ninguna relación:

Anibal CARDONA PARADEDA, " en averiguacion , quien posteriormente fuera puesto en libertad el dia 30 *del mismo mes y año , habiendose comunicado lo expresado en los Ra – dios Nros .* 378 *y* 379 *respectivamente de esta Unidad. – El dia* 18 *de mayo*

La relación *liberada* busca una etiqueta de tipo persona, luego una expresión regular que coincide con "puesto en libertad" y por último una fecha, por lo que en este texto se debería haber extraído una 3-upla { *Anibal CARDONA PARADEDA*, *liberada*, *día* 30 *del mismo mes* }, o en efecto la 3-upla { *Anibal CARDONA PARADEDA*, *liberada*, 18 *de mayo* }. Sin embargo, aplicar el algoritmo a este fragmento de texto no arroja resultados.

La primer 3-upla no se encuentra debido a que en las anotaciones manuales que forman parte del *ground truth* no se anotó "30 del mismo mes" como una fecha, por lo que esa relación no se extrae. Con respecto a la segunda, esta no se arroja como resultado debido a que la ventana de tolerancia está fijada en 10 *tokens* para este patrón, por lo que para poder anotar automáticamente esta relación debe de haber como máximo 10 *tokens* entre las 2 entidades nombradas, y entre ellas encontrarse las palabras "puesto en libertad". En este caso hay 38 *tokens* entre "Anibal CARDONA PARADEDA" y "18 de mayo" por lo que no se extrae la segunda relación.

6.3.3. Lematización

A continuación se analizan los resultados de la extracción de relaciones para el reconocedor implementado, configurado con lematización para las herramientas que se utilizaron en *LUZ* mencionadas en 3.4.

Utilizando la lematización con *Stanza* se extraen un total de 46 relaciones, logrando extraer más que sin lematizar. Éstas se dividen de la forma que muestra la gráfica siguiente:

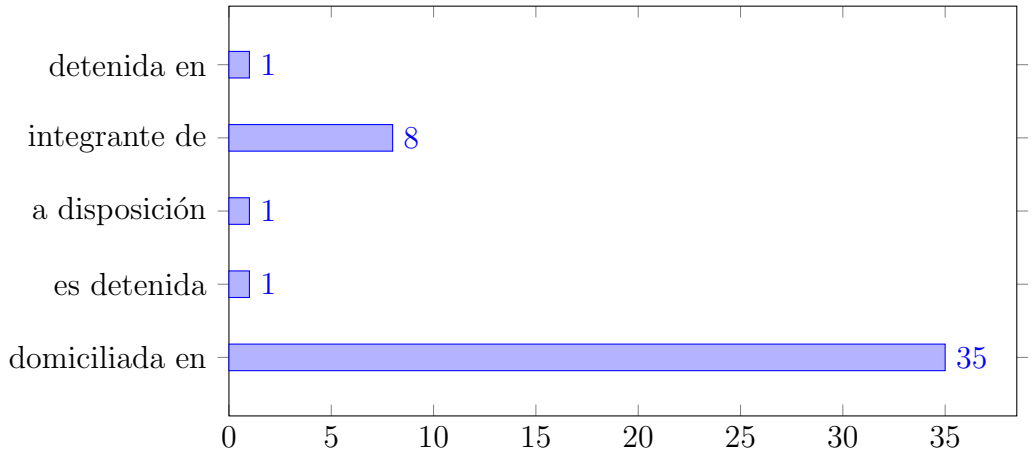


Figura 6.2: Relaciones reconocidas utilizando lematización con Stanza

La siguiente gráfica muestra el detalle por categoría de las 47 relaciones obtenidas utilizando el reconocedor configurado con lematización *FreeLing*:

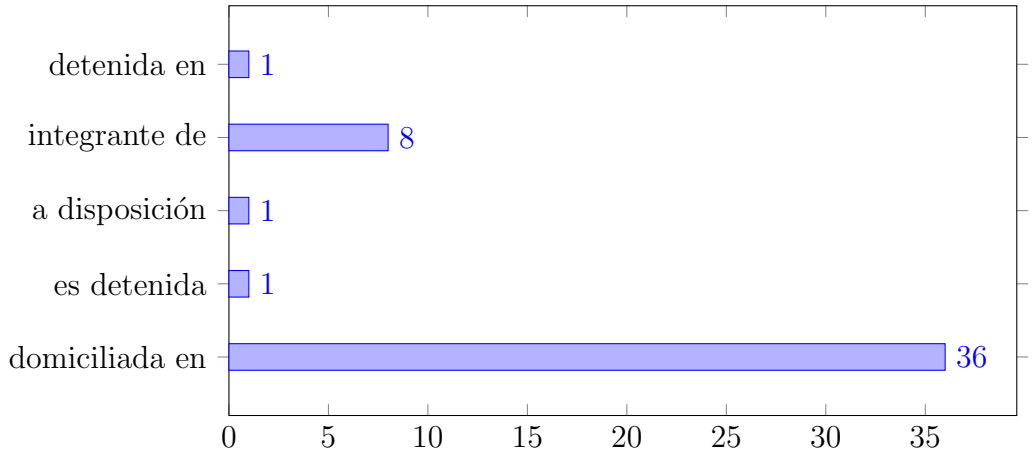


Figura 6.3: Relaciones reconocidas utilizando lematización con FreeLing

Se puede observar que en las relaciones obtenidas con ambos lematizadores

hay un aumento de 6 en la relación *integrante de* en comparación con las mostradas en la gráfica 6.1 para las relaciones obtenidas sin lematizadores. Esto se debe a que el patrón reconoce el texto "integrante de", pero al haber lematizado el texto el patrón reconoce expresiones del tipo "integrantes de" también, por lo que en los 6 ejemplos nuevos detectados en el *ground truth* se obtuvieron estas relaciones al tener una lista de personas, luego la expresión regular y por último una organización, como se puede ver en el siguiente ejemplo:

$Jacinto\ YANEZ\ y\ Carlos\ MACHIN\ ,\ integrantes\ de\ la\ Dirección\ del\ P.C$

Debido a cómo está implementado, para este ejemplo el algoritmo reconoce la relación $\{Carlos\ MACHIN,\ integrante\ de,\ Dirección\ del\ P.C\ \}$, pero no la de "Jacinto Yanez", ya que el algoritmo toma solo entidades nombradas seguidas en la lista de estas para el texto.

Cabe resaltar que no es necesario usar lematización para encontrar estos ejemplos nuevos, ya que agregando patrones para plurales también se reconocerían.

En la gráfica 6.3 se puede notar que hay una relación nueva de tipo *domiciliada en*, en comparación con las otras 2 gráficas anteriores, esto ocurre debido a los siguientes aspectos:

- Si bien el patrón busca por el texto "domiciliado", la lematización de *Stanza* y *FreeLing* es distinta para esta palabra, ya que para el primero el lema es "domiciliado" y para el segundo es "domiciliar". Esto se debe a que estas herramientas asignan categorías gramaticales diferentes, para *Stanza* es un adjetivo mientras que para *FreeLing* es un verbo.
  Por lo tanto el reconocedor con lematización va a buscar por distintas palabras lematizadas para cada lematizador.
- El ejemplo que difiere contiene la palabra "domicilia", el cual para ambos lematizadores tiene lema "domiciliar".

Por lo tanto utilizando la lematización de *Stanza* no se reconoce la relación, pero en cambio utilizando la de *FreeLing* sí. A continuación se presenta el ejemplo explicado:

Ana Maria BATLLE MIRANDA , oriental casada de 48 años . domicilia en Florida No.235

Esta diferencia se da debido a una mala transcripción ya que para que la oración estuviera bien formada debería decir "se domicilia en". En ese caso sería más sencillo reconocer que *domicilia* es un verbo.

6.4. Evaluación de la generación de los grafos de conocimiento

En esta sección se presentarán algunos de los resultados que se pueden obtener con consultas *SPARQL* sobre la base de conocimiento generada a partir de los grafos construidos como salida del pipeline.

Al igual que para la evaluación del resto de las áreas de este proyecto, se utilizó el conjunto *ground truth* definido en 5 para la generación de los grafos de conocimiento. Se utilizó la versión libre de *GraphDB* [14] para generar y consultar la base de conocimiento.

En principio se muestra una consulta básica que devuelve la cantidad total de entidades nombradas encontradas en la base de conocimiento según su clasificación.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX luz: <http://mh.uy/luz#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
select (?type as ?Clase) (count(?res) AS ?Cantidad) {
    luz:NamedEntity owl:equivalentClass/owl:unionOf/rdf:rest*/rdf:first ?type .
    ?res a ?type .
}
group by ?type
```

keyboard shortcuts

Filter query results — Showing results from 1 to 4 of 4. Query took 0.2s, today at 16:01.

	Clase	Cantidad
1	luz:Organization	"837"^^xsd:integer
2	luz:Date	"698"^^xsd:integer
3	luz:Person	"1671"^^xsd:integer
4	luz:Place	"788"^^xsd:integer

Figura 6.4: Cantidad de entidades nombradas por clase

Se puede notar que la totalidad de las entidades es de 3994 como se mencionó anteriormente.

Por otro parte, los objetivos de este proyecto enfatizan la necesidad de recuperar la *provenance* de las ocurrencias de las entidades nombradas y las relaciones reconocidas de los textos. A partir de esto, se arman algunas consultas que permiten obtener esta información fácilmente.

6.4.1. Ocurrencias de entidades nombradas

A continuación se presenta una consulta que refleja la *provenance* de todas las entidades nombradas filtradas para un texto en particular.

```
BASE <http://mh.uy/luz/resource/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX luz: <http://mh.uy/luz#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
select distinct (concat(?roll, "_", ?sheet) as ?Rollo_Hoja) ?entity ?entity_label (?
begin as ?Comienzo) (?end as ?Final)
where {
    ?entity a/^(owl:equivalentClass/owl:unionOf/rdf:rest*/rdf:first) luz:NamedEntity ;
            rdfs:label ?entity_label ;
            luz:hasOccurrence ?occ .
    ?occ luz:occurrenceBegin ?begin ;
         luz:occurrenceEnd ?end ;
         luz:hasOccurredIn [ luz:roll ?roll ;
                             luz:sheet ?sheet ] .
    filter(?roll = "643" && ?sheet = "0039")
}
```

keyboard shortcuts

Filter query results

Showing results from 1 to 71 of 71. Query took 0.4s, minutes ago.

	Rollo_Hoja	entity	entity_label	Comienzo	Final
1	"643_0039"	http://mh.uy/luz/resou	"Ejército Nacional"	"2002"^^xsd:integer	"2019"^^xsd:integer
2	"643_0039"	http://mh.uy/luz/resou _lleja_1	"Jefatura de Policía d e Lava- lleja"	"870"^^xsd:integer	"904"^^xsd:integer
3	"643_0039"	http://mh.uy/luz/resou	"Liceo N°13"	"3404"^^xsd:integer	"3414"^^xsd:integer
4	"643_0039"	http://mh.uy/luz/resou	"R.O.E"	"1114"^^xsd:integer	"1119"^^xsd:integer
5	"643_0039"	http://mh.uy/luz/resou	"10/6/975"	"204"^^xsd:integer	"212"^^xsd:integer
6	"643_0039"	http://mh.uy/luz/resou	"17/6/975"	"2711"^^xsd:integer	"2719"^^xsd:integer
7	"643_0039"	http://mh.uy/luz/resou	"20 de marzo"	"2029"^^xsd:integer	"2040"^^xsd:integer

Figura 6.5: Ocurrencias de las entidades nombradas en la hoja 643_0039

6.4.2. Correferencias entre entidades nombradas

Las correferencias se registran como propiedades entre ocurrencias de entidades nombradas como se detalló en el capítulo 3.5. La siguiente consulta permite obtener las correferencias que existen en cada hoja y también su *provenance*.

```
BASE <http://mh.uy/luz/resource/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX luz: <http://mh.uy/luz#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX : <http://mh.uy/luz#>
select (concat(?n_roll, "_", ?n_sheet) as ?Rollo_Hoja)
(concat(?ent1_label, " (", str(?ent1_begin), ", ", str(?ent1_end), ")") as ?Antecedente)
(concat(?ent2_label, " (", str(?ent2_begin), ", ", str(?ent2_end), ")") as ?Correferente)
where {
    ?ent1 rdfs:label ?ent1_label ;
          luz:hasOccurrence [ luz:hasOccurredIn <sheet/677_0630> ;
                              luz:hasCoreference [ luz:hasOccurredIn ?sheet ;
                                                   ^luz:hasOccurrence/rdfs:label ?ent2_label ;
                                                   luz:occurrenceBegin ?ent2_begin ;
                                                   luz:occurrenceEnd ?ent2_end ] ;
                              luz:occurrenceBegin ?ent1_begin ;
                              luz:occurrenceEnd ?ent1_end ] .
    ?sheet luz:roll ?n_roll ;
           luz:sheet ?n_sheet .
}
order by ?sheet
```

Press Alt+En keyboard shortcuts

Filter query results — Showing results from 1 to 7 of 7. Query took 0.1s, minutes ago.

	Rollo_Hoja	Antecedente	Correferente
1	"677_0630"	"30/7/975 (1352, 1360)"	"30/7/975 (1648, 1656)"
2	"677_0630"	"ALVAREZ (2762, 2769)"	"ALVAREZ (2917, 2924)"
3	"677_0630"	"P.C (1732, 1735)"	"P.C (2179, 2182)"
4	"677_0630"	"P.C. (1011, 1015)"	"P.C. (1577, 1581)"
5	"677_0630"	"P.C. (1577, 1581)"	"Dirección del P.C. (1891, 1909)"
6	"677_0630"	"P.C. (1577, 1581)"	"P.C (1732, 1735)"

Figura 6.6: Correferencias entre entidades nombradas en la hoja 677_0630

6.4.3. Relaciones entre entidades nombradas

Para obtener todas las relaciones de la base de conocimiento y su *provenance* basta con consultar todas las ocurrencias de relaciones (*RelationOccurrence*) y la *provenance* de las ocurrencias de las entidades nombradas que participan en cada instancia de relación.

```
BASE <http://mh.uy/luz/resource/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX luz: <http://mh.uy/luz#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
select (concat(?n_roll, "_", ?n_sheet) as ?Rollo_Hoja) ?relation (?relation_label as ?Relacion)
(concat(?ent_label, " (", str(?begin), ", ", str(?end), ")") as ?Entidad)
(?ent_type as ?Clase)
where {
    ?relation luz:hasRelationOccurrence [ luz:hasEntityOccurrence ?ent_occ ;
                                          luz:hasOccurredIn ?sheet ] ;
              a/rdfs:label ?relation_label .
    ?ent_occ luz:hasOccurredIn ?sheet ;
             luz:occurrenceBegin ?begin ;
             luz:occurrenceEnd ?end ;
             ^luz:hasOccurrence [ a ?ent_type ;
                                  rdfs:label ?ent_label] .
    ?sheet luz:roll ?n_roll ;
           luz:sheet ?n_sheet .
}
order by ?relation
```

keyboard shortcuts

Filter query results

Showing results from 1 to 80 of 80. Query took 0.2s, minutes ago.

	Rollo_Hoja	relation	Relacion	Entidad	Clase
1	"643_0014"	http://mh.uy/luz/resource/relation/r643_i0014_REL1	"domicilio de una persona"	"MARIA CARMEN GOMEZ CIANO (632, 657)"	luz:Person
2	"643_0014"	http://mh.uy/luz/resource/relation/r643_i0014_REL1	"domicilio de una persona"	"Charrúa 1111 (703, 715)"	luz:Place
3	"643_0014"	http://mh.uy/luz/resource/relation/r643_i0014_REL2	"domicilio de una persona"	"RICARDO NELSON ARMANDO MIRITI (755, 784)"	luz:Person
4	"643_0014"	http://mh.uy/luz/resource/relation/r643_i0014_REL2	"domicilio de una persona"	"Manuel Haedo 1234.- (826, 845)"	luz:Place

Figura 6.7: Relaciones extraídas

Cabe destacar que cada instancia de relación en los resultados arrojados por la consulta está representada por dos filas que corresponden a las dos entidades nombradas envueltas en la relación. Habiendo dicho esto, se verifica el número de filas totales según la cantidad de relaciones que se mencionan en la evaluación de relaciones (6.1).

A continuación se pueden ver las consultas realizadas para cada tipo de relación que es reconocida por *LUZ* en el conjunto *ground truth*.

Es importante aclarar que la ubicación de las entidades nombradas en la hoja no se incluyen en los resultados simplificando así la consulta. Sin embargo, la recuperación de esta información es análoga a la de la consulta anterior.

Domicilio de una persona

```
BASE <http://mh.uy/luz/resource/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX luz: <http://mh.uy/luz#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX schema: <https://schema.org/>
select ?Relacion (concat(?roll, "_", ?sheet) as ?Rollo_Hoja) ?Nombre ?Domicilio
where {
    ?Relacion a luz:HomeLocation ;
              luz:hasDomiciled/schema:name ?Nombre ;
              luz:where/rdfs:label ?Domicilio ;
              luz:hasRelationOccurrence/luz:hasOccurredIn [ luz:roll ?roll ;
                                                            luz:sheet ?sheet ] .
    filter(?roll = "677" && ?sheet = "0575") .
}
order by ?Rollo_Hoja
```

keyboard shortcuts

Filter query results

Showing results from 1 to 1 of 1. Query took 0.1s, minutes ago.

	Relacion	Rollo_Hoja	Nombre	Domicilio
1	http://mh.uy/luz/resource/relation/r677_i0575_REL1	"677_0575"	"William"	"calle Regimiento 9 Nro. 123"

Figura 6.8: Relaciones *domiciliada en* en la hoja 677_0575

Persona integrante de organización

```
BASE <http://mh.uy/luz/resource/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX luz: <http://mh.uy/luz#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX schema: <https://schema.org/>
select ?Relacion (concat(?roll, "_", ?sheet) as ?Rollo_Hoja) ?Nombre ?Organizacion
where {
    ?Relacion a luz:Membership ;
              luz:hasRelationOccurrence/luz:hasOccurredIn [ luz:roll ?roll ;
                                                            luz:sheet ?sheet ] .
    optional { ?Relacion luz:hasMember/schema:name ?Nombre }
    optional { ?Relacion luz:membershipHasOrganization/schema:name ?Organizacion }
    filter(?roll = "643" && ?sheet = "0079") .
}
order by ?Rollo_Hoja
```

Press Alt+En keyboard shortcuts

Filter query results

Showing results from 1 to 1 of 1. Query took 0.1s, moments ago.

	Relacion	Rollo_Hoja	Nombre	Organizacion
1	http://mh.uy/luz/resource/relation/r643_i0079_REL1	"643_0079"	"Enrique Y@NTAS OB@@TI"	"U.J.C."

Figura 6.9: Relaciones *integrante de* o *integraba* en la hoja 643_0079

Detención de una persona

```
BASE <http://mh.uy/luz/resource/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX luz: <http://mh.uy/luz#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX schema: <https://schema.org/>
select ?Relacion (concat(?roll, "_", ?sheet) as ?Rollo_Hoja) ?Nombre ?Fecha ?Lugar
where {
    ?Relacion a luz:Arrest ;
              luz:hasRelationOccurrence/luz:hasOccurredIn [ luz:roll ?roll ;
                                                            luz:sheet ?sheet ] .
    optional { ?Relacion luz:hasArrested/schema:name ?Nombre }
    optional { ?Relacion luz:when/rdfs:label ?Fecha }
    optional { ?Relacion luz:where/rdfs:label ?Lugar }
}
order by ?Rollo_Hoja
```

Press Alt+En keyboard shortcuts

Filter query results — Showing results from 1 to 2 of 2. Query took 0.2s, minutes ago.

	Relacion	Rollo_Hoja	Nombre	Fecha	Lugar
1	http://mh.uy/luz/resource/relation/r643_i0029_REL1	"643_0029"	"Hector Arami BUSTAMANTES"	"9 de enero de 1975"	
2	http://mh.uy/luz/resource/relation/r677_i0913_REL3	"677_0913"	"CARME CASTILLARES"		"Facultad de Humanidades"

Figura 6.10: Relaciones *es detenida* y *detenida en*

Se puede notar que el primer resultado corresponde a la relación *es detenida* y el segundo a la relación *detenida en*.

En el anexo .4 se encuentran las consultas utilizadas.

Capítulo 7

Conclusiones

Este proyecto se enfocó en facilitar el análisis de documentos históricos para esclarecer eventos ocurridos en la última dictadura cívico-militar. En pos de esto se plantearon y siguieron estos dos objetivos: realizar tareas de extracción de información de los documentos transcriptos en *LUISA* y almacenar esta información de manera que luego sea posible crear una base de conocimiento navegable.

A lo largo de las etapas del proyecto se respetó la confidencialidad de la información contenida en los textos con los que se trabajó, evitando utilizar medios online para cualquier requerimiento que implique el uso de documentos enteros. Esto incluye procesamiento, distribución y almacenamiento.

Dado que los objetivos planteados son bastante amplios, se propuso como solución una plataforma configurable y que facilite la integración de nuevas y mejores herramientas.

El primer objetivo implicó estudiar distintos algoritmos de reconocimiento de entidades nombradas, extracción de relaciones y resolución de correferencias. Sin embargo al momento de construir una solución se hizo especial énfasis en el reconocimiento de entidades nombradas ya que resolver bien este problema construye cimientos fuertes para el resto de las tareas de extracción de información.

Para el reconocimiento de entidades nombradas se investigaron e integraron las siguientes herramientas: *FreeLing*, *NLTK*, *SpaCy* y *Stanza*. Además se implementaron dos reconocedores, uno a partir de patrones y otro a partir de diccionarios. Estos 6 reconocedores fueron evaluados y comparados según su capacidad de reconocer entidades nombradas así como de clasificarlas.

Como para este proyecto es importante la precisión del reconocimiento y clasificación de las entidades nombradas pero también se necesita extraer la mayor cantidad posible de información, en la comparación se puso foco en la *medida F* obtenida para cada reconocedor. De esto se desprendió que el reconocedor que más armonía alcanza entre *precisión* y *recuperación* para las categorías lugar, organización y persona es *Stanza* y para fechas es *FreeLing*. Los reconocedores implementados en base a patrones y diccionarios demostraron también tener un buen desempeño, en particular ser muy precisos, lo que se podría aprovechar ejecutándolos previo a *Stanza* y *FreeLing* según cada tipo de entidad. Además con algunas mejoras mencionadas en 6.1 podrían mejorar notablemente sus métricas. Cabe destacar que si bien se vieron afectadas la *precisión* y *recuperación* de los resultados debido a errores en las transcripciones de los textos, en paralelo a este proyecto se está trabajando en obtener transcripciones de mejor calidad con las que se podrían alcanzar mejores resultados.

La tarea de extracción de relaciones se llevó a cabo utilizando un enfoque basado en patrones identificados a partir del análisis de distintos textos del dominio. Al no contar con un conjunto de evaluación con anotaciones manuales de relaciones, se hizo un análisis manual notando una alta precisión en las relaciones anotadas ya que en general todos los resultados que se obtuvieron para el *ground truth* fueron acertados. A su vez, los resultados con errores observados en 6.3 no se deben a fallas en el algoritmo, sino que en general son errores arrastrados desde la etapa de NER o de las mismas transcripciones de los textos.

Además, se implementaron variantes con lematizaciones para los textos, las cuales arrojaron aún mejores resultados. Esto se deduce de que se encontraron al menos 6 relaciones más y no se perdió ninguna de las extraídas anteriormente. Cabe resaltar también que no se notaron grandes diferencias entre los lematizadores utilizados.

En cuanto a la resolución de correferencias, este proyecto se centró en las que se dan exclusivamente entre entidades nombradas proponiendo un algoritmo que busca desambiguar entidades nombradas encontradas en un mismo texto. En este caso también se hizo un análisis manual de donde se deduce que el algoritmo propuesto no es robusto en casos donde una entidad es abreviación de la otra, ya que, por ejemplo, en algunos casos hay iniciales de palabras que son omitidas al momento de revisar la posible abreviación.

Además, si bien el formato de cadena propuesto para la representación de las correferencias en algunos casos puede brindar referencias transitivas erróneas, en general aporta información valiosa y facilita el análisis de los textos procesados. Se entiende que como primera aproximación a la resolución de esta tarea sienta buenas bases para continuar trabajando.

Para lograr el segundo objetivo se implementó la generación de un grafo de conocimiento por cada texto procesado y además se definió una ontología propia para representar el conocimiento obtenido por *LUZ*. A partir de las consultas de evaluación realizadas se puede concluir que la ontología y los grafos generados satisfacen ampliamente las necesidades propuestas como son la sencilla recuperación de la información recolectada en etapas anteriores y la *provenance* de cada ocurrencia de entidad nombrada. Sin embargo, esta tiene algunos puntos débiles como, por ejemplo, el hecho de tener que modificar la ontología para representar nuevos tipos de relaciones, o la necesidad de implementar un desglose para cada formato de fecha reconocido para así representarla correctamente utilizando la ontología *OWL Time*.

Capítulo 8

Trabajo futuro

A continuación se describen algunas mejoras reconocidas para el sistema implementado y que quedaron por fuera del alcance del proyecto.

8.1. Entidades nombradas

Entrenar modelos de Stanza

Aunque se tiene una forma de entrenar modelos a través de la librería *SpaCy* como se explica en la sección 3.6, queda pendiente el entrenamiento de modelos de *Stanza* a través de *LUZ* y evaluar los resultados en contraste con los demás reconocedores de entidades nombradas investigados.

Construir un reconocedor mixto de entidades nombradas

Si bien *LUZ* ofrece muchos reconocedores de entidades nombradas distintos, como trabajo futuro se propone crear un clasificador mixto donde para cada categoría se use el reconocedor que tenga mejor desempeño para esa categoría.

Si se tomaran los que están integrados hasta el momento, el reconocedor mixto debería usar *FreeLing* para el reconocimiento de fechas y *Stanza* para reconocer lugares, organizaciones y personas.

Reconocer fechas relativas

Hasta el momento se reconocen fechas en distintos formatos como entidades nombradas, pero todas las que se pueden encontrar se conforman por día, mes

y/o año. Como trabajo futuro se podrían reconocer fechas relativas al día en el que se escribió el documento, por ejemplo si en alguno aparece una fecha de cuándo fue escrito y en alguna parte dice “en el día de ayer” o “el pasado martes” sería útil poder reconocer y descifrar esa fecha.

8.2. Relaciones

Utilizar distintos tipos de algoritmos

Para el reconocimiento de relaciones se utilizó un algoritmo de extracción basado en patrones, sin poder explorar otros distintos como los mencionados en la sección 2.1. Como posible trabajo futuro se podrían implementar otros algoritmos para evaluar y comparar resultados como se hizo para el reconocimiento de entidades nombradas.

Calcular métricas de la extracción de relaciones

Al igual que como se tiene para el reconocimiento de entidades nombradas sería de utilidad tener métricas para el reconocimiento de relaciones, de modo de poder evaluar qué tan precisa es la implementación realizada para este proyecto y también qué tanta recuperación tiene. Con esta información y suficientes textos anotados se podrían sacar más conclusiones sobre la solución implementada.

Evaluar lematizadores

Relacionado a lo anterior, sería bueno tener la posibilidad de evaluar los lematizadores implementados ya que permitiría una mejor observación del desempeño de los mismos.

Más tipos de relaciones

Se propone como trabajo futuro agregar nuevos tipos de relaciones que ayuden a extraer más información. Por ejemplo locación de organizaciones, parentezco entre personas, o secuestros.

Relaciones n-arias

El sistema implementado reconoce únicamente relaciones binarias. Como trabajo futuro se podría agregar la extracción de relaciones compuestas por más de 2 entidades. Para lograrlo se debería integrar un nuevo módulo a la etapa de extracción de relación y modificar la etapa de generación del grafo de conocimiento.

8.3. Correferencias

Modificar alcance

Hasta el momento la resolución de correferencias es únicamente a nivel de cada texto. Se propone como trabajo futuro que las cadenas de correferencias construidas abarquen la base de conocimiento completa para facilitar el cruzamiento de información entre textos.

Evaluación

Al igual que para la extracción de relaciones, sería bueno tener la posibilidad de evaluar el desempeño del algoritmo implementado para tomar mejores conclusiones.

8.4. Grafo de conocimiento

Representación de lugares

Facilitaría las consultas a la base de conocimiento y el análisis de la información extraída modificar la ontología construida para que represente las entidades de tipo lugar extendiendo una ontología de geolocalización.

8.5. Pipeline

Modificar el preprocesamiento

Como se explica en la sección 4.2, el motor de preprocesamiento recibe un texto y una lista de reglas a aplicar. Como trabajo futuro a la solución propuesta del pipeline, se podría agregar la posibilidad de que el usuario del sistema pueda modificar este conjunto de reglas dinámicamente, sin la necesidad de tener que liberar una nueva versión del sistema para modificarlas.

8.6. Ground truth

Nuevas anotaciones

Para poder evaluar y comparar algoritmos de resolución de correferencias y extracción de relaciones se necesitará construir un conjunto *ground truth* que cuente con anotaciones confiables de ambas cosas.

Bibliografía

[1] Steven Bird, Ewan Klein y Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. O'Reilly Media, Inc., 2009.

[2] Christian Bizer, Maria-Esther Vidal y Michael Weiss. "Resource Description Framework". En: *Encyclopedia of Database Systems*. Ed. por Ling Liu y M. Tamer Özsu. New York, NY: Springer New York, 2017, págs. 1-4. ISBN: 978-1-4899-7993-3. DOI: 10.1007/978-1-4899-7993-3_905-3. URL: https://doi.org/10.1007/978-1-4899-7993-3_905-3.

[3] Lucía Cantamutto et al. *Resolución de correferencias para la captura de eventos*. Universidad Nacional del Sur, Argentina y Universidad de Deusto-DeustoTech, España, 2015.

[4] Lautaro Cardozo, Lía Rivero y Guillermo Zorrón. *Ontología del sistema de búsqueda Luz*. URL: https://gitlab.fing.edu.uy/lia.rivero/proyecto-luz/-/raw/ontologia-luz/luz.ttl.

[5] Lautaro Cardozo et al. "Anotación automática sobre documentos del proyecto LUISA". En: 2019.

[6] Xavier Carreras et al. "FreeLing: An Open-Source Suite of Language Analyzers". En: *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC'04)*. 2004.

[7] *Clase DateTimeDescription en OWL Time*. URL: https://www.w3.org/TR/owl-time/#time:DateTimeDescription.

[8] *Clase Organization en schema.org*. URL: https://schema.org/Organization.

[9] *Clase Person en schema.org*. URL: https://schema.org/Person.

[10] *Clase Place en schema.org*. URL: https://schema.org/Place.

[11] *Cruzar*. URL: https://cruzar.uy/.

[12] *Dandelion API*. URL: https://dandelion.eu/.

[13] Lucía Gandioli. *A la justicia*. URL: https://sdr.fic.edu.uy/a-la-justicia/.

[14] *GraphDB™ Free Edition*. URL: https://www.ontotext.com/products/graphdb/graphdb-free/.

[15] Tom Gruber. "Ontology". En: *Encyclopedia of Database Systems*. Ed. por Ling Liu y M. Tamer Özsu. New York, NY: Springer New York, 2016, págs. 1-3. ISBN: 978-1-4899-7993-3. DOI: 10.1007/978-1-4899-7993-3_1318-2. URL: https://doi.org/10.1007/978-1-4899-7993-3_1318-2.

[16] Marti A. Hearst. "Automatic Acquisition of Hyponyms from Large Text Corpora". En: *COLING 1992 Volume 2: The 14th International Conference on Computational Linguistics*. 1992. URL: https://www.aclweb.org/anthology/C92-2082.

[17] Rachel Heery. "What Is RDF". En: 1998.

[18] Graeme John Hirst. "Anaphora in natural language understanding: A survey". Department of Engineering Physics, Research School of Physical Sciences, The Australian National University, 1979.

[19] Matthew Honnibal e Ines Montani. "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing". 2017.

[20] Daniel Jurafsky y James Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Vol. 2. Feb. de 2008.

[21] Jan-Christoph Klie et al. "The INCEpTION Platform: Machine-Assisted and Knowledge-Oriented Interactive Annotation". En: *Proceedings of the 27th International Conference on Computational Linguistics: System Demonstrations*. 2018. URL: https://aclanthology.org/C18-2002.

[22] G. Klyne y Jeremy J. Carroll. "Resource description framework (rdf) concepts and abstract syntax". En: 2003.

[23] *Librería RDFLib*. URL: https://github.com/RDFLib/rdflib.

[24] *LUISA*. URL: https://mh.udelar.edu.uy/luisa/.

[25] V. Tresp M. Nickel K. Murphy y E. Gabrilovich. *A Review of Relational Machine Learning for Knowledge Graph*. 2016.

[26] *Módulo sklearn.metrics.precision_recall_fscore_support*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html.

[27] Mike Mintz et al. "Distant supervision for relation extraction without labeled data". En: *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*. Suntec, Singapore: Association for Computational Linguistics, ago. de 2009, págs. 1003-1011. URL: https://www.aclweb.org/anthology/P09-1113.

[28] M.F. Moens. *Information Extraction: Algorithms and Prospects in a Retrieval Context*. The Information Retrieval Series. Springer Netherlands, 2006. ISBN: 9781402049934. URL: https://books.google.com.uy/books?id=t5oMg54hBxwC.

[29] McGuinness N. F. Noy. *A Guide to Creating Your First Ontology*. 2000.

[30] Hiroki Nakayama. *seqeval: A Python framework for sequence labeling evaluation*.
Software available from https://github.com/chakki-works/seqeval. 2018. URL: https://github.com/chakki-works/seqeval.

[31] *Programa analyze de Freeling*. URL: https://freeling-user-manual.readthedocs.io/en/v4.2/analyzer/#the-easy-way-using-the-analyze-script.

[32] Peng Qi et al. "Stanza: A Python Natural Language Processing Toolkit for Many Human Languages". En: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. 2020. URL: https://nlp.stanford.edu/pubs/qi2020stanza.pdf.

[33] Karthik Raghunathan et al. *A Multi-Pass Sieve for Coreference Resolution*. Computer Science Department, Stanford University, 2010.

[34] Álvaro Rico (Coord.) *Investigación histórica sobre la dictadura y el terrorismo de estado en Uruguay (1973-1985)*. Centro de Estudios Interdisciplinarios Uruguayos, Universidad de la República, Uruguay, 2008.

[35] Emili Sapena, Lluís Padró y Jordi Turmo. *RelaxCor: An Open Source Coreference Resolution System*. TALP Research Center, Universitat Politecnica de Catalunya, 2010.

[36] Emili Sapena, Lluís Padró y Jordi Turmo. *RelaxCor: Coreference Resolution system*. URL: http://nlp.lsi.upc.edu/relaxcor/.

[37] *Spanish pipeline optimized for CPU*. URL: https://spacy.io/models/es#es_core_news_md.

[38] Erik F. Tjong Kim Sang y Fien De Meulder. "Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition". En: *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*. 2003, págs. 142-147. URL: https://aclanthology.org/W03-0419.

ANEXOS

.1. Expresiones regulares

A continuación se muestra la lista de patrones implementados para el reconocedor basado en expresiones regulares.

```
{
"rules": [
  {
    "ref": "Nombres de calle",
    "label": "LOC",
    "regexp": "((?:calle|Calle|CALLE|bulevar|Bulevar|
       BULEVAR|boulevar|Boulevar|BOULEVAR|avenida|Avenida
       |AVENIDA|camino|Camino|CAMINO(?:av|Av|AV|avda|Avda
       |bv|Bv|BV|cno|Cno|CNO)(?:\\.?))(?:.*?)(?:(?:(?:nro
       |Nro|NRO|no|No|NO)(?:\\.?)){0,1} \\d+(?:,? (?:apto
       |Apto|APTO|ap|Ap|AP|hab|Hab|HAB)(?:\\.?) \\d+){0,1
       }|| S\\/N|,? (?:apto|Apto|APTO|ap|Ap|AP|hab|Hab|
       HAB)(?:\\.?) \\d+))(?: [a-z]{2,}|,|\\.)"
  },
  {
    "ref": "Domiciliadx en con nro",
    "label": "LOC",
    "regexp": "(?:d|D)omiciliad(?:a|o) en (?:el |la ){0,1
       }(.*?(?:\\d+(?:(?: apto| Apto| APTO| ap| Ap| AP|
       hab| Hab| HAB)(?:\\.?) \\d+){0,1}| S\\/N|(?: apto|
        ap| hab)(?:\\.?) \\d+))"
  },
  {
    "ref": "Domiciliadx en sin nro",
    "label": "LOC",
    "regexp": "(?:d|D)omiciliad(?:a|o) en (?:el |la ){0,1
       }(.*?)(?:\\.-| [a-z]{2,} |,)"
  },
  {
    "ref": "Finca",
    "label": "LOC",
    "regexp": "((?:f|F)inca (?:(?:de .*? (?:\\d+))|(?:(?:
       no|No)\\.? \\d+)) (?:(?:apto|Apto|APTO|ap|Ap|AP|
```

```
      hab|Hab|HAB)(?:\\.?) \\d+)?)"
  },
  {
    "ref": "Escuela o Liceo",
    "label": "LOC",
    "regexp": "(?:(?:e|E)scuela|(?:l|L)iceo) (?:(?:(?:(?:
      no|No|nro|Nro)\\.? )?\\d+)|[A-Z][A-Za-zñÑ]+)[^A-Za
      -zñÑ]"
  },
  {
    "ref": "Nombres APELLIDOS",
    "label": "PER",
    "regexp": "(?:|[^A-Za-zñÑ])((?:[A-ZÑ][a-zñ]+ )+(?:(?:
      de ){0,1}[A-ZÑ]{2,} ?)+)[^A-Za-zñÑ]"
  },
  {
    "ref": "APELLIDOS Nombres",
    "label": "PER",
    "regexp": "(?:|[^A-Za-zñÑ])([A-ZÑ]{2,}(?: [A-ZÑ]{2,})
      *(?: de [A-ZÑ]+){0,1},?(?: [A-ZÑ][a-zñ]+)+)[^A-Za-
      zñÑ]"
  },
  {
    "ref": "DDHHMM<MES>YYY",
    "label": "DATE",
    "regexp": "(\\d{6}[A-Z]{3}\\d{3})"
  },
  {
    "ref": "DD MM YYYY",
    "label": "DATE",
    "regexp": "((?:3[01]|[12][0-9]|0?[1-9])(?:[\\-/.])(?:
      0?[1-9]|1[1-2])(?:[\\-/.])\\d{3,4})"
  },
  {
    "ref": "Día de Mes de Año",
    "label": "DATE",
    "regexp": "((?:3[01]|[12][0-9]|0?[1-9]) de (?:Enero|
      Febrero|Marzo|Abril|Mayo|Junio|Julio|Agosto|Se(?:p
```

```
        ?)tiembre|Octubre|Noviembre|Diciembre) (?:de(?:l)
        ?|del año) \\d{2,4})",
52      "caseSensitive": false
53    },
54    {
55      "ref": "Año XX (o XXX, o XXXX)",
56      "label": "DATE",
57      "regexp": "(?<![A-Za-z])(año \\d{2,4})",
58      "caseSensitive": false
59    },
60    {
61      "ref": "Requerido Numero",
62      "label": "REQ",
63      "regexp": "\\((Req(?:\\.{0,1}) Nro(?:\\.{0,1}) \\d+)
          \\)"
64    }
65
66  ]
67  }
```

.2. Instalación de INCEpTION

A continuación se muestra la guía desarrollada para la instalación y configuración de *INCEpTION*.

.2.1. Descargar INCEpTION

Se puede descargar *INCEpTION* en su versión 0.11.2 desde este enlace.

.2.2. Ejecutar INCEpTION

Para ejecutar esta herramienta es necesario contar con Java en su versión 8 o posterior instalado en el sistema.

El servidor se levanta en el puerto 8080 con el comando $java\ -jar\ inception-app-standalone-0.12.2.jar$.

Las credenciales por defecto son:

- Usuario *admin*

- Contraseña *admin*

.2.3. Importar el proyecto pre configurado

1. Click en *Projects* en la barra arriba a la izquierda.
2. Click en *Browse*.
3. Seleccionar el archivo comprimido después de descargarlo de https://drive.google.com/file/d/16LpNnusP-H_-5zxXXqzYVYgZWIz77w3b7/view?usp=sharing.
4. Click en *Upload*.

.2.4. Creación de usuarios

Sólo un administrador puede crear usuarios siguiendo los siguientes pasos:

1. Click en el botón *Administración* en la barra arriba a la derecha.
2. Click en *Users* en la sección de la izquierda.
3. Click en *Create*.
4. Completar los datos requeridos seleccionando *ROLE_USER*.
5. Activar la cuenta con el checkbox.
6. Click en *Save*.

.2.5. Agregar usuarios al proyecto

Sólo un administrador puede agregar usuarios a un proyecto siguiendo los siguientes pasos:

1. Click en el botón *Administración* en la barra arriba a la derecha.
2. Click en *Projects* en la sección de la izquierda.
3. Seleccionar el proyecto.
4. Ir a la pestaña *Users*.
5. Buscar el usuario que se quiera agregar.
6. Configurar los permisos de *Annotator*.

.3. Instructivo de anotaciones con INCEpTION

A continuación se muestra la guía desarrollada para las anotaciones y curaciones manuales con INCEpTION que se utilizaron para obtener el conjunto *ground truth* utilizado para la evaluación del reconocimiento de entidades nombradas de *LUZ*.

.3.1. Empezar a anotar

Al loguearse en INCEpTION se debe seleccionar el proyecto que se quiere trabajar, hacer click en **Annotation** y luego elegir un texto que se muestre con letras negras.

.3.1.1. Generar una anotación

Una vez en el texto, elegir **Named Entity** antes de hacer anotación alguna. Luego de esto, seleccionar los *tokens* en el texto para generar una anotación y finalmente elegir la categoría en el campo **value**.

.3.1.2. Borrar anotación

Si es necesario borrar una anotación errónea, simplemente se selecciona y se hace click en **Delete** en el panel de la derecha.

Si una entidad anotada cuenta con más o menos *tokens* de los que debería se tiene que borrar la anotación y volver a hacerla como corresponde. Sin embargo, **cuando una entidad está etiquetada con una categoría incorrecta** no es necesario borrar la anotación y hacerla de nuevo, se la puede seleccionar y cambiar la etiqueta en el panel de la derecha.

.3.1.3. Confirmar

Una vez terminada la anotación del texto, click en **Finish document** (ícono de candado). Ten en cuenta que sólo un administrador puede deshacer esta acción.

.3.2. Información necesaria antes de anotar

- La **granularidad** de las anotaciones es a nivel de *token* y no de caracter. Esto significa que una anotación contendrá un *token* enteramente y no solo una parte de él.
- Es posible que una anotación esté compuesta por *tokens* de sentencias consecutivas.
- Los *tokens* de una anotación **deben** ser consecutivos.

Etiquetas: Entre la capas de las que se disponen se encuentra **Named Entity**. Esta capa a su vez se puede clasificar en varias categorías entre las que se encuentran:

- PER
- LOC
- ORG
- DATE

Se pide que se utilicen estas etiquetas para anotar **entidades seguras** que son aquellas que se ven claramente que son:

- nombre de una persona,
- nombre de un lugar,
- nombre de una organización (o sigla),
- y fecha propiamente dicha

No se anotarán:

- personas, lugares u organizaciones que no dispongan del nombre de los mismos;
- fechas implícitas como *día de ayer*, *últimos 40 años*, etc.

.4. Consultas SPARQL

A continuación se muestran las consultas utilizadas para evaluar la recuperación de información de los grafos de conocimiento generados con *LUZ*.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX luz: <http://mh.uy/luz#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
select (?type as ?Clase) (count(?res) AS ?Cantidad) {
    luz:NamedEntity owl:equivalentClass/owl:unionOf/rdf:rest*/rdf:first ?type .
    ?res a ?type .
}
group by ?type
```

Figura 1: Total de entidades nombradas según su clasificación

```
BASE <http://mh.uy/luz/resource/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX luz: <http://mh.uy/luz#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
select distinct (concat(?roll, "_", ?sheet) as ?Rollo_Hoja) ?entity ?entity_label
    ↪ (?begin as ?Comienzo) (?end as ?Final)
where {
    ?entity a/^(owl:equivalentClass/owl:unionOf/rdf:rest*/rdf:first) luz:
        ↪ NamedEntity ;
           rdfs:label ?entity_label ;
           luz:hasOccurrence ?occ .
    ?occ luz:occurrenceBegin ?begin ;
         luz:occurrenceEnd ?end ;
         luz:hasOccurredIn [ luz:roll ?roll ;
                             luz:sheet ?sheet ] .
}
```

Figura 2: Ocurrencias de entidades nombradas

```
BASE <http://mh.uy/luz/resource/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX luz: <http://mh.uy/luz#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
select (concat(?n_roll, "_", ?n_sheet) as ?Rollo_Hoja)
(concat(?ent1_label, " (", str(?ent1_begin), ", ", str(?ent1_end), ")") as ?
    ↪ Antecedente)
(concat(?ent2_label, " (", str(?ent2_begin), ", ", str(?ent2_end), ")") as ?
    ↪ Correferente)
where {
    ?ent1 rdfs:label ?ent1_label ;
          luz:hasOccurrence [ luz:hasOccurredIn ?sheet ;
                              luz:hasCoreference [ luz:hasOccurredIn ?sheet ;
                                                   ^luz:hasOccurrence/rdfs:label ?
                                                       ↪ ent2_label ;
                                                   luz:occurrenceBegin ?ent2_begin ;
                                                luz:occurrenceEnd ?ent2_end ] ;
                    luz:occurrenceBegin ?ent1_begin ;
                              luz:occurrenceEnd ?ent1_end ] .
    ?sheet luz:roll ?n_roll ;
           luz:sheet ?n_sheet .
}
order by ?sheet
```

Figura 3: Correferencias entre entidades nombradas

```
BASE <http://mh.uy/luz/resource/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX luz: <http://mh.uy/luz#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
select (concat(?n_roll, "_", ?n_sheet) as ?Rollo_Hoja) ?relation (?relation_label
    ↪ as ?Relacion)
(concat(?ent_label, " (", str(?begin), ", ", str(?end), ")") as ?Entidad)
(?ent_type as ?Clase)
where {
    ?relation luz:hasRelationOccurrence [ luz:hasEntityOccurrence ?ent_occ ;
                                          luz:hasOccurredIn ?sheet ] ;
              a/rdfs:label ?relation_label .
    ?ent_occ luz:hasOccurredIn ?sheet ;
          luz:occurrenceBegin ?begin ;
            luz:occurrenceEnd ?end ;
            ^luz:hasOccurrence [ a ?ent_type ;
                       rdfs:label ?ent_label] .
    ?sheet luz:roll ?n_roll ;
          luz:sheet ?n_sheet .
}
order by ?relation
```

Figura 4: Relaciones entre entidades nombradas

```
BASE <http://mh.uy/luz/resource/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX luz: <http://mh.uy/luz#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX schema: <https://schema.org/>
select ?Relacion (concat(?roll, "_", ?sheet) as ?Rollo_Hoja) ?Nombre ?Domicilio
where {
    ?Relacion a luz:HomeLocation ;
              luz:hasDomiciled/schema:name ?Nombre ;
              luz:where/rdfs:label ?Domicilio ;
              luz:hasRelationOccurrence/luz:hasOccurredIn [ luz:roll ?roll ;
                                                            luz:sheet ?sheet ] .
}
order by ?Rollo_Hoja
```

Figura 5: Instancias de relación *domiciliada en*

```
BASE <http://mh.uy/luz/resource/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX luz: <http://mh.uy/luz#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX schema: <https://schema.org/>
select ?Relacion (concat(?roll, "_", ?sheet) as ?Rollo_Hoja) ?Nombre ?Organizacion
where {
    ?Relacion a luz:Membership ;
              luz:hasRelationOccurrence/luz:hasOccurredIn [ luz:roll ?roll ;
                                                            luz:sheet ?sheet ] .
    optional { ?Relacion luz:hasMember/schema:name ?Nombre }
    optional { ?Relacion luz:membershipHasOrganization/schema:name ?Organizacion }
}
order by ?Rollo_Hoja
```

Figura 6: Instancias de relación *integrante de* e *integraba*

```
BASE <http://mh.uy/luz/resource/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX luz: <http://mh.uy/luz#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX schema: <https://schema.org/>
select ?Relacion (concat(?roll, "_", ?sheet) as ?Rollo_Hoja) ?Nombre ?Fecha ?Lugar
where {
    ?Relacion a luz:Arrest ;
              luz:hasRelationOccurrence/luz:hasOccurredIn [ luz:roll ?roll ;
                                                            luz:sheet ?sheet ] .
    optional { ?Relacion luz:hasArrested/schema:name ?Nombre }
    optional { ?Relacion luz:when/rdfs:label ?Fecha }
    optional { ?Relacion luz:where/rdfs:label ?Lugar }
}
order by ?Rollo_Hoja
```

Figura 7: Instancias de relación *detenida en* y *es detenida*

```
BASE <http://mh.uy/luz/resource/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX luz: <http://mh.uy/luz#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX schema: <https://schema.org/>
select ?Relacion (concat(?roll, "_", ?sheet) as ?Rollo_Hoja) ?Nombre ?Organizacion
where {
    ?Relacion a luz:Release ;
             luz:hasRelationOccurrence/luz:hasOccurredIn [ luz:roll ?roll ;
                                                           luz:sheet ?sheet ] .
    optional { ?Relacion luz:hasMember/schema:name ?Nombre }
    optional { ?Relacion luz:membershipHasOrganization/schema:name ?Organizacion }
}
order by ?Rollo_Hoja
```

Figura 8: Instancias de relación *liberada*

```
BASE <http://mh.uy/luz/resource/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX luz: <http://mh.uy/luz#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX schema: <https://schema.org/>
select ?Relacion (concat(?roll, "_", ?sheet) as ?Rollo_Hoja) ?Nombre ?Organizacion
where {
    ?Relacion a luz:Disposal ;
             luz:hasRelationOccurrence/luz:hasOccurredIn [ luz:roll ?roll ;
                                                           luz:sheet ?sheet ] .
    optional { ?Relacion luz:hasPersonAtTheDisposal/schema:name ?Nombre }
    optional { ?Relacion luz:theDisposalOf/schema:name ?Organizacion }
}
order by ?Rollo_Hoja
```

Figura 9: Instancias de relación *a disposición de*

```
BASE <http://mh.uy/luz/resource/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX luz: <http://mh.uy/luz#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX time: <http://www.w3.org/2006/time#>
select ?date ?label ?day ?month ?year ?hour ?minute ?second ?dayOfWeek ?dayOfYear
    ↪ ?monthOfYear ?unitType ?timeZone ?week
where {
    ?date a luz:Date ;
    optional { ?date rdfs:label ?label }
    optional { ?date time:day ?day }
    optional { ?date time:month ?month }
    optional { ?date time:year ?year }
    optional { ?date time:hour ?hour }
    optional { ?date time:minute ?minute }
    optional { ?date time:second ?second }
    optional { ?date time:dayOfWeek ?dayOfWeek }
    optional { ?date time:dayOfYear ?dayOfYear }
    optional { ?date time:monthOfYear ?monthOfYear }
    optional { ?date time:unitType ?unitType }
    optional { ?date time:timeZone ?timeZone }
    optional { ?date time:week ?week }
}
order by ?year ?month ?day ?hour ?minute ?second
```

Figura 10: Fechas reconocidas ordernadas de manera creciente