Universidad de la República

Facultad de Ingeniería

Extracción automática de pistas para la generación de crucigramas a partir de textos en inglés

Informe de Proyecto de Grado presentado por

Arturo Collazo, Santiago Berruti y Diego Sellanes

como requisito de graduación de la carrera de Ingeniería en Computación de Facultad de Ingeniería de la Universidad de la República

Supervisores

Aiala Rosá
Luis Chiruzzo

Montevideo, 29 de julio de 2023

Agradecimientos

Agradecemos a nuestras familias y amigos que siempre estuvieron apoyándonos en todo momento, en los buenos y aún más en los no tan buenos. Siempre a nuestro lado incondicionalmente, este trabajo es gracias a ustedes.

Un grato agradecimiento a todas las personas del equipo de etiquetado que formaron parte: Santiago Góngora, Laura Musto, Sofía Sánchez, Ignacio Sastre, Luciana Agosto y Paola Martinez. También un gran agradecimiento a Juan Pablo Filevich por ayudar con la integración del sistema. Sin todos ustedes no hubiese sido posible concretar gran parte del trabajo presentado.

Finalmente, un especial agradecimiento a nuestros tutores Aiala Rosá y Luis Chiruzzo, nos han guiado y han participado con mucho entusiasmo y dedicación constantemente durante todo el transcurso del proyecto. Ha sido un placer para nosotros poder trabajar en conjunto con ustedes.

Resumen

En este proyecto se plantea el diseño y la implementación de un sistema de generación de crucigramas de forma automática a partir de textos para niños en inglés. Se trabaja sobre la existente aplicación del proyecto CINACINA de la Universidad de la República, el cual consiste en una herramienta utilizada para la enseñanza de inglés en diversos centros educativos del Uruguay.

La aplicación incluye varias aplicaciones lúdicas para la enseñanza de inglés, una de las cuales permite generar crucigramas. Previamente, los crucigramas se generaban en forma dinámica a partir de conjuntos de pistas de una base estática. Con la mejora presentada, los crucigramas son generados de forma dinámica y variada, tomando como base un texto de entrada. Es decir, el sistema propuesto es capaz de extraer de forma automática, a partir de un texto ingresado en inglés, pares de pistas y definiciones relacionadas al texto y con esto generar un crucigrama completo.

En particular, se desarrolla el módulo de extracción de pares $\prec$ *definiendum: definición* $\succ$ de posibles pistas para la generación del crucigrama entero. La implementación del módulo se basa en un corpus de textos no etiquetados, extraídos a partir del sitio web "ReadWorks", consistente en 400 textos en inglés para niños, desde los cuales se obtienen los pares relevantes para crucigramas.

La implementación propuesta utiliza diversas herramientas de procesamiento del lenguaje natural, donde distintos métodos se aplican de manera secuencial con el fin de obtener los pares. Cada método está basado en reglas y/o patrones comunes encontrados en los textos de entrada. Adicionalmente, se implementa un clasificador con un enfoque neuronal, capaz de clasificar las pistas generadas por el módulo anterior como buenas o malas. Este modelo es entrenado a partir de un corpus etiquetado manualmente por integrantes del proyecto e interesados en el mismo. Finalmente, se trabaja en la creación de una API para así soportar la integración con la aplicación ya existente sobre los crucigramas.

Respecto a los resultados obtenidos, por cada texto del corpus se generan en promedio 6 pistas, donde el módulo tiene una accuracy del 72 % de las pistas generadas. Estos resultados superan el método definido como línea base. Además, el clasificador implementado obtiene un 84 % de accuracy y un 78 % de F1, superando ampliamente el clasificador definido como linea base. Todos los objetivos son satisfechos por el proyecto, agregando el nuevo módulo al sistema.

Palabras clave: Generación de crucigramas, Extracción de pistas, Procesamiento del lenguaje natural, Clasificación de texto

Índice general

Capítulo 1

Introducción

Aprender inglés es una habilidad fundamental en la realidad actual, donde abre muchas puertas en términos de educación, comunicación y conocimiento. En general, lo mejor es que este aprendizaje ocurra en edades tempranas, particularmente en la escuela. Si bien en muchos centros uruguayos la enseñanza de inglés ya se encuentra normalizada, no es así para muchos otros casos.

Es por esto que el programa "Inglés sin Límites" de la Dirección De Políticas Lingüísticas de la ANEP, focaliza sus esfuerzos en mejorar y proveer una enseñanza de inglés para las escuelas del Uruguay donde los recursos son escasos o no tienen actualmente un plan para la enseñanza de inglés. En estos centros educativos, la mayoría de las veces los recursos no son suficientes para dar una educación de inglés completa y variada. En muchos centros, por ejemplo los rurales, el acceso a Internet es muy escaso, lo que imposibilita usar otros métodos de enseñanza como las videoconferencias. La enseñanza de ingles día a día es cada vez más importante en las nuevas generaciones, donde parece indispensable aprender este idioma. Hoy en día, la realidad es que no todas las escuelas pueden proveer este servicio, y es por esto que en este proyecto se busca contribuir con el desarrollo del proyecto "Capacitación en Inglés en el Aula con Inteligencia Artificial" (CINACINA).

CINACINA es un proyecto que combina investigación, extensión y enseñanza, desarrollado por docentes de la Facultad de Ingeniería y de la Facultad de Información y Comunicación de la Udelar. Cuenta con una plataforma web con diferentes juegos y actividades que sirven de apoyo para la enseñanza del inglés, en especial de alumnos de tempranas edades.

En este proyecto de grado, es de interés uno de los módulos de la plataforma: el generador de crucigramas. Estos crucigramas cuenta con diferentes categorías, tamaños de tableros, y una lista de definiciones en inglés que corresponden a palabras que hay que llenar en el tablero. El objetivo es que los estudiantes completen el crucigrama descifrando las palabras a las que corresponden estas definiciones.

Este crucigrama es muy limitado ya que solo se pueden seleccionar ciertas categorías como colores, animales, etc. Además, estas definiciones fueron extraídas con métodos basados en reglas simples, otras muchas generadas manualmente, y aunque sean de un nivel de inglés simple, es posible que si el estudiante esta recién comenzando a aprender el idioma no cuente con el contexto necesario como para conocer la palabra o entender la definición.

De esta problemática surge la idea de crear una herramienta que genere pares de palabras y definiciones que puedan ser utilizadas en un crucigrama a partir de un texto, por ejemplo, un cuento. De esta forma, un

docente podría entregar a sus alumnos un cuento para que lean, ingresar el cuento en esta herramienta y generar un crucigrama de forma automática utilizando palabras y definiciones extraídas del propio texto. Los alumnos pueden luego completar el crucigrama utilizando el cuento como referencia.

Es importante que las definiciones sean extraídas del propio texto, ya que como mencionamos antes, si se extraen las definiciones de un diccionario, pueden contener palabras que los alumnos nunca hayan leído o que no sean aptas para su edad. Podemos ver un ejemplo con el cuento de Caperucita Roja, se podría extraer una definición para el lobo como *"A wild carnivorous mammal of the dog family, living and hunting in packs"*, esta definición puede contener palabras muy complicadas para un niño que recién esta comenzando a aprender inglés. Pero por otro lado, se podría llegar a extraer una definición para lobo como *"Animal that pretends to be Little Red Riding Hood's grandmother so that he can eat her"*. El objetivo de esta herramienta sería obtener una definición de este estilo, donde la mayoría de palabras son extraídas del propio texto, y la definición tiene sentido dentro del mismo. Los niños pueden leer el cuento y encontrar que esa definición corresponde al lobo y de esta manera llenar el crucigrama. Es decir, la complejidad de las palabras utilizadas y redacción debe ser baja y deben ser autocontenidas en los textos.

Es importante destacar que crear crucigramas de forma manual es una tarea difícil y puede ocupar mucho tiempo para un docente. También puede ocurrir que el docente que presente el cuento a los niños no tenga tanto conocimiento de inglés como para formar el crucigrama. Por lo que tener una herramienta que reconozca posibles palabras para el crucigrama, forme sus definiciones y arme el crucigrama de forma rápida y automática puede tener muchas ventajas.

1.1. Planteo del problema

Particularmente en este proyecto, se trabaja en la generación automática de crucigramas temáticos a partir de un cuento. Estos crucigramas son herramientas didácticas de interés que permiten evaluar conceptos clave del idioma, tales como el dominio del vocabulario, del lenguaje y la comprensión lectora. La generación de los crucigramas implica invertir tiempo en la lectura del texto para encontrar palabras clave o pistas y sus definiciones para los lectores. Es por ello que, a través del uso de herramientas de procesamiento del lenguaje natural, se reduce significativamente el tiempo dedicado a generar crucigramas como tareas lúdicas de enseñanza. En consecuencia, el problema que se resuelve es la extracción de pistas o pares $\prec$*definiendum: definición*$\succ$ de los textos. Llamamos definiendum al término que buscamos definir, palabra que será incluida en el crucigrama, y definición a la pista brindada en el crucigrama a los estudiantes a partir del texto, es decir, la oración que define a su respectivo definiendum.

Este problema, que se enmarca dentro del procesamiento del lenguaje natural, forma parte de una tarea más general: la extracción de información. Esta tarea consiste en tomar un texto de entrada y devolver en un formato preestablecido una serie de datos estructurados. Luego, estos datos se pueden utilizar en diversos casos de uso, como almacenarlos en una base de datos, utilizarlos en un sistema de recomendación, generar un corpus, o como en este caso, generar un crucigrama.

En cuanto a la obtención de definiendums variados, no solo se centran en objetos o entidades en particular, sino también en verbos y patrones encontrados en las oraciones. Además, se busca que las pistas generadas combinen información distribuida a lo largo del texto para generar pistas interesantes y enriquecedoras que

prueben la comprensión lectora de los niños.

1.2. Resolución del problema

En esta sección, presentamos una explicación simplificada de nuestro objetivo y cómo se visualizaría una posible solución al problema. Concretamente, se procura construir una aplicación que permita a un docente introducir un texto en inglés y generar automáticamente un crucigrama a partir de dicho texto. Por ejemplo, supongamos que un docente presenta el siguiente texto a sus alumnos:

What country is just south of the United States? It's Mexico! Mexico is part of the continent North America. Mexico is shaped like a hook with a wide top. Like the U.S., it has water on two sides. On its west side is the Pacific Ocean. On its east side is the Gulf of Mexico.The bottom end of the hook curves north. This part of Mexico is called the Yucatan Peninsula. A peninsula is a piece of land that has water on most sides. The Yucatan Peninsula has the Gulf of Mexico on its west and north sides. It has the Caribbean Sea on its east side.

El docente ingresará el texto en esta aplicación desarrollada, la cual analizará automáticamente el texto y extraerá información relevante para crear las pistas del crucigrama. Por ejemplo:

1. A partir de la oración "Mexico is part of the continent North America" puede concluir que una defincón de "Mexico" es "Part of the continent North America" ya que el verbo "is" suele especificar la naturaleza o características de algo. Aunque tal vez luego al tomar la oración "Mexico is shaped like a hook with a wide top" forme otra pista para Mexico de la forma "Shaped like a hook with a wide top", por lo que podría decidir unir estas dos pistas y crear una sola definición para "Mexico" de la forma "Part of the continent North America that is shaped like a hook with a wide top".

2. Al encontrar la oración "On its west side is the Pacific Ocean", la aplicación podría entender que se refiere a "Mexico". Una vez que la aplicación entienda mejor el contexto de la oración podria formar una defincón para "Pacific Ocean" como "Something found on the west side of Mexico". Sin embargo en este crucigrama no queremos pistas que definan un concepto de múltiples palabras, por lo que la aplicación deberá elegir si definir "Pacific" u "Ocean". Tal vez decida definir "Pacific" y crear una pista del tipo "(_____ Ocean): Something found on the west side of Mexico", sugiriendo que la palabra en el crucigrama completa el espacio _____.

3. A partir de las oraciones "This part of Mexico is called the Yucatan Peninsula" y "A peninsula is a piece of land that has water on most sides", la aplicación podría interpretar que la definición de "peninsula" también aplica a "Yucatan Peninsula" y crear una definición para "Yucatan" como "(_____ Peninsula): A piece of land that has water on most sides".

4. Analizando las oraciones "The Yucatan Peninsula has the Gulf of Mexico on its west and north sides" e "It has the Caribbean Sea on its east side", la aplicación podría entender que el pronombre "It" refiere a la "Yucatan Peninsula" y generar una definición para "Sea" como "(Caribbean _____): Something located on the east side of the Yucatan Peninsula".

La aplicación también podría aprovechar información adicional que no se encuentre de forma explícita en el texto y utilizar sinónimos para mejorar las definiciones. Por ejemplo, en lugar de "Something" en la definición

de “Sea”, podría usar “Body of Water” para formar una definición más completa como “(Caribbean _____): Body of Water located on the east side of the Yucatan Peninsula”.

Con base en el análisis, la aplicación generaría los siguientes pares de palabras y pistas que podemos visualizar en el formato ≺ *definiendum* - *definición* ≻ antes mencionado:

1. ≺ ***Mexico*** - *Part of the continent North America that is shaped like a hook with a wide top* ≻
2. ≺ ***Pacific*** - *(_____ Ocean): Something found on the west side of Mexico* ≻
3. ≺ ***Yucatan*** - *(_____ Peninsula): A piece of land that has water on most sides* ≻
4. ≺ ***Sea*** - *(Caribbean _____): Body of Water located on the east side of the Yucatan Peninsula* ≻

Una vez obtenidos los pares de palabras y pistas, estos se ingresarán en un módulo de creación de crucigramas y se generará una interfaz gráfica para resolverlo. A modo ilustrativo, en la figura 1.1 se ve el formato de un posible crucigrama completado.

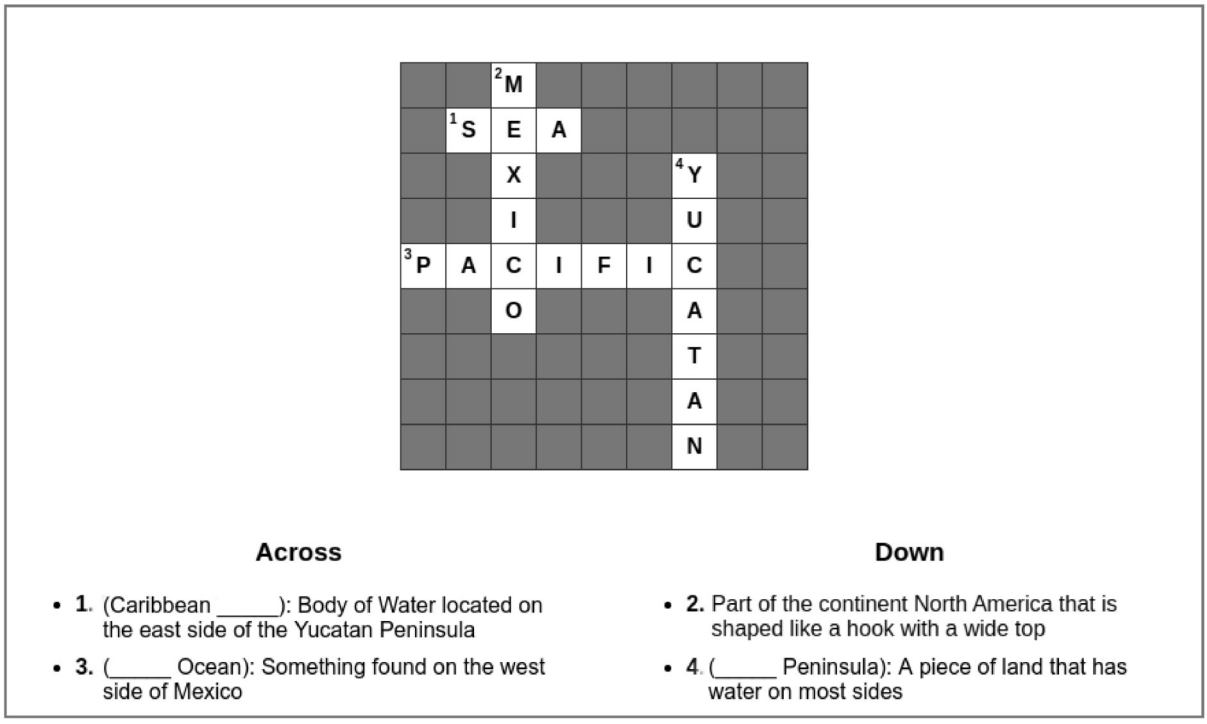

Figura 1.1: Ejemplo de crucigrama para el cuento *Where is Mexico?*

En resumen, se busca desarrollar una aplicación en la cual se pueda introducir un texto en inglés y mediante el análisis del contexto, los pronombres, el tipo de oraciones que forman una definición, la unión de diferentes oraciones y la creación de definiciones más completas, se genere un crucigrama atractivo y educativo. Posteriormente, este crucigrama se integrará en una plataforma existente para que los alumnos puedan resolverlo.

1.3. Objetivos planteados

El objetivo del trabajo presentado es el desarrollo de una herramienta que permita la extracción de pares ≺ *definiendum: definición* ≻ a partir de textos para niños en inglés, que permita generar de forma automática

crucigramas. Aún más, se busca integrar esta funcionalidad al sistema en actual uso antes mencionado (CINA-CINA). En particular, se pueden enumerar los siguientes objetivos específicos:

1. Desarrollar el módulo de extracción de pistas.
2. Generar un corpus de pistas a partir de los textos utilizados como referencia para el desarrollo del módulo.
3. Evaluar el desempeño del módulo generado.
4. Implementar un clasificador basado en aprendizaje automático y aprendizaje profundo que permita evaluar y mejorar la calidad de las pistas generadas.
5. Integrar la versión final de la implementación en el sistema ya existente.

1.4. Resultados esperados

Se procura alcanzar todos los objetivos planteados previamente, es decir:

1. Resumir el estado del arte.
2. Implementar un módulo funcional que permita generar crucigramas de forma automática.
3. Integrar el módulo desarrollado al sistema.
4. Generar un corpus que pueda ser usado para la generación de pistas con aprendizaje automático.

1.5. Organización del documento

En la siguiente sección se brinda una guía acerca de la organización del informe del proyecto. El documento está dividido en los siguientes capítulos:

- En el Capítulo 1 se presenta la introducción al proyecto.
- El Capítulo 2 consiste de seis secciones:
  - En la primera sección se desarrollan conceptos básicos necesarios para el entendimiento del proyecto.
  - En la segunda y tercera sección se brinda un resumen del estado del arte en tareas similares, estudios relacionados y proyectos de grado anteriores.
  - En la cuarta sección, se aborda en detalle tanto las herramientas de procesamiento de lenguaje natural utilizadas, como las tecnologías elegidas.
  - Finalmente, se comenta brevemente el corpus de datos utilizado.
- En el Capítulo 3 se presenta el flujo de la implementación, desde el paso inicial de extracción y formación de las definiciones, el desarrollo de cada uno de los grupos de métodos incluyendo las estructuras utilizadas y el bosquejo del código hasta la curación final de las pistas, aplicando validaciones sobre la tupla generada. Se presenta con detalle cada funcionalidad implementada para la generación de pistas finales. Finalmente, se presenta la arquitectura final del pipeline.
- En el Capítulo 4 se comenta toda la experimentación llevada a cabo en el proyecto. Se puede dividir en las siguientes temáticas:
  - En primer lugar, se comenta acerca de la exploración de los datos.

  - Luego, se explica el procedimiento iterativo y experimental utilizado para la construcción de los métodos presentados previamente.
  - Por otro lado, se explica detalladamente la tarea de generación del corpus etiquetado.
  - Finalmente, se detallan los experimentos relacionados con el clasificador: las arquitecturas experimentadas, el entrenamiento correspondiente y, por último, los resultados obtenidos.

- En el Capítulo 5 se comentan las conclusiones y el trabajo futuro del proyecto.
- *Apéndice Bibliotecas*: Se presentan las librerías utilizadas en el proyecto, se brinda una noción de sus contenidos y para que fueron utilizadas en el mismo.
- *Apéndice Transformers*: Se explica en profundidad los fundamentos utilizados en muchas de las herramientas neuronales que utilizan transformers.
- *Apéndice Mejoras en Resolvedor de Correferencias*: Se explica en detalle los problemas clásicos y las mejoras implementadas para la herramienta de resolución de correferencias.
- *Apéndice Métricas de performance*: Se comentan nociones básicas de las métricas utilizadas para la evaluación del módulo de extracción de pares y del clasificador.
- *Apéndice Planificación del proyecto*: Se brinda de forma detallada la planificación del proyecto, así como la comparación de lo esperado contra la ejecución actual.

Capítulo 2

Revisión de antecedentes

En el siguiente capítulo se brindan los conceptos necesarios para comprender nuestro proyecto. Se presentan los fundamentos teóricos como los trabajos previos utilizados para el desarrollo del mismo. Los fundamentos teóricos proporcionan las bases para comprender las herramientas utilizadas en la implementación de los métodos de extracción de pistas, así como la experimentación y construcción del clasificador. Por otro lado, los trabajos relacionados descritos en este capítulo ofrecen una comprensión del contexto del proyecto y también sirven de inspiración para ciertas ideas que se describirán en profundidad más adelante, como los métodos de extracción de pistas y la arquitectura del clasificador.

2.1. Marco teórico

En esta sección, se presentan los conceptos fundamentales para una mejor comprensión del trabajo realizado. Se comenzará con los conceptos básicos, como el de *definiendum, definición* y *crucigrama*, que se utilizarán a lo largo del informe, junto con el marco de las disciplinas donde se desarrolla el proyecto y las temáticas centrales correspondientes. Si el lector desea profundizar en estos conceptos, puede consultar los apéndices correspondientes.

2.1.1. Conceptos básicos

Como se mencionó en el capítulo anterior, uno de los objetivos principales del proyecto es crear crucigramas en inglés a partir de textos para niños. Por lo tanto, es importante definir algunos conceptos relacionados:

Definiendum: Término que se está definiendo.

Definición: Explicación que describe a un definiendum dado.

A continuación, se presenta un ejemplo de un par definiendum y definición:

Definiendum: **Apple** Definición: **Red fruit many people love**

Crucigrama: Un crucigrama es un tablero (cuadrícula) con cuadrados negros y blancos que funcionan como separadores y celdas para completar respectivamente, y un conjunto de pistas. Las pistas corresponden a definiciones de ciertos definiendums que deben ser colocados, letra por letra, en cada cuadrado alfabético adyacente,

de modo que se forme una palabra. Los cuadrados negros no contienen ningún carácter y se completa el crucigrama cuando todos los cuadrados blancos contienen un carácter. Los términos a resolver dentro del crucigrama deben ser de una sola palabra. En la figura 2.1 [1], se muestran algunos ejemplos de crucigramas en inglés para niños, como los que se pretenden desarrollar en este proyecto.

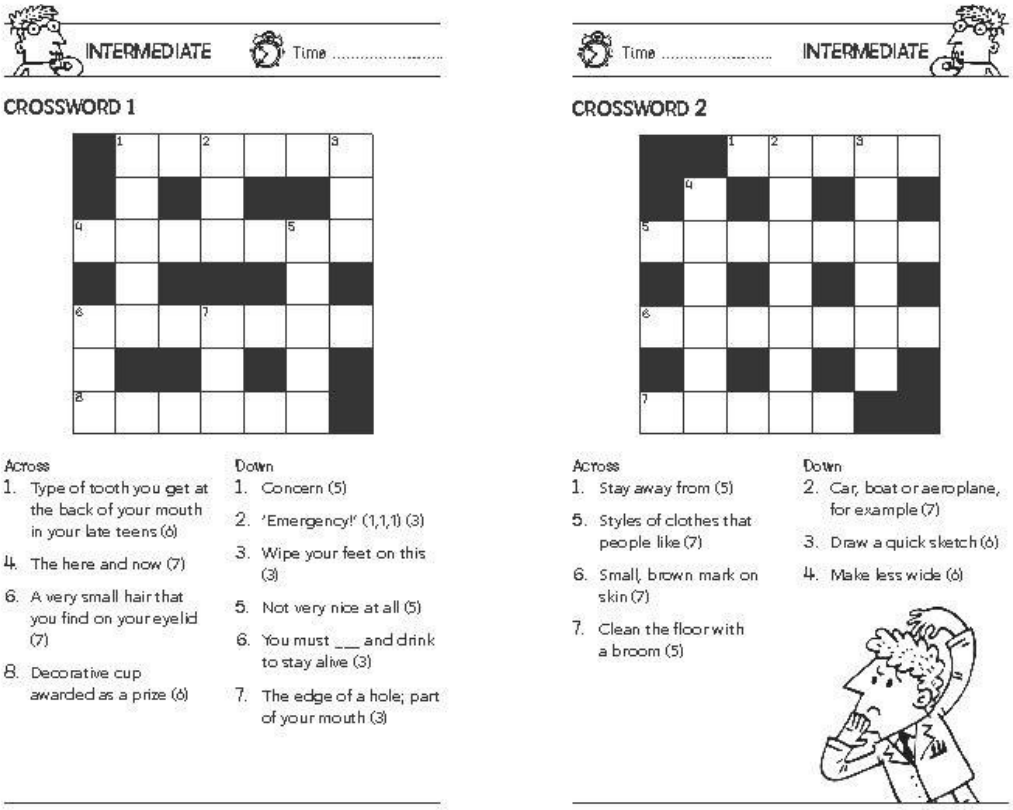

Figura 2.1: Ejemplos de crucigrama de *Crosswords for kids in English* (2023)

Extracción automática de definiciones: Tarea que consiste en obtener, a partir de un texto, una palabra (definiendum) y su posible definición que se encuentra contenida en el mismo.

Generación automática de crucigramas: Tarea que consiste en construir, mediante un algoritmo, un tablero y su distribución de cuadrados blancos y negros, así como la lista de pistas (definiciones) y definiendums correspondientes.

2.1.2. PLN

El Procesamiento del Lenguaje Natural (PLN o NLP por sus siglas en inglés) es una subdisciplina de la Inteligencia Artificial que busca resolver, mediante el uso de computadoras, tareas relacionadas con el lenguaje

[1] Tomado de *Crosswords for kids in English* (2023)

humano. Esto permite la comunicación entre humanos y computadoras a través del lenguaje natural, así como la resolución de tareas que implican algún tipo de procesamiento de texto o habla (Jurafsky & Martin (2009)).

Las aplicaciones del área de PLN son numerosas y en constante expansión, y muchos métodos comunes resultan útiles en su resolución. Sin embargo, existen tareas tradicionales en la literatura, tales como la traducción automática, la clasificación de textos, el resumen de textos, la extracción de información y la generación de texto. Este trabajo se enfoca en una sub-tarea específica de la extracción de información, y también se trabaja en la clasificación de textos para la generación del corpus y el entrenamiento del clasificador. Como se verá en las siguientes secciones, se emplean varias herramientas, algoritmos y modelos de PLN para resolver estos problemas.

2.1.3. Expresiones regulares

Las expresiones regulares, también conocidas como regex, son patrones utilizados para buscar y manipular cadenas de caracteres. Estas expresiones se utilizan en programación y en herramientas de procesamiento de texto para realizar búsquedas y reemplazos de texto con gran precisión y rapidez. Las expresiones regulares son especialmente útiles en procesamiento de lenguaje natural para encontrar patrones en grandes cantidades de texto, como palabras clave, fechas, correos electrónicos y otros tipos de información. (Jurafsky & Martin (2023))

Por ejemplo, si se dispone de un archivo de texto con un gran número de direcciones de correo electrónico, se puede utilizar una expresión regular para buscar y extraer todas las direcciones de correo electrónico en el archivo. Una expresión regular básica para buscar direcciones de correo electrónico podría ser `[\w.-]+@[\w.-]+`, que encontraría cualquier cadena de caracteres que contenga un conjunto de letras, números, puntos y guiones seguido de una arroba y otro conjunto similar de caracteres. Con esta expresión regular, todas las direcciones de correo electrónico del archivo se podrían extraer en cuestión de segundos.

Esta herramienta es de utilidad en el proyecto para la implementación de parsers en los métodos de extracción de pistas.

2.1.4. Part of Speech Tagging (POS Tagging)

Las clases de POS (categoría gramatical de una palabra) se dividen en dos tipos: cerradas y abiertas. Las clases cerradas son aquellas cuyos miembros (palabras con esa categoría) son constantes, mientras que las abiertas son clases que aumentan su número de palabras en el lenguaje con el tiempo. Por ejemplo, las preposiciones pertenecen a las clases cerradas, mientras que los sustantivos son parte de las clases abiertas. (Jurafsky & Martin (2009))

La importancia de POS en el procesamiento del lenguaje radica en la gran cantidad de información que proporciona acerca de una palabra y su entorno, tanto para las clases más grandes como los verbos o los sustantivos, como para las distinciones más finas, como los pronombres posesivos o personales.

El etiquetado de categorías gramaticales, conocido en inglés como Part-of-Speech Tagging o POS tagging, es el proceso en el cual se identifica la categoría de cada palabra en el contexto de un corpus (texto). Las etiquetas pueden ser de tipo sustantivo, pronombre, verbo, adjetivo, entre otros. Hoy en día, es una tarea ampliamente utilizada y existen diversas herramientas y modelos de lenguaje que permiten realizarla. El gran desafío de la tarea de etiquetado es resolver la ambigüedad del lenguaje (Bird (1995)). Es decir, la entrada o input de esta tarea es una secuencia de tokens $x_1, x_2, ..., x_n$, y la salida es una secuencia de etiquetas $y_1, y_2, ..., y_n$. El

etiquetado es una tarea de desambiguación, ya que dado una palabra se debe asignar la etiqueta correcta, pero esta puede tener varias etiquetas posibles dependiendo de su uso. (Jurafsky & Martin (2009))

Usualmente, los algoritmos de POS Tagging reciben una cadena de palabras como entrada y retornan una cadena o lista de etiquetas, correspondientes a cada palabra de la entrada.

Los *tags* o etiquetas que se utilizan dependen del lenguaje en el que se trabaje. En la tabla 2.1 se presentan las etiquetas más utilizadas para el ingles [2].

Tag	Significado	Ejemplos
ADJ	adjetivo	small, new, good
ADP	adposición	on, of, at
ADV	adverbio	already, still, now
CONJ	conjunción	and, or, but
DET	determinante	the, a, some
NOUN	nombre	home, cost, Uruguay
PRON	pronombre	he, she, their
VERB	verbo	say, told, playing

Cuadro 2.1: Tags más utilizados en POS tagging para inglés.

A continuación se muestra un ejemplo de resolución de POS Tagging para una oración simple:

Bobby is a good boy.

Existen dos grandes tipos de algoritmos de POS tagging: el POS tagging que solo indica la categoría gramatical de las palabras:

*Bobby(**NOUN**) is(**VERB**) a(**DET**) good(**ADJ**) boy(**NOUN**).*

Por otro lado, el POS tagging detallado, también conocido como análisis morfológico, permite extraer información adicional de las palabras, como su número, género, lema y persona, además de proporcionar información sobre la conjugación verbal para obtener detalles sobre el tiempo y la voz, por ejemplo

*Bobby(**NNP**) is(**VBZ**) a(**DET**) good(**JJ**) boy(**NN**).*

En este segundo ejemplo se puede observar que se utilizaron tags distintos, correspondientes al conjunto de etiquetas utilizadas por el "Penn Treebank" (Marcus et al. (1999)). Estas etiquetas incluyen información adicional característica de cada palabra y varían dependiendo del modelo utilizado. En la tabla 2.2 se presentan las etiquetas mencionadas previamente utilizadas para el "Penn Treebank".

[2]Etiquetas tomadas de *Universal POS tags* (2014)

Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coord. conj.	*and, but, or*	NNP	proper noun, sing.	*IBM*	TO	"to"	*to*
CD	cardinal number	*one, two*	NNPS	proper noun, plu.	*Carolinas*	UH	interjection	*ah, oops*
DT	determiner	*a, the*	NNS	noun, plural	*llamas*	VB	verb base	*eat*
EX	existential 'there'	*there*	PDT	predeterminer	*all, both*	VBD	verb past tense	*ate*
FW	foreign word	*mea culpa*	POS	possessive ending	*'s*	VBG	verb gerund	*eating*
IN	preposition/ subordin-conj	*of, in, by*	PRP	personal pronoun	*I, you, he*	VBN	verb past participle	*eaten*
JJ	adjective	*yellow*	PRP$	possess. pronoun	*your, one's*	VBP	verb non-3sg-pr	*eat*
JJR	comparative adj	*bigger*	RB	adverb	*quickly*	VBZ	verb 3sg pres	*eats*
JJS	superlative adj	*wildest*	RBR	comparative adv	*faster*	WDT	wh-determ.	*which, that*
LS	list item marker	*1, 2, One*	RBS	superlatv. adv	*fastest*	WP	wh-pronoun	*what, who*
MD	modal	*can, should*	RP	particle	*up, off*	WP$	wh-possess.	*whose*
NN	sing or mass noun	*llama*	SYM	symbol	*+,%, &*	WRB	wh-adverb	*how, where*

Figura 2.2: Tags utilizados para el dataset PTB, obtenido de Jurafsky & Martin (2009).

En el proyecto, es de interés presentar esta tarea ya que se utilizara en varias funciones de post-procesamiento y en varios métodos de extracción de pistas.

2.1.5. Análisis de constituyentes

El análisis de constituyentes es la tarea de obtener los constituyentes de una oración dada. Un árbol de constituyentes es una estructura de datos que representa la estructura sintáctica de una oración de acuerdo con una gramática libre de contexto. Este árbol es el resultado de analizar la oración en sus constituyentes, por lo que es un conjunto ordenado de todos los constituyentes sintácticos de una oración. El orden se basa en cómo los constituyentes se descomponen en subconstituyentes. Los constituyentes pueden ser una palabra o un conjunto de palabras dentro de la oración. (Jurafsky & Martin (2023))

Los nodos internos representan categorías no terminales de la gramática utilizada, mientras que las hojas representan categorías terminales.

Ejemplo 2.1: Consideremos la siguiente oración:

A strong wind may be part of a storm.

En la figura 2.3 podemos ver la salida del análisis de constituyentes y su respectivo árbol.

A partir de la estructura del árbol, se pueden identificar los nodos intermedios según el tipo de sintagma y las hojas correspondientes al análisis léxico de las palabras. Dentro del proyecto, estos arboles permiten analizar las oraciones, y luego extraer ciertos patrones de estas en los métodos de extracción de pistas.

2.1.6. Análisis de dependencias

Otra forma de representar la estructura de una oración es el análisis de dependencias, una tarea que consiste en identificar las relaciones sintácticas entre las palabras de una oración. Estas relaciones binarias se definen en términos de dependencias, donde cada palabra de la oración se relaciona con otra palabra de la oración a la

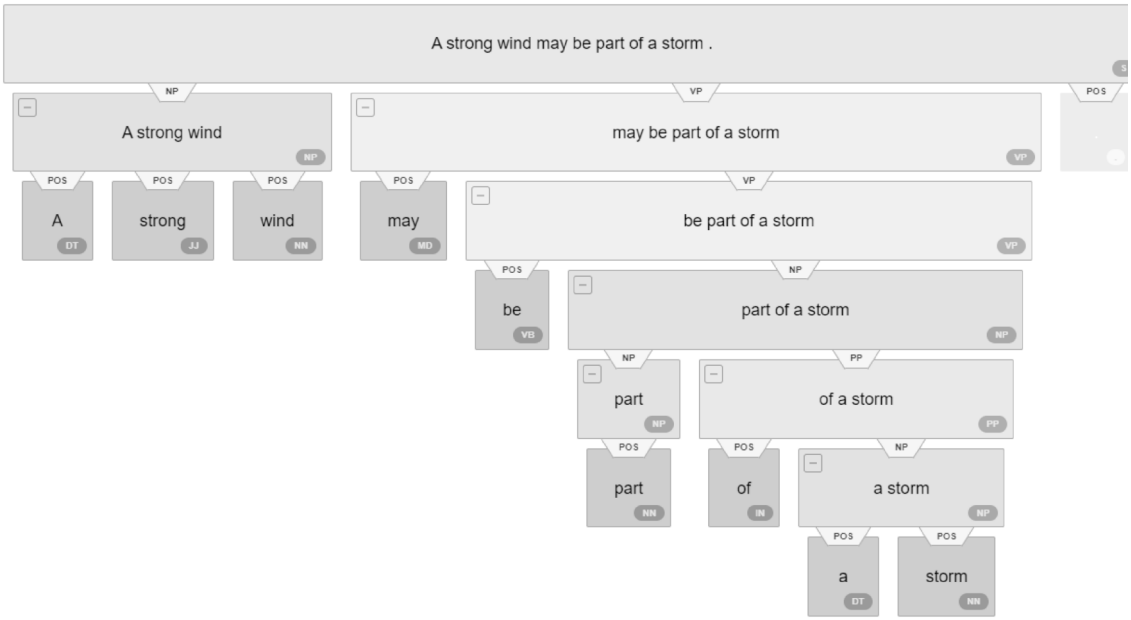

Figura 2.3: Ejemplo de árbol de constituyentes.

cual depende. El árbol de dependencias es una estructura de datos que representa estas relaciones en forma de un grafo acíclico dirigido, donde los nodos representan las palabras y las aristas representan las dependencias. (Jurafsky & Martin (2023))

Estas relaciones binarias son llamadas relaciones de dependencia (dependency relations) y dan la base a estas estructuras o arboles de dependencias, en la tabla 2.4 podemos ver una lista de las relaciones utilizadas en el 'Universal Dependency set' (de Marneffe et al. (2014)). Estas relaciones son dirigidas, es decir, tienen un sentido asignado. Esta dirección indica cuál es la palabra *head, raíz o núcleo* y cuál es la *child o dependiente* de la relación.

Ejemplo 2.2: Considérese la oración:

I prefer the morning flight through Denver.

En la figura 2.5 podemos ver la salida del análisis de dependencias para el ejemplo.

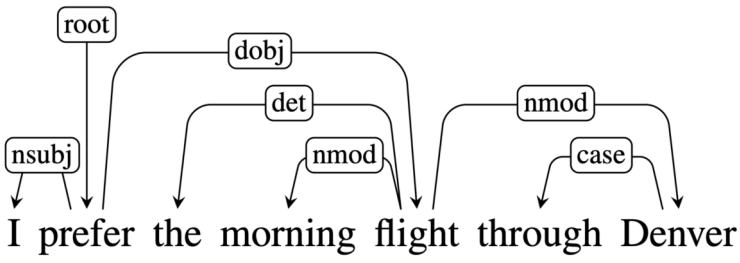

Figura 2.5: Ejemplo de árbol de dependencias extraido de Jurafsky & Martin (2009)

Clausal Argument Relations	Description
NSUBJ	Nominal subject
DOBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
XCOMP	Open clausal complement
Nominal Modifier Relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
APPOS	Appositional modifier
DET	Determiner
CASE	Prepositions, postpositions and other case markers
Other Notable Relations	**Description**
CONJ	Conjunct
CC	Coordinating conjunction

Figura 2.4: Relaciones de dependencia utilizadas en 'Universal dependency Set' (de Marneffe et al. (2014))

En el proyecto, esta herramienta es utilizada para analizar oraciones y reconocer patrones mas complejos que los encontrados utilizando análisis de constituyentes presentado previamente.

2.1.7. Etiquetado de roles semánticos

El etiquetado de roles semánticos (en inglés *Semantic Role Labeling* o *SRL*) es el proceso por el cual se le asignan etiquetas a los constituyentes de los verbos dentro de una oración indicando su rol semántico dentro de ella (Màrquez et al. (2008)). En este proceso se busca entender cómo se relacionan los participantes de la oración con sus eventos, tratando de responder preguntas tales como *"¿Quién le hizo qué a quién?"*.

Los roles semánticos o relaciones temáticas son los diferentes roles que un sintagma nominal puede interpretar en relación a la acción o estado que describe el verbo principal de la oración. Entre los roles principales se encuentran:

- **Agente**: Es quien deliberadamente causa el evento.
- **Experimentador**: Es quien experimenta el evento.
- **Paciente**: Es el participante directamente más afectado por la acción.
- **Benefactor**: Hace referencia a quien recibe cierto resultado acerca de cierta acción.
- **Instrumento**: Participante o elemento utilizado para llevar a cabo la acción.
- **Punto de inicio**: Argumento que marca el inicio (usualmente temporal o espacial) de cierto evento.
- **Punto de fin**: Argumento que marca el fin (usualmente temporal o espacial) de cierto evento.

Se ha buscado establecer un estándar para estos roles de manera que puedan ser utilizados por programas que puedan ser ejecutados por una computadora (Punyakanok et al. (2008)). En este trabajo se utiliza el estándar

propuesto por PropBank. En este modelo se encuentran las categorías mostradas en la tabla 2.2.

Etiqueta	Rol Semántico
ARG0	Agente
ARG1	Paciente, Experimentador
ARG2	Benefactor, Instrumento
ARG3	Punto de inicio, Benefactor
ARG4	Punto de fin

Cuadro 2.2: Etiquetas de SRL y sus respectivos roles.

Se puede notar que en ocasiones, dependiendo del verbo, un argumento puede referirse a un rol semántico en particular. Sin embargo, esta relación no es estática y varía según la estructura argumental de los verbos.

Ejemplo 2.3: Considérese la siguiente oración:

Britain had colonies in North America.

Analizándola a través de SRL se obtiene:

Figura 2.6: Ejemplo de etiquetado de roles semánticos.

Se puede observar en la figura 2.6 cómo se asignan los roles semánticos respecto al verbo *had*, donde el ARG0 se refiere al Agente y el ARG1 al paciente.

En el contexto del proyecto, la herramienta se utiliza para la implementación de un método que construye pistas a partir de la estructura argumental de los verbos.

2.1.8. Resolución de correferencias

La resolución de correferencias es la tarea de agrupar las referencias a una misma entidad en un texto. Es relevante para simplificar y mejorar la comprensión en muchas tareas de procesamiento de lenguaje natural. Jurafsky & Martin (2023)

En la figura 2.7 se muestra un ejemplo de esta tarea, donde se agrupan las palabras que hacen referencia a una misma entidad en el texto, y cada grupo se llama cadena de correferencias. Este problema se suele analizar de forma similar a un problema de clustering. Cada cadena contiene un representante principal llamado “Head” o núcleo. En este caso, existen dos cadenas, uno constituido por las flechas azules y otro por las flechas rojas. En cada caso, debe haber un núcleo para cada cadena. Parece natural definir a “Nader” como núcleo de la cadena azul, pero en el caso de la cadena rojo, no es claro cuál es su núcleo, por falta de contexto. Este es un problema importante que se intenta resolver en el apendice B.2.

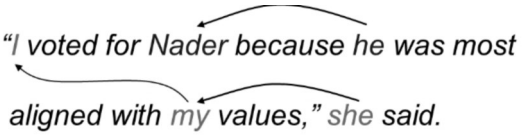

Figura 2.7: Ejemplo de resolución de correferencias.

La tarea de reemplazar correferencias es una tarea derivada de la resolución de correferencias, que consiste en sustituir todas las ocurrencias de un mismo cluster por su "Head" o núcleo. Este proceso es útil para simplificar el texto y facilitar la identificación de las entidades involucradas. A continuación, se presenta un ejemplo extraído de los textos trabajados:

Ejemplo 2.4: Tomemos el siguiente fragmento:

Bobby is a good boy. He has a sister. Her name is Anna.

En este caso, se forman dos agrupaciones o clusters: *[Bobby, He]* y *[Anna, Her]*. Al reemplazar cada una de estas agrupaciones por su núcleo, se obtiene el siguiente resultado:

Bobby is a good boy. Bobby has a sister. Bobby's sister name is Anna.

De esta manera, se pueden identificar las entidades involucradas en cada oración y facilitar su comprensión. Por ejemplo, al aislar la oración *"He has a sister"*, se puede inferir que el pronombre *He* hace referencia a la entidad *Bobby*.

La resolución de correferencias es una tarea fundamental del proyecto, se utiliza como parte inicial del procesamiento de los textos: permite extraer información y simplficar los textos para la extracción de pistas.

2.1.9. Named entity recognition

Una 'named entity' es cualquier entidad en el texto que pueda ser nombrada con un nombre propio, estas pueden ser tanto organizaciones, como lugares o persona. La tarea de reconocer estas entidades es denominada Named Entity Recognition (NER), como su nombre sugiere, trata de encontrar o identificar las palabras o grupos de palabras (spans) que sean entidades. (Jurafsky & Martin (2009))

Hay varias clases de entidades en los métodos que hacen reconocimiento de entidades. En el proyecto actual, son de interés:

- **Organization**: Reconoce organizaciones o grupos de personas.
- **Place**: Reconoce lugares geográficos.
- **Person**: Reconoce nombres de personas.

También se suelen extender las definiciones para otras entidades de interés, que no son usualmente consideradas 'named entities' como fechas, eventos, entidades temporales o expresiones numéricas como precios u otras expresiones de interés. Sin embargo, no consideraremos estos casos.

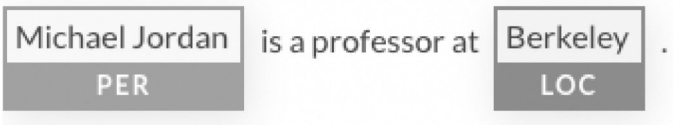

Figura 2.8: Ejemplo de reconocedor de entidades obtenido a partir de la demo de *AllenNLP* (2020).

Se puede observar en la figura 2.8 un texto con sus respectivas clasificaciones de entidades y las clases mencionadas anteriormente.

En el marco del proyecto, los reconocedores de entidades permiten el desarrollo de ciertos métodos de extracción de pistas centrados en las entidades presentes en los textos.

2.1.10. Representaciones vectoriales de palabras

Los *word embeddings*, también conocidos como representaciones vectoriales de palabras, son una técnica fundamental en el procesamiento del lenguaje natural (PLN) que busca representar palabras o frases en un espacio vectorial continuo de baja dimensionalidad. Estas representaciones capturan la semántica y las relaciones contextuales entre palabras, permitiendo que los algoritmos de aprendizaje automático puedan procesar y analizar el texto de manera más efectiva. Los *embeddings* son esenciales para una amplia gama de aplicaciones de PLN, como clasificación de texto, análisis de sentimientos, traducción automática y generación de texto, entre otras. (Jurafsky & Martin (2023))

Existen varios métodos para generar word embeddings, siendo los más populares Word2Vec (Mikolov et al. (2013)), GloVe (Pennington et al. (2014)) y FastText (Bojanowski et al. (2016)). Estos modelos aprenden las representaciones vectoriales a partir de grandes conjuntos de datos de texto, basándose en la idea de que palabras con contextos similares tienden a tener significados similares. Al representar las palabras en un espacio vectorial, es posible realizar operaciones algebraicas para medir la similitud semántica entre palabras, identificar sinónimos o incluso resolver analogías.

Los embeddings son de interés para la implementación de ciertas arquitecturas del clasificador, como se verá en detalle en el capítulo de experimentación.

2.1.11. Clasificación supervisada

Una tarea esencial en el área de Inteligencia Artificial es la de clasificación. La clasificación supervisada es una técnica de aprendizaje automático que tiene como objetivo enseñar a un modelo a asignar categorías a elementos, utilizando un conjunto de datos etiquetado como referencia. En este enfoque, se proporcionan ejemplos de entrada junto con sus respectivas etiquetas, permitiendo que el algoritmo aprenda la relación entre las características de entrada y las etiquetas de salida. Los modelos de clasificación supervisada se utilizan en una amplia variedad de aplicaciones, como el reconocimiento de imágenes, la detección de spam, análisis de

sentimiento, el diagnóstico médico y la predicción del comportamiento del consumidor, entre otras. (Steven Bird (2019))

Esta tarea consiste en predecir una posible clase para un ejemplo dado, utilizando un modelo estadístico y una lista finita de clases previamente definidas. El proceso consta de dos fases principales: entrenamiento y predicción. Durante la fase de entrenamiento, el algoritmo ajusta sus parámetros internos para minimizar el error entre las predicciones del modelo y las etiquetas de los datos de entrenamiento. Una vez que el modelo ha sido entrenado, se utiliza para realizar predicciones en datos nuevos y no etiquetados, asignándoles una categoría basada en el conocimiento adquirido durante el entrenamiento. Es fundamental evaluar el desempeño del modelo utilizando métricas adecuadas, como la precisión, la exhaustividad (‘recall‘), entre otras. para asegurar que el modelo tiene una capacidad de generalización adecuada y evitar problemas como el sobreajuste.

En este proyecto, se trabaja sobre la tarea de clasificación supervisada (en particular, clasificación de texto) en la construcción del corpus y del clasificador. El clasificador de pistas generadas es una instancia de esta tarea.

Clasificación de texto

La clasificación de texto es una subtarea del Procesamiento del Lenguaje Natural (PLN) que se enfoca en asignar categorías o etiquetas a segmentos de texto, como documentos, párrafos, oraciones o palabras, según su contenido semántico. Esta tarea es esencial en una amplia variedad de aplicaciones. En este proyecto se aplicó para clasificar las pistas generadas en buenas o malas. La clasificación de pistas en buenas o malas, entre otras. Los algoritmos de clasificación de texto pueden ser supervisados, semi-supervisados o no supervisados, dependiendo del tipo de datos de entrenamiento disponibles y la naturaleza del problema a resolver. (Steven Bird (2019))

En el contexto de la clasificación de texto supervisada, se entrena un modelo utilizando un conjunto de datos etiquetado, que consiste en ejemplos de texto y sus respectivas categorías. Los algoritmos de aprendizaje automático comunes utilizados para la clasificación de texto incluyen Naive Bayes, Máquinas de Vectores de Soporte (SVM) y Redes Neuronales Artificiales (ANN). En los últimos años, los modelos de lenguaje preentrenados basados en arquitecturas de redes neuronales profundas, como BERT, GPT y sus variantes, han demostrado un rendimiento sobresaliente en tareas de clasificación de texto. Estos modelos aprenden representaciones semánticas ricas y generalizables a partir de grandes cantidades de datos de texto no etiquetados y luego se adaptan a tareas específicas mediante un proceso de fine-tuning en conjuntos de datos etiquetados más pequeños.

2.1.12. Modelos de lenguaje

Los modelos de lenguaje son herramientas de inteligencia artificial que se entrenan para comprender y generar texto en lenguaje humano. Estos modelos utilizan redes neuronales profundas y algoritmos de aprendizaje automático para procesar y analizar grandes volúmenes de texto, con base en el contexto y la información previamente aprendida, los modelos de lenguaje pueden completar fragmentos de texto, responder preguntas, traducir entre idiomas y generar contenido creativo, entre otras tareas. (Jurafsky & Martin (2023))

Existen varias arquitecturas distintas que varían en complejidad y rendimiento. Los modelos de lenguaje se utilizan en todo el proyecto, tanto en los métodos de generación de pistas como en el clasificador. En particular, se emplean los transformers (una arquitectura de modelos de lenguaje) en la mayoría de los casos como modelos

preentrenados mediante bibliotecas de PLN. Además, se utilizó como candidato en el clasificador de pistas generadas. Estos modelos, especialmente los transformers, se fundamentan teóricamente en el apéndice B.

2.2. Trabajos relacionados

En esta sección se presentan los trabajos que sirvieron como referencia para el proyecto. Estos proporcionaron ideas y sentaron las bases para el desarrollo de los métodos de generación de pistas y construcción del clasificador.

2.2.1. Proyectos de grado anteriores

Se presentan los proyectos anteriores del área que tienen puntos en común con nuestro proyecto. Estos aportan la base fundamental de la aplicación de interés, sobre la cual se implementa la mejora en el proyecto actual. También aportan ideas en términos de desarrollo tanto en la generación de pistas como construcción del clasificador.

Aplicaciones Lúdicas de Soporte a la Enseñanza de Lenguas

El proyecto presentado surge de un trabajo realizado en 2019 (Tosi & Percovich (2019)), donde se desarrollaron herramientas lúdicas para la enseñanza del inglés bajo el marco del proyecto "Inglés sin Límites" de la ANEP. El objetivo de este proyecto fue construir una plataforma que proporcionara una base de varios juegos para que los docentes puedan construir actividades de enseñanza.

En el marco de este proyecto se propusieron tres juegos o herramientas lúdicas: la Batalla Naval, la Sopa de Letras y los Crucigramas. Si bien inicialmente nuestra propuesta consistía en continuar y ampliar las funcionalidades de esta ultima herramienta, los crucigramas, también se aplica hoy en día para el módulo de definiciones de las sopas de letras (no fue el objetivo principal, pero se reutiliza la implementación del módulo).

Para la generación de los crucigramas se ha creado un corpus a partir de grandes volúmenes de texto, en particular utilizando Simple English Wikipedia y Ducksters. Se han utilizado métodos de extracción de patrones y heurísticas de clasificación de la salida de los patrones para filtrar los pares correctos e incorrectos de "definiendum" y "definición" junto con su categoría (de entre "house", "animals" y "color"). Este corpus funciona como una especie de "base de datos" para obtener los pares necesarios y luego generar los crucigramas.

El método de extracción de pistas utilizado se basa en heurísticas y analiza los corpus mencionados previamente. Este método toma una oración como entrada y produce un posible par "definiendum"-"definición". Se fundamenta en el reconocimiento del verbo "to be" y el análisis de patrones en estas oraciones. Aunque se realizaron experimentos con diferentes herramientas de implementación en el proyecto, la versión final se implementó utilizando analizadores de dependencias. En la figura 2.9 se muestra la estructura que se busca reconocer en estas oraciones, seguido por la figura 2.10, donde se presenta un ejemplo del patrón mencionado en la oración *"Lava is melted or liquid rock"*.

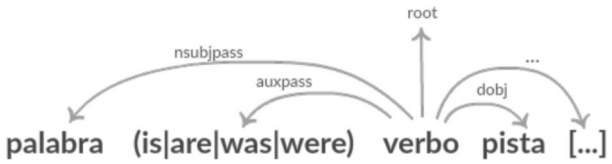

Figura 2.9: Patrón de interés para extracción de pistas en Tosi & Percovich (2019) y su respectivo análisis de dependencias.

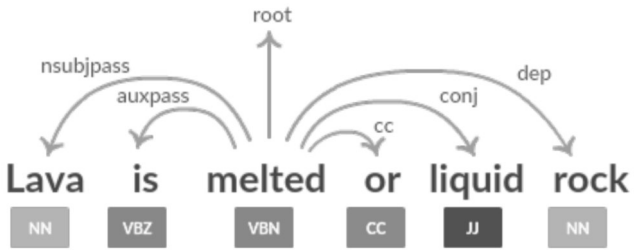

Figura 2.10: Ejemplo de oración y análisis de dependencias extraído de Tosi & Percovich (2019).

Respecto a los crucigramas generados por la aplicación, estos se generan a partir de dos opciones seleccionadas previamente por los usuarios. En primer lugar, se puede elegir el tamaño del tablero, que puede ser, por ejemplo: 8x8, 10x10, 12x12, 15x15. En segundo lugar, se puede seleccionar la categoría de las pistas, que incluyen “house”, “animals”, “colors”, entre otras. A continuación, se seleccionan del corpus generado en el desarrollo del proyecto la cantidad de pistas acordes al tamaño del tablero y con la categoría deseada. Finalmente, se genera el tablero correspondiente al crucigrama mediante un algoritmo. Si bien los crucigramas generados son de alta calidad, presentan una desventaja en cuanto a la poca diversidad de pistas y la falta de versatilidad para aprender nuevos conceptos.

Finalmente, se presenta la plataforma con las herramientas lúdicas propuestas, las cuales fueron integradas y utilizadas en eventos de la ANEP y visitas a centros educativos para su implementación.

Este proyecto, presenta las bases de la aplicación en la cual se desarrolla nuestro proyecto (donde se trabaja la nueva funcionalidad de la plataforma desarrollada en Tosi & Percovich (2019)). Además, la heurística mencionada anteriormente, fue reutilizada en nuestro proyecto y se utilizó como línea base en el módulo de extracción.

Extracción de definiciones y generación automática de crucigramas a partir de textos de prensa

El objetivo de este proyecto anterior es similar al proyecto actual, ya que ambos trabajan en la construcción y generación de crucigramas. En ese proyecto se presenta un pipeline de procesamiento de lenguaje natural que utiliza patrones para extraer pares de definiciones de textos de prensa (Esteche & Romero (2015)). Además, se desarrolla un algoritmo greedy en Prolog para construir los tableros de los crucigramas.

Este proyecto anterior ha servido como guía e inspiración para muchos de los métodos y patrones utilizados en el proyecto actual. La metodología iterativa utilizada en la investigación también ha sido útil para explorar y desarrollar posibles candidatos. Sin embargo, existen diferencias importantes entre ambos trabajos que diferencian los enfoques de la extracción de definiciones.

En primer lugar, el idioma es diferente; mientras que en el proyecto actual se utiliza el inglés, en Esteche y Romero 2015 se trabaja con el español. Debido a que son idiomas distintos, los patrones no se construyen de la misma manera y se les da importancia a diferentes estructuras sintácticas.

En segundo lugar, la naturaleza de los textos de entrada es diferente; en este proyecto anterior se utilizan noticias que emplean un lenguaje técnico y más complejo. En el proyecto actual, los textos son más simples y acotados, lo que dificulta la extracción de definiciones y requiere de métodos de extracción más variados y amplios.

En tercer lugar, los requerimientos son distintos; en este proyecto anterior se utiliza un complemento externo para obtener definiciones, mientras que en nuestro proyecto se requiere la extracción de definiciones autocontenidas y más simples.

Finalmente, (Esteche & Romero (2015)) obtuvo buenos resultados, con una precisión del 73 % en la generación de pistas y la capacidad de generar crucigramas correctamente utilizando los métodos descritos previamente.

Generación de preguntas y respuestas para comprensión lectora en inglés utilizando modelos neuronales

Aunque este proyecto anterior difiere en objetivos del proyecto actual, comparte algunos puntos en común y provee un entendimiento sobre textos para niños. Ambos trabajos se basan en la misma distribución de datos, es decir, ambos trabajamos sobre cuentos breves, simples en ingles para niños. (Rischewski & Berger (2022))

En este proyecto, el objetivo fue obtener una lista de pares de “preguntas” y “respuestas” a partir de un texto para niños, utilizando modelos neuronales. Para ello, se entrenaron diversas arquitecturas basadas en corpus tradicionales como SQuAD (Rajpurkar et al. (2016)), y se seleccionó el modelo con los mejores resultados. De esta manera, la aplicación permite realizar actividades lúdicas, juegos y evaluaciones de comprensión lectora para niños.

Sobre este proyecto, se toman ideas principalmente respecto a la construcción y entrenamiento del clasificador. Si bien la tarea de generar preguntas y respuestas es distinta a la de nuestro proyecto, los enfoques neuronales nos sirvieron de inspiración para los clasificadores (particularmente, el seleccionar modelos preentrenados y realizar un finetuning de los mismos). Además, se utilizan métodos de curación y preprocesamiento de los datos que nos fueron de utilidad al momento de la experimentacion.

2.2.2. Otros trabajos

Esta sección proporciona una breve introducción a los artículos relevantes que influyeron en el desarrollo del proyecto. Las ideas y metodologías utilizadas en estos trabajos fueron fundamentales para el desarrollo del proyecto, ya que están estrechamente relacionados con su alcance.

Automatic Generation of Crossword Puzzles

En este artículo Rigutini et al. (2012), se propone un sistema en capas para la generación automática de crucigramas. El primer grupo de capas utiliza técnicas de procesamiento de lenguaje natural (PLN) para obtener entidades y acciones relacionadas con ellas. El segundo grupo de capas utiliza programación dinámica para generar el crucigrama.

Aunque el proyecto actual no se centra en modificar o implementar un algoritmo de generación de tableros de crucigramas (segundo objetivo principal del artículo), la arquitectura presentada para el análisis de texto y extracción de definiciones es interesante y ofrece un enfoque diferente a la tarea de extracción de definiciones en comparación con trabajos previos. Como se muestra en la figura 2.11, la arquitectura del sistema para el submódulo de PLN es compleja, con varias capas que analizan las oraciones de manera secuencial.

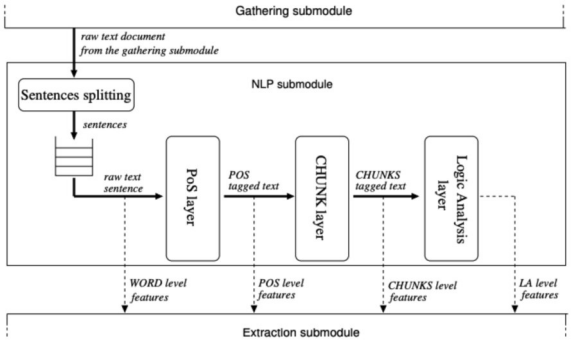

Figura 2.11: Arquitectura de submódulo de NLP (Tomada de Rigutini et al. (2012))

Como se verá más adelante en la solución presentada, esta fue basada e inspirada en cierta forma en este procesamiento secuencial para la extracción de definiciones.

It is AI's Turn to Ask Humans a Question: Question-Answer Pair Generation for Children's Story Books

Con el objetivo de generar evaluaciones y aplicaciones lúdicas para niños, este artículo Bingsheng Yao (2021) muestra cómo, a través de métodos neuronales, es posible generar una lista de pares de preguntas y respuestas auto contenidas en el texto. Los autores trabajan sobre la tarea de "Question Answering" implementando un pipeline de generación de preguntas y respuestas; generando un corpus de 10,580 preguntas explícitas e implícitas a partir de 278 textos para niños.

Basándonos en la solución propuesta, ideamos un método de "QA" que será descrito más adelante, el cual fue modificado para nuestro caso de uso con el fin de conseguir los pares deseados para los crucigramas. Además, adaptamos la idea del "ranking" para construir el clasificador de pistas. En la figura 2.12 se puede ver la arquitectura propuesta en el artículo. La arquitectura de referencia se encuentra disponible en *FairytaleQA: Question Answering Generation System* (2022).

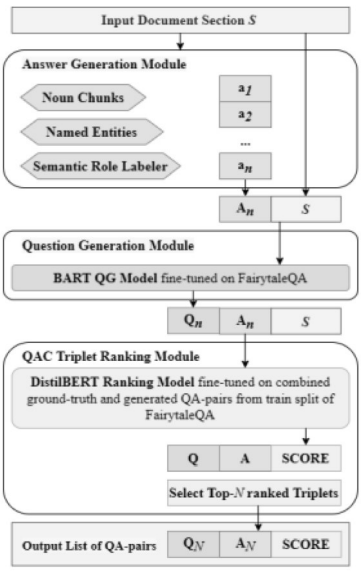

Figura 2.12: Arquitectura de QAG (tomada de Bingsheng Yao (2021))

2.3. Herramientas

En esta sección se describen las herramientas utilizadas en la implementación del sistema, así como las alternativas analizadas y las razones por las que se eligió cada una de ellas. Es importante tener en cuenta que los resultados del módulo dependen del rendimiento específico de cada uno de los modelos en estas bibliotecas, que pueden variar y ser eventualmente suplantadas por alternativas similares.

Además, muchas de estas bibliotecas ofrecen soluciones para varias de las tareas presentadas en el sistema, pero después de experimentar con diferentes combinaciones, se eligieron las que mejor funcionaron en cada tarea en particular. Si el lector desea profundizar en estas bibliotecas, se presentan y se brinda una noción de sus contenidos en el apéndice A.

2.3.1. POS taggers

Existen varias bibliotecas y herramientas populares en Python que facilitan la tarea de etiquetado de categorías gramaticales. En este proyecto se experimentaron con las siguientes:

- **NLTK**: Incluye varios etiquetadores POS pre-entrenados, como el etiquetador basado en el modelo HMM (Hidden Markov Model) y el etiquetador de perceptrón promedio.
- **spaCy**: Proporciona etiquetadores POS precisos y eficientes basados en modelos de aprendizaje profundo y permite la personalización mediante el entrenamiento de modelos en datos etiquetados específicos del dominio.
- **Stanza** (**StanfordNLP**): Ofrece etiquetadores POS de alta precisión y admite una amplia variedad de idiomas.

Estas librerías y herramientas ofrecen soluciones robustas y eficientes para el etiquetado de categorías gramaticales en Python. Para los textos trabajados en este proyecto, se obtuvieron mejores resultados con el POS Tagger de spaCy, por lo que se utilizó en la aplicación desarrollada.

2.3.2. Analizadores de constituyentes

En el desarrollo de la aplicación se evaluaron diversas alternativas para el análisis de constituyentes. A continuación, se mencionan algunas de las opciones consideradas:

- **spaCy**: Esta herramienta de procesamiento de lenguaje natural proporciona un analizador de constituyentes basado en BERT, que permite obtener árboles sintácticos precisos y eficientes.
- **Stanza** (**StanfordNLP**): Este analizador posee una alta precisión y utiliza modelos de aprendizaje profundo entrenados en grandes conjuntos de datos de árboles gramaticales.
- **AllenNLP**: Incluye varios modelos para el análisis de constituyentes, como el analizador de árboles de Span-Constituency basado en el modelo ELMo y el analizador de árboles de constituyentes basado en BERT.

Después de comparar estas alternativas basándonos en diversos ejemplos, al detectar un error en alguno de los análisis, probábamos otra alternativa. Si esta demostraba ser más robusta en la mayoría de los casos, la elegíamos como la opción final. Tras evaluar los resultados, se determinó que el modelo de AllenNLP ofrecía un mejor rendimiento, por lo tanto, fue integrado en la aplicación desarrollada.

2.3.3. Analizadores de dependencias

En la siguiente lista se presentan las alternativas analizadas para analizadores de dependencias:

- **spaCy**: Incluye un analizador de dependencias basado en modelos de aprendizaje profundo. Además, proporciona herramientas para la visualización y manipulación de árboles de dependencias, lo que facilita el desarrollo y la comprensión del análisis de dependencias.
- **Stanza** (**StanfordNLP**): Proporciona un analizador de dependencias basado en el algoritmo de Eisner. Aunque su rendimiento es inferior a los modelos de aprendizaje profundo, sigue siendo una opción robusta y ampliamente utilizada en el campo de PLN.
- **NLTK (Natural Language Toolkit)**: También proporciona un analizador de dependencias basado en el algoritmo de Eisner. Por lo tanto, posee el mismo problema que Stanza y puede ser inferior a otros competidores.
- **AllenNLP**: Provee un analizador de dependencias basado en el modelo de lenguaje ELMo, lo que permite obtener resultados precisos y confiables.

En este caso, se eligió el analizador de dependencias de spaCy debido a que obtuvo los mejores resultados en el proyecto.

2.3.4. Etiquetadores de roles semánticos

En este caso, sólo se evaluó la librería AllenNLP, que incluye modelos de aprendizaje profundo de vanguardia para el etiquetado de roles semánticos. Entre ellos, destacan el modelo basado en la arquitectura BiLSTM-CRF y el modelo basado en BERT, ambos con un buen rendimiento según los experimentos realizados.

2.3.5. Resolución de correferencias

A continuación, se enumeran algunas de las principales librerías y herramientas en Python para la resolución de correferencia:

- **spaCy**: Se incluye una extension popular, llamada neuralcoref que agrega un resolvedor de correferencia basado en redes neuronales a spaCy.
- **AllenNLP**: Incluye modelos de aprendizaje profundo de vanguardia para la resolución de correferencia, como el modelo basado en la arquitectura de clustering end-to-end.
- **Stanza (Stanford NLP)**: El resolvedor de correferencia de Stanford es conocido por su alta precisión y puede utilizarse fácilmente en Python.

En este caso, se ha elegido utilizar AllenNLP debido a los resultados superiores que ha obtenido en la tarea de resolución de correferencias. Además, se implementaron algunas mejoras del resolvedor, detalladas en C.

2.3.6. Reconocedores de entidades nombradas

A continuación, se enumeran algunas de las principales librerías y herramientas en Python para el reconocimiento de entidades:

- **spaCy**: Proporciona herramientas para la visualización y manipulación de las entidades reconocidas (algunos métodos de utilidad para sustituir, formar objetos y funciones basadas en las cadenas de correferencias). Incluye un reconocedor de entidades de alta calidad y rendimiento basado en aprendizaje profundo.
- **NLTK (Natural Language Toolkit)**: Incluye un reconocedor de entidades basado en expresiones regulares y en la técnica de "chunking". Aunque NLTK es fácil de usar, su rendimiento en el reconocimiento de entidades puede ser inferior al de otras librerías más modernas.
- **Stanza (anteriormente StanfordNLP)**: Proporciona reconocedores de entidades precisos y admite una amplia variedad de idiomas.
- **AllenNLP**: Incluye modelos de aprendizaje profundo de vanguardia para el reconocimiento de entidades, como el modelo basado en BERT.
- **Hugging Face Transformers**: Proporciona varios modelos neuronales pre-entrenados para el reconocimiento de entidades, que pueden adaptarse a la tarea mediante finetuning en conjuntos de datos etiquetados específicos. Los modelos basados en arquitecturas de redes neuronales profundas, como BERT y RoBERTa, han demostrado obtener resultados de vanguardia en el reconocimiento de entidades.

En este caso, se utiliza uno de los modelos de reconocimiento de entidades de spaCy, que al experimentar, se ha encontrado que proporciona los mejores resultados.

2.3.7. WordNET

WordNet (Miller (2006)) es una base de datos léxica y semántica desarrollada en la Universidad de Princeton, que tiene como objetivo facilitar el análisis y procesamiento del lenguaje natural. Esta base de datos organiza palabras en inglés en conjuntos de sinónimos llamados "synsets", y establece relaciones semánticas entre ellos, como hiperonimia, hiponimia, meronimia, holonimia, antonimia, entre otras. WordNet es ampliamente utilizado en aplicaciones de procesamiento del lenguaje natural, como la desambiguación del sentido de las palabras, expansión de consultas en sistemas de búsqueda y análisis de similitud semántica. Un ejemplo de uso de WordNet en Python es a través de la librería NLTK (Natural Language Toolkit).

2.3.8. Librerías para clasificación de texto

En Python, existen diversas librerías para la clasificación de texto, pero dos de las más prominentes son scikit-learn y Hugging Face Transformers. Ambas ofrecen enfoques diferentes y complementarios para abordar problemas de clasificación de texto, dependiendo de las necesidades y requisitos de cada proyecto.

- **scikit-learn**: Proporciona una amplia gama de algoritmos de clasificación, como Naive Bayes, SVM, Decision Trees, Random Forest y Logistic Regression. Es una excelente opción para comenzar con la clasificación de texto y aprender sobre diferentes algoritmos y técnicas. Además, scikit-learn ofrece herramientas para la extracción de características (como Bag of Words y TF-IDF), la selección de características, la validación cruzada y la evaluación del rendimiento del modelo.
- **Hugging Face Transformers**: Los modelos neuronales proporcionados pueden adaptarse a la tarea de clasificación de texto mediante ajuste fino en conjuntos de datos etiquetados específicos de esta tarea. Los Transformers han demostrado un rendimiento de vanguardia en una amplia variedad de tareas de PLN, incluida la clasificación de texto. Al utilizar estos modelos pre-entrenados, se pueden abordar problemas de clasificación de texto más complejos y lograr una mayor precisión incluso con grandes volúmenes de datos. Sin embargo, es importante tener en cuenta que los modelos de Transformers suelen ser más intensivos en recursos y requieren más tiempo de entrenamiento en comparación con los algoritmos de scikit-learn.

En el proyecto se utilizan ambas librerías, ya que se experimentaron con varios métodos de ambos casos.

2.4. Corpus de datos

El cuerpo de datos utilizado en este proyecto fue recomendado por los tutores y se obtuvo de un proyecto de grado: Colombato & Musso (2022). Este corpus está compuesto por cuentos en inglés adecuados para niños con diferentes niveles de dificultad, y se recopilaron de la página "ReadWorks" (*ReadWorks* (2008)). Los cuentos se dividen en 5 niveles, los cuales aumentan la dificultad de comprensión y complejidad de la temática. Los textos de nivel 1 y 2 son relativamente cortos y utilizan un léxico sencillo, mientras que los textos de nivel 3, 4 y 5 son más complejos, largos y requieren un mayor nivel de comprensión. En este proyecto, se utilizaron mayoritariamente los cuentos de nivel 1 y algunos de nivel 2, que suman un total de 400.

Capítulo 3

Diseño de patrones para generar pistas

El objetivo de esta etapa es recibir un texto y formar un conjunto de pares $\prec$ *definiendum: definición* $\succ$. La extracción de definiciones se aborda mediante diversos enfoques, creando patrones basados en distintos conceptos. En cada texto que se quiera analizar para extraer sus definiciones, se ejecutan todos los métodos y se forma un conjunto de definiciones resultantes.

Algunos patrones cuentan con etapas de preprocesamiento y postprocesamiento, mientras que en otros estas etapas no son necesarias y solo basta con el patrón en sí mismo. Estos distintos patrones se basan en el uso de herramientas de procesamiento de lenguaje natural, mediante la creación de reglas a partir de un análisis de los textos y herramientas de aprendizaje automático.

En este capítulo se explicará cómo se crearon los distintos patrones utilizados para formar las definiciones y cómo funciona cada uno de ellos. Los distintos patrones utilizados para formar los pares de definiciones se agrupan en las siguientes categorías:

- Métodos basados en reglas
    1. Patrones basados en expresiones regulares
    2. Patrones basados en roles semánticos
    3. Patrones extendidos
    4. Patrones basados en entidades nombradas
- Métodos basados en QA (*Question Answering*)

3.1. Métodos basados en reglas

Estos métodos se basan en analizar un texto buscando que se cumplan ciertas estructuras y una vez identificadas, crear reglas para interpretarlas, extraer sus diferentes partes y finalmente crear las definiciones.

El enfoque inicial para detectar estas estructuras o "patrones" que se repiten en los textos surge del análisis de una gran cantidad de cuentos. Se extraen definiciones de forma manual y se exploran las similitudes entre estas definiciones, prestando especial atención a sus contextos. Ilustremos este procedimiento con el cuento "*A baby polar bear grows up*" extraído del corpus "ReadWorks" previamente mencionado:

Polar bears live in ice and snow. A polar bear baby is a cub. A cub is born with its eyes closed. It does not have much hair. A cub drinks its mother's milk. The mother keeps the cub warm. The cub grows bigger. Soon the cub can walk. Its mother shows it how to hunt. She shows it how to swim. The cub likes to play. It rolls in the snow. The cub grows stronger. The cub learns to swim. It can find its own food. Now the cub can live by itself.

Al leer este cuento, parece que el tema principal son los "cubs", es decir, los osos polares bebés. En principio, parecería que solo hay tres definiciones que se pueden obtener: una para "cub", otra para "cub's mother" y otra para "Polar bears". Recordando que el objetivo de este trabajo es obtener pares de ≺*definiendum: definición*≻ (ó ≺*palabra: pista*≻) para formar un crucigrama y evaluar la comprensión del estudiante sobre el texto, no parece muy interesante crear un crucigrama con solo tres términos. Sin embargo, en una segunda lectura, se pueden identificar muchos otros términos en el cuento que si bien no son entidades "protagonistas" o que no se les da tanta atención o énfasis, aún así se pueden formar definiciones a partir de ellas.

Ejemplo 3.1: Considérese la oración:

Polar bears live in ice and snow.

Como se mencionó antes, "Polar bears" es el término inicial que intuitivamente se buscaría definir, donde una posible definición es "Animals that live in ice and snow". Sin embargo, también existen otros dos candidatos en la oración: "ice" y "snow". Se pueden formar definiciones para estos otros dos términos de la forma "Environment where polar bears live". Al ampliar el espectro de definiciones, se generan nuevas reglas y se pueden formar crucigramas más completos. Otras definiciones que se podrían formar para este cuento son:

- ≺ *Eyes closed: state of a newborn cub* ≻
- ≺ *Walk: a skill a growing cub learns* ≻
- ≺ *Snow: an environment in which a cub rolls* ≻
- ≺ *Strength: something a cub gains as it grows* ≻

Después de extraer múltiples definiciones de forma manual, es necesario encontrar contextos similares entre las definiciones para poder construir las reglas y patrones para extraer nuevas definiciones de forma automática. Por ejemplo, después de extraer diferentes definiciones de forma manual, es posible obtener el siguiente conjunto (Contexto ≺ ***definiendum****: definición*≻). El contexto puede ser la oración de donde se obtuvo el término a definir o un conjunto de oraciones que ayudaron a formar la definición.

- A polar bear baby is a cub. ≺ ***Cub****: Polar bear baby* ≻
- Mexico is part of the continent North America. ≺ ***Mexico****: part of the continent North America* ≻
- The moon is a dark place. ≺ ***Moon****: a dark place* ≻

Esta fue una breve simplificación del proceso para encontrar un patrón, pero en realidad este proceso de extraer definiciones de forma manual para luego comparar sus contextos y buscar estructuras similares es muy

largo y agotador. Por lo que se propuso buscar una forma más eficiente de encontrar patrones.

Entre los primeros patrones formados se nota que muchos de ellos son formados en torno a verbos. Se toma quién realiza una acción, sobre quién o qué la realiza, lugar dónde ocurre y en qué momento. Luego, se forman definiciones sobre el agente que realiza la acción o sobre el objeto que fue afectado por la acción, pero ambas definiciones son expresadas por el verbo que los une. Para entenderlo mejor, veamos más ejemplos de definiciones extraídas manualmente de cuentos pertenecientes a ReadWorks, siguiendo la estructura (Contexto $\prec$ ***definiendum**: definición*$\succ$) antes vista:

- Wild pigs eat leaves, roots, and fruits. $\prec$ ***Leaves**: Part of a plant that wild pigs eat* $\succ$
- Noisy birds use their big beaks to eat fruit. $\prec$ ***Fruit**: Food that noisy birds eat* $\succ$
- More than a thousand years ago, the Aztecs lived in Mexico. $\prec$ ***Mexico**: Place where the Aztecs lived* $\succ$

En estos ejemplos, podemos ver que dos de las definiciones utilizan el verbo “eat” y otro utilizan el verbo “live”. En ambos casos, las definiciones resultantes son similares: se toma a quien realiza la acción o el lugar donde ocurre y se forma una definición entorno al verbo utilizado.

De esta observación surge una nueva idea para buscar patrones: elegir un verbo y buscar oraciones en las que esté dicho verbo o una de sus conjugaciones. De esta forma, ya podemos comparar estas oraciones, ver si se pueden construir definiciones similares y formar un patrón. Dado esto, se crea un sistema que recorre todos los cuentos de referencia y se cuenta qué verbos se utilizan más. Luego, podemos listar todas las oraciones de todos los textos que tengan ese verbo y buscar patrones que sabemos que se van a repetir con más frecuencia en el corpus, pensando que idealmente se puedan repetir también en otros textos usados por docentes que sean del mismo nivel de *ReadWorks*. Otra forma de buscar patrones fue analizando qué uso se les da a otros tipos de palabras dentro de los textos, como las preposiciones o los adjetivos.

Una vez que se tiene una idea de cómo formar un patrón, es necesario crear un programa que pueda identificarlo de forma automática utilizando las reglas que se han creado. Más adelante se explicará cómo crear definiciones mezclando oraciones, pero por el momento, se explicará cómo extraer definiciones utilizando oraciones de forma aislada. La única relación entre las distintas oraciones es para saber a qué se refiere un pronombre que pueda aparecer. Para esto, cada vez que se recibe un texto, se deben seguir los siguientes pasos:

1. Resolver correferencias.
2. Segmentar las oraciones.
3. Recorrer cada oración y buscar que se cumplan las reglas creadas.
4. Etapa de postprocesamiento para mejorar definiciones.

Por lo tanto, lo primero es resolver las correferencias. Como se explicó en la sección 2.1.8, la resolución de correferencias es la tarea de agrupar las referencias de una misma entidad en un texto. Esto se traduce en reemplazar todas las referencias de una misma entidad en un texto por la entidad misma, de tal forma que si

tenemos una oración aislada, siempre sepamos de quién o qué se está hablando. Esto se puede entender mejor con un ejemplo. Para ello, tomemos el cuento "Maria Makes a Snake" perteneciente a Readworks:

Marco and Maria are camping in the yard. They have their flashlights and some snacks. Marco swings his flashlight around in the tent. Hoot! Hoot! He makes scary noises. Maria is not scared. She remembers what she learned in school about shadows. A solid object in front of light makes a shadow. She shines her flashlight on the side of the tent. She puts her hand in front of the light. She twists her hand around. "Look, a snake!" Maria says. Marco jumps. Then he sees that it is only a shadow. They laugh and laugh together.

Tomemos ahora una oración al azar, como "He makes scary noises". Si tenemos esta oración aislada, no sabemos a quién se refiere el pronombre"He". Como se verá más adelante, en nuestra estrategia para formar definiciones, queremos ser capaces de tomar una oración de forma aislada y poder formar una definición. Para ello, deberíamos ser capaces de saber que "He" se refiere a "Marco". Para esto, utilizamos el módulo de resolución de correferencias de spaCy y al haber recuperado las cadenas de correferencias entre pronombres y nombres propios, sustituimos cada pronombre con su nombre propio asociado, obteniendo:

Marco and Maria are camping in the yard. Marco and Maria have their flashlights and some snacks. Marco swings his flashlight around in the tent. Hoot ! Hoot ! Marco makes scary noises. Maria is not scared. Maria remembers what she learned in school about shadows. A solid object in front of light makes a shadow. Maria shines her flashlight on the side of the tent. Maria puts her hand in front of the light. Maria twists her hand around. " Look, a snake ! " Maria says. Marco jumps. Then Marco sees that it is only a shadow. They laugh and laugh together.

Ahora podemos tomar una oración y saber de quién o qué se está hablando, por ejemplo: "Marco makes scary noises". Es importante tener en cuenta que estas herramientas de resolución de correferencias no son perfectas y, en algunos casos, pueden resolver mal las correferencias o dejar referencias sin resolver, tal como en la última oración: "They laugh and laugh together". Existen otros pronombres que están sin resolver, por ejemplo en la oración "Maria shines her flashlight on the side of the tent", donde "her" también hace referencia a María. Sin embargo, estos pronombres fueron excluidos a propósito, ya que al generar las definiciones no se desea repetir los nombres. Por ejemplo, si creáramos una definición para María utilizando la oración "María shines María flashlight on the side of the tent", obtendríamos algo como "Person that shines María flashlight on the side of the tent", donde el término a definir se incluye en la definición, algo que se quiere evitar.

Una vez completado este paso, podemos seguir por separar las oraciones para poder analizarlas de forma aislada. Esto se realiza con la herramienta de separacion de oraciones[1]. Al ejecutar este módulo, obtenemos una lista con todas las oraciones del texto:

- *Marco and Maria are camping in the yard.*
- *Marco and Maria have their flashlights and some snacks.*
- *Marco swings his flashlight around in the tent.*
- *Hoot!*
- *Hoot!*

[1] *SentenceSplitter* de *SpaCy* (2016)

- *Marco makes scary noises.*
- *Maria is not scared.*
- *Maria remembers what she learned in school about shadows.*
- *A solid object in front of light makes a shadow.*
- *Maria shines her flashlight on the side of the tent.*
- *Maria puts her hand in front of the light.*
- *Maria twists her hand around.*
- *"Look, a snake!" Maria says.*
- *Marco jumps.*
- *Then Marco sees that it is only a shadow.*
- *They laugh and laugh together.*

Una vez que se han separado todas las oraciones del texto, podemos proceder a la parte más extensa de la generación de pistas. En esta etapa, se recorren las oraciones para verificar que se cumplan las reglas previamente creadas y así identificar patrones que puedan servir como pistas.

En las siguientes secciones (3.1.1, 3.1.2, 3.1.3, 3.1.4), se explicarán los diferentes patrones utilizados para resolver este punto. Se verán las diferentes herramientas y tecnologías para crear este paso.

Por último, se cuenta con una etapa para efectivamente generar las pistas. Esta etapa también se utiliza para filtrar las pistas que no son adecuadas para el crucigrama o mejorar las pistas generadas. Se proporciona una explicación más detallada de esto en la sección 3.4.

3.1.1. Patrones con expresiones regulares

Este tipo de patrones se basa en el uso de expresiones regulares. Para formar estas expresiones, se estudia la composición de los árboles de constituyentes de distintas oraciones que comparten verbos en común. Cuando se encuentra una generalidad entre los distintos árboles, se crean reglas para extraer distintas partes de la oración y formar las definiciones.

El siguiente ejemplo ilustra este enfoque. Entre los verbos más utilizados en todos los cuentos analizados está "eat". Al analizar la siguiente oración:

Bears eat the meat

En lugar de tratar de definir quién realiza la acción, se toma un enfoque diferente y se define al objeto que se ve afectado por la acción. Es decir, en lugar de tratar de definir "Bears", se define "meat". Si asumimos que lo que viene después del verbo "eat" (en este caso "meat") entra en la categoría de "comida", podríamos definirla en el contexto de la oración como "Food that bears eat".

Crear una regla para que una computadora pueda identificar las distintas partes de la oración parece fácil en este caso, ya que lo que tenemos a la derecha de "eat" es lo que queremos definir y luego formamos una definición tomando lo que está a la izquierda de "eat", en Inglés. Pero no siempre tenemos oraciones tan sencillas. Con una simple variación de la oración, la regla ya no funciona:

Bears are animals that eat meat

Si se forma una definición para "meat" utilizando la misma regla, se obtiene: "Food that bears are animals that eat". O peor aún, si tenemos la siguiente oración:

Deer are animals that eat the leaves when they are hungry

En este caso, incluso resulta complejo identificar qué es lo que se quiere definir.

En esta sección se aborda la búsqueda de alternativas para visualizar las oraciones y encontrar elementos en común, así como la exploración de otros formatos para representar sus generalidades. Para ello, se utiliza el análisis de árboles de constituyentes. A continuación, en la figura 3.1, se examina la sintaxis de la primera oración mencionada.

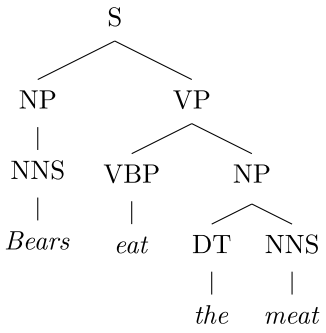

Figura 3.1: Árbol de constituyentes de la oración *Bears eat the meat*

Los árboles de constituyentes son un conjunto ordenado de todos los constituyentes sintácticos de una oración, lo que implica que aunque los actores que se relacionan con el evento que representa el verbo cambien, la estructura del árbol tendrá similitudes. Por ejemplo, podemos cambiar al animal quién "come" o lo que está "comiendo" para encontrar qué partes del árbol se mantienen iguales. A continuación, en las figuras 3.2 y 3.3, se muestran algunos ejemplos:

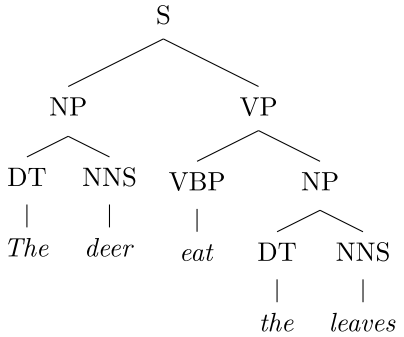

Figura 3.2: Árbol de constituyentes de la oración *The deer eat the leaves*

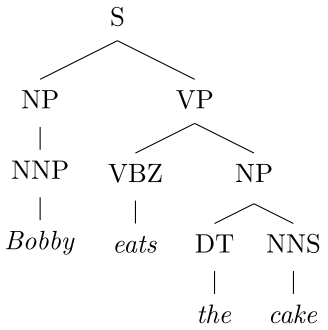

Figura 3.3: Árbol de constituyentes de la oración *Bobby eats the cake*

Al analizar estos árboles, notamos una estructura común. Todos comienzan dividiendo la oración en un sintagma nominal (NP o SN) y uno verbal (VP o SV). Si tomamos el subárbol formado por el SN, sus nodos terminales son quienes realizan la acción "eat" y quienes nos ayudarán a formar la posible definición: "Food that [nodos terminales de SN] eat". Por otro lado, el sintagma verbal se divide en dos subgrupos de palabras, uno que refiere al verbo mismo y el otro a un sintagma nominal. En este caso, los nodos terminales del SN indican lo que queremos definir.

Los ejemplos que acabamos de ver se representaron en forma de árbol para una mejor visualización, pero también podemos representarlos con *bracket notation*:

(ROOT (S (NP (NNS Bears)) (VP (VBP eat) (NP (DT the) (NNS meat)))))
(ROOT (S (NP (DT The) (NNS deers)) (VP (VBP eat) (NP (DT the) (NNS leaves)))))
(ROOT (S (NP (NNS Bobby)) (VP (VBZ eats) (NP (DT the) (NNS meat)))))

Recordemos ahora nuestra definición de expresiones regulares: "*...Las expresiones regulares son una forma de describir patrones para poder validar o encontrar una cadena de caracteres dentro de un texto...*". Esta definición parece ser útil para nuestro objetivo, sin embargo, es importante destacar que las expresiones regulares están diseñadas para trabajar con lenguajes regulares y el inglés no es un lenguaje regular, sino un lenguaje natural con una complejidad aun mayor a la de una gramática libre de contexto. Por lo tanto, estas no pueden reconocerse o analizarse con expresiones regulares.

Aunque hay muchas estructuras lingüísticas en el inglés que no se pueden expresar mediante expresiones regulares, podemos suponer que las oraciones que vamos a analizar son simples, ya que provienen de cuentos para niños y es probable que puedan describirse con expresiones regulares.

Dejando eso de lado, intentemos aplicar expresiones regulares para reconocer las partes necesarias de forma automática.

Patrón eat: La extracción de los grupos constituyentes de una oración que cumple con este patrón (es decir, contiene al verbo *eat*), se hace a partir de la expresión regular:

```
.*\(S(\(NP .*)(\(S.*)?\(VP\(VB\w{0,1} eat[s]?\).?(\(NP .*)
```

Figura 3.4: Segmentos a recuperar de patrón *eat*: NP del sujeto en azul, verbo en rojo, NP del complemento en verde.

1. Obtener el NP anterior al verbo (figura 3.5):

```
ROOT (S (NP (NNS Bears)) (VP (VBP eat) (NP (DT the) (NNS meat)))))
```

Figura 3.5: Oración capturada - patrón eat.

2. Utilizar como frontera o delimitador el verbo en análisis (figura 3.6):

```
ROOT (S (NP (NNS Bears)) (VP (VBP eat) (NP (DT the) (NNS meat)))))
```

Figura 3.6: Oración analizada con detección del verbo.

3. Recuperación del grupo NP a la derecha del verbo (figura 3.7):

```
ROOT (S (NP (NNS Bears)) (VP (VBP eat) (NP (DT the) (NNS meat)))))
```

Figura 3.7: Oración totalmente analizada - patrón eat.

Luego de capturar estas tres partes, podemos separarlas y obtener un término a definir y su definición. Las palabras en verde corresponden al término a definir, mientras que las partes en azul y rojo se utilizan para formar la definición. En este caso, dado que el verbo es "eat", se asume que lo que se va a definir es una comida. Por lo tanto, llegamos a la siguiente formación:

$$\prec \textit{[NP (verde)]: Food that [NP (Azul)] [verb (rojo)]} \succ$$

Por lo tanto, para el ejemplo anterior, obtenemos:

$$\prec \textit{The meat: Food that Bears eat} \succ$$

Patrón *is called:* La extracción de los grupos constituyentes de una oración que cumple con este patrón, se hace a partir de la expresión regular:

```
.*\(S (\(NP .*)(\(S.*)?\(VP \(VB\w{0,1} is\) \(\w{2,3} \(\w{2,3} called\).*\(S\s(\(NP\s.*)\)\)
```

Figura 3.8: Segmentos a recuperar de patrón *is called*: NP del sujeto en azul, verbo en rojo, NP del complemento en verde

Para ejemplificar este patrón, analicemos la oración:

One kind of green apple is called Granny Smith.

1. Obtención del NP inmediato anterior al verbo (figura 3.9):

```
ROOT (S (NP (NP (CD One) (NN kind)) (PP (IN of) (NP (JJ green) (NN apple)))) (VP (VBZ is) (VP (VBN called) (S (NP (NNP Granny) (NNP Smith))))))
```

Figura 3.9: Extracción del NP (segmento azul).

2. Utilizar como frontera o delimitador el verbo (figura 3.10):

```
ROOT (S (NP (NP (CD One) (NN kind)) (PP (IN of) (NP (JJ green) (NN apple)))) (VP (VBZ is) (VP (VBN called) (S (NP (NNP Granny) (NNP Smith))))))
```

Figura 3.10: Captura del verbo (segmento rojo).

3. Recuperación del grupo NP a la derecha del verbo (figura 3.11):

```
ROOT (S (NP (NP (CD One) (NN kind)) (PP (IN of) (NP (JJ green) (NN apple)))) (VP (VBZ is) (VP (VBN called) (S (NP (NNP Granny) (NNP Smith))))))
```

Figura 3.11: Extracción del primer NP a partir del verbo funcionando de delimitador (segmento verde).

Una vez que hemos capturado estas tres partes, podemos separarlas y extraer un término a definir junto con su definición. Las palabras resaltadas en azul representan el término a definir, mientras que las partes resaltadas en rojo y verde se utilizan para construir la definición. Por lo tanto, llegamos a la siguiente formación:

$\prec$ *[NP (azul)]: Something that [is called (rojo)] [NP (verde)]* $\succ$

Por lo tanto, para el ejemplo anterior, obtenemos:

$\prec$ *One kind of green apple: Something that is called Granny Smith* $\succ$

Los patrones basados en *regex* siguen el algoritmo 1 para la recuperación, generación y filtro de los pares retornados.

Algorithm 1 Extracción de pistas vía patrón de exp. regulares

```
Input story
Output set of clues
updated_story ← runCorreference(story, models)
sentences ← splitStory(updated_story)
results ← emptyList()
for sentence in sentences do
    pair ← findPattern(sentence)
    if pair exists? then
        clue, term ← getTuple(pair)
        context ← sentence
        results ← (clue, term, context)
    end if
end for
return results
```

3.1.2. Patrones SRL

Los patrones SRL se basan en el uso de la tarea **Semantic Role Labeling** utilizando el modelo entrenado por AllenNLP. La idea principal de estos patrones es identificar un verbo objetivo dentro de una oración y determinar si se cumplen diferentes reglas deseadas para los roles semánticos. Luego, a partir del verbo y los diferentes argumentos identificados, se forman los pares $\prec$ *definiendum: definición* $\succ$.

Ejemplo 3.2: Considere la siguiente oración:

Bobby is a good boy.

Como se muestra en la figura 3.12, a partir de SRL se obtiene:

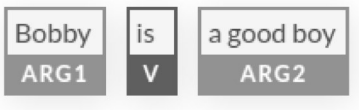

Figura 3.12: Roles semánticos del Ejemplo 3.2 obtenidos a partir de la demo de *AllenNLP* (2020).

Con esta información, se identifica un posible par candidato $\prec$ *definiendum: definición* $\succ$. El ARG1 "Bobby" es aquello que se quiere definir, y el ARG2 "a good boy" es su definición en el contexto de esa oración.

Esta información es muy útil al formar patrones, ya que permite aislar las partes de la oración que son relevantes. Al enfrentarse a una oración más compleja que la vista anteriormente, SRL permite extraer los pares

deseados.

Ejemplo 3.4: Si se tiene:

However, it is known that Bobby is a good boy and he likes to play football.

Se detectan distintos roles para distintos verbos, estos se presentan en las figuras 3.13, 3.14, 3.15 y 3.16.

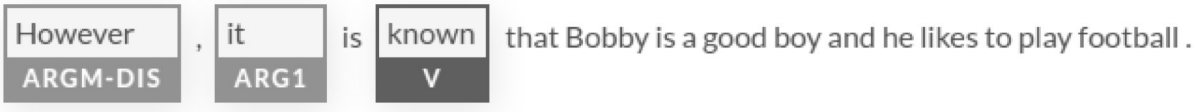

Figura 3.13: Roles semánticos del Ejemplo 3.4 para el verbo "known".

Figura 3.14: Roles semánticos del Ejemplo 3.4 para el verbo "is".

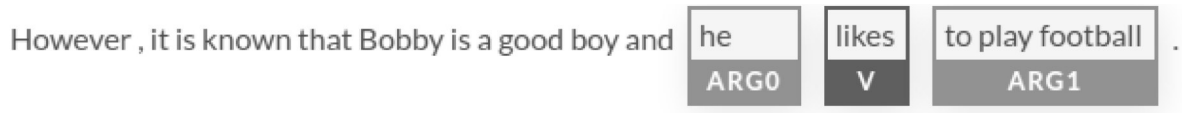

Figura 3.15: Roles semánticos del Ejemplo 3.4 para el verbo "likes".

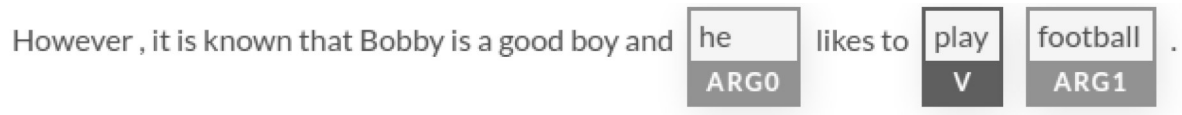

Figura 3.16: Roles semánticos del Ejemplo 3.4 para el verbo "play".

A pesar de la complejidad de los verbos y roles que puedan aparecer en la oración, todavía es posible identificar el patrón deseado para extraer la definición deseada, como se muestra en el ejemplo presentado.

En todos los patrones se busca una combinación específica de roles semánticos. Estas combinaciones pueden ser (ARG0, ARG1) y/o (ARG1, ARG2). A partir de ahora, cuando se haga referencia a estas combinaciones, se llamará *argA* al primer elemento de esa tupla y *argB* al segundo elemento de la tupla.

A continuación se detallan distintos patrones que se basan en esta herramienta.

Patrón *to be* Al contar los verbos más utilizados en los cuentos analizados, se identificó que las conjugaciones más frecuentes eran del verbo "to be", como "is" y "are". Estos verbos se usan comúnmente para describir lo que algo es, por lo que resulta conveniente utilizar un patrón que involucre estos verbos, ya que la relación entre

un definiendum y su definición es la misma: explicar lo que algo es.

A partir de este razonamiento se forma un nuevo patrón. Este patrón se basa en encontrar conjugaciones del verbo "to be", específicamente "is", "are", "was" o "were". Luego se identifican los roles semánticos que acompañan a este verbo en busca de una combinación de las conjugaciones mencionadas. Si se cumplen todas las condiciones, es decir, si se encuentra una de las conjugaciones del verbo "to be" junto a una combinación de roles semánticos válida, se formará el par $\prec$ *definiendum: definición* $\succ$ de la forma $\prec$ *argA: argB* $\succ$, donde el *argB* describe al *argA*. Un ejemplo de este patrón se usó anteriormente para describir los patrones SRL en el Ejemplo 3.2.

Es importante tener en cuenta que las conjugaciones del verbo "to be" no siempre se usan para describir lo que algo es o fue. Todas las conjugaciones que se utilizan en este patrón también pueden ser utilizadas como auxiliares de otro verbo en el presente o pasado continuo, o como auxiliar de una voz pasiva junto con un participio. Por ejemplo, podría haber una oración como "Bobby is playing football", en la cual es importante distinguir que en este caso el verbo "is" no se usa para representar lo que Bobby es, sino algo que está haciendo. Los roles semánticos pueden ayudar a identificar estos casos en los que uno de los verbos que se necesita está en la oración, pero no cumple la función requerida. En este caso, no hay roles asociados al verbo "is", pero sí existen roles asociados al verbo "playing", aunque no son relevantes para este patrón.

Patrón *have* El verbo "have" es otro de los verbos que se utilizó con frecuencia en una gran cantidad de cuentos del conjunto de datos de ReadWorks. A continuación, se analizan algunas oraciones donde aparece este verbo:

- **Oración 3.1:** Chameleons have super sight.
- **Oración 3.2:** Cleopatra had beauty and charm.
- **Oración 3.3:** England had 13 colonies.
- **Oración 3.4:** Mae Jemison had wanted to visit space since she was a child.

En las primeras tres oraciones, se puede observar que el verbo "have" refiere a algo que alguien o algo posee. Al encontrar los roles semánticos que acompañan al verbo "have", se valida que siempre se cumple que lo que se tiene o se tuvo corresponde a la clasificación *ARG1* y quien lo tiene es el *ARG0*. Por lo tanto, parece posible formar un patrón en el cual el definiendum corresponda a lo que se tiene, y la definición se pueda construir a partir de una descripción de lo que se tiene y de quién lo tiene.

Por ejemplo, al analizar la **Oración 3.1**, se podría identificar que "super sight" es algo que se debe definir y, en el contexto de esa oración, una forma de definirlo sería "Sense that Chameleons have".

El mismo razonamiento se puede utilizar para la segunda oración, formando el par $\prec$ *definiendum: definición*$\succ$ de la forma $\prec$ *beauty and charm: Features that Celopatra had* $\succ$.

En la tercera oración, se tiene a "13 colonies" como ARG1. No es fácil encontrar una categoría o descripción específica para este caso, pero se podría utilizar de manera más general la palabra "Thing" que, según el diccionario de Cambridge, toma los siguientes significados: *"used to refer in an approximate way to an idea, subject, event, action, etc."* o *"used to refer in an approximate way to an object or to avoid naming it"*. Por lo tanto, si se utiliza "thing" como categoría, se podría formar el par $\prec$ *definiendum: definición* $\succ$ como $\prec$*13 colonies: Thing that England had* $\succ$. Más adelante, se reemplaza el término "Thing" por "Something" ya que suena más natural.

Notemos que en la cuarta oración también aparece una conjugación del verbo "have", pero en este caso actúa como auxiliar para permitir el uso del verbo "wanted" en pasado perfecto. Al analizar los roles semánticos de esta oración, se puede observar que el verbo "had" no tiene ninguno asociado, lo que termina actuando como filtro al generar la pista.

Al analizar todos estos casos, se puede formar un patrón en el que se busca encontrar conjugaciones del verbo "have". En específico, se buscan las conjugaciones "have", "has" o "had". Siguiendo los pasos definidos para los patrones basados en SRL, además de encontrar uno de estos verbos en la oración, se busca que los roles semánticos para el verbo sean ARG0 y ARG1. Si se cumplen las condiciones, se puede formar el par $\prec$ *definiendum: definición* $\succ$ de la siguiente manera:

$$\prec \textit{[Arg1]: Something that [Arg0] [verb]} \succ$$

Para simplificar el capítulo, se asumirá que siempre se utiliza la palabra "Something" como categoría del ARG1, ya que en algunos casos no se tiene una palabra específica para referirse a lo que se quiere definir. En la sección 3.4.3, se explorará una siguiente etapa en el pipeline completo donde se intenta buscar una categoría más específica para el definiendum.

Patrón *like* En este patrón se intenta capturar las oraciones en las cuales la palabra "like" actúa como un verbo. Recordemos que "like" puede ser usada como verbo para expresar un gusto o un sentimiento, pero también puede ser usada como preposición para ejemplificar o explicitar semejanza.

Podemos reflejar estos casos en las siguientes dos oraciones:

Bobby likes to play basketball.
Bobby plays sports like basketball.

En la primera oración, "likes" actúa como verbo para describir que a "Bobby" *le gusta* jugar al básquetbol. En la segunda oración, la palabra "like" actúa como preposición para describir que el básquetbol es uno de los deportes que le gusta a Bobby. Si analizamos los roles semánticos de la primera oración, obtenemos el siguiente resultado, donde like solo es detectado como verbo en el caso correspondiente:

Figura 3.17: Ejemplos de roles semánticos en sus oraciones, obtenidos a partir de la demo de *AllenNLP* (2020).

Utilizando los roles semánticos de las oraciones, podemos diferenciar estos dos casos para formar un patrón e identificar los pares ≺ *definiendum: definición* ≻ de la siguiente manera:

≺ *[Arg1]: Something that [Arg0] likes* ≻

Para la primera oración del ejemplo 3.17 podemos formar el par ≺ *to play basketball: Something that Bobby likes* ≻.

Otro verbo que se usa para describir emociones satisfactorias es el verbo *love*. Por lo general se usan de forma similar con la diferencia de que este segundo verbo describe emociones más intensas. Se puede utilizar esta similitud para extender el patrón y capturar los dos verbos. En concreto todas los verbos y conjugaciones que se utilizan en este patrón son *like, likes, liked, love, loves, loved*. Si se encuentra uno de esos verbos en la oración y se identifica que existe el par de argumentos Arg0 y Arg1, se forma el par ≺*definiendum: definición*≻:

≺ [*Arg1*]:*Something that* [*Arg0*] [*like* | *likes* | *liked* | *love* | *loves* | *loved*] ≻

Patrón *live* Este patrón se captura mediante el verbo "live", el cual se usa usualmente para identificar dónde o cuándo alguien o algo vive o vivió. Para introducir el patrón veamos los siguientes ejemplos:

1. Aztecs lived in Mexico.
2. Dinosaurs lived in prehistoric ages.

En la primera oración, podemos identificar que el verbo *lived* se utiliza para identificar el *lugar donde* los aztecas vivieron. Por otro lado, en la segunda se utiliza para identificar *cuándo* vivieron los dinosaurios.

Siguiendo la lógica de todos los patrones SRL, parece que debería haber alguna forma de definir "Mexico" para la primera oración y "prehistoric ages" para la segunda. Pero parecería que estas definiciones no se

formarían de la misma forma, ya que se quiere explicar que "Mexico" *es el lugar donde* los aztecas vivieron y "prehistoric ages" *es la época cuando* los dinosaurios vivieron. Por lo tanto, habría que poder separar estos dos casos. Este problema se puede resolver nuevamente con la ayuda de los roles semánticos. Analicemos las dos oraciones:

Figura 3.18: Ejemplos de roles semanticos en sus respectivas oraciones

Se observa en la figura 3.18 que en estos casos se tiene un Arg0, tal como se ha visto en muchos otros patrones, con la diferencia de que en estos no se encuentra junto a un Arg1. En estos ejemplos, se presentan los "modificadores", los cuales sirven para proporcionar información adicional sobre algún término en la oración. Existen diferentes tipos de modificadores, pero en este caso nos interesa en particular los modificadores de tiempo y lugar, estos se representan en la herramienta de AllenNLP mediante las etiquetas ARGM-LOC y ARGM-TMP, respectivamente.

Es posible distinguir de forma automática los dos tipos de oraciones que se presentaron anteriormente, y así formar el patrón *live* de dos formas:

1. Si se encuentra una conjugación del verbo "live", un Arg0 y un ARGM-LOC, formamos el par $\prec$ *definiendum: definición* $\succ$ como: $\prec$ [*ARGM-LOC*]: *Place where* [*Arg0*] [*live* | *lives* | *lived*] $\succ$.

2. Si se encuentra una conjugación del verbo "live", un Arg0 y un ARGM-TMP, formamos el par $\prec$ *definiendum: definición* $\succ$ como: $\prec$ [*ARGM-TMP*]: *Time when* [*Arg0*] [*live* | *lives* | *lived*] $\succ$.

Por lo tanto, si aplicamos esto a las oraciones previamente vistas, formamos las definiciones $\prec$ *Mexico: Place where Aztecs lived* $\succ$ y $\prec$ *Prehistoric ages: Time when dinosaurs lived.* $\succ$.

Patrón General SRL (Patrón NER SRL) Aunque este patrón utiliza la herramienta SRL que hemos visto en otros patrones, tiene un enfoque diferente. En lugar de buscar un patrón a partir de un verbo específico, buscamos que se cumplan reglas específicas para cualquier verbo que pueda aparecer en el cuento.

Hemos visto en otros patrones de SRL que, después de segmentar las oraciones, las recorremos en busca de un verbo específico para nuestro patrón. A partir de ese verbo, buscamos ciertos roles dentro de la oración para formar una pista que tenga sentido para ese verbo.

Al analizar estos patrones, se ha observado una estructura común y se han identificado ciertos términos que se están pasando por alto. De ahí surge la idea de no solo crear patrones específicos para cada verbo, sino de desarrollar un patrón que sea aplicable a cualquier verbo que se encuentre en el texto, logrando así una mayor cantidad de pistas sin la necesidad de analizar todos los verbos posibles. Para ejemplificar, consideremos algunos verbos que hemos analizado previamente utilizando oraciones extraídas de los cuentos "A Shell is Great for Protection" y "A Kangaroo's Life Cycle":

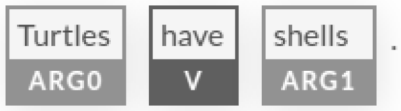

Figura 3.19: Roles semánticos de la oración "Turtles have shells.".

Figura 3.20: Roles semánticos de la oración "A grown-up kangaroo can be bigger than a person.".

Tomando como ejemplo la oración mostrada en la figura 3.19, utilizando el patrón "have" podemos obtener el siguiente par: $\prec$*Shells*: *Something that turtles have*$\succ$. Sin embargo, podemos notar que este no es el único término que podemos definir. Si utilizamos esa oración, también podemos definir a "Turtles" de una forma bastante simple. Podemos crear una definición de la forma $\prec$*Turtles*: *Something that have shells.*$\succ$. Por otro lado, con la oración mostrada en la figura 3.20, también podemos ver que se puede formar una definición para "Kangaroo" de la forma $\prec$*Kangaroo*: *Something that that can be bigger than a person.*$\succ$. Este nuevo patrón puede capturar definiciones para oraciones con verbos que no habían sido contemplados.

Se puede apreciar que estas dos definiciones siguen un mismo formato (utilizando la nomenclatura previamente definida de *ArgA* y *ArgB*): se toma el *ArgA* y se lo define como una combinación del verbo, el *ArgB* y un verbo modal en caso de haberlo. Por lo tanto, a partir de esta observación, surge la idea de formar un nuevo patrón. Se recorren las oraciones de forma habitual y cada vez que se encuentra un verbo, se identifican los roles semánticos asociados a ese verbo. Si cumple una de las combinaciones de roles semánticos antes mencionadas, se forma una definición de este estilo:

$$\prec [ArgA]\text{: } \textit{Something that } (\text{opt}[\textit{argm-mod}])\ [\textit{verb}]\ [ArgB] \succ$$

Así como este patrón tiene la ventaja de poder obtener muchas pistas sin tener que analizar patrones diferentes para cada tipo de oración, por otro lado tiene algunas desventajas. En primer lugar, las definiciones formadas pueden llegar a ser muy similares a las oraciones, haciendo que las pistas sean fáciles de resolver. En segundo lugar, las pistas formadas pueden ser muy genéricas y no muy precisas, lo que puede generar confusiones o ambigüedades al momento de resolverlas.

El enfoque de SRL se basa en el algoritmo 2, con tres bloques bien definidos: preprocesamiento, generación de la tupla y postprocesamiento (curación y validación de la salida).

Algorithm 2 Extracción de pistas vía patrón SRL general.

```
Input story, NLPmodels
Output set of clues
sentences ← preProcess(story, NLPmodels)
for sentence in sentences do
    verbs ← predictVerbFromSRLPredictor(srlPredictor)
    for verb in verbs do
        if verb is an expected verb then
            [arg0, arg1] ← retrieveArgsFromVerb(verb)
            if argumentsExist(arg0, arg1) then
                term ← arg0
                clue ← buildTerm(arg1, verb)
                if resultIsCompliance(clue, term, verb, sentence) then
                    results ← zipResult(clue, term, sentence)
                end if
            end if
        end if
    end for
end for
return results
```

3.1.3. Patrones extendidos

Hasta el momento, todos los patrones vistos para extraer pares $\prec$ *definiendum: definición* $\succ$ funcionan en el contexto de una sola oración. Cada oración se utiliza de forma aislada, y la única relación entre las diferentes oraciones de un texto se establece al momento de resolver las correferencias. Sin embargo, en un intento por formar pistas más complejas para el crucigrama, se presentan en esta sección diferentes patrones en los cuales las pistas se forman relacionando oraciones en lugar de aislarlas.

Ejemplo 3.5: A modo de introducción, considerar el siguiente fragmento de texto:

Mount Rushmore is a rock sculpture. It was carved into a mountain from 1927 to 1941.

De forma análoga a los otros patrones, procedemos de las siguiente forma: primero, resolver las correferencias; luego, segmentar cada oración y extraer los pares de definiciones.

En la segunda oración del ejemplo, existe una correferencia entre "Mount Rushmore" e "It". Después de resolverla y separar las oraciones, se tiene:

Mount Rushmore is a rock sculpture
Mount Rushmore was carved into a mountain from 1927 to 1941

Utilizando el patrón "to be" y el patrón general de SRL, podemos extraer dos pares de definiciones: $\prec$ *Mount Rushmore: a rock sculpture* $\succ$ y $\prec$ *Mount Rushmore: carved into a mountain* $\succ$. Si estamos pensando en formar un crucigrama con lo que se ha extraído, es razonable pensar que no queremos repetir términos en las pistas que

se ofrecen, es decir, preferimos evitar que haya dos pistas diferentes para "Mount Rushmore". Además, ninguno de los dos pares de definiciones extraídos parece especialmente interesante por sí solo. Por lo tanto, podríamos pensar que es mejor unirlos y formar un solo par: $\prec$ *Mount Rushmore: a rock sculpture that was carved into a mountain* $\succ$.

En este razonamiento se basan los patrones extendidos. Utilizan como base otros patrones, y cuando se encuentra un par $\prec$ *definiendum: definición* $\succ$ con ciertos patrones, se busca en el resto del texto otras oraciones donde exista una correferencia del definiendum y se intenta encontrar una *acción* para ese definiendum. . Si se encuentra una acción, se unen las dos oraciones.

Para entender qué es una "acción", recordar el patrón general de SRL. En este patrón, se buscaba cualquier verbo dentro de una oración y luego se verificaba si había alguna combinación de roles semánticos "válida". Luego, se encontraba una categoría para el definiendum y se formaba una definición de la forma: [Categoría] + that + [verb] + [argB]. En los patrones extendidos, se utiliza el mismo concepto, pero en lugar de usar una "categoría" para referirse al término a definir, se usa una definición previamente extraída con otro patrón.

Ejemplo 3.6: Consideremos el siguiente fragmento de texto:

...Bears are apex predators. They eat small mammals, like foxes.

En primer lugar, se extrae la definición de "Bears" como "apex predators". Luego, en la segunda oración, se encuentra la acción que realizan los osos, que es "eat small mammals, like foxes". En lugar de formar dos definiciones separadas utilizando el patrón *to be* y el patrón general de SRL, podemos unirlas para formar una definición más completa: $\prec$ *Bears: apex predators that eat small mammals, like foxes* $\succ$. De esta manera, creamos una pista más completa que utiliza diferentes partes del texto y no necesitamos encontrar una categoría adicional para "Bears".

Por lo que la regla general es que luego de encontrar una "definiciónA" y una "acciónA" para un mismo término, se unen formando una definciónB: [definiciónA] + that + [acciónA].

Los patrones base en los cuales se intenta "extender" son el patrón "to be" y el patrón "is called" ambos pertenecientes a los patrones SRL. Estos dos patrones tienen en común que cuando se encuentra una oración que los cumpla, no se necesita crear una lógica extra para formar la definición como se vio en otros patrones, la definición está en su totalidad como un argumento de los roles semánticos y por eso forman buenos candidatos como patrones para extender. Por ejemplo, el resto de patrones SRL siempre se forman de la forma [Categoría] + that + [acción], siendo la categoría algo fijo o creado en el momento. Al extender estos patrones, las definiciones resultantes tendrían la estructura [Categoría] + that + [acciónA] + that + [acción B], las cuales no forman buenas definiciones. Por eso solo se usan los patrones "to be" e "is called", donde la definición ya funciona como una categoría o un sinónimo.

3.1.4. Patrones basados en entidades nombradas

Estos patrones no entran en ninguna de las otras categorías.

Patrón GPE ORG: Es común que en este tipo de textos se hable de lugares donde ha ocurrido un evento, donde vive una persona u otros casos similares. En este patrón se trata de identificar cuando se habla de uno de estos lugares y formar una definición a partir de su contexto.

Ejemplo 3.7: Para introducir el patrón veamos la siguiente oración:

Finlay lived in Cuba

Este fragmento fue extraído del cuento "Mosquito Man", donde se habla del médico y científico Juan Carlos Finlay. Lo que nos interesa de esta oración es el lugar donde Finlay vivió, es decir, Cuba.

Una vez identificado que Cuba es un lugar, podemos usar el resto de la oración para formar una definición. Con esta información, se forma el par $\prec$ *definiendum: definición* $\succ$ de la forma $\prec$ *Cuba: Place where Finlay lived* $\succ$.

Analizando una gran cantidad de cuentos, se identifica que la estructura de esta oración es común. Intentamos identificar oraciones en las cuales sucede algo y al final de la oración se explica dónde sucede esto. En concreto, nos enfocamos en oraciones con la siguiente estructura:

[Evento] + in + [Lugar]

Una vez identificado esto, formamos el par $\prec$ *definiendum: definición* $\succ$ de la forma $\prec$ *Lugar: Place where [Evento]* $\succ$.

Ejemplo 3.8: Podemos ver otro ejemplo en esta oración:

Many chili peppers are grown in Mexico

Donde identificamos el evento "Many chili peppers are grown" y el lugar donde ocurre esto, que es "Mexico", y así se forma una definición para México de la forma: *Place where many chili peppers are grown.*

Hasta este momento, se ha omitido la tarea de reconocer que Cuba o México son lugares geográficos. De manera similar al patrón general de SRL, este patrón utiliza como auxiliar la tarea de reconocimiento de entidades nombradas (NER). Para resolver esta tarea, utilizamos el módulo de Spacy, que es capaz de identificar países, ciudades y estados, etiquetándolos con la categoría "GPE", una abreviación de *Geopolitical entity.*

Como indica el nombre de este patrón, esta es solo una de sus caras. Aprovechando todo lo que hemos visto hasta ahora, identificamos otro tipo de oraciones en los cuentos con una estructura muy similar. Existen muchos casos en los cuales se habla de dónde ocurre un evento, pero no necesariamente se refieren a un lugar. Como se ve en el siguiente fragmento:

Lebron James plays basketball at NBA

Notamos una cierta similitud entre esta oración y lo visto hasta el momento. En este caso tenemos el evento "Lebron James plays basketball" y el lugar donde ocurre, que es "NBA". Hay que tener en cuenta que hasta

ahora solo se identificaban entidades geográficas o geopolíticas, y en este caso, NBA no entra en esas categorías, por lo que el patrón hasta el momento no es capaz de identificarla para definirla.

Para ampliar el patrón, el módulo de NER de Spacy proporciona una clasificación para la NBA, que en este caso se clasifica como ORG. Esta categoría se utiliza para clasificar compañías, agencias o instituciones, es decir, organizaciones en general.

Es importante señalar que en este caso la preposición para referirse a la NBA no es "in" sino "at". Por lo tanto, ampliamos el patrón para identificar los casos:

[Evento] + [in | at] + [Lugar | Organización]

Por lo tanto, también necesitamos modificar las definiciones formadas, y en el caso de que se identifique una organización y no un lugar geográfico, se forma la definición: *Organization where [Evento].*

Finalmente, podemos formar una definición para NBA de la siguiente manera: *Organization where Lebron James plays basketball.*

3.2. Métodos basados en QA (*Question Answering*)

En este método se utiliza un modelo pre-entrenado de *Question Answering* (QA) al cual se le puede ingresar un texto, hacer preguntas sobre el mismo y obtener respuestas basadas en el contexto previamente ingresado. De esta manera, se pueden hacer preguntas sobre posibles términos a definir y el modelo proporciona pistas para el término.

Sistema de Q&A:

Para ilustrar el método, se puede analizar el siguiente cuento "The Great Sphinx" extraído de ReadWorks:

The Great Sphinx is one of the largest statues in the world. It is in Egypt. Egypt is a country in Africa. The Great Sphinx has the head of a man but the body of a lion. You will see its nose is missing. Nobody knows for sure what happened to it. This amazing statue is over four thousand years old! It is one of the oldest statues in the world. The Ancient Egyptians made it. They made the statue by cutting into a huge rock. It probably wasn't easy to make a statue that big!

A simple vista, se puede elegir el término "Egypt" como un buen candidato para definir. Para obtener una definición, se puede preguntar al modelo de QA de HuggingFace[2] "What is Egypt?", definiendo como el contexto de entrada el cuento completo y obtener la respuesta "A country in Africa", por lo que con esa simple pregunta podemos obtener un par $\prec$ *definiendum: definición*$\succ$ de la forma $\prec$ *Egypt: A country in Africa* $\succ$.

Sin embargo, hay un paso intermedio que se debe considerar: elegir un término. Aunque es fácil leer el cuento y seleccionar términos para definir, se necesita una forma automática de encontrarlos. La solución elegida es

[2]https://huggingface.co/tasks/question-answering

pasar el cuento por los módulos de reconocimiento de entidades de Spacy y de HuggingFace. Por ejemplo, si se ejecutan estos módulos con el cuento anterior, se obtiene la siguiente lista de entidades:

- *Ancient Egyptians*
- *Egypt*
- *Great*
- *Africa*

Se puede ver que el término "Egypt" seleccionado anteriormente está en esta lista. Al hacer la misma pregunta para los otros términos, se obtienen las siguientes respuestas:

- *What are Ancient Egyptians?* ***Respuesta:*** *They made the statue by cutting into a huge rock.*
- *What is Great?* ***Respuesta:*** *One of the largest statues in the world.*
- *What is Africa?* ***Respuesta:*** *Egypt is a country.*

Se puede notar que algunas respuestas no parecen ser muy precisas. El modelo de QA de HuggingFace no siempre responde correctamente a estas preguntas, por lo que se debe ser cauteloso. También se puede observar que las entidades reconocidas para definir tampoco fueron las mejores. Por ejemplo, la definición encontrada para "Great" tiene sentido dentro de este contexto, pero hubiera sido mejor si las entidades reconocidas fueran más precisas, como "The Great Sphinx'. Por lo que se debe tener cuidado con las limitaciones de estas herramientas.

Para tratar de filtrar estas pistas, utilizamos una propiedad que nos proporciona este modelo de QA. Cada vez que se hace una pregunta, el modelo devuelve un porcentaje de confianza en la respuesta. Por ejemplo, para las preguntas recién hechas, el modelo ha devuelto los siguientes valores redondeados de confianza:

- *What is Egypt? A country in Africa.* ***Confianza:*** *0.57*
- *What are Ancient Egyptians? They made the statue by cutting into a huge rock.* ***Confianza:*** *0.21*
- *What is Great? One of the largest statues in the world.* ***Confianza:*** *0.63*
- *What is Africa? Egypt is a country.* ***Confianza:*** *0.45*

Estos valores pertenecen al rango entre 0 y 1. Cuanto más cerca de 0 significa que el modelo no confía en la respuesta que devolvió, y cuanto más cerca de 1, más seguro está de que su respuesta es correcta. Por lo tanto, podemos establecer un umbral de confianza para descartar las respuestas menos seguras. Por ejemplo, si el umbral es 0.5, solo utilizaríamos las pistas generadas para "Egypt" y "Great".

Como se verá más adelante en nuestra estrategia general para el sistema completo de generación de crucigramas, hemos decidido crear una gran cantidad de patrones y métodos para obtener pistas. En esta estrategia, queremos que la precisión de las pistas sea alta, lo que significa que preferimos que los métodos generen pocas

pistas, pero que sean de buena calidad. Por lo tanto, hemos elegido 0.85 como nuestro umbral de confianza para aceptar una pista generada por este método.

Al iterar sobre este método, también nos dimos cuenta de algunas mejoras que podríamos implementar. En primer lugar, al reconocer las entidades, las herramientas de Spacy y HuggingFace también nos proporcionan una categoría para cada entidad identificada. Por ejemplo, si encontramos el nombre de una persona en un cuento, estas herramientas nos devolverán el nombre junto a la categoría "Person". Utilizando esta información, podemos formular preguntas más precisas y efectivas al modelo de QA, como por ejemplo preguntar quién es esa persona en lugar de qué es esa persona, utilizando la pregunta: *Who is [Nombre de Persona]?* Estas preguntas suelen producir respuestas más precisas y con un índice de confianza mayor.

Por otro lado, también nos dimos cuenta de que en algunos cuentos, al hacer una pregunta sobre una entidad, el nombre de la entidad también aparece en la respuesta. Esto puede ser problemático, ya que no queremos incluir pistas en el crucigrama en las que la respuesta ya esté contenida en la propia pista. Si detectamos que esto ocurre, descartamos la pista.

La extracción, generación y filtro de tuplas mediante QA se explica en el algoritmo 3.

Algorithm 3 Recuperación de pistas vía QA

```
Input story
Output set of clues
entities ← retrieveEntitiesFromStory()
for entity in entities do
    if entity is person? then
        question ← who is {entity}?
    else
        question ← what is {entity}?
    end if
    answer ← retrieveAnswerFromStory(question)
    if answer is compliance? then
        results ← answer
    end if
end for
return results
```

3.3. Resumen de Patrones

En las secciones anteriores se presentaron todos los métodos y sus patrones que forman parte de la implementación final del pipeline. A modo de resumen, se presentan los patrones descritos previamente:

- **Patrón eat:** Este patrón se basa en reconocer oraciones que contengan el verbo "eat" y luego obtener los pares utilizando análisis de constituyentes y expresiones regulares.

- **Patrón is called:** Patrón similar al patrón "eat", reconoce oraciones que contengan el verbo "is called" y luego obtiene los pares mediante análisis de constituyentes y expresiones regulares.

- **Patrón like:** Se basa en reconocer oraciones con el verbo "like". Se implementa utilizando roles semánticos para analizar la estructura argumental del verbo y extraer las pistas correspondientes.
- **Patrón live:** Se reconocen oraciones con el verbo "live". Mediante el uso de análisis de constituyentes y expresiones regulares, se construyen los pares buscados.
- **Patrón to be:** Este patrón se basa en reconocer oraciones que contengan conjugaciones del verbo "to be" (is, was, were). Se implementa utilizando roles semánticos para identificar la estructura argumental del verbo y formar los pares de definiendum y definición correspondientes.
- **Patrón have:** Este patrón se basa en reconocer oraciones con el verbo "have". Utilizando roles semánticos, se identifica la estructura argumental del verbo para luego extraer los pares deseados.
- **Patrón live SRL:** Este patrón busca encontrar pistas en oraciones en las que el patrón "live" no haya obtenido resultados. Para esto, se analiza la estructura argumental del verbo "live" utilizando roles semánticos y se utilizan estos argumentos para generar los pares buscados.
- **Patrón ner SRL:** Este patrón busca generalizar los patrones que utilizan roles semánticos. Para lograr esto, se deja de buscar un verbo en particular y se buscan combinaciones de argumentos en los verbos y entidades nombradas relacionadas para generar las pistas necesarias.
- **Patrón gpe org:** Este patrón se basa en reconocer entidades con nombre, especialmente en el caso de lugares y organizaciones relacionadas con eventos mencionados en el texto. Posteriormente, se generan las definiciones de estas entidades nombradas a partir de los eventos.
- **Patrón to be extended:** Este patrón es una generalización del patrón "to be" y busca enriquecer las pistas mediante distintas oraciones que se refieren a lo mismo en el texto.
- **Patrón is called extended:** Es una ampliación del patrón "is called". Se busca enriquecer las pistas mediante distintas oraciones que se refieren a lo mismo en el texto.
- **Patrón QA:** Este patrón busca generar pistas utilizando un sistema de preguntas y respuestas (QA) sobre los textos. Para esto, se formulan una serie de preguntas sobre las entidades nombradas en el texto y se utilizan las respuestas obtenidas como definiendum.

3.4. Generación de pistas

Hasta este punto se explicaron diferentes métodos y patrones para la creación de pares $\prec$ *definiendum: definición* $\succ$. Estos pares no siempre están listos para ser usados dentro de un crucigrama como pistas. En esta sección se explicarán diferentes etapas de postprocesamiento antes de dejar disponibles estas definiciones como pistas para un crucigrama. Finalmente todos estos métodos y funciones de generación de pistas se unen en el pipeline final, explicado en detalle en 3.5.

3.4.1. Filtrar términos

Como se mencionó anteriormente, existen casos en los que el resolvedor de correferencias no funciona como se desea, ya sea debido a un error en la herramienta o porque, dado el contexto, no es posible reemplazar ciertos pronombres por un referente. Un ejemplo de esto se ilustró en la sección 3.1 con el cuento "María Makes a Snake", donde no fue posible sustituir todos los pronombres. En la última oración"They laugh and laugh together.", el pronombre personal "They" no se resuelve, y en realidad debería haber sido reemplazado

por “Marco and María”. Para obtener pistas de calidad en un crucigrama valioso, es necesario filtrar términos que no tiene sentido incluir, como los pronombres demostrativos y personales “it, this, that, etc.”. Cuando se intenta crear una definición para uno de estos pronombres, se omite ese término y se continúa buscando otras definiciones.

Luego, en el proceso de generación de definiciones, es posible que algunas oraciones con pronombres sin resolver cumplan con las reglas o patrones establecidos. Por ejemplo, si se tuviera la oración “They are very good kids”, al analizar los roles semánticos se encontraría lo siguiente:

Figura 3.21: Roles semánticos de la oración “They are very good kids”.

En el caso de la oracion presentada en 3.21, la combinación de roles semánticos sería identificada por el patrón “to be” de SRL, lo que intentaría formar una definición de la forma $\prec$*They: Very good kids.*$\succ$. Sin embargo, esta definición no sería adecuada para ser utilizada en un crucigrama, debido a que el pronombre “They” no es un término concreto. Es por ello que, antes de agregar cualquier definición a las finales, se revisa que los términos estén permitidos y se filtran o descartan las definiciones que contengan pronombres demostrativos o personales sin resolver.

3.4.2. Aislar término

Puede que los lectores hayan notado una discordancia entre el funcionamiento de los crucigramas estipulados en los capítulos uno y dos y los pares de definiciones extraídos hasta el momento. En esta sección se estipula una restricción para los crucigramas: *los términos a resolver dentro del crucigrama deben ser de una sola palabra*, sin embargo, en el transcurso de este informe se han visto distintos ejemplos de definiendums en los cuales el término a definir no es una sola palabra si no un conjunto de ellas.

Por ejemplo, ciertos pares $\prec$ *definiendum: definición* $\succ$ extraídos con los distintos métodos son:

- **Ejemplo 1:** $\prec$ *A Snake: a reptile that moves its tail* $\succ$
- **Ejemplo 2:** $\prec$ *The five senses: sight , touch , smell , hearing , and taste* $\succ$
- **Ejemplo 3:** $\prec$ *Beauty and charm: Features Cleopatra had* $\succ$
- **Ejemplo 4:** $\prec$ *Solar system: the name for the sun , planets , and other smaller bodies* $\succ$
- **Ejemplo 5:** $\prec$ *Statue of Liberty: a symbol of freedom* $\succ$
- **Ejemplo 6:** $\prec$ *A large crown on her head: Something a woman has* $\succ$
- **Ejemplo 7:** $\prec$ *Ice cream: Food Bobby likes* $\succ$
- **Ejemplo 8:** $\prec$ *Super sight: Sense chameleons have* $\succ$

Surge la incógnita de cómo ajustar estos pares para que puedan ser utilizados en los crucigramas.

El primer paso es reducir los definiendums a una sola palabra, analicemos los ejemplos recientemente vistos. El primer caso es "A snake", analicemos su estructura:

*A(**DET**) Snake(**NN**)*: a reptile that moves its tail

A simple vista parece que el término que de verdad importa es el sustantivo "snake" mientras que el determinante "A" puede ser descartado. Recordemos que los determinantes son palabras que siempre acompañan a un sustantivo y no son términos que queremos que aparezcan como pistas de un crucigrama, por lo que en una primera instancia parece acorde remover todos los determinantes que aparezcan en los definiendums. Por lo que este caso en concreto se reduciria al nuevo par $\prec$ *snake: a reptile that moves its tail* $\succ$ y ya puede usarse en un crucigrama.

El segundo caso es "The five senses", aplicando lo que se acaba de ver en este caso se puede remover el determinante "The" y se obtiene "five senses". Este parece un *definiendum* más adecuado pero todavía podría ser más reducido. La pista (sight , touch , smell , hearing , and taste) podría usarse simplemente para describir a "senses" por lo que removemos el término "five" etiquetado como número y permanecemos solamente con el término "senses", un sustantivo nuevamente.

Parece que hay un cambio de lógica en este momento. Se debe decidir qué remover y qué quedarse. Una posible solución es quedarse solo con los sustantivos. Analicemos el siguiente ejemplo para intentar resolver este problema:

*Beauty(**NN**) and(**CC**) charm(**NN**)*: Features Cleopatra had

En este caso tenemos dos sustantivos únidos por la conjunción "and". Cabe mencionar que la categoría "features" para referirse a "beauty and charm" fue introducida de forma manual, pero si fuera necesario podría modificarse para referirse en singular a una sola característica de "Cleopatra". Bajo esta condición, cualquiera de los sustantivos podría usarse, por lo que podemos formar dos nuevos pares $\prec$ *Beauty: Feature Cleopatra had* $\succ$ y $\prec$ *Charm: Feature Cleopatra had* $\succ$. Esto es posible porque las conjunciones son palabras que unen palabras o frases, por lo que podemos detectar una nueva regla: si existen sustantivos separados por una conjunción, podemos separarlos en múltiples definiciones.

A pesar de todo, existen casos en los que no parece posible reducir un definiendum a una sola palabra. Veamos la estructura del Ejemplo 4:

*Solar(**NN**) system(**NN**)*: the name for the sun , planets , and other smaller bodies

En este ejemplo se tiene un *compound noun*, estos son sustantivos que están formados usualmente por la combinación de dos sustantivos o un adjetivo y un sustantivo, actuando como una sola unidad. Remover cualquier palabra del definiendum hace que la definición pierda su sentido. También tenemos el caso de $\prec$ *Statue of Liberty: a symbol of freedom* $\succ$, donde parece que el definiendum es "*irreducible*". Por lo que se puede concluir que en algunos casos, otro enfoque es necesario.

Para resolver esto, surge la idea de seleccionar una palabra del definiendum y usar el resto en su definición. Por ejemplo, para el caso de "solar system", podríamos elegir la palabra "solar" para ser el candidato a aparecer

en el tablero del crucigrama y formar la definición: $\prec$ *Solar: (_____ system) the name for the sun , planets , and other smaller bodies* $\succ$. Para el caso de "Statue of Liberty" notamos que la palabra "Statue" ya está en la definición, la palabra "of" es una preposición y como ya fue visto, no parece el término más adecuado para resolver en un crucigrama, por lo que nos deja "liberty", formando la nueva definición: $\prec$ *Liberty: (Statue of _____) a symbol of freedom* $\succ$.

En conclusión, para resolver el problema de definiendums formados por múltiples palabras, primero se intenta reducir el definiendum a una sola palabra, enfocándose en los sustantivos. En el caso de que el definiendum sea "irreducible", se elige unos de sus sustantivos según ciertas reglas y se incluye el resto de él en la definición.

3.4.3. Búsqueda de categorías

En capítulos anteriores, se ha mencionado que con diferentes métodos y patrones se forman pares $\prec$ *definiendum: definición* $\succ$ donde la definición tiene la forma "*Something that ...*". Al momento de razonar estos patrones se explicaba que se puede formar una definición clasificando el definiendum y explicando algo sobre este.

Por ejemplo, al describir el **patrón have** dentro de los patrones SRL se analizó la oración "*Chameleons have super sight.*" y a partir del verbo y los roles semánticos se podría llegar a formar la definición, $\prec$ *super sight: Sense that Chameleons have* $\succ$. En este ejemplo se sugirió que la categoría o clase del definiendum "*super sight*" es *Sense*. Esta sugerencia fue formada manualmente y se explicó que en algunos casos no es fácil pensar una categoría que describa al definiendum por lo que se la clasifica con el término genérico "*Something*".

Recordemos que el propósito de este trabajo es formar pistas para crucigramas para enriquecer la compresión lectora de niños, por lo que el término "*Thing*" no parece ser siempre la mejor opción para cumplir este propósito. Parece razonable tratar de clasificar estos términos en una categoría más acertada. En esta sección se explica el proceso para intentar clasificar estos términos de forma automática.

Existen diferentes procesos para intentar resolver este problema. Es importante tener en cuenta que una palabra puede tener un significado diferente según su contexto, por lo que es importante tener toda la oración al momento de clasificar.

En todos los métodos que se necesite clasificar al definiendum, se trata de clasificar siempre de la misma forma. El primer intento es haciendo uso de la tarea *Name Entity Recognition*. Como se explica en el marco teórico, esta es una tarea en la que se busca clasificar entidades nombradas. La herramienta usada en este trabajo para resolver esto es el módulo NER de Spacy. En este módulo existen distintas categorías en las que se puede clasificar a un término, en el proyecto se utilizaron las categorias presentadas en el cuadro 3.1.

Categoría	**Descripción**
ORG	Companies, agencies, institutions
NORP	Nationalities or religious or political groups
PERSON	People, including fictional
LOC	Geopolitical entity, i.e. countries, cities, states

Cuadro 3.1: Subconjunto de categorías del módulo NER de Spacy

En los casos que se clasifica como ORG y NORP, se remplaza el término “Something that” por “Group who”. En los casos donde se clasifica como PERSON se reemplaza el término “Something” por Person y cuando se clasifica como LOC se reemplaza por Place.

Es importante destacar que con este método sólo se identifican entidades nombradas por lo que muchos de los términos que busquemos clasificar van a quedar por fuera sin importar la eficacia de la herramienta utilizada.

El segundo método utilizado para clasificar es mediante la búsqueda de hiperónimo. Como mencionamos, no todos los términos van a ser clasificados mediante el reconocimiento de entidades nombradas como es el caso de “*super sight*”. Tampoco es fácil encontrar un sinónimo o hiperónimo de ese mismo término, pero podemos situar esta parte del procesamiento luego de haber aislado los términos como fue explicado en la sección 3.4.25. En este caso “super” es un adjetivo por lo que solo habría que buscar un hiperónimo de “sight”. Para resolver este problema se utiliza la herramienta WordNet de NLTK.

Como fue explicado previamente, esta base de datos léxica proporciona una estructura de grafo para las palabras, donde podemos obtener relaciones entre ellas basadas en su semántica. La idea inicial es reemplazar la palabra definiendum por otra que pueda ser utilizada en la definición. Es decir, se busca una palabra que pueda describir de alguna manera al definiendum, sin decir explícitamente qué es, o dicho de otra manera, una palabra que lo incluya. Demostraremos esto con ejemplos más claros, como se muestra en la tabla 3.2.

Definiendum	**Hiperónimo**
Cocodrile	Animal
Apple	Fruit
Guitar	Musical Instrument
Pineapple	Fruit
Elephant	Animal
Brain	Body Part

Cuadro 3.2: Definiendums y posibles hiperónimos de ejemplos simples.

Ahora, realizar esta tarea involucra cierto pensamiento sobre el significado de una palabra y sobre qué otra podría describirlo de forma clara. Muchas veces esto es fácil, pero en otros casos no lo es tanto, como se puede ver en la tabla 3.3, que consiste en ejemplos más complejos y menos comunes en el lenguaje infantil.

Definiendum	**Hiperónimo**
Beagel	Dog Breed
Polynomial	Mathematical Function
La Sagrada Familia	Cathedral
Laptop	Computer
Telescope	Optical Instrument
Cello	String Instrument

Cuadro 3.3: Definiendums y sus hiperónimos en ejemplos complejos.

Entonces se idea un algoritmo que permite abarcar la mayoría de los casos, teniendo en cuenta que debido a la baja dificultad de las palabras, se parte de una lista de palabras que constituyen las posibles categorías (se

muestran algunos ejemplos de categorías en la tabla 3.4) para los definiendums. El algoritmo diseñado consiste en:

1. Se define una lista prefija de categorías y el valor por defecto "Something".
2. Se itera sobre los hiperónimos de la palabra usando WordNet.
3. Cuando se encuentra un hiperónimo definido en la lista de categorías, se retorna esta palabra.
4. Si no se encuentra ningún hiperónimo en la lista, se sigue iterando hasta llegar al root de WordNet.
5. Si se llega al root y no se ha encontrado ningún hiperónimo en la lista, se devuelve "Something".

Podemos ver estos mismos pasos en el pseudocódigo 4. Un detalle del algoritmo es que en general, a medida que se itera sobre los hiperónimos, estas palabras son más abstractas y complejas, y tienden a alejarse de la definición o idea inicial de la palabra, por lo que se agrega un *threshold* de iteraciones al algoritmo, para no seguir iterando sobre palabras abstractas que no están incluidas en las categorías iniciales.

Categorías
animal
food
fruit
clothing
country
location
instrument
musical_instrument
plant
tool
activity
action
relative
feeling
city
region
sensation

Cuadro 3.4: Ejemplos de categorías.

A modo de demostrar cómo se utiliza este módulo en la aplicación, se brinda un ejemplo. Consideremos que tenemos dos definiendums: "carrot" y "skin". Debemos encontrar un hiperónimo para cada uno que se encuentre dentro de las categorías definidas. Por lo tanto, haremos uso de WordNet para hallarlos, como se muestra en la imagen 3.22.

Por un lado, al iterar sobre "carrot", rápidamente en una iteración obtenemos "plant root" que se encuentra en las categorías como "plant" y, por lo tanto, en una sola iteración podemos colocar como hiperónimo a "plant". Luego, su definición correspondiente comenzará como: "*Plant* that ...".

Algorithm 4 Obtenión del hiperónimo de una palabra

Input definendum
Output hypernym
$hypernym \leftarrow getHypernymFromWordNet(definendum)$
if hypernym in categoryList? **then**
return Nill
end if
if hypernym in categoryList? **then**
return hypernym
else
while notInRootOfWordnet() and not hypernym in categoryList? **do**
$hypernym \leftarrow getHypernymFromWordNet(hypernym)$
end while
end if
return 'something'

Por otro lado, nótese que "skin" tiene hiperónimos más complejos que no se adecuan a las categorías ni parecen útiles en el caso de uso de pistas para niños. En el árbol 3.22., se puede observar que "skin" tiene un hiperónimo padre que es "surface" (superficie), el cual es difícil de comprender de forma intuitiva. Al profundizar un nivel en el árbol, nuevamente se encuentra una palabra poco común y difícil de aprender para niños que están aprendiendo otro idioma, como "artifact". Además, los dos niveles anteriores a la raíz son (objeto - objeto físico) y (entidad - objeto físico), los cuales son aún más complejos. Por lo tanto, se establecen límites para evitar estos casos. La primera limitación es el uso de un conjunto de categorías esperadas o aceptadas. La segunda es limitar la profundidad en la exploración del árbol de hiperónimos. De lo contrario, las iteraciones seguirán sin éxito hasta llegar a "entity" o al máximo número de iteraciones permitido. En ese caso, el algoritmo devolverá el resultado por defecto "Something" y su definición tendrá la forma: "*Something* that ...".

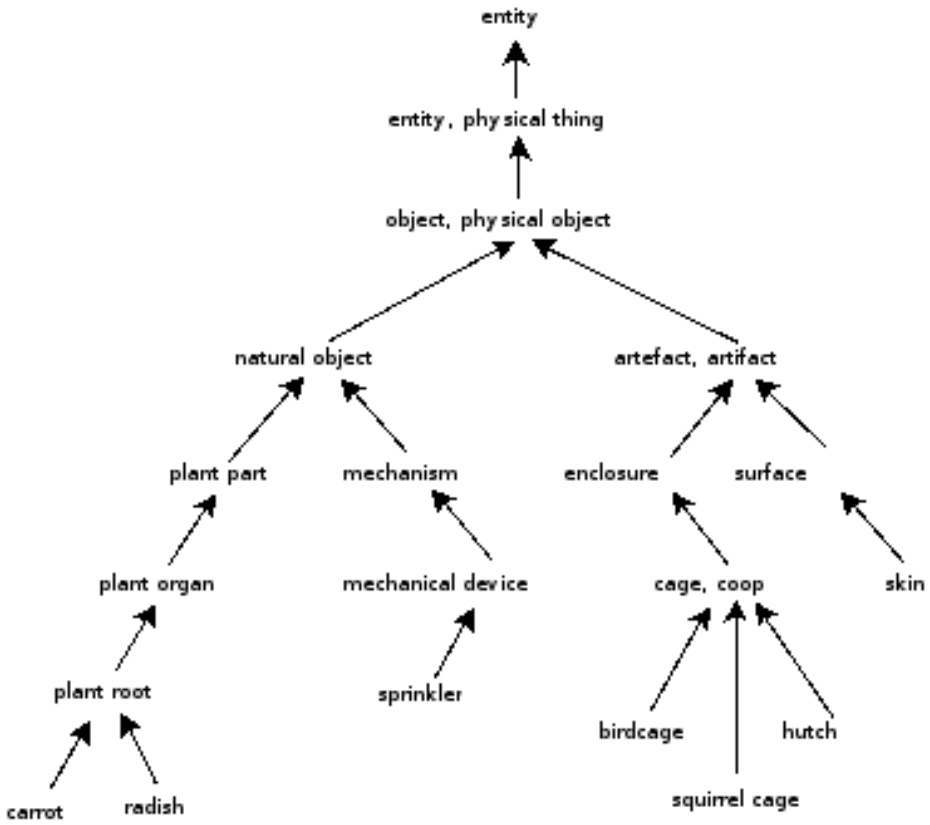

Figure 1. "is a" relation example

Figura 3.22: Ejemplo de archivo utilizado para la tarea de etiquetado.

3.5. Arquitectura del pipeline

En esta sección se presenta la arquitectura del pipeline. En las secciones anteriores de este capítulo, se explicaron en detalle los patrones implementados, así como las funciones de preprocesamiento y postprocesamiento necesarias para tener un pipeline completo, que tome un texto como entrada y genere los pares correspondientes como salida. A continuación, se presenta la versión final del módulo implementado.

El pipeline consta de un módulo orquestador encargado de llevar paso a paso el procesamiento del cuento de entrada. El cuento se procesa a través de todos los patrones, generando así un conjunto de pares de salida.

Para comenzar, se cargan los modelos y funciones auxiliares necesarios para ejecutar el preprocesamiento común a todos los patrones (los patrones implementados en el pipeline se encuentran detallados en la sección previa 3.3). Luego, cada patrón es invocado por el orquestador para generar los pares de palabras $\prec$*definiendum: definición*$\succ$ de interés. Una vez generados los pares, se someten a un postprocesamiento que incluye un filtro de calidad para seleccionar solo aquellas tuplas que cumplen con los requisitos establecidos. La salida final es el conjunto de tuplas seleccionadas. La secuencia de pasos y la integración entre los componentes se detallan en la figura 3.23.

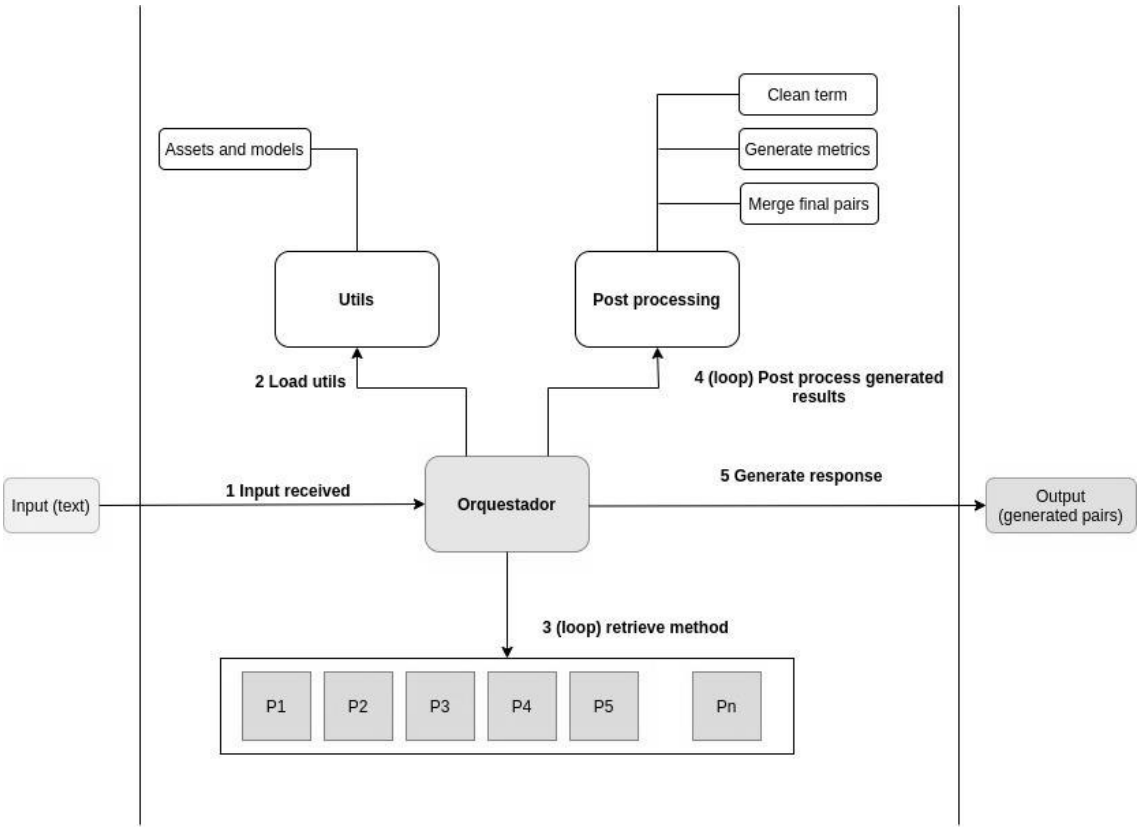

Figura 3.23: Arquitectura del pipeline.

Capítulo 4

Experimentación

La etapa de experimentación del proyecto consiste en analizar el conjunto de datos proporcionado, explorar su estructura y descubrir patrones y conexiones entre las entidades del texto y los eventos que se describen. Posteriormente, se procede a construir los métodos mencionados previamente y calcular las métricas necesarias para su evaluación. Finalmente, con el fin de crear un clasificador que funcione como filtro final para la salida generada, se describe la tarea de clasificación manual realizada.

4.0.1. Exploración del corpus

Como se mencionó previamente, el conjunto de datos "ReadWorks" utilizado en este proyecto se divide en cinco niveles. En este caso, se utilizaron mayoritariamente los cuentos correspondientes al nivel 1 y algunos del nivel 2, que presentan una menor dificultad y se acercan más a los casos de uso previstos en la plataforma.

Los textos de este nivel suelen ser cortos, con una extensión de uno o dos párrafos. En su mayoría, se trata de fábulas o cuentos para niños, o bien de información sencilla sobre diferentes temas, como presidentes, eventos históricos importantes, instrumentos o animales. El ejemplo 4.1 que se muestra a continuación es un extracto del corpus que ilustra esta característica.

Ejemplo 4.1: Cuento Wonderful Trees

Waving in the breeze. Birds lay eggs and nest, Squirrels climb up to rest. Many fruits to be eaten, Maple syrup to sweeten. Cool shade from the sun, Bright leaves by the ton. So if you could please, Take care of our trees.

En el ejemplo 4.1 se pueden observar las características previamente descritas: un párrafo breve con frases y oraciones sencillas y cortas que se refieren a un tema central, en este caso los árboles y lo que sucede en ellos.

Para llevar a cabo esta exploración, se procedió a leer los cuentos y a tomar notas manuales de posibles pistas que pudieran ser extraídas por docentes y utilizadas para crear un crucigrama. Esta tarea permitió generar cierta intuición acerca de los datos y, posteriormente, intentar generalizar el procedimiento para construir métodos o patrones que permitan extraer las definiciones de forma automática. Es importante destacar que no es una tarea sencilla extraer una gran cantidad de pistas a partir de estos cuentos, debido a su brevedad y simplicidad.

En una primera iteración se obtuvieron 150 pistas de forma manual de 25 cuentos. Se tomaron cuentos al azar de nivel 1 y se anotaron los términos, sus definiciones y la o las oraciones de donde se obtuvieron. Además, en una primera instancia, se clasificaron subjetivamente según su posible dificultad.

Por ejemplo, del cuento "Human's Best Friends"[1] se obtuvieron, entre otras, las siguientes parejas:

- **Text:** A breed is like a family of dogs.
  - Term: Breed
  - Definition: Family of dogs.
  - Difficulty: Easy
- **Text:** The most popular breed is the Labrador Retriever.
  - Term: Labrador Retriever
  - Definition: The most popular breed.
  - Difficulty: Easy
- **Text:** The world's largest dog is the Great Dane. This dog might not fit in laps, but it has a place in its owners' hearts.
  - Term: Great Dane
  - Definition: This dog might not fit in laps, but it has a place in its owners' hearts.
  - Difficulty: Medium
- **Text:** Sometimes parent dogs from different breeds have a puppy. When this happens, the puppy is called a mixed-breed or a mutt. A mutt will often act like both its different parents.
  - Term: Mutt
  - Definition: The puppy from different breeds parents.
  - Difficulty: Hard

Las posibilidades para la dificultad eran: Easy, Medium y Hard. En un principio, surgió esta idea como una forma de diferenciar las pistas en el futuro para estudiantes con diferentes niveles de inglés o para descartar pistas que fueran demasiado complejas. Sin embargo, la idea no se mantuvo, ya que resultaba difícil obtener esa información y no se encontró mucha utilidad en ella. Es por esto, que esta variable no es considerada en ninguna otra parte del proyecto.

Entre estas definiciones, ya se podían observar algunas de las ideas que se utilizaron más adelante, como el patrón del verbo "to be", el uso de correferencias y la combinación de múltiples oraciones.

Cabe destacar que esta exploración no se limitó únicamente al inicio del proyecto, sino que se llevó a cabo de forma constante durante todo el proceso. A medida que se iban construyendo los métodos descritos en el capítulo 3 y se revisaban los resultados obtenidos, se realizaba una iteración constante sobre los textos. En otras palabras, se leía y se revisaba continuamente el material para probar los métodos desarrollados hasta el momento y obtener nuevas ideas para seguir avanzando.

[1] No se incluye la totalidad de este cuento en el informe debido a que es un poco más largo que el resto de cuentos del nivel 1. No obstante, su variedad de definiciones sirve para ilustrar las ideas obtenidas en esta etapa de experimentación.

4.1. Experimentación en métodos de extracción de pares

En esta sección se describe el proceso de experimentación llevado a cabo en la construcción del sistema y los métodos y patrones utilizados. Se presentan tres fases: la fase inicial, que describe cómo se inició el desarrollo de la extracción de pistas; la fase iterativa (o intermedia), que se centra en el proceso seguido para construir nuevos métodos y mejorar los existentes; y la fase final, donde se detallan los cambios finales realizados en la solución antes de obtener la última versión.

4.1.1. Fase inicial

Después de haber generado una gran cantidad de pistas de forma manual, se inició la fase de desarrollo de los métodos y patrones. El enfoque inicial para la recuperación de pistas consistía en detectar verbos entre las pistas generadas manualmente y a partir de ahí, construir una pista. Sin embargo, estas pistas iniciales no eran de buena calidad: eran poco intuitivas, ambiguas, redundantes y agramaticales, lo que significa que estaban mal escritas o no tenían sentido. Estas inconsistencias fueron corregidas de forma iterativa, sin perder generalidad.

Se decidió trabajar con verbos, ya que eran los términos más fáciles de encontrar entre las pistas generadas a mano. Además, se optó por trabajar con estos, ya que este enfoque se había utilizado en trabajos anteriores como el reconocimiento del verbo “to be” utilizado en Tosi & Percovich (2019). Después de leer los textos, se pudo generalizar esta idea y construir métodos similares a partir de otros verbos. De esta manera, se logró conseguir, por ejemplo, el patrón “eat” o el patrón “is called”.

Se inició el proceso de selección de verbos a partir de las pistas generadas manualmente. Cada vez que se creaba un patrón, se analizaban los resultados utilizando los textos de “ReadWorks” con los que se trabajó. Sin embargo, esta estrategia no resultó ser la más efectiva, ya que algunos de los patrones creados generaban pocas pistas. Para abordar esta problemática, se decidió realizar un análisis de los verbos más utilizados en todos los textos del corpus, lo que permitió identificar los verbos más recurrentes y relevantes para el proceso de extracción de información. Se puede contemplar la lista de verbos más utilizados en la tabla 4.1. Una vez elegido un verbo, se verificaba manualmente que la forma en la que se utilizaba fuera consistente en distintas oraciones, con el objetivo de obtener patrones más sólidos y consistentes. Para este fin, se desarrolló un programa que listaba todas las oraciones que contenían un verbo determinado, lo que permitía analizarlas de forma secuencial y asegurar la consistencia del patrón.

Para los primeros métodos, se comenzó utilizando expresiones regulares para la coincidencia de estos patrones. Sin embargo, esto presenta un problema, ya que las expresiones regulares no son adecuadas para una gramática libre de contexto. Como se explica en el capítulo 3, debido a que el léxico utilizado en los cuentos para niños es muy simple y las oraciones no tienen muchos constituyentes complejos, se intentó utilizar expresiones regulares en algunos patrones (que permitían capturar la mayoría de los casos). En otros patrones, las expresiones regulares no fueron suficientes y como alternativa, se utilizó SRL para capturar los patrones. El uso de SRL permitió, además, capturar información de la estructura argumental de los verbos, como se demostró en secciones anteriores. Esto permitió generalizar los verbos por su estructura argumental en lugar de por la palabra en particular que se utiliza.

Durante esta etapa, se realizó la configuración del ambiente de trabajo local, lo que incluyó la instalación y acceso a las librerías necesarias para el análisis y procesamiento de los textos. En este proceso, se enfrentaron requerimientos precisos de versionado y limitantes de hardware, en particular de memoria.

Verbo	Frecuencia	Verbo	Frecuencia
is	824	made	128
are	433	see	96
can	320	were	94
was	314	make	91
have	262	had	89
has	148	eat	88
called	144	use	84
be	130	live	84
grow	75	get	84
help	76	do	81
go	78		

Cuadro 4.1: Frecuencia de los 21 verbos más utilizados en los cuentos de Nivel 1 de ReadWorks

4.1.2. Fase intermedia

Durante esta etapa del desarrollo del módulo, se obtuvieron resultados parciales satisfactorios, aunque no suficientes. Surgió un conflicto entre la generalidad de un patrón y la calidad de las pistas obtenidas, ya que cuanto más general era un patrón, más pistas se obtenían, pero de menor calidad, mientras que cuanto más específico era un patrón, aumentaba su precisión pero disminuía la cantidad de pistas obtenidas. Se llegó a la conclusión de que era más importante tener pocas pistas buenas que muchas pistas malas. Luego de esta decisión, se observó que debido a la brevedad de los cuentos y el aumento de la especificidad de los métodos, la cantidad de pistas obtenidas era muy reducida. Para abordar estos problemas, se decidió centrarse en la creación de una mayor cantidad de métodos que abarcaran una gama más amplia de posibles candidatos a pistas, con el fin de encontrar más pistas posibles en textos cortos sin comprometer la calidad. Aunque los métodos basados en verbos habían sido efectivos hasta ese momento, se hizo necesario encontrar más fuentes de definiciones y pistas para los crucigramas.

Por un lado, se observó que en muchos textos se mencionaban personas, animales, lugares o entidades específicas, y se desarrollaron patrones que tomaban como *definiendum* estas palabras en lugar de los verbos. Se utilizaron herramientas como NER, resolvedores de correferencias y question answering para construir estos patrones. Por otro lado, se notó que en los textos se desarrollaba un tema central y se crearon patrones extendidos que unificaban varias oraciones en una única definición con el mismo *definiendum*.

Por otra parte, también se intentó integrar modelos preentrenados como T5, que funcionan como codificador-decodificador convirtiendo las tareas en secuencias entrantes y salientes de texto. Sin embargo, se encontró que estos modelos requieren una cantidad mucho mayor de datos que el conjunto de datos utilizado en el proyecto, que en este caso fue el conjunto de datos *ReadWorks*.

Finalmente, se llevó a cabo una iteración continua de los nuevos patrones y extensiones de métodos, revisando los resultados y corrigiendo errores para mejorar el método hasta que se considerará suficientemente consistente.

4.1.3. Fase final

Después de finalizar las iteraciones de desarrollo e implementación de patrones y métodos, se procedió a la fase final del proyecto. Durante esta etapa se analizaron los resultados obtenidos por los métodos y el sistema

en general, con el objetivo de identificar cuáles fueron los métodos que más contribuyeron a la generación de pares. Además, se llevó a cabo un análisis exhaustivo de los errores encontrados, los cuales se dividieron en dos categorías: errores de implementación y errores intrínsecos en los modelos y herramientas utilizadas. Los primeros se corrigieron en una etapa final de ajustes. En cuanto a los segundos, al no haber sido responsables de la implementación de dichos modelos y herramientas, quedaron fuera de nuestro alcance, pero aun así se propusieron mejoras o modelos alternativos

Durante esta fase, también se implementaron mejoras adicionales, tales como la funcionalidad para limpiar términos innecesarios de algunos definiendums, permitiendo la construcción de pistas donde el definiendum se limitara a una única palabra y fuera factible de insertar en un crucigrama restringido a sustantivos o adjetivos. Esto permitió que el definiendum fuera el sujeto o una característica de éste, en lugar de palabras que no fueran relevantes, como conjunciones, determinantes u otros.

4.2. Métricas obtenidas

Después de ejecutar el pipeline actual, se obtuvieron los siguientes resultados para los textos considerados del corpus “ReadWorks” (400 textos): se generaron un total de 2321 tuplas, con una media de 6.02 pistas por texto. En la tabla 4.2 se puede realizar un análisis desglosado por método, donde los patrones con mayor frecuencia generan un mayor número de pistas finales.

Método	Cant. pistas generadas	Media pistas generadas/cuento
Patrón eat	47	0.136628
Patrón is_called	35	0.127778
Patrón to_be	538	1.455041
Question Answering	32	0.088398
Patron to_be_extended	887	2.376731
Patron have	276	0.815934
Patrón ner_srl	218	0.547684
Patron gpe_org	49	0.136364
Patrón live	15	0.028986
Patrón like	53	0.204360
Patrón live_srl	74	0.223164
Patrón is_called_extended	0	0.000000

Cuadro 4.2: Tabla de métricas sobre generación de pistas por modelo

Estos resultados preliminares son muy prometedores, ya que se logra generar un promedio de 6 pistas por cuento, lo que permite la construcción de crucigramas.

Vale la pena destacar que el patrón “is called extended” no generó ninguna pista. Este patrón pertenece a la categoría de patrones extendidos los cuales fueron pensados en un principio para extender el patrón “to be” (el cual sí fue exitoso). En la sección 3.1.3, se comentó que los patrones que eran aptos para extender eran aquellos que formaban pistas con palabras que estaban en su totalidad en el texto, sin agregar lógica extra ni otras palabras de soporte como el “that”. En teoría el patrón “is called” también cumplía esto, pero nunca se vio un ejemplo en los cuentos, era un caso teórico que podría funcionar y era fácil de implementar por lo que se optó integrarlo al sistema. De todas formas, viendo los resultados más adelante, se puede comprobar que

efectivamente ninguno de los cuentos analizados cumplía con este patrón.

4.3. Etiquetado de pistas

Dadas las pistas obtenidas anteriormente, es de interés saber cuáles de ellas son acertadas y adecuadas para la construcción de crucigramas. Conocer los resultados obtenidos con este sistema ayuda a medir su eficacia y compararlo con otras implementaciones para determinar si se han alcanzado los objetivos propuestos.

Sin embargo, determinar si las pistas generadas son buenas o malas no es una tarea fácil, ya que se requiere revisar manualmente cada pista para determinar su validez. Para abordar este problema, se llevó a cabo una tarea en colaboración con estudiantes y profesores de la Facultad de Comunicación (FIC) y la Facultad de Ingeniería (FIng) para etiquetar los pares obtenidos. Se constituyó un equipo de 11 personas para clasificar un conjunto de datos de 2321 pistas.

Después de clasificar cada pista, se pueden obtener métricas sobre cada patrón. A partir de estos resultados, es posible construir un clasificador automático capaz de clasificar futuras pistas como buenas o malas de forma automática, sin necesidad de revisarlas manualmente. Al utilizar un clasificador automático, se pueden ahorrar recursos y tiempo valioso.

En la siguiente subsección, se abordará en profundidad cómo se llevó a cabo la tarea de etiquetado.

4.3.1. Tarea de etiquetado

El objetivo principal de esta tarea fue etiquetar las pistas generadas previamente como buenas o malas. Para lograrlo, es fundamental contar con los definiendum y las definiciones generados previamente, ya que se utilizan en el crucigrama como “término a resolver” y “pista”, respectivamente.

De todas formas, es importante recordar que estas definiciones dependen totalmente del contexto del cuento en muchos casos. Se pueden tener pares aislados de $\prec$*definiendum: definición*$\succ$ que, sin un contexto, no se puede determinar si están correctamente asociados. Por ejemplo, consideremos el siguiente par generado:

$$\prec \textit{Jake: A big, strong horse.} \succ$$

Es imposible clasificar una pista como buena o mala sin conocer el cuento al que pertenece o sin tener un fragmento de texto del cual se haya obtenido la pista. Por esta razón, se proporciona un contexto para cada par de definición y definiendum. Por ejemplo, en este caso también se proporcionó al revisor la oración “Jake was a big, strong horse”. De esta manera, el revisor puede verificar rápidamente si la pista es correcta.

En la mayoría de los casos, el contexto consiste en una sola oración, ya que como se explica en el capítulo 3, en la mayoría de los patrones y métodos utilizados para generar pistas, se segmentan las oraciones y se extraen las definiciones una por una. Sin embargo, existen otros métodos en los que se utilizan varias oraciones para formar la pista, en cuyo caso el contexto serían todas las oraciones involucradas. También está el caso del método QA, que es un modelo que extrae definiciones de todo el texto al mismo tiempo. En este caso, no se sabe si se extrajo la definición de una sola oración o de varias, por lo que para los pares generados con este método, se incluye la totalidad del cuento como contexto. Este dato fue fácil de incluir, ya que se decidió obtener desde

un principio cuando se comenzó a implementar el sistema, ayudando a depurar las pistas generadas de manera rápida y sencilla.

Además, se decidió proporcionar a los revisores el nombre de cada cuento para cada pista, así como también los archivos de los cuentos. De esta forma, los revisores pueden tener más contexto y corroborar si las pistas son buenas o malas de manera más precisa, ya que esto no siempre es posible solo con una oración. En algunos casos, una sola oración puede no proporcionar suficiente información. Como se ha visto previamente, las herramientas utilizadas para resolver las correferencias pueden actuar de manera incorrecta, asignando correferencias a un pronombre que pertenece a otra persona. Por ejemplo, uno de los pares generados es:

≺*Memorial, (Jr _____): Black walls that show the names of many soldiers that fought for the United States.*≻

A simple vista, se puede entender que se está hablando de un monumento llamado "Jr Memorial" que consiste en muros negros que muestran los nombres de soldados. Sin embargo, alguien que esté familiarizado con el "Jr Memorial" sabrá que esta definición es incorrecta. Para asegurarse de que las definiciones son precisas, se incluyó el nombre del cuento que las generó. Al revisar el cuento "What is a Memorial?", se encuentra lo siguiente:

...The Martin Luther King Jr. Memorial is a large stone statue. The statue honors Martin Luther King Jr. He worked for peace among all people. The Vietnam Veterans Memorial has two special walls. They are black walls that show the names of many soldiers. They fought and died for the United States in the Vietnam War.

Se puede observar que se generó una definición para "Jr Memorial" que en realidad corresponde a "The Vietnam Veterans Memorial". Esto sucedió debido a que el resolvedor de correferencias asignó "Jr Memorial" a "They" y se generó el siguiente contexto: *Jr Memorial are black walls that show the names of many soldiers. Jr Memorial fought and died for the United States in the Vietnam War.* Al leer este contexto, puede parecer que la pista es correcta, pero al abrir el cuento se puede ver que la pista es incorrecta.

Para resolver este problema, se decidió que el contexto sirve para marcar pistas como incorrectas de forma rápida, pero para marcar pistas como buenas debemos entrar al texto y corroborar que la pista es verdaderamente buena.

Finalmente, se incluyen otras dos columnas vacías llamadas "clase" y "gramatical" donde los anotadores deben llenar la información sobre si la pista era buena o no. Estas dos categorías se explican en detalle más adelante.

Resumiendo, se agruparon todos los pares generados en un archivo con las siguientes columnas:

- **Definiendum**: Término a adivinar en el crucigrama.
- **Definición**: Pista para poder adivinar el término a rellenar en el crucigrama.
- **Clase**: Espacio en blanco para poder marcar si la pista es buena o mala.
- **Gramatical**[2]: Espacio en blanco para marcar si la pista está bien escrita.

[2]Esta información se utilizo para la evaluación del modulo de extracción de pistas y no para el clasificador

- **Contexto**: Oración o conjunto de oraciones de donde se obtuvo la pista.
- **Text Name**: Nombre del cuento de donde se obtuvo la pista.
- **Método**: Método o patrón utilizado para generar la pista (no relevante para los anotadores).

En la figura 4.1 se puede observar un ejemplo de archivo utilizado para el etiquetado.

definiendum	definicion	clase	gramatical	contexto	text_name	metodo
You	Thing that can play mini golf	0	1	You can play mini golf with parents and friends !	Mini Golf.csv	patron_ner_srl
Florida	Place where There are many mini golf courses	1	1	There are many mini golf courses in Florida	Mini Golf.csv	patron_gpe_org
rambutan	One of some fruits	1	1	One of some fruits is the rambutan.	Rambutan.csv	patron_to_be
rambutan	a small fruit	1	1	A rambutan is a small fruit.	Rambutan.csv	patron_to_be
white	the fruit	0	0	On the inside, the fruit is white.	Rambutan.csv	patron_to_be
fruit	a little see - through , like a grape	0	0	the fruit is a little see - through, like a grape.	Rambutan.csv	patron_to_be
fruit	a very popular fruit	0	0	the fruit is a very popular fruit in Asia !	Rambutan.csv	patron_to_be
rambutan	a small fruit that can be red , orange , yellow , or green	1	1	Sentence: A rambutan can be red, orange, yellow, or green..	Rambutan.csv	patron_to_be_extended
rambutan	a small fruit that is covered in bristles	1	1	Sentence: And A rambutan is covered in bristles..	Rambutan.csv	patron_to_be_extended
rambutan	a small fruit that is covered in bristles	1	1	Sentence: And A rambutan is covered in bristles..	Rambutan.csv	patron_to_be_extended
fruit	a little see - through , like a grape that is white	0	1	Sentence: On the inside, the fruit is white..	Rambutan.csv	patron_to_be_extended
fruit	a very popular fruit that is white	0	1	Sentence: On the inside, the fruit is white..	Rambutan.csv	patron_to_be_extended

Figura 4.1: Ejemplo de archivo utilizado para la tarea de etiquetado.

Durante el proceso de iteración, se descubrió que en algunos casos la pista generada correspondía al término especificado, pero presentaba errores gramaticales, como malas estructuras o verbos mal conjugados. A pesar de estos errores, la pista era comprensible y podía servir para evaluar la comprensión lectora. Para no descartar estas pistas de antemano, se decidió agregar una columna para que los anotadores indicaran la gramaticalidad de las pistas, con el fin de señalar estos casos. De esta manera, una pista podría ser correcta pero no estar bien escrita. La clasificación fue dividida en tres categorías, 1, 0 y -1. A continuación, se explica el criterio para puntuar cada una de las pistas con estos números.

Clasificación de clase

Allí, el criterio de aprobación sobre una pista generada se basa en los siguientes ítems:

- Pista correcta, etiqueta [1]
  En este caso, la pista es clasificada como buena, considerándose que tiene coherencia, cohesión y lleva a resolver el término. Ejemplo de pista correcta:
  - Contexto: The White House has a swimming pool and a movie theater.
  - Tupla: $\prec$ ***Pool*** *- (Swimming _____): Something The White House has* $\succ$
- Pista incorrecta, etiqueta [0]
  En este caso, la pista puede tener varias interpretaciones posibles, no está clara o no es suficientemente específica para guiar al alumno a resolver el término. Por lo tanto, se considera una pista incorrecta. Ejemplo de pista incorrecta:
  - Contexto: Some people have lived in Alaska for a long time.
  - Tupla: $\prec$ ***people****: Alaska_time* $\succ$
- Cuento corrupto, etiqueta [-1]
  Las pistas descartadas son aquellas que se identifican como inválidas debido a errores en los datos de entrada, como textos que hacen referencia a imágenes y, por lo tanto, no son utilizables para el módulo de extracción. Esta clase se excluye del conjunto de datos final que se envía a los modelos de aprendizaje automático. Ejemplo de pista descartada:

- Contexto: Life in a Rain Forest Photos.comMacaw Macaws live in the tallest trees, which make up Emergent Layer.ine
- Tupla: ≺ ***trees****: Place where Life in a Rain Forest Photos.comMacaw Macaws live* ≻

Las condiciones fueron propuestas inicialmente por el equipo del proyecto y posteriormente fueron revisadas y ajustadas por todos los participantes.

Clasificación gramatical

- Pista gramatical, etiqueta [1]
  La pista es gramaticalmente correcta.
  - Contexto: So Franklin D. Roosevelt came up with plans to add more jobs.
  - Tupla: ≺ ***Roosevelt*** *- (Franklin D _____): President that came with plans to add more jobs* ≻
- Pista agramatical, etiqueta [0]
  La pista tiene errores gramaticales o de formación.
  - Contexto: Green iguanas eat leaves, flowers, and fruit
  - Tupla: ≺ ***iguanas*** *- (Green _____): large that eat leaves , flowers , and fruit* ≻
- Pista incoherente, etiqueta [-1]
  Misma clasificación que antes, son aquellas en las que el texto está corrupto o presenta algún tipo de problema que impide su uso en el proceso de generación de pistas.
  - Contexto: This painting Photos.comis titled Breaking Home Ties. It was painted by Thomas Hovenden, an Irish-born artist. Thomas moved to the United States from Ireland in 1863. He painted this scene in 1890.
  - Tupla: ≺ ***Irish****: Breaking Home Ties* ≻

Proceso y metodología

Para llevar a cabo la tarea de etiquetado, primero se formó un equipo de etiquetadores con la ayuda de los tutores, formado por estudiantes que cursaban módulos de extensión y sus docentes, los cuales accedieron a tener reuniones semanales para ayudar con la tarea. En las primeras etapas, se brindó contexto a los etiquetadores a través de presentaciones sobre el proyecto, su motivación y, sobre todo, se les explicó en qué consistía la tarea de etiquetado.

Se decidió dividir las pistas en archivos de 100 cada uno para poder distribuir mejor el trabajo. El formato de columnas explicado anteriormente no fue el inicial; gracias a la retroalimentación del equipo, se incluyeron los datos necesarios y se ordenaron las columnas para que fuera fácil leer los archivos y etiquetar.

Durante el proceso, surgieron diversas mejoras en los métodos utilizados. Se decidió emplear el término “Something” en las definiciones, en lugar de “Thing”, que se utilizaba inicialmente, ya que resultaba más apropiado. Además, se propuso separar la clasificación en *clase* y en *gramatical*. También se optó por agrupar las pistas de un mismo cuento en orden, para poder etiquetar todas las pistas relacionadas sin tener que ingresar al cuento varias veces.

Una vez que se tuvo un buen formato para los archivos y se logró transmitir el objetivo de la tarea, se creó un conjunto de 50 pistas de prueba que todos los etiquetadores clasificaron por su cuenta. En una siguiente reunión, compartieron sus resultados y se llegó a un acuerdo sobre qué pistas eran buenas o malas, ya que podían existir ambigüedades y casos borde. De esta forma, se mitigó el impacto que podría tener la subjetividad de la tarea.

Durante el etiquetado, se resolvieron dudas por correo electrónico y se utilizaron las reuniones agendadas para discutir temas más complejos, compartir sugerencias o inquietudes. El proceso duró aproximadamente dos meses y se lograron etiquetar todas las pistas generadas.

Finalmente, se logró obtener resultados muy buenos en la construcción del corpus. En la tabla 4.3 se muestran dichos resultados. Se observa que solamente 112 pistas fueron marcadas como inválidas (5 %), lo cual corresponde a errores en los textos de entrada. Además, se obtuvo un alto porcentaje de pistas gramaticales (87 %) y de pistas válidas (72 %), lo cual indica un buen desempeño del pipeline. Estos resultados se presentan en mayor detalle en la sección de resultados del pipeline (sección 4.5.1). Para el entrenamiento del clasificador y la evaluación del pipeline, se utilizarán las 2209 pistas válidas, descartando las 112 pistas provenientes de textos inválidos.

Categoría	**Cantidad**
Pistas Buenas	1598
Pistas Malas	611
Gramatical	1937
Agramatical	272
Válidas	2209
Inválidas	112

Cuadro 4.3: Resultados de anotación de las tuplas generadas

4.4. Construcción del clasificador

En esta sección se detallan los aspectos de la experimentación del clasificador, incluyendo su entrenamiento y los resultados obtenidos. El objetivo del clasificador es determinar si una pista es buena o mala, dado un par de términos definiendum y definición y su respectivo contexto. Además, las métricas utilizadas para la experimentación y evaluación de los modelos se fundamentan en el apéndice D.

4.4.1. Pruebas iniciales

Al comienzo del desarrollo del clasificador, se consideraron varias posibles arquitecturas para el mismo. En una etapa inicial se construyeron y experimentaron varios clasificadores de distintos niveles de complejidad, con el objetivo de obtener una línea base con el modelo más simple y luego poder mejorar a partir de ella. Estos clasificadores fueron inspirados en Esteche & Romero (2015), Kamran Kowsari (2019) y Bingsheng Yao (2021).

Las arquitecturas propuestas inicialmente fueron:

- Distancia entre centroides
- K-Nearest Neighbours

- Naive Bayes
- Arboles de decisión
- Gradient Boosting
- Multi layer Perceptron
- BERT
- DistilBERT

Para esta etapa inicial, se llevó a cabo una prueba de concepto con 50 pistas etiquetadas para validar que los clasificadores pudieran ajustarse a ellas. Se comprobó que todos los métodos descritos anteriormente funcionaron correctamente en esta etapa y, por lo tanto, se decidió continuar con ellos para realizar un entrenamiento posterior y analizar el mejor modelo posible.

4.4.2. Arquitecturas

En esta sección se describen con más detalle las arquitecturas de los clasificadores presentados en la sección previa.

Distancia entre centroides

Este clasificador se basa en el método utilizado en el proyecto de grado de Esteche & Romero (2015) para la tarea similar de determinar si una pista es buena o mala. En dicho proyecto, se calcula la distancia entre el definiendum y la media de la definición, considerando también la clase de la pista. La idea es comparar la clase de la pista con el vector que representa su definición y verificar si coinciden o tienen sentido. La solución propuesta es similar, pero se basa en el contexto del cual se obtuvo la información para construir el par.

En este clasificador, se calculan dos centroides:

- Centroide del contexto.
- Centroide del par "definiendum" + "definición".

Luego se calcula la distancia entre ambos y se compara con un umbral que se define como el límite donde un par es considerado bueno o malo (hiperparámetro del modelo). Si la distancia es menor al umbral, se considera buena y, de lo contrario, mala. Cabe destacar que es necesaria una representación vectorial de las palabras, para lo cual se utiliza "FastText embedding" (aunque se probaron otros, este dio los mejores resultados). Este clasificador se eligió como línea base por su simplicidad y por su uso previo en un proyecto de grado relacionado.

Clasificadores Aprendizaje Automático

Estos clasificadores hacen uso del embedding FastText mencionado anteriormente Bojanowski et al. (2016) para representar tanto el definiendum como la definición y su contexto. Los clasificadores que se utilizaron en esta etapa son: KNN, NB, Árboles de decisión y Gradient Boosting. (Kamran Kowsari (2019))

Para entrenar estos clasificadores, se utiliza el centroide del embedding del "contexto" + $\prec$ SEP $\succ$ + "definiendum" + $\prec$ SEP $\succ$ [3] + "definición". En algunos casos se realizó una búsqueda manual de hiperparámetros para tratar de mejorar la precisión de los modelos.

[3] Token especial para la separación utilizado usualmente en los modelos de lenguaje

Clasificadores Aprendizaje Profundo

En esta sección se presentan tres clasificadores de aprendizaje profundo: MLP Rusli et al. (2020), BERT (Devlin et al. (2019)) y distilBERT (Sanh et al. (2019)). Si bien BERT y distilBERT son modelos de lenguaje y no clasificadores en sí, se pueden utilizar para la tarea de clasificación agregando capas al final de las redes (estas arquitecturas se detallan en el Apéndice B). En el proyecto se utiliza la solución por defecto proporcionada por la librería Huggingface[4] (Wolf et al. (2020)). Estas capas difieren según el modelo utilizado, por ejemplo, en el caso de BERT se compone de una capa de "dropout" seguida de una capa "densa". Para simplificar el nombre de estos clasificadores, que son el modelo de lenguaje con las capas de clasificación agregadas, simplemente los llamamos por el nombre del modelo de lenguaje.

Todos ellos reciben como entrada el contexto, el definiendum y la definición, concatenados y separados por el token especial $\prec$ SEP $\succ$, y producen un puntaje para cada clase (buena o mala).

Las arquitecturas se basan en el artículo de referencia Bingsheng Yao (2021), donde se implementó un módulo "ranker" para pares de preguntas y respuestas a partir de la salida de un modelo anterior.

4.4.3. Búsqueda de hiperparámetros

Con respecto a la optimización de los modelos, dado que se cuenta con una amplia variedad de patrones, no se llevó a cabo una búsqueda exhaustiva de hiperparámetros en todos los casos. En muchos de ellos, particularmente en los clasificadores de aprendizaje automático, se realizó una búsqueda manual de los hiperparámetros, sin embargo, no se llevó a cabo una búsqueda detallada.

Distancia entre centroides

En el caso particular del baseline, dada la naturaleza del método, se realizó una búsqueda completa del espacio de hiperparámetros, en este caso del threshold. Para esto se calcularon la distancia mínima y máxima posibles entre todos los ejemplos de los datos etiquetados, ya que este es el rango posible para el valor del threshold: $[0{,}3, 6]$. Para cada valor posible del threshold se calculó la precisión y f1-macro obtenidos. Se puede observar en las figuras 4.2 que la precisión aumenta a medida que aumenta el threshold (es trivial), pero además se buscó maximizar la medida f1 en la figura 4.3, donde se aprecia cómo crece al inicio y luego decae.

[4]La librería provee una clase AutoModelForSequenceClassification que agrega las capas necesarias para la clasificación

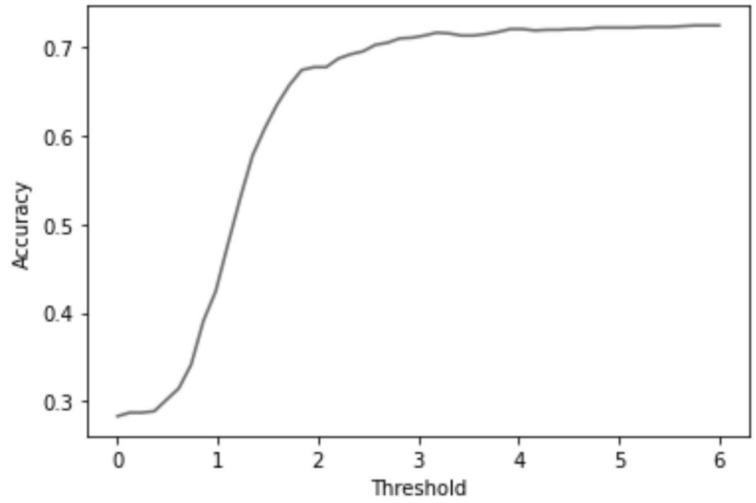

Figura 4.2: Gráfico de acierto al variar threshold.

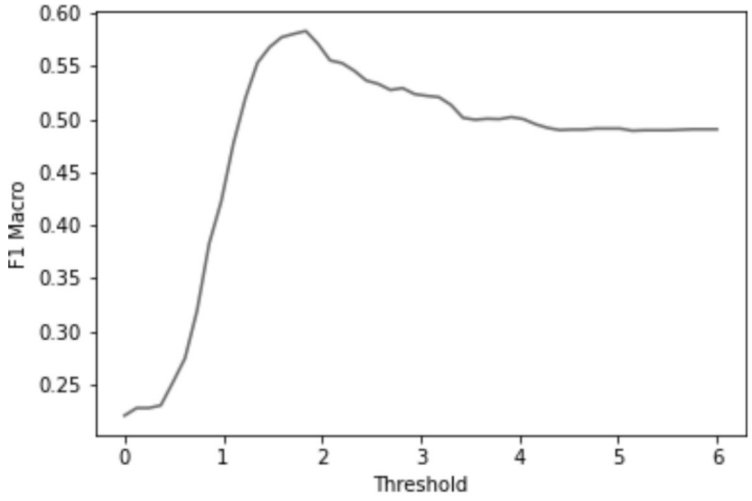

Figura 4.3: Gráfico de f1 macro al variar threshold.

La figura 4.3 muestra que el valor del *threshold* que maximiza la f1 es 1.8, por lo que este valor fue seleccionado para el método.

4.4.4. Entrenamiento

Para el entrenamiento de los modelos, se realizó en primera instancia un proceso de 5-fold cross-validation para evaluar el desempeño de los diferentes modelos en términos de acierto, precisión, recuperación y f1-macro. Luego, se seleccionaron los tres modelos con mejor rendimiento para realizar un segundo entrenamiento, dividiendo los datos en subconjuntos de entrenamiento y validación (con un split de 80 % para entrenamiento y 20 % test) para medir nuevamente precisión, recuperación y f1-macro, y elegir el mejor modelo. Los resultados de estas métricas para cada modelo se pueden encontrar en la tabla 4.4, donde se observa que DistilBERT es el mejor modelo en términos de f1-macro en la primera instancia de entrenamiento

Luego, se presentan en la tabla 4.5 los resultados obtenidos con los mejores modelos y se comparan con el baseline.

Modelo	Accuracy	Precisión	Recall	F1-Macro
KNN	0.72	0.62	0.57	0.57
Dec Tree	0.64	0.55	0.55	0.55
Grad Boosting	0.71	0.67	0.59	0.59
MLP	0.73	0.69	0.58	0.59
NB	0.66	0.52	0.51	0.49
BERT	0.77	0.78	0.74	0.70
DistilBERT	**0.84**	**0.81**	**0.75**	**0.78**

Cuadro 4.4: Métricas obtenidas para los clasificadores entrenados con 5-fold cross-validation.

Modelo	Accuracy	Precision	Recall	F1 (macro)
Centroides	0.66	0.60	0.56	0.58
BERT	0.77	0.78	0.74	0.70
DistilBERT	0.84	0.81	0.75	0.78

Cuadro 4.5: Resultados obtenidos sobre *test* en la segunda instancia de entrenamiento, sobre el conjunto de test.

4.5. Resultados

En esta sección se presentan los resultados obtenidos de los experimentos realizados en el proyecto. Se divide en dos partes: por un lado, los resultados del pipeline y las pistas generadas luego de ser clasificadas manualmente y por otro lado los resultados del clasificador entrenado con el corpus etiquetado.

4.5.1. Resultados del pipeline

Una vez que el *pipeline* está disponible, se ejecutan los métodos y se clasifican por clase y categoría semántica. La tabla 4.6 detalla los valores recuperados una vez clasificados por clase (correctitud), mientras que la tabla 4.7 detalla los valores obtenidos según si son gramaticales o no. Además, se adjuntan gráficas que permiten visualizar la ponderación de cada una de las clases.

Método	Total	Clase 0: No	Clase 1: Sí	Accuracy
Patrón eat	53	9	44	0.83
Patrón gpe org	49	14	35	0.71
Patrón have	276	96	180	0.65
Patrón is called	35	18	17	0.48
Patrón like	54	19	35	0.65
Patrón live	15	13	2	0.13
Patrón live SRL	74	12	62	0.84
Patrón ner SRL	217	54	163	0.75
Patrón to be	538	104	434	0.81
Patrón to be extended	887	264	623	0.70
Patrón qa	11	8	3	0.28

Cuadro 4.6: Métricas obtenidas para los métodos ejecutados desde el pipeline (correctitud).

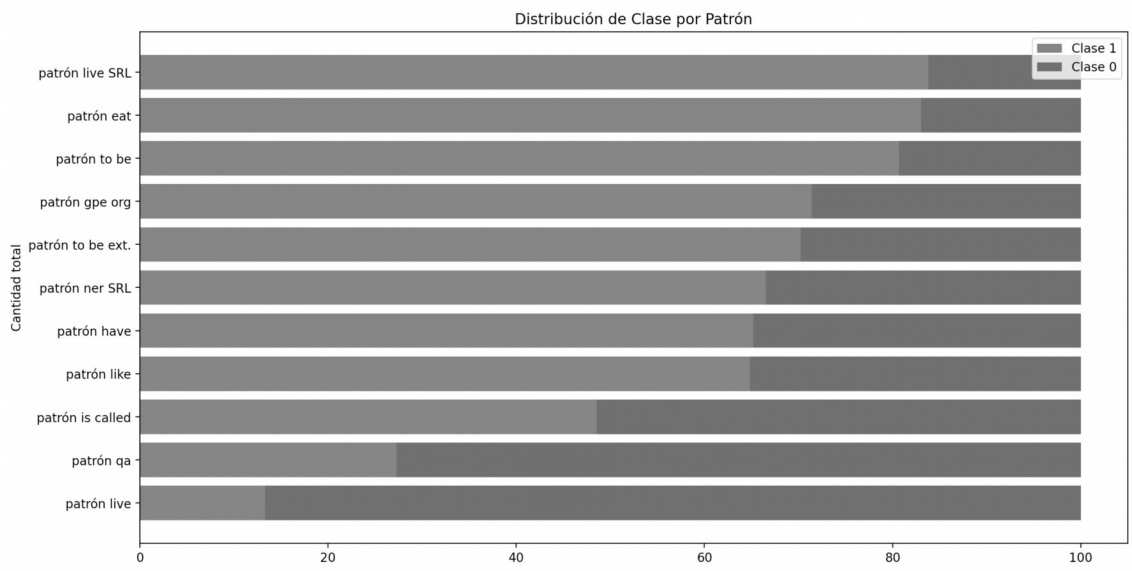

Figura 4.4: Distribución de pistas buenas (Clase 1) y malas (Clase 0) por método (porcentualmente)

Analizando los gráficos de la figura 4.4, notamos que en general los métodos tienen buena performance para la extracción de pares. En particular, la mayoría se encuentra por encima del 60 % de accuracy, siendo únicamente los patrones "is called", "qa" y "live" los que caen por debajo de esta medida.

Además, como se muestra en la figura 4.5, el patrón "to be extended" parece ser el más efectivo en la extracción de pistas, con más de 850 pistas extraídas para los cuentos, mientras que el resto cae bastante por debajo de esta línea. Aun así, todos los patrones logran extraer una gran cantidad de pistas en su conjunto.

Método	**Total**	**Agramatical**	**Gramatical**	**Accuracy**
Patrón eat	53	7	46	0.87
Patrón gpe org	49	7	42	0.86
Patrón have	276	29	247	0.90
Patrón is called	35	4	31	0.86
Patrón like	54	3	51	0.94
Patrón live	15	6	9	0.60
Patrón live SRL	74	9	65	0.88
Patrón ner SRL	217	38	179	0.82
Patrón to be	538	20	518	0.96
Patrón to be extended	887	147	740	0.83
Patrón qa	11	2	9	0.81

Cuadro 4.7: Métricas obtenidas para los métodos ejecutados desde el pipeline (gramatical).

Finalmente, en la tabla 4.8 se muestran los resultados globales del pipeline, donde se presenta el accuracy de 72 % para las pistas buenas y del 87 % de pistas gramaticales, con el total de pistas extraídas siendo 2209.

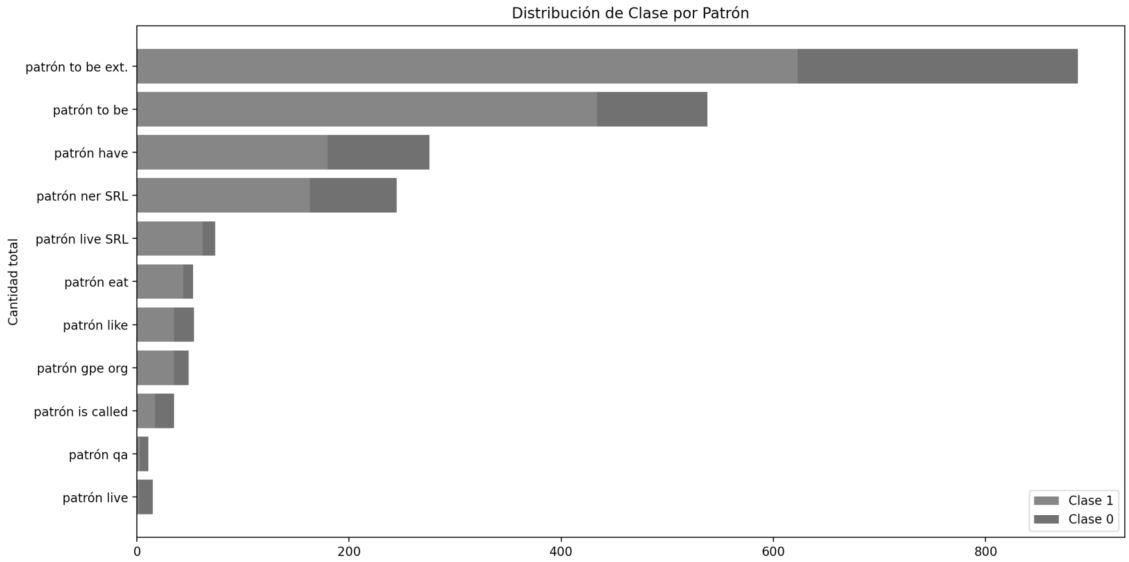

Figura 4.5: Distribución de pistas obtenidas por método y por clase.

Pistas Buenas	**Pistas Malas**	**Gramatical**	**Agramatical**
1598	611	1937	272

Cuadro 4.8: Resultados globales del pipeline.

4.5.2. Resultados del clasificador

Después de experimentar con diversas arquitecturas de distintas complejidades y realizar varios entrenamientos para identificar el método más adecuado para el caso de uso, se logró construir un clasificador de pares definiendum y definición que superó el rendimiento del *baseline* seleccionado, aumentando la precisión de 0,66 a 0,84 y la F1 de 0,58 a 0,78.

Además, se utilizó un conjunto de datos etiquetados creado en colaboración entre estudiantes de la Facultad de Comunicación y la Facultad de Ingeniería, el cual también resulta útil para analizar los resultados del módulo. En resumen, este módulo de clasificación permite un mejor filtrado de las pistas, separando las útiles de las que no lo son. Cabe destacar que este módulo es independiente y no se integró con el módulo de generación de pistas.

Capítulo 5

Conclusiones y Trabajo Futuro

En este capítulo, se presentan los resultados obtenidos en este proyecto de grado, se evalúan los aportes del trabajo y se identifican las dificultades encontradas. Además, se discuten las posibles extensiones del proyecto y se realiza una autocrítica que permita identificar qué aspectos pueden ser mejorados en futuros trabajos.

Se logró construir un sistema de generación de pistas para crucigramas que permite generar pistas de manera automática a partir de cuentos en inglés para niños, utilizando herramientas del Procesamiento de Lenguaje Natural, lo que puede ahorrar tiempo y esfuerzo en la creación manual de pistas. Además, se construyó una API para integrar el sistema en otras aplicaciones de crucigramas existentes.

Se lograron generar 2209 pistas a partir de 400 cuentos, y se evaluó la precisión del sistema mediante la clasificación manual de las pistas generadas. Se trabajó con un equipo de 11 personas para etiquetar estas pistas y se logró que las pistas generadas tuvieran una accuracy del 72 %, lo que supera al *baseline* del módulo de generación de pistas (método "to be") tomado de trabajos anteriores, implementado inicialmente en Esteche & Romero (2015). Además, se construyó un clasificador para determinar si una pista generada es correcta o no, que alcanza una precisión del 84 % y una medida f1 del 78 %.

Se identificaron dificultades durante el proyecto, como la necesidad de hardware específico, como el uso de GPU para utilizar algunas bibliotecas (que permitiría acelerar el procesamiento de los textos) y paquetes de procesamiento de lenguaje natural. También se reconoció que algunos métodos de generación de pistas son más eficaces que otros, lo que puede generar repetitividad en las pistas generadas. Sin embargo, se reconoce que esto se debe en parte a que algunos métodos no generaron tantas pistas como otros.

Se logró integrar el sistema en el módulo de crucigramas de CINACINA mediante la API desarrollada, tanto para los crucigramas como para la sopa de letras (funcionalidad adicional que no estipulada al inicio del proyecto pero también se pudo aplicar este modulo en el caso de uso).

En cuanto a las posibles extensiones del proyecto, se podría continuar trabajando en la mejora de los métodos de generación de pistas. El sistema es modular y permite la integración de nuevos métodos o reglas para generar pistas, lo que amplía su capacidad de adaptación. Se puede continuar explorando más modelos de procesamiento de lenguaje natural, como los basados en redes neuronales. Esto podría hacerse utilizando modelos de lenguaje como GPT, que se intentó utilizar durante el desarrollo del proyecto, pero cuyo uso era muy restringido en ese momento. Aunque durante la etapa final del proyecto, se observó que el uso de la serie de GPT se había

vuelto cada vez más accesible y popular en la comunidad científica. En ese momento, ya habían surgido nuevas versiones y mejoras de este modelo de procesamiento de lenguaje natural, lo que permite una mayor accesibilidad y facilidad de uso. Por lo tanto, se considera que sería interesante explorar la aplicación de estos modelos en trabajos futuros, dado que su uso está cada vez más extendido y su potencial en la generación de pistas para crucigramas puede ser significativo. Estos trabajos podrían ser bien mediante ingeniería de "prompts" con modelos preentrenados populares como la API (Interfaz de Programación de Aplicaciones) de ChatGPT, o bien mediante el entrenamiento o "finetuning" de algún modelo acorde para la cantidad de datos disponible. También se puede continuar utilizando los métodos ya utilizados en el proyecto pero ampliando las reglas y patrones, iterando sobre los resultados obtenidos.

En cuanto a la autocrítica, se reconoce que el proyecto ha sido un éxito en términos de resultados y aportes, pero también se identifican áreas que pueden ser mejoradas en trabajos futuros. Se reconoce la necesidad de resolver los problemas de hardware y de mejorar los métodos de generación de pistas para reducir la repetitividad de las pistas. También se reconoce la necesidad de mejorar el proceso de etiquetado manual de las pistas generadas, con el fin de mejorar la precisión del clasificador automático.

En cuanto a la gestión del proyecto, se destaca la importancia del trabajo en equipo y la utilización de herramientas de comunicación y gestión de proyectos para coordinar el trabajo y mantener el progreso. Se mantuvo una comunicación constante con el equipo de etiquetado manual de pistas, lo que permitió completar la tarea en el plazo establecido.

En conclusión, el sistema de generación de pistas para crucigramas ha sido un éxito en términos de resultados y aportes. Se ha logrado desarrollar un sistema que permite la generación automática de pistas a partir de cuentos en inglés para niños, con una precisión que supera al *baseline* tomado de trabajos anteriores. El diseño es modular y permite la integración de nuevos métodos o reglas para generar pistas, lo que amplía su capacidad de adaptación y mejora su precisión. Aunque se han identificado algunas dificultades y áreas que pueden ser mejoradas en trabajos futuros, se reconoce que el proyecto ha sido un gran avance en la generación automática de pistas de crucigramas y puede tener aplicaciones prácticas en el campo de la educación.

Apéndice A

Bibliotecas

En este apéndice se presenta una breve introducción a las bibliotecas y frameworks considerados y analizados para su uso en la aplicación. Estas bibliotecas proveen modelos y funciones muy útiles para las herramientas que se comentarán en la siguiente sección.

Todas las bibliotecas seleccionadas tienen compatibilidad con Python, un requerimiento excluyente en este caso, ya que se decidió implementar el módulo en este lenguaje y de preferencia de código abierto.

A.1. SpaCy

SpaCy (Honnibal & Montani (2017)) es una biblioteca de procesamiento de lenguaje natural de código abierto desarrollada en Python[1]. La biblioteca se enfoca en proporcionar soluciones prácticas y eficientes para tareas comunes del procesamiento del lenguaje, como la tokenización, análisis morfosintáctico, reconocimiento de entidades nombradas y análisis de dependencias.

La mayor ventaja de SpaCy es su capacidad para ofrecer múltiples modelos y configuraciones, lo que permite a los desarrolladores ajustar los hiperparámetros, medir las métricas y obtener visualizaciones de las herramientas utilizadas. Entre sus principales funcionalidades, incluye modelos pre-entrenados para varios idiomas, lo que facilita la implementación rápida de soluciones de PLN. En este proyecto se utilizaron y analizaron varias herramientas dentro de la biblioteca, como la representación vectorial de palabras, los modelos reconocedores de entidades, analizadores de sintaxis, etiquetadores de roles semánticos, separadores de oraciones, tokenizadores y POS taggers.

En resumen, SpaCy es una herramienta versátil y eficiente para el procesamiento del lenguaje natural con una gran comunidad que proporciona un fácil soporte, lo que lo convierte en una excelente opción para el desarrollo.

A.2. Natural Language Toolkit (NLTK)

El Natural Language Toolkit (NLTK) Bird et al. (2009) es una biblioteca de código abierto para el procesamiento del lenguaje natural, desarrollada en Python[2]. NLTK está diseñado principalmente para la investigación

[1]Disponible en *SpaCy Github* (2016)
[2]Disponible en *NLTK Github* (2005)

y la enseñanza en el campo de la lingüística computacional y la inteligencia artificial.

A diferencia de SpaCy, que se enfoca en ofrecer soluciones prácticas y eficientes para tareas comunes de procesamiento de lenguaje natural, NLTK se centra más en la exploración y análisis detallado del lenguaje, lo que lo convierte en una herramienta esencial para la investigación y el aprendizaje en PLN. La biblioteca ofrece una amplia variedad de algoritmos y técnicas, permitiendo a los usuarios experimentar y personalizar sus soluciones para adaptarse a sus necesidades específicas. A pesar de no ser tan eficiente en términos de rendimiento como SpaCy, sigue siendo ampliamente utilizado y apreciado por su enfoque pedagógico y la diversidad de recursos disponibles para investigadores y estudiantes en el campo del procesamiento del lenguaje natural.

Entre las funcionalidades de NLTK utilizadas en el proyecto se encuentran el tokenizador, el etiquetador de categorías gramaticales (POS tagger), el reconocedor de entidades con nombre y la interfaz con la base léxica WordNet (Miller (2006)). Además, se experimentó con la visualización de árboles de constituyentes.

En resumen, NLTK es una herramienta muy útil para la investigación en PLN, con una amplia variedad de recursos y técnicas disponibles para su uso y experimentación.

A.3. Stanza (Stanford NLP)

Stanza (anteriormente conocida como Stanford NLP) (Qi et al. (2019)) es una biblioteca de procesamiento del lenguaje natural desarrollada por el grupo de investigación en inteligencia artificial de la Universidad de Stanford. Esta biblioteca de código abierto, implementada en Python[3], proporciona una serie de herramientas y funcionalidades para analizar y procesar textos. Stanza se basa en los avances más recientes en modelos de aprendizaje profundo y ofrece alta precisión en diversas tareas de PLN, como la tokenización, el análisis morfológico, el análisis de dependencias y el reconocimiento de entidades nombradas.

Una de las ventajas de Stanza es su fácil integración y uso en proyectos de PLN. Además de los modelos de aprendizaje profundo, también proporciona acceso a los modelos clásicos de Stanford CoreNLP a través de una interfaz en Python. Stanza se centra en ofrecer soluciones precisas y eficientes en el procesamiento del lenguaje natural, y su capacidad para manejar una amplia variedad de idiomas lo convierte en una herramienta valiosa para investigadores y desarrolladores del área.

A.4. AllenNLP

AllenNLP (Gardner et al. (2017)) es una biblioteca de procesamiento del lenguaje natural desarrollada por el Allen Institute for Artificial Intelligence (AI2). Escrita en Python [4], esta biblioteca se basa en el popular framework de aprendizaje profundo PyTorch, proporcionando un conjunto de componentes modulares y flexibles para la construcción y experimentación de modelos de PLN de última generación.

A diferencia de otras bibliotecas de PLN, AllenNLP se enfoca en la investigación y el desarrollo de modelos de aprendizaje profundo para resolver tareas complejas, como la respuesta a preguntas, el análisis de sentimientos, la traducción automática y la generación de texto. La biblioteca proporciona una amplia variedad de

[3]El código está disponible en *Stanza Github* (2019)
[4]Disponible en *AllenNLP Github* (2020)

herramientas para cargar datos, tokenizar y construir modelos personalizados, lo que facilita la implementación y experimentación con modelos de PLN.

Una de las principales características de AllenNLP es su capacidad para preentrenar y realizar fine-tuning modelos de lenguaje contextualizados, como BERT y sus variantes, lo que permite a los investigadores y desarrolladores aprovechar el poder de los modelos de lenguaje preentrenados en sus aplicaciones de NLP. Además, AllenNLP ofrece una variedad de modelos preentrenados y herramientas de evaluación para ayudar a los usuarios a comenzar rápidamente con sus proyectos de PLN.

En resumen, AllenNLP es una biblioteca avanzada y centrada en la investigación que facilita el desarrollo y la experimentación con modelos de PLN de última generación para abordar desafíos complejos en el campo del procesamiento del lenguaje natural.

A.5. Huggingface Transformers

Hugging Face Transformers (Wolf et al. (2020)) es una biblioteca de Python desarrollada por la empresa Hugging Face, que se ha convertido en un estándar en la comunidad de procesamiento del lenguaje natural y aprendizaje profundo. La biblioteca ofrece una amplia gama de modelos de lenguaje preentrenados y arquitecturas que pueden ser fácilmente integrados y adaptados en diversas aplicaciones. Transformers proporciona una interfaz unificada y fácil de usar para el preentrenamiento, el fine-tuning y la implementación de estos modelos en tareas como la generación de texto, el análisis de sentimientos, la traducción automática, y la respuesta a preguntas.

Además de su fácil acceso a modelos de lenguaje preentrenados, Hugging Face Transformers también ofrece una serie de herramientas y recursos para simplificar el desarrollo y la experimentación con modelos de NLP. La biblioteca es compatible con frameworks de aprendizaje profundo populares, como PyTorch y TensorFlow, lo que permite a los usuarios trabajar con sus preferencias y aprovechar las ventajas de cada framework [5]. En resumen, Hugging Face Transformers es una biblioteca de vanguardia y altamente versátil que ha facilitado la implementación en una amplia gama de aplicaciones de PLN.

A.6. Scikit Learn

Scikit-learn (Pedregosa et al. (2011)) es una biblioteca de aprendizaje automático de código abierto desarrollada en Python que se enfoca en proporcionar algoritmos y herramientas eficientes y fáciles de usar para la minería y análisis de datos. Aunque no se especializa en procesamiento del lenguaje natural, Scikit-learn ofrece una amplia variedad de algoritmos de clasificación, regresión, clustering y reducción de dimensionalidad que pueden ser aplicados a tareas de PLN. La biblioteca también incluye herramientas para selección de modelos, validación cruzada y evaluación del rendimiento, lo que facilita el desarrollo y ajuste de modelos de aprendizaje automático [6].

En el contexto de PLN, Scikit-learn proporciona módulos para la extracción de características de texto, como la vectorización de palabras y la transformación de texto en representaciones numéricas. Estas representaciones se pueden utilizar junto con algoritmos de aprendizaje automático para abordar tareas como la clasificación de

[5] Codigo en *Huggingface Github* (2020)
[6] Disponible en *scikit-learn Github* (2020)

texto, el análisis de sentimientos y la agrupación de documentos. En resumen, Scikit-learn es una biblioteca ampliamente utilizada en la comunidad de aprendizaje automático que, aunque no fue específicamente diseñada para PLN, ofrece una gran cantidad de algoritmos y herramientas que pueden ser aplicados con éxito en diversas tareas de procesamiento del lenguaje natural.

Apéndice B

Transformers

En este apéndice se presenta una breve introducción a la arquitectura de Transformers, utilizada por varias de las herramientas implementadas, así como el método de Questions Answering utilizado para la extracción de definiciones y algunas de las arquitecturas del clasificador implementado. La información presentada se basa principalmente en el trabajo original de los autores de Transformers Vaswani et al. (2017).

La arquitectura de Transformers se desarrolló para resolver tareas en las que la unidad de trabajo son secuencias de entrada y salida, permitiendo manejar dependencias de largo alcance. Se basa en mecanismos de atención para computar representaciones de la entrada y la salida sin necesidad de convolución o alineación de secuencias.

El método de Questions Answering, por otro lado, se utiliza para extraer información específica de un texto en respuesta a una pregunta. En este caso, se utiliza para extraer las definiciones de términos de un cuento. En cuanto al clasificador implementado, se utilizaron varias arquitecturas de redes neuronales, incluyendo LSTM, CNN y Transformers. Cada arquitectura fue evaluada en función de su desempeño y complejidad para seleccionar la mejor opción para el caso de uso.

Arquitectura del *transformer* El *transformer* se compone de dos elementos fundamentales: el codificador y el decodificador. Estos elementos están acompañados de componentes auxiliares para el preprocesamiento y el posprocesamiento, así como un último componente para el posprocesamiento de la salida. Estos componentes se pueden visualizar en la figura B.1.

El proceso de codificación del *transformer* consiste en generar una representación numérica de la secuencia de entrada de símbolos (palabras). El codificador está compuesto por dos subcapas: una capa de atención múltiple (*multi-head self-attention mechanism*) y una red neuronal simple completamente conectada (*fully connected feed forward*).

Por otro lado, el proceso de decodificación del *transformer* implica la generación de una secuencia de salida de forma secuencial. El decodificador está compuesto por tres subcapas: la primera es un mecanismo de atención múltiple con máscara (*inner attention* multi-cabeza), la segunda es un mecanismo de atención múltiple (*multi-head attention*) y la última subcapa es, nuevamente, una red neuronal completamente conectada (*fully-connected*).

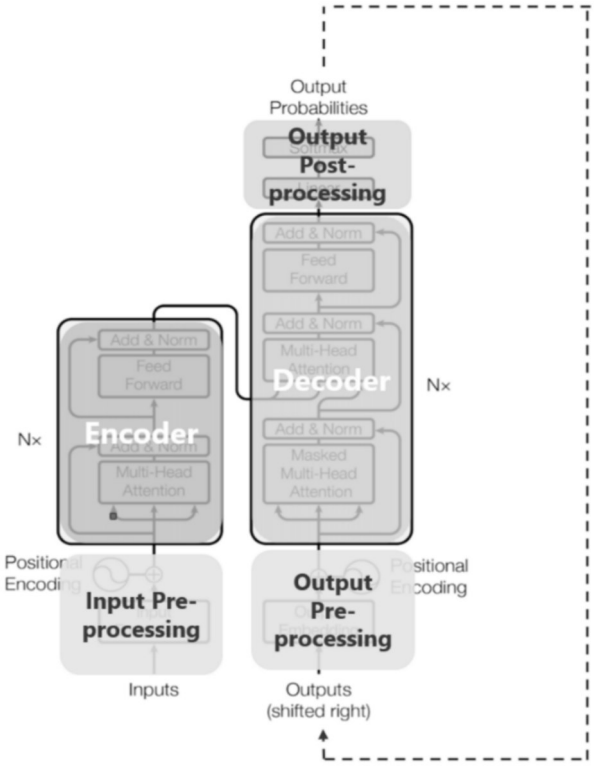

Figura B.1: Componentes principales de un transformer

B.1. Attention is all you need

A diferencia de los modelos recurrentes o convolucionales, esta arquitectura se basa en mecanismos de atención y se compone de capas de atención interna (*intra-attention*) y capas neuronales completamente conectadas tanto para el codificador como para el decodificador. La atención es una función que relaciona una entidad *query* con un par de valores *key-value*, ambos representados por vectores. La salida de la función de atención se calcula como una suma ponderada, donde los pesos se obtienen mediante una función de compatibilidad entre la *query* y la *key*.

La ventaja de la arquitectura *transformer* es que permite una gran escalabilidad y el procesamiento de secuencias más largas en comparación con los modelos anteriores. El entrenamiento se realizó utilizando una infraestructura con una sola máquina con 8 GPU. Para la pareja de idiomas inglés-alemán, se utilizó el conjunto de datos WMT 2014 English-German, que consta de aproximadamente 4 millones de pares de oraciones. Para la pareja inglés-francés, se utilizó el conjunto de datos WMT 2014 English-French, que consta de 36 millones de pares de oraciones.

B.2. Question Answering

Los modelos de lenguaje permiten predecir la siguiente palabra de un flujo de palabras a partir de texto. Pueden estar basados en reglas (denominados estáticos) o en redes neuronales, y calculan las probabilidades dada una secuencia de entrada. Estos modelos se clasifican según la cantidad de palabras que toman como insumo para obtener la probabilidad de la siguiente palabra, existiendo modelos unigramas, bigramas o n-gramas.

Dentro de la recuperación de información a partir de textos se encuentran los sistemas de preguntas y respuestas (QA). Los modelos y sistemas de QA pueden ser generativos y cerrados, donde la respuesta es generada por completo por el modelo, generativos y abiertos, donde el modelo genera la respuesta a partir del contexto, o extractivos, donde el modelo extrae la respuesta del contexto. Es importante destacar que existen otras clasificaciones de modelos QA según su alcance este limitado o no a cierto dominio.

Apéndice C

Mejoras en Resolvedor de Correferencias

En este apéndice se presentan las mejoras vistas, implementadas y utilizadas para el resolvedor de correferencias de AllenNLP. Este análisis de problemáticas con el modelo y sus soluciones fue realizado por la empresa "NeuroSys" y se detalla en profundidad en M. Maślankowska (2021).

Los modelos de resolvedores de correferencias actuales, incluido el de AllenNLP, utilizan métodos estadísticos para encontrar qué palabras o 'tokens' hacen referencia a la misma entidad o elemento del texto. Esto lleva a confusión y ambigüedad en algunos casos, y por lo tanto los modelos parecen estar incompletos cuando no se adapta una forma efectiva de tratar con estos casos.

Los principales problemas que presentan en general los modelos de resolutores de correferencias, en particular el de AllenNLP, son:

- Clusters sin alguna mención con significado alguno (por ejemplo, un cluster compuesto solo con pronombres). Esto puede generar que cualquiera de ellos reemplace a todos, pero al no tener contenido semántico el texto se convierte en agramatical.

- Tratar la primera aparición de un cluster como su elemento principal y no el más significativo, esto puede hacer que si un pronombre aparece antes que un nombre propio, entonces se reemplacen todas las ocurrencias por el pronombre.

- Clusters en spans anidados, es decir, menciones anidadas de distintas correferencias.

En el primer problema, la solución propuesta es fácil: si un cluster no contiene ningún significado semántico, este se elimina. La idea de esto es que, en primer lugar, el cluster no sirve, y reemplazar los distintos elementos por el "HEAD" no agrega valor, sino ruido al texto. Podemos ver en la figura C.1 cómo al sustituir las correferencias, el texto pierde sentido, y al eliminar el cluster, por lo menos, no empeora el resultado obtenido.

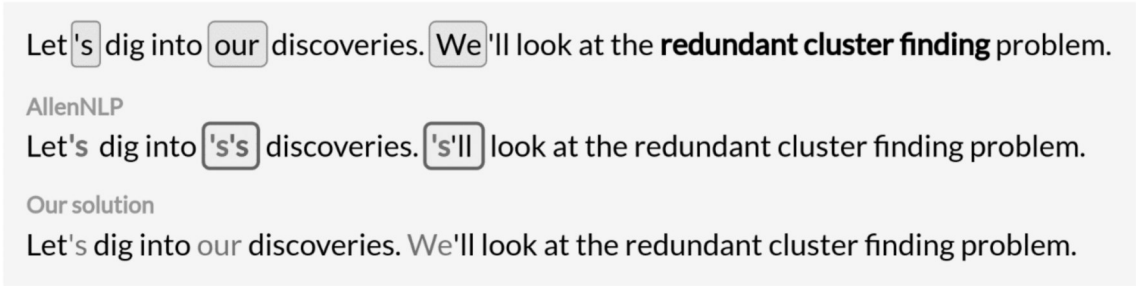

Figura C.1: Problema de cluster sin significado.

Para el segundo problema, debemos buscar otro nuevo candidato como representante o "HEAD" del cluster a reemplazar, y no utilizar la primera mención del cluster como representante. Entonces, se propone utilizar la primera mención del cluster que sea un sustantivo, y de tal forma asegurar que posea algún contenido semántico. Como se muestra en la figura C.2, la primera mención del cluster hace que al reemplazarlo, el texto quede con menos información que al inicio, mientras que al tomar el sustantivo, solucionamos de forma correcta las correferencias.

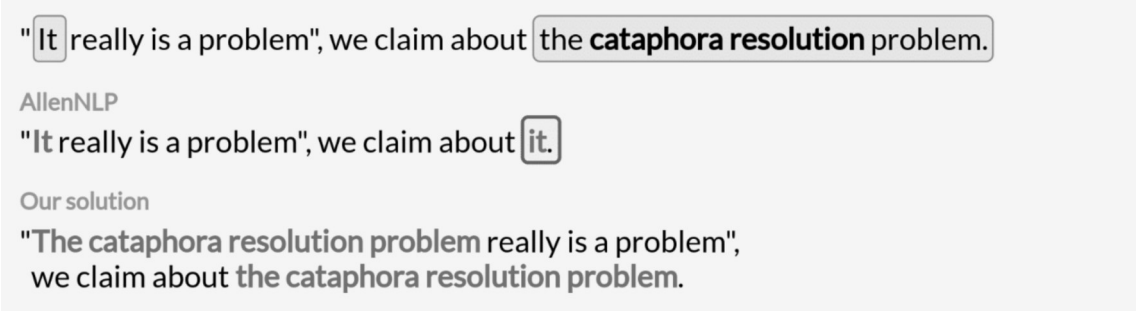

Figura C.2: Problema de cluster con un representante incorrecto.

Finalmente, en el tercer y último problema considerado, el modelo de AllenNLP intenta resolver ambos clusters al mismo tiempo, y esto hace que el texto en algunos casos quede agramatical. Para esto, se propone solo reemplazar aquellos spans lo más adentro posible que pertenezcan a un cluster, es decir, resolver parcialmente las correferencias en los casos que haya varios clusters en un mismo span. Aunque esto puede parecer que no aporte el mejor resultado, genera que los textos luego de reemplazar las correferencias sean siempre gramaticales. Se muestra en la figura C.3 un ejemplo de esto; podemos ver cómo inicialmente el algoritmo de reemplazo deja el texto de forma incorrecta, mientras que la solución propuesta resuelve correctamente las correferencias.

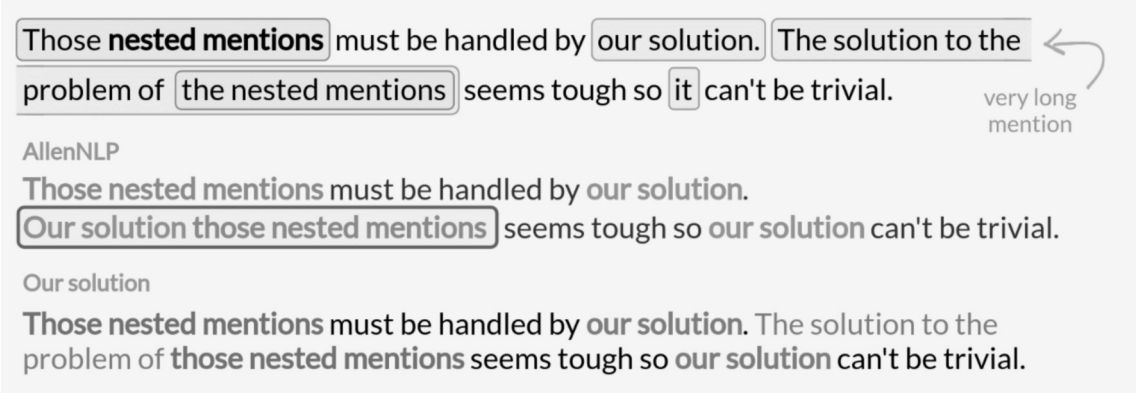

Figura C.3: Problema de menciones anidadas.

Además, en el artículo de referencia se menciona la utilización del resolvedor de correferencias de Hugging Face para combinar ambos y obtener aún mejores resultados, pero no se utilizó en este proyecto, ya que las mejoras previas fueron suficientes para alcanzar los objetivos.

Todas las mejoras descritas previamente fueron implementadas en Python para el modelo de AllenNLP, y se encuentran en el siguiente repositorio NeuroSys (2021). Este mismo se utilizó como guía y referencia para implementar las mejoras.

Apéndice D

Métricas de performance

En el siguiente apéndice se describen las definiciones y nociones básicas de las métricas utilizadas para evaluar lo desarrollado en la aplicación, tanto para los métodos de reglas generados para la extracción de pares de pistas y definiciones, como para el clasificador. Es necesario examinar de alguna forma los resultados para saber de forma objetiva qué tan buenos son. Para ello, es necesario entender ciertos conceptos para analizar los resultados.

En primer lugar, es importante destacar que un clasificador asigna una clase a cada instancia, y esta instancia además tiene una clase verdadera. De esta forma, podemos determinar si la clasificación realizada por el clasificador es correcta o no. Una forma más gráfica de entender esto es a través de la matriz de confusión, como se muestra en la figura D.1, la cual representa un ejemplo para un clasificador de dígitos.

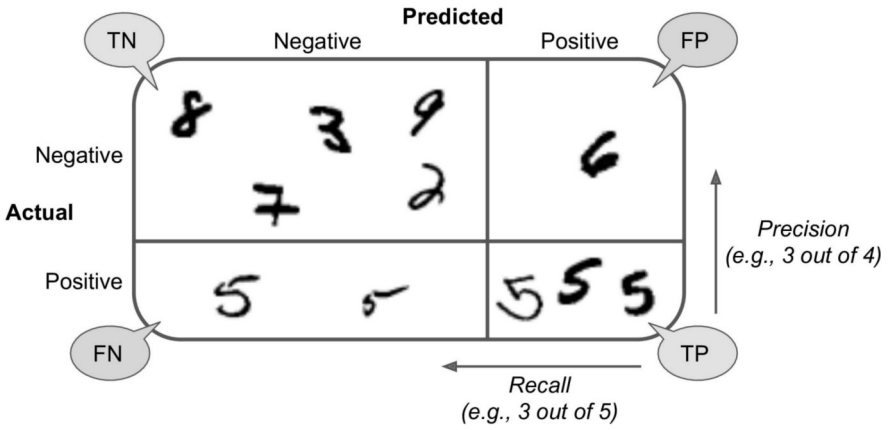

Figura D.1: Matriz de confusión para un clasificador de dígitos (Tomada de Géron (2019)).

Entonces, una matriz de confusión es una matriz que compara las clases verdaderas y las predichas por el clasificador Géron (2019). Las divisiones[1] de la matriz son:

- **Verdaderos positivos (VP)**: Estos casos son clasificados correctamente por el clasificador, donde la clase verdadera de la instancia es positiva y también es la predicha por el clasificador (esquina inferior derecha de la matriz).

[1]Consideramos el caso de un clasificador binario ya que es el caso pertinente al proyecto

- **Falsos positivos (FP)**: Estos casos son incorrectos respecto a la clasificación, su clase real es negativa y la clase asignada es positiva (esquina superior derecha).
- **Falsos negativos (FN)**: Estos casos también son clasificados incorrectamente, su clase es positiva pero el clasificador lo clasifica negativo (esquina inferior izquierda).
- **Verdaderos negativos (VN)**: Estos casos son clasificados correctamente (esquina superior izquierda), tanto la clase real de la instancia como la clasificación es negativa.

Con estas definiciones obtenidas a partir de la matriz de confusión, podemos definir ciertas métricas esenciales utilizadas en el proyecto para evaluar el clasificador y el módulo de extracción[2]:

- **Precisión**: Esta medida representa, de todos los clasificados como positivos, cuáles de ellos fueron clasificados correctamente.
$$Precisión = \frac{VP}{VP + FP} \tag{D.1}$$
- **Recall**: En este caso, la medida indica qué porcentaje de los positivos no clasificó nuestro sistema como tales.
$$Recall = \frac{VP}{VP + FN} \tag{D.2}$$
- **F1**: Esta medida es una media de ambas.
$$F1_{macro} = \frac{2 \times Precisión \times Recall}{Precisión + Recall} \tag{D.3}$$
- **Exactitud** (también llamada accuracy): Esta medida nos dice qué porcentaje de las clasificaciones son buenas, es decir, mide simplemente qué porcentaje de veces coincide la clase real con la clasificada.
$$Accuracy = \frac{VP + VN}{VP + FP + VN + FN} \tag{D.4}$$

Estas métricas fueron las utilizadas para evaluar la performance de los clasificadores explicados en el capítulo de experimentación. Además, notemos que para el método de generación de pistas, se pudo confirmar si estas eran buenas o malas, es decir, si medimos el porcentaje de salidas buenas contra todas las salidas, tenemos una medida de accuracy para el módulo de generación de pistas. Por lo tanto, utilizamos estas para medir la performance en ambos casos.

Por otro lado, en el caso del clasificador, al tener instancias preliminares de evaluación para medir con qué clasificadores realizar una búsqueda más extensa, con los mejores candidatos, se realiza una partición de los datos llamada validación cruzada y de esta forma obtener resultados más confiables para las métricas presentadas previamente.

La idea de este método es poder deshacerse de los sesgos de los datos, que podrían tener configuraciones favorables en algunos casos para algunos modelos. Inicialmente se divide la data en dos conjuntos: data de entrenamiento y data de test. Luego, como se muestra en la figura D.2, se divide la data de entrenamiento en particiones iguales, en la imagen son 5. Luego, se generan 5 instancias de entrenamiento donde se varían las instancias utilizadas para entrenar como las instancias para evaluar. Se toman las métricas en cada iteración y finalmente el resultado es la media de todas las métricas tomadas.

[2]Donde VP son los verdaderos positivos, FP los falsos positivos, VN los verdaderos negativos y FN los falsos negativos.

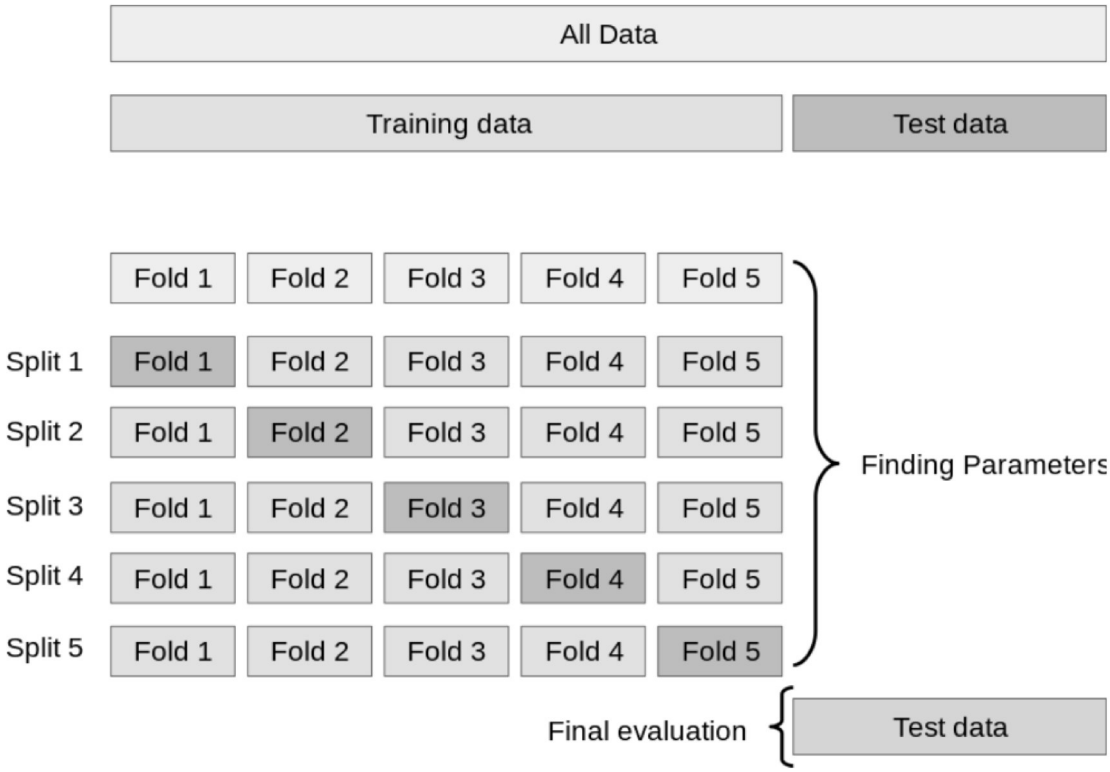

Figura D.2: Partición de datos para 5-fold cross-validation (Tomada de Géron (2019))

En el caso del proyecto, este paso generó los mejores candidatos de los modelos a utilizar. Y luego se llevó a cabo un paso final donde se realiza el entrenamiento con todas las particiones y se evalúa frente al set de test definido inicialmente.

Apéndice E

Planificación del proyecto

En el siguiente apéndice se describe la planificación planteada al inicio del proyecto, basada en la propuesta inicial al comienzo del mismo. Además, mostramos la ejecución de la misma a modo de comparación.

E.1. Planificación esperada

En la figura E.1 podemos ver la planificación presentada en la propuesta al inicio del proyecto en abril de 2022. La misma consistía en nueve meses de trayectoria de proyecto, con dos meses iniciales para búsqueda bibliográfica y planificación del alcance del mismo. Luego, se dedicarían 4 meses para el desarrollo de los métodos de extracción de definiciones, así como la construcción del corpus de datos obtenido. A mediados de este desarrollo también se trabajaría con la construcción de un clasificador. Finalmente, se llevaría a cabo una evaluación de los resultados y la escritura del informe final.

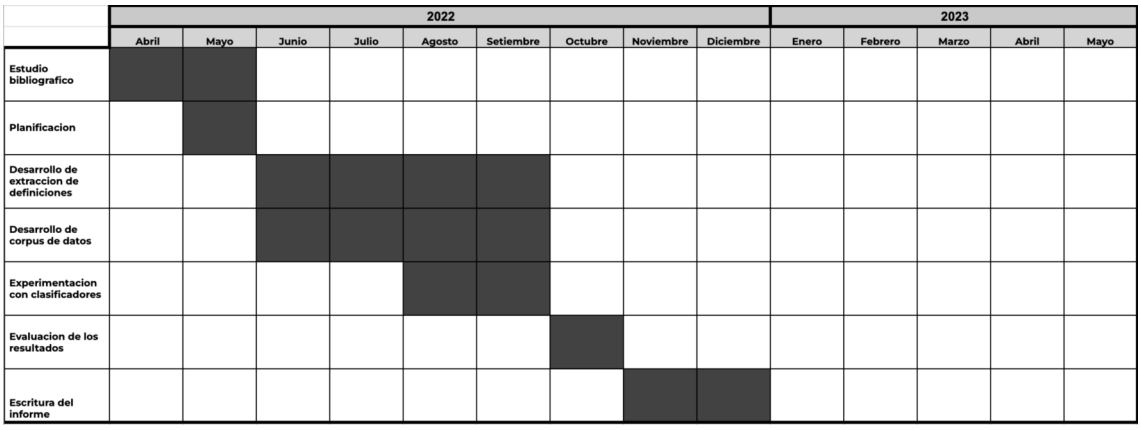

Figura E.1: Planificación planteada en la propuesta del proyecto.

E.2. Ejecución actual

En la figura E.2 se presenta la ejecución del proyecto mes a mes. A grandes rasgos se nota que la planificación fue seguida, tomada como una guía para alcanzar los objetivos planteados en la propuesta. Sin embargo, puede notarse como a mediados del proyecto se dilatan las tareas finales. Esto era esperado ya que la planificación fue tomada como una guía y a lo largo del proyecto se recalculó la planificación de los meses siguientes. En conclusión, la ejecución del proyecto es acorde a lo planificado al inicio y teniendo en cuenta los cambios a

lo largo del proyecto, se realizaron más iteraciones sobre algunas de las tareas (por ejemplo, el desarrollo de métodos o la experimentación sobre el clasificador) para conseguir mejores resultados.

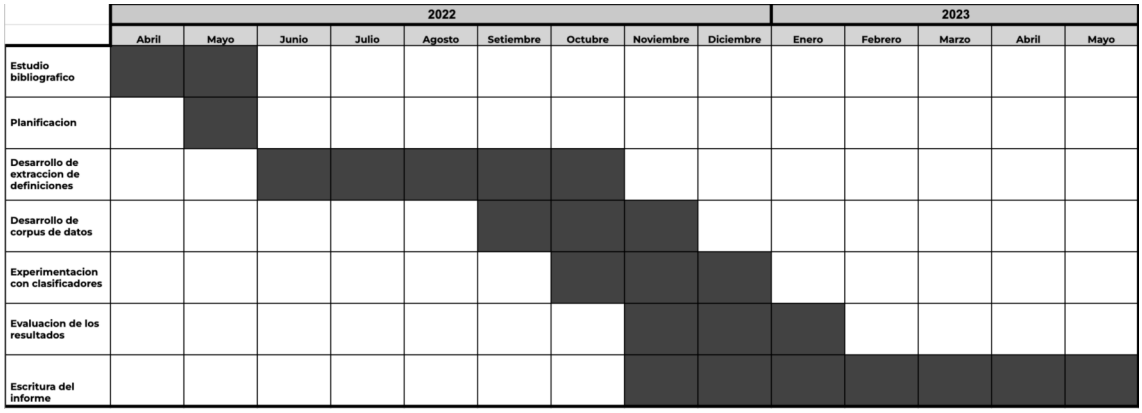

Figura E.2: Ejecución del proyecto.

Referencias

AllenNLP (2020), `https://allenai.org/allennlp/software/allennlp-library`.

AllenNLP Github (2020), `https://github.com/allenai/allennlp`.

Bingsheng Yao, e. a. (2021), 'It is ai's turn to ask humans a question: Question-answer pair generation for children's story books', *Rensselaer Polytechnic Institute* .

Bird, S. (1995), *NLTK: the natural language toolkit.*, In Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions (pp. 69-72), USA.

Bird, S., Klein, E. & Loper, E. (2009), *Natural language processing with Python: analyzing text with the natural language toolkit*, O'Reilly Media, Inc.

Bojanowski, P., Grave, E., Joulin, A. & Mikolov, T. (2016), 'Enriching word vectors with subword information'.

Colombato, F. R. & Musso, A. N. (2022), 'Evaluación y simplifación de textos para enseñanza de inglés', *Universidad de la Repulica, UdelaR* .

Crosswords for kids in English (2023), `https://www.kidspuzzlesandgames.co.uk/puzzle-sheets/type/crossword`. Accessed: 2023-03.

de Marneffe, M.-C., Dozat, T. & Silveira, N. (2014), *Universal Stanford dependencies: A cross-linguistic typology*, European Language Resources Association (ELRA).

Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. (2019), 'Bert: Pre-training of deep bidirectional transformers for language understanding', *Google AI Language* .

Esteche, J. & Romero, R. (2015), 'Extracción de definiciones y generación automática de crucigramas a partir de textos de prensa', *Facultad de Ingeniería, UdelaR* .

FairytaleQA: Question Answering Generation System (2022), `https://github.com/WorkInTheDark/FairytaleQA_QAG_System`. Accessed: 2022.

Gardner, M., Grus, J., Neumann, M., Tafjord, O., Dasigi, P., Liu, N. F., Peters, M., Schmitz, M. & Zettlemoyer, L. S. (2017), *AllenNLP: A Deep Semantic Natural Language Processing Platform.*

Géron, A. (2019), *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, O'Reilly, USA.

Honnibal, M. & Montani, I. (2017), *spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing.*

Huggingface Github (2020), `https://github.com/huggingface/transformers`.

Jurafsky, D. & Martin, J. H. (2009), *Speech and Language Processing*, Pearson, New Jersey, NJ.

Jurafsky, D. & Martin, J. H. (2023), 'Speech and language processing', https://web.stanford.edu/~jurafsky/slp3/2.pdf.

Kamran Kowsari, K. Jafari, M. H. S. M. L. B. (2019), 'Text classification algorithms: A survey', *Machine Learning on Scientific Data and Information* .

M. Maślankowska, P. M. (2021), 'How to make an effective coreference resolution model', https://neurosys.com/blog/effective-coreference-resolution-model. Accessed: 2022-05-06.

Marcus, M. P., Santorini, B., Marcinkiewicz, M. A. & Taylor, A. (1999), *Treebank-3*.

Mikolov, T., Chen, K., Corrado, G. & Dean, J. (2013), *Efficient Estimation of Word Representations in Vector Space*.

Miller, G. A. (2006), *WordNet: a lexical database for English*, USA.

Màrquez, L., Carreras, X., Litkowski, K. C. & Stevenson, S. (2008), 'Semantic role labeling: An introduction to the special issue', *Computational Linguistics* .

NeuroSys (2021), 'Coreference resolution in nlp with allennlp and huggingface', https://github.com/NeuroSYS-pl/coreference-resolution. Accessed: 2022-05-06.

NLTK Github (2005), https://github.com/nltk/nltk.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. et al. (2011), 'Scikit-learn: Machine learning in python', *Journal of machine learning research* **12**(Oct), 2825–2830.

Pennington, J., Socher, R. & Manning, C. D. (2014), *GloVe: Global Vectors for Word Representation*.

Punyakanok, V., Roth, D. & tau Yih, W. (2008), 'The importance of syntactic parsing and inference in semantic role labeling', *University of Illinois at Urbana-Champaign* .

Qi, P., Zhang, Y., Zhang, Y., Bolton, J. & Manning, C. D. (2019), *Stanza: A Python Natural Language Processing Toolkit for Many Human Languages*.

Rajpurkar, P., Zhang, J., Lopyrev, K. & Liang, P. (2016), *SQuAD: 100,000+ Questions for Machine Comprehension of Text*, Stanford University.

ReadWorks (2008), https://www.readworks.org/. Accessed: 2022-03.

Rigutini, L., Diligenti, M., Maggini, M. & Gori, M. (2012), 'Automatic generation of crossword puzzles', *International Journal on Artificial Intelligence Tools (IJAIT)* **21**, 22.

Rischewski, T. & Berger, G. (2022), 'Generación de preguntas y respuestas para comprensión lectora en inglés utilizando modelos neuronales', *Universidad de la Repulica, UdelaR* .

Rusli, A., Young, J. C. & Iswari, N. M. S. (2020), Identifying fake news in indonesian via supervised binary text classification, *in* '2020 IEEE International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology (IAICT)', pp. 86–90.

Sanh, V., Debut, L., Chaumond, J. & Wolf, T. (2019), 'Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter', *Hugging Face* .

scikit-learn Github (2020), https://github.com/scikit-learn/scikit-learn.

SpaCy (2016), `https://spacy.io/`.

SpaCy Github (2016), `https://github.com/explosion/spaCy`.

Stanza Github (2019), `https://github.com/stanfordnlp/stanza`.

Steven Bird, Ewan Klein, E. L. (2019), *Natural Language Processing with Python*, O'Reilly, USA.

Tosi, A. & Percovich, A. (2019), 'Aplicaciones lúdicas de soporte a la enseñanza de lenguas', *Universidad de la Repulica, UdelaR* .

Universal POS tags (2014), `https://universaldependencies.org/u/pos/`. Accessed: 2022-04.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. & Polosukhin, I. (2017), Attention is all you need, *in* 'Advances in neural information processing systems'.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q. & Rush, A. M. (2020), *HuggingFace's Transformers: State-of-the-art Natural Language Processing*.

Glosario

API (Interfaz de Programación de Aplicaciones) Conjunto de reglas y protocolos que permiten la interacción y comunicación entre diferentes software o componentes, facilitando la integración y el acceso a funcionalidades de una aplicación. 76

baseline Punto de referencia inicial o una línea de base utilizada para comparar o medir el progreso, el rendimiento o los cambios en un proyecto o sistema. 70

hiperónimo Palabra que engloba categorías o clases más amplias que incluyen términos más específicos relacionados. 53

pipeline Secuencia de pasos o etapas ordenadas de manera que la salida de cada paso se convierte en la entrada del siguiente, comúnmente utilizado en procesos de procesamiento de datos. 5

threshold Valor límite o umbral que se utiliza para determinar si un dato o una medida específica alcanza un nivel predeterminado, a partir del cual se toman decisiones o se activan acciones. 54

tupla Estructura de datos que permite almacenar múltiples elementos o valores en un solo objeto inmutable. 5