Universidad Zaragoza
1542

Trabajo Fin de Grado

Verbalización de ontologías para su explotación mediante modelos de lenguaje

Verbalization of ontologies for exploitation by language models

Autor

Adrián Espino Candalija

Directores

Jorge Raúl Bernad Lusilla

Jorge Gracia del Río

Titulación

Grado en Ingeniería en Informática

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2022

AGRADECIMIENTOS

Agradezco a mis directores de TFG Jorge Raúl Bernad Lusilla y Jorge Gracia del Río por su ayuda continua durante la realización del trabajo siempre dispuestos a resolver mis dudas y dar apoyo ante los contratiempos.
También agradecer a mi familia y amigos por crear un entorno amigable y sano durante la realización del grado.

RESUMEN

Un intérprete de lenguaje natural o modelo del lenguaje (ML) es una red neuronal que sigue la arquitectura de los *transformers*, que a día de hoy son el estado del arte en muchas técnicas de NLP (Natural Language Processing). El objetivo de un modelo del lenguaje probabilístico es aprender la probabilidad conjunta de secuencias de palabras sobre un vocabulario. Sin embargo esto se vuelve cada vez más complicado conforme aumenta el número de palabras debido a la maldición de la dimensionalidad.

Una ontología es una estructura orientada a un área temática que guarda información sobre propiedades y relaciones entre una serie de conceptos y categorías.

Los modelos de lenguaje no son capaces de procesar conocimiento estructurado como el contenido por una ontología, por ello buscamos implementar un método multilingüe que extraiga el contenido explicito en las ontologías obteniendo una descripción escrita en lenguaje ingles natural. El objetivo principal de este trabajo es emplear un corpus de verbalizaciones de distintas ontologías para ser procesado por un modelo del lenguaje capaz de encontrar nuevas relaciones de equivalencia entre las ontologías empleadas. Adicionalmente el método busca hacer más accesible y explicable el conocimiento que contiene la ontología a humanos no expertos en ontologías.

Existen en la web varios métodos de verbalización ya implementados, han sido utilizados como fuente de inspiración con objeto de no reinventar la rueda e implementar un método de verbalización útil y de buena calidad.

Los resultados obtenidos indican que el corpus de verbalizaciones ha sido útil para encontrar nuevas relaciones de equivalencia, la mejoría de la medida *F1-score* depende del modelo del lenguaje empleado, se ha conseguido un aumento medio del 8.361 % en modelos *BERT base* y de un 4.5 % en modelos *BERT large*.

Palabras clave: *ontología, transformers, modelo del lenguaje, red neuronal, procesamiento de lenguaje natural, OWL, Word embedding, BERT, Roberta.*

Índice

Capítulo 1

Introducción

1.1. Motivación

En la actualidad, el aumento incontrolable de información no estructurada en la web (mensajes, comentarios, valoraciones, textos,...) ha creado una necesidad de buscar métodos eficientes para procesar esta información, denominada información en lenguaje natural. Los modelos del lenguaje basados en *transformers* son capaces de procesar dichos textos escritos en lenguaje natural.

Un intérprete de lenguaje natural o modelo del lenguaje consiste en una red neuronal que analiza texto escrito en lenguaje natural para hallar la probabilidad de que una secuencia de palabras aparezcan en ese mismo orden respecto al vocabulario completo de palabras.

Este trabajo busca emplear modelos de lenguaje para encontrar relaciones de equivalencia entre conceptos pertenecientes a diferentes ontologías, en la actualidad existen modelos capaces de procesar texto en varios idiomas, sin embargo la mayoría de los modelos más grandes y capaces se centran en el procesamiento de lenguaje natural en inglés. El desarrollar un método multilingüe permite obtener una descripción escrita en lenguaje natural (verbalización) en cualquier idioma siempre que haya sido adaptada la traducción al mismo.

Por sí misma la semántica implícita en los modelos del lenguaje ya es capaz de predecir relaciones de equivalencia entre ontologías, al verbalizar el conocimiento estructurado contenido en las ontologías se puede complementar a los modelos de lenguaje de forma que predigan con mayor exactitud relaciones de equivalencia.

1.2. Contexto

En un principio se estudiaron varios proyectos ne la web que trataban sobre la verbalización de ontologías, algunos señalaban la existencia de idiomas desde los cuales es posible obtener una mejor verbalización, otros exponen los beneficios de emplear refinamiento semántico para eliminar la redundancia.

También se estudió la existencia de herramientas/aplicaciones de uso libre que proveían de un servicio de verbalización, como *NaturalOWL* [1] o *DogmaModeler* [2] que es capaz de verbalizar en varias lenguas.

Haciendo uso de las estrategias observadas se pretende implementar un método que combine los máximos aspectos positivos posibles.

1.3. Objetivos

El objetivo principal de este trabajo es emplear un conjunto de verbalizaciones de distintas ontologías para ser procesado por un modelo del lenguaje capaz de encontrar nuevas relaciones de equivalencia entre las ontologías empleadas. Se pretende adaptar el modelo para poder traducir cualquier ontología a las lenguas: español, inglés, francés y portugués.

Para cumplir este objetivo van a seguirse las siguientes instrucciones:

1. Formación en los conocimientos necesarios sobre ontologías, herramientas para la explotación de ontologías y modelos de lenguaje natural.

2. Estudio del estado del arte en la verbalización de ontologías.

3. Propuesta y formalización de distintas estrategias para verbalizar una ontología.

4. Implementación y pruebas de las verbalizaciones propuestas usando modelos de lenguaje y evaluación con humanos.

5. Preparación de la memoria y presentación del trabajo fin de grado.

1.4. Herramientas

El método implementado y su evaluación mediante modelos de lenguaje hacen uso de varias librerías, herramientas y datos:

– Ontologies dataset: han sido empleadas varias ontologías durante la implementación del método de extracción de información, estas un una ontología sobre pizzas [3] y un conjunto de datos completo sobre conferencias y sus equivalencias [4].

– Jena: Framework de java de código abierto, permite extraer determinada información de grafos RDF. [5]

– Protégé: Programa de escritorio utilizado para la gestión de ontologías, es capaz de crear ontologías desde cero, añadir propiedades y conceptos. También cuenta con un razonador. [6]

– Pellet Reasoner: Razonador de código abierto en Java empleado para inferir nueva información de los modelos. [7]

– OWL API Semantic Web: Librería de Java empleada en el proyecto para traducir un modelo en formato RDF/OWL a formato XML con objetivo de extraer nueva información. [8]

– Deep Translator: Librería de Python utilizada para traducir el identificador/nombre de las entidades y propiedad a la lengua objetivo. Utiliza el traductor de Google (GoogleTranslator). [9]

– Langid: Librería de Python que permite detectar con precisión la lengua origen de la expresión. [10]

– Librerías destinadas a trabajar con grandes conjuntos de datos (Datasets) [11] y modelos de lenguaje (Transformers) [12].

– Kaggle: plataforma gratuita que pose a disposición de los usuarios un entorno de ejecución para el análisis predictivo y el machine learning. [13]

– Overleaf: Herramienta para la redacción de documentos.

1.5. Estructura de la Memoria

La memoria está compuesta por cinco capítulos incluyendo el actual capítulo introductorio *(Capítulo 1)*. En el *Capítulo 2* SE introducirán los conceptos básicos necesarios para entender el resto del documento. En el *Capitulo 3* se verán las ideas anteriores a este proyecto y se comentarán las razones de la decisión final. En el *Capítulo 4* se explicará el proceso que llevó a la implementación y los componentes clave del proyecto. En el *Capítulo 5* se expondrán los métodos de valoración planteados y los

resultados obtenidos. En el *Capitulo 6* se explican las conclusiones a las que ha dado lugar el proyecto y el trabajo futuro.

Capítulo 2

Conceptos previos

En este apartado se verá una introducción a los conceptos básicos sobre ontologías y modelos de lenguaje necesarios para el trabajo.

2.1. Bases de las ontologías

Las ontologías son usadas para capturar el conocimiento sobre un dominio de interés. Una ontología es una representación de conceptos junto a sus propiedades y relaciones. Los elementos principales en una ontología son las clases, propiedades e individuos: [14]

- Clase: una clase es una representación concreta de un concepto. Se describen mediante declaraciones que indican los requerimientos para ser miembro de ésta, un ejemplo de una clase puede ser *Persona*, que representa el concepto de persona con atributos como su edad, peso o nombre.
- Individuo: Los individuos representan objetos de un dominio de interés, siempre deben tener asignada una clase principal, dos ejemplos de individuos pueden ser *Guillermo* y *Fernando* que pertenecen a la clase *Persona*.
- Propiedad: son relaciones binarias entre individuos o entre un individuo y un literal, un ejemplo de propiedad puede ser *esAmigo*, que representa una relación entre dos individuos de la clase Persona como podrían ser *Guillermo* y *Fernando*, las propiedades pueden ser vistas como las características de las clases.
- Literal: es un dato que suele ser enlazado mediante una propiedad a un individuo, puede presentarse en varias formas como una cadena, un entero, una fecha...

Toda propiedad posee un dominio (conjunto de clases que pueden poseer esta propiedad) y un rango (conjunto de clases a las que puede referenciar la propiedad).

Por ejemplo para la propiedad *comproPizza* el dominio sería la clase *Cliente* ya que hace la acción de comprar la pizza y el rango seria *Pizza* ya que es la clase que recibe la acción.

Dentro de una ontología existen identificadores denominados *IRIs* (International Resource Identifiers) empleados para identificar a la misma ontología y sus elementos, una *IRI* tiene la siguiente forma: <*http://www.semanticweb.org/ontologies/2021/PizzaTutorial*>.

Empleamos identificadores llamados *URIs* (Uniform Resource Locators) para referenciar objetos contenidos en una ontología, una *URI* tiene la siguiente forma: *http://www.semanticweb.org/ontologies/2021/PizzaTutorial#CheesyPizza*.

La sintaxis más utilizada para representar grafos sobre ontologías es la denominada *RDF* (Resource Description Framework) [15], un grafo *RDF* tiene nodos y arcos que vinculan pares de nodos, esto se representa como un conjunto de tripletas *RDF* donde cada tripleta contiene un nodo sujeto, un predicado y un nodo objeto.

Los nodos son *IRI*, literales o nodos en blanco. A los nodos en blanco se les puede dar un identificador local de documento denominado identificador de nodo en blanco. Los predicados son *IRI* y se pueden interpretar como una relación entre los dos nodos o como la definición de un valor de atributo (nodo objeto) para algún nodo sujeto.

Figura 2.1: Grafo *RDF* que muestra tres objetos y dos propiedades, expresa que el individuo *AmericanaPizza1* posee una propiedad llamada *hasTopping* cuyo individuo asignado es *MozzarellaTopping3*, además de que *AmericanaPizza1* es un individuo de la clase *AmericanaPizza*

En el contexto de la web, una ontología suele representarse mediante los modelos *OWL* o *RDFS* [16], donde el lenguaje de ontologías *OWL* está diseñado para ser usado en aplicaciones que necesitan procesar el contenido de la información en lugar de únicamente representar información[17], *RDFS* es una extensión semántica de *RDF*, proporciona mecanismos para describir grupos de recursos relacionados y las relaciones entre estos recursos[15].

Las ontologías suelen ser almacenadas en ficheros de textos, las principales sintaxis son:

- RDF/XML: es una sintaxis definida por la *World Wide Web Consortium* (W3C) para expresar un grafo *RDF* mediante un documento XML. [15]
- Turtle: Permite describir el grafo *RDF* de una forma más compacta, es más sencillo de entender para los humanos y también de modificar manualmente, expresa la información en tripletas que cuentan con un sujeto, un predicado y un objeto. [18]
- OWL/XML: se trata de sintaxis orientada al uso de *OWL 2*, se ajusta al esquema XML por tanto son muy similares salvo porque este suele ser más detallado lo que lleva a formar documentos a menudo muy grandes y costosos de analizar por un computador. [19]

Una ontología debe de ser consistente, es decir, si se añade un individuo a una clase de la que no cumple las restricciones, la ontología se volverá inválida por un problema de consistencia y deberá ser corregida. En aplicaciones destinada a facilitar el trabajo con ontologías la detección de incoherencias suele llevarse a cabo mediante un razonador semántico emitiendo mensajes de ejecución que ayuden al usuario a corregir el problema. Las clases suelen estar organizadas en una jerarquía de clases y subclases conocida como taxonomía. Esta se representa como un árbol donde cada nodo tiene uno o varios nodos padre de los que hereda sus requerimientos y propiedades. Las subclases especializan a sus clases padre, por ejemplo *Perro* es subclase de *Animal*, lo que quiere decir que todos los elementos del conjunto *Perro* cumplen las condiciones para pertenecer al conjunto *Animal*.

2.2. Introducción a los modelos de lenguaje

Un modelo de lenguaje es el componente central en los sistemas de procesamiento de lenguaje natural. La mayoría de los modelos de lenguaje utilizados actualmente son redes neuronales, compuestas por un gran número de neuronas y múltiples capas.

Las capas neuronales se emplean para hallar una representación distribuida de cada palabra *(word embedding)* [20], esto es, transformar cada palabra recibida en un vector de números dentro de un espacio vectorial, que permite obtener mejoras en rendimiento en tareas de procesamiento del lenguaje natural mediante la agrupación de palabras similares, es decir palabras con significados similares tienen representaciones vectoriales similares. Estos son los *word embedding* estáticos, es necesaria una gran cantidad de texto bien formado para entrenar la red neuronal que asigna las representaciones vectoriales a las palabras. [21]

Sin embargo este enfoque tiene algunas limitaciones:

- Al añadir una palabra dentro del espacio vectorial no se tiene en cuenta el contexto de esta, lo que puede causar que el modelo falle en casos en que una palabra tenga más de un significado, por ejemplo la palabra *play* en inglés cambia su significado dependiendo del contexto *play the piano* (tocar el piano) frente a *play football* (jugar al fútbol).

- Las palabras que no se encuentran en el vocabulario del modelo no tienen una representación en el espacio vectorial. Esto causa que deban reentrenarse al menos un vector por cada palabra del vocabulario si queremos añadir la nueva palabra, por ejemplo, en *word2vec* [22] teniendo un vocabulario de 1.000.000 palabras a un tamaño del vector de 300 parámetros, el total de parámetros a entrenar seria de 600.000.000.

Los *word embedding* que emplean BERT y Roberta son contextuales, lo que significa que a diferencia de los *word embedding* estáticos estos si tienen en cuenta el contexto de la palabra a la hora de asignarle una representación dentro del espacio vectorial.

Un modelo de lenguaje probabilístico devuelve la probabilidad de que la secuencia de palabras que se recibe sea correcta, interpretan el lenguaje natural mediante un algoritmo que establece reglas para el contexto de las palabras, después el modelo emplea estos contextos para predecir palabras con gran precisión o para escribir nuevas oraciones. [23]

Un modelo de lenguaje de red neuronal explota su capacidad de aprendizaje para reducir el impacto de la llamada *maldición de la dimensionalidad*, esta maldición se refiere a la necesidad de una gran cantidad de ejemplos de entrenamiento para lograr aprender funciones complejas, esta maldición aparece cuando se aumenta el número de variables de entrada y se necesita discriminar una gran cantidad de combinaciones de estas variables necesitando al menos un ejemplo por cada combinación de valores relevante, en este caso las variables serían las palabras, el problema surge por la gran cantidad de secuencias de palabras que deben ser discriminadas. [24] [25]

La probabilidad de una secuencia de palabras del vocabulario de un modelo (conjunto de palabras únicas del corpus del texto) se obtiene empleando la probabilidad de cada palabra dado el contexto de las palabras detrás de ella, la forma en que se calcula esta probabilidad depende del modelo, por ejemplo empleando la *regla de probabilidad de la cadena (Teorema de Bayes)*:

$$P(w_1, w_2, ..., w_{t-1}, w_t) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)...P(w_t|w_1, w_2, ..., w_{t-1})$$

La mayoría de modelos del lenguaje competitivos tienen una estructura codificador-decodificador, el codificador recibe una secuencia de símbolos (x1, ..., xn) a la que asigna una secuencia de representaciones continuas z = (z1, ..., zn). Dado el vector de contexto z, el decodificador genera una secuencia de salida (y1, ..., ym) de símbolos para cada elemento. En cada paso el modelo es autorregresivo, consumiendo los símbolos generados previamente como entrada adicional al recibir la siguiente entrada. El *transformer* puede tener varias capas apiladas conectadas tanto por el codificador como por el decodificador. [26]

Los modelos contextuales como BERT o Roberta solo emplean la parte del codificador de la estructura de los *transformers*, los codificadores tienen varias capas que funcionan de acuerdo al mecanismo de atención. [26]

2.2.1. Usos de los modelos de lenguaje

Los modelos de lenguaje son el componente principal de los procesadores de lenguaje natural (NLP), algunas de las aplicaciones de estos son las siguientes:

- Reconocimiento de voz: se procesa el audio del habla para crear asistentes de voz como *Siri* o *Alexa*.
- Traducción entre lenguas: consiste en la traducción de información, *Google Translate* emplea un modelo de lenguaje para esto.
- Análisis de información: se analizan las cadenas de texto buscando relaciones entre palabras.
- Análisis del sentimiento detrás de una cadena de texto: se emplean para comprender el sentimiento detrás de una oración escrita en texto, pueden utilizarse para analizar opiniones sobre productos en la web.

2.2.2. Procedimiento de ejecución

Normalmente un modelo de lenguaje primero se entrena con una gran cantidad de información general para después ser re-entrenado para una tarea específica.

Los datos de entrada en lenguaje natural deben de pasar por un preprocesameinto del que se encarga el componente llamado *tokenizer* antes de ser utilizados por un modelo de lenguaje. Este proceso es capaz de eliminar palabras respecto al texto utilizado en la entrada, lo que desemboca en un menor número de parámetros necesarios para entrenar el modelo. La labor del tokenizador de los modelos BERT y Roberta consiste en recibiendo una cadena de entrada, dividirla en subpalabras (tokens) contenidas en el vocabulario del modelo entrenado con anterioridad para cada subpalabra, este vocabulario esta contenido por cerca de 50000 tokens distintos con los que se puede construir casi cualquier secuencia de palabras usada como entrada. Cada token tiene asignado un número.

El número máximo de tokens que puede manejar el tokenizador de *BERT* y *Roberta* en una misma cadena de entrada es de 510, a estos se agregan los tokens iniciales por defecto llamados *[CLS]* (CLS significa *classification*, aparece al principio de cada oración y contiene información sobre el resto de tokens de la cadena) y *[SEP]* (se trata de un token separador, ayuda al modelo a saber que token pertenece a que oración). Existe una diferencia entre el tokenizador de BERT y el de Roberta, ya que Roberta no cuenta con *Next Sentence Prediction* debe analizar cada parte a los lados del token *[SEP]* por separado mientras que BERT es capaz de analizar la oración completa.

[CLS] the man went to the store [SEP] he bought a gallon of milk [SEP]

El procedimiento que vamos a seguir para entrenar y utilizar un modelo del lenguaje es el siguiente:

1. Primero deben prepararse las verbalizaciones descargadas para ello se emplea el *tokenizer*, todo tokenizador de un modelo del lenguaje tiene un límite sobre cuantos tokens puede manejar en una sola oración, por ejemplo los modelos del lenguaje como *GPT-1, GPT-2 y GPT-3* [27] pueden llegar a almacenar 1024 tokens.

   [CLS] the man went to the store [SEP] he bought a gallon of milk [SEP]

2. A continuación se enmascaran un porcentaje de los tokens contenidos en los datos guardados que serán los que el modelo de lenguaje tratará de predecir.

   [CLS] the man [MASK] to the store [SEP] he bought a gallon [MASK] milk [SEP]

Entrenamos el modelo de lenguaje con estos datos enmascarados indicando el número de épocas que deseamos que se ejecuten, una época es una ejecución del entrenamiento, al final de cada época se genera un modelo entrenado, este entrenamiento no es más que una continuación al entrenamiento previo que han sufrido todos los modelos de lenguaje.

3. Posteriormente codificamos la información contenida en la tripleta de ficheros CSV de equivalencias (train, val, test) para poder ser usada como entrada para los modelos de lenguaje, finalmente entrenamos por segunda vez de forma orientada a la clasificación de parejas de conceptos en equivalentes o no equivalentes. Para esto se entrena el modelo con entradas con la siguiente forma:

   [CLS] Concepto1 [SEP] Concepto2 [MASK] [SEP]

   Donde el elemento que ha sido enmascarado corresponde a la relación de equivalencia entre ambos conceptos, valdrá 0 en caso de ser no equivalentes y 1 en caso de ser equivalentes. De esta el modelo devuelve la palabra la palabra mas probable en la posición enmascarada basándose en el contenido de *Concepto1* y *Concepto2*.

4. Por último empleamos el modelo ya entrenado para tratar de predecir la equivalencia o no equivalencia de las parejas de conceptos contenidas en los ficheros CSV de test comparando las predicciones con la realidad.

2.2.3. Modelos de lenguaje a utilizar en el trabajo

Los modelos de lenguaje que vamos a utilizar durante el trabajo son algunos de los mas pupulares: BERT y Roberta. Ambos han sido pre-entrenados empleando un corpus de datos en inglés de forma no supervisada, con textos sin procesar sin ninguna etiqueta en ellos. [28]

Como se comentó anteriormente BERT y Roberta son modelos que solamente emplean la parte del codificador del modelo codificador-decodificador de los *transformers*, el codificador se basa en capas de atención apiladas una encima de otra, el codificador de la versión base de estos modelos cuenta con 12 capas, mientras que la versión extendida o *large* cuenta con 24 capas, estas capas son útiles ya que colaborando entre si permiten que el modelo entienda el contexto y las dependencias de las secuencias. [29]

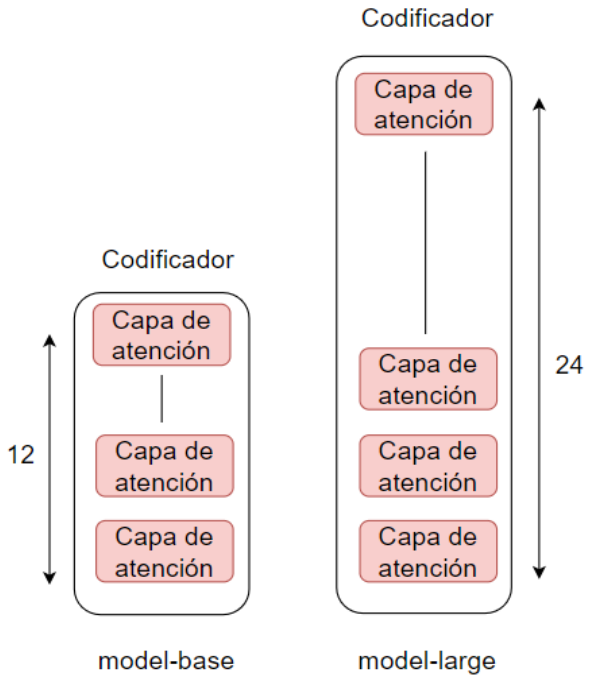

Figura 2.2: Comparación de las capas del codificador entre un modelo base y un modelo *large*

Las versiones base de BERT y Roberta emplean cerca de 110 millones de parámetros mientras que las versiones ampliadas emplean alrededor de 340 millones, por ello la versión base ocupa menos espacio en memoria pero la versión *large* es mucho más precisa. [30]

Los modelos BERT han sido entrenados empleando dos métodos:

- Modelado de lenguaje enmascarado (MLM): el modelo enmascara aleatoriamente el 15 % de las palabras de cada oración que recibe, luego se pasa la oración enmascarada por modelo teniendo este que tratar de predecir las palabras enmascaradas.

- Predicción de la siguiente oración (NSP): en modelo concatena pares de oraciones enmascaradas de forma aleatoria durante el pre-entrenamiento, luego el modelo debe tratar de predecir si realmente esas oraciones estaban escritas en ese orden en el texto original.

Los modelos Roberta a diferencia de los modelos *BERT* solamente han sido entrenados empleando el método de Modelado de lenguaje enmascarado (MLM).

El modelo de BERT ampliado que utilizaremos en este trabajo es especial ya que ha sido pre-entrenado con un modelado de lenguaje enmascarado (MLM) modificado, este nuevo método consiste en que además de enmascarar la palabra completa también se enmascaran todas las subpalabras correspondientes a esa palabra al mismo tiempo.

La principal diferencia entre los modelos BERT y Roberta es el método de enmascaramiento de tokens, mientras que en *BERT* el enmascaramiento se realiza una sola vez durante la preparación de los datos, en la que cada oración se enmascara de 10 maneras distintas, en *Roberta* el enmascaramiento se realiza durante el entrenamiento del modelo, cada vez que se añade una oración contenida en un pequeño lote de oraciones se enmascara, la cantidad de versiones enmascaradas de la oraciones no es limitada como en BERT, por tanto puede ser enmascarada de un mayor número de formas que en BERT.

Capítulo 3

Estado del Arte

En este apartado se hará un apunte sobre las ideas relacionadas con verbalización anteriores a este proyecto que fueron analizadas como base de estudio para la elaboración del método final.

3.1. Estrategias y métodos de verbalización

En el siguiente apartado van a ser expuestas las diferentes ideas, estrategias y métodos de verbalización vistas durante la búsqueda de información: *Filtrado mediante refinamiento semántico* [31], *Natural OWL* [1], *Expressing OWL axioms by English sentences* [32] y *Dogma Modeler* [2].

3.1.1. Natural OWL

Esta herramienta permite recibir instrucciones del usuario para evitar mencionar determinada información, por ejemplo no mencionar que el vino está hecho de un solo tipo de uva. [1]

El procedimiento que sigue el programa para crear una verbalización a partir de la ontología es el siguiente:

1. En primer lugar se identifican fragmentos de información con una estructura predefinida para transformarlos en tripletas de la forma <target, property, value>.
   ClassAssertion(NamedClass target) --> < *target,instanceOf,NamedClass* >

2. Posteriormente se obtiene la verbalización de las tripletas guardadas siguiendo un patrón de verbalización asignado a cada tripleta, al resultado se le aplica lexicalización y se añaden oraciones predefinidas con objeto de mejorar la continuidad del texto.

Usando este método empleando la siguiente entrada en tripletas:

SubClassOf(*:Aryballos :Vase* ***)***
SubClassOf(*:Aryballos ObjectHasValue(:exhibitTypeCannedDescription 'An aryballos was a small spherical vase with a narrow neck, in which the athletes kept the oil they spread their bodies with' xsd:string)* ***)***
DatatypePropertyAssertion(*:periodDuration :archaicPeriod '700 BC to 480 BC' xsd:string* ***)***
DatatypePropertyAssertion(*:periodCannedDescription :archaicPeriod 'The archaic period was when the Greek ancient city-states developed' xsd:string* ***)***
DataPropertyAssertion(*:techniqueCannedDescription :blackFigureTechnique 'In the black-figure technique, the silhouettes are rendered in black on the pale surface of the clay, and details are engraved' xsd:string* ***)***

Obtenemos la siguiente verbalización:

'This is an aryballos, a kind of vase. An aryballos was a small spherical vase with a narrow neck, in which the athletes kept the oil they spread their bodies with. This aryballos was found at the Heraion of Delos and it was created during the archaic period. The archaic period was when the Greek ancient citystates developed and it spans from 700 bc to 480 bc. This aryballos was decorated with the black-figure technique. In the black-figure technique, the silhouettes are rendered in black on the pale surface of the clay, and details are engraved. This aryballos is currently in the Museum of Delos.'

Esta herramienta solo es capaz de recibir como entrada ontologías escritas en inglés y solo puede verbalizar como destino a esta misma lengua, lastimosamente el código de esta herramienta no es abierto por tanto no pudimos observar detalles de implementación, no cuenta con documentación actualizada y la guía de instalación era muy confusa y antigua por tanto no se pudo probar.

3.1.2. Dogma Modeler

Se trata de la herramienta multilenguaje capaz de verbalizar la ontología dada a lenguaje natural en la lengua que se desee siempre y cuando se haya creado la plantilla del idioma correspondiente. [2]

Hace uso de una plantilla para cada idioma, de esta forma es posible aumentar el número de lenguas posibles simplemente escribiendo la plantilla correspondiente al idioma deseado. Sin embargo la estructura de la plantilla es bastante compleja, lo que puede dificultar la tarea de modificarla o escribir una nueva. En la web pueden encontrarse algunos ejemplos de plantillas creadas por la comunidad, se han encontrado plantillas para inglés, árabe y español.

La plantilla está compuesta por reglas, cada regla tiene un identificador y contiene la información necesaria sobre cómo se desea verbalizar la información correspondiente al identificador.

```
   <Constraint xsi:type="Subset">
<Text> -[Subset] Si un(a) </Text>
<Object index="0"/>
<Role index="child"/>
<Text>un(a) </Text>
<Object index="child"/>
<Text>, entonces eso(a)  </Text>
<Object index="0"/>
<Role index="parent"/>
<Text>un(a) </Text>
<Object index="parent"/>
</Constraint>
```

Figura 3.1: Fragmento de plantilla del idioma español del método de verbalización *Dogma Modeler*.

3.1.3. Expressing OWL axioms by English sentences

Se trata de un proyecto de código abierto que tiene como objetivo verbalizar los axiomas OWL en lenguaje natural en inglés. Para ello crearon un método compuesto por varios pasos que toma decisiones dependiendo del rango de aparición de patrones.

Para poder trabajar con ontologías de gran tamaño optaron por emplear la API de java llamada *API OWL*[8], de evitan tener problemas de memoria a la hora de analizar ontologías grandes, para la detección de patrones emplearon herramientas estándar de procesamiento de texto de Unix (*grep*, *sed* y *awk*. [32]

El método está compuesto por los siguientes pasos:

1. En primer lugar, convierten toda ontología del corpus a la sintaxis funcional OWL.

2. Después generan automáticamente listas de los identificadores que contiene la ontología: clases, individuos, propiedades, etc. Programaron las herramientas de Unix para reemplazar cada aparición de un identificador por una cadena que representa su tipo. Este proceso genera un nuevo archivo en el que cada axioma de la ontología original ha sido reemplazado por una cadena que representa su estructura lógica.

3. Cuentan el número de ocurrencias de cada patrón único y los resultados se convierten en un conjunto de hechos de Prolog para su posterior análisis. En este paso tuvieron que ordenar manualmente los datos para corregir algunos casos complejos, como los literales de cadenas entre comillas que contenían cadenas entre comillas.

4. Luego calcularon la frecuencia para cada función utilizando dos medidas: (a) la cantidad de ontologías en las que la función se usó al menos una vez, y (b) la cantidad de axiomas que usan la función en general.

Function	Ontology Frequency	Percent	Axiom Frequency	Percent
SubClassOf	190	94 %	468812	74.0 %
EquivalentClasses	94	46 %	6082	1.0 %
ObjectPropertyRange	92	45 %	2275	0.4 %
ObjectPropertyDomain	91	45 %	2176	0.3 %
DisjointClasses	88	43 %	94390	14.9 %
SubObjectPropertyOf	75	37 %	2511	0.4 %
InverseObjectProperties	63	31 %	1330	0.2 %
TransitiveObjectProperty	59	29 %	221	0.0 %
FunctionalObjectProperty	56	28 %	1129	0.2 %
Total	203	100 %	633791	100 %

Tabla 3.1: Tabla que recoge los resultados de realizar una conteo sobre los axiomas detectados durante el análisis de las ontologías empleadas por el método.

5. Para abordar la estructuración de la información, observaron los patrones de argumentos para cada función de axioma, distinguiendo tres casos: (a) todos los argumentos son simples (es decir, atómicos); (b) todos los argumentos complejos (no atómicos); (c) argumentos mixtos (algunos atómicos, algunos no atómicos).

Functor	All Simple	Percent	All Complex	Mixed	Percent
SubClassOf	297293	63 %	978(0.2 %)	170541	37 %
EquivalentClasses	1222	20 %	0	4860	80 %
DisjointClasses	94390	100 %	0	0	0 %
Total	392905	69 %	978(0.2 %)	175401	31 %

Tabla 3.2: Tabla que recoge el conteo de los patrones observados en los parámetros de las funciones de axiomas más frecuentes: patrones simples, complejos y mixtos.

6. Para abordar la complejidad semántica (es decir, tamaño del axioma), contaron las frecuencias de patrones de argumentos detallados, haciendo abstracción de términos atómicos:

   – Un pequeño número de patrones cubre la mayoría de los axiomas del corpus. Por lo tanto, los cinco patrones principales cubren el 91,9

   – Todos los patrones frecuentes (es decir, los 20 principales) pueden expresarse en una sola oración sin problemas de complejidad semántica

   – En los patrones donde un argumento es simple y el otro es complejo (es decir, *subclassOf* y *EquivalentClasses*), el argumento simple siempre viene primero.

3.1.4. Filtrado mediante refinamiento semántico

El refinamiento semántico se emplea para teniendo en cuenta el significado de la información que ha sido procesada ser capaz de desechar nueva información que exprese conocimientos ya almacenados, es decir, eliminar la redundancia en el resultado de la verbalización.[31]

Eliminar la redundancia mejora la calidad de la verbalización, adicionalmente el refinamiento semántico también es útil para evitar la verbalización ambigua de expresiones lógicas.

3.2. Elección final

De entre los métodos analizado se decidió imitar la utilización de plantillas que emplea Dogma Modeler por ser una solución sencilla de abordar para lograr un verbalizador multilenguaje, lo cual era uno de nuestros objetivos. Decidimos basarnos en la forma en que se identificaban diferentes tipos de información a partir de estructuras predefinidas en la herramienta *NaturalOWL*, añadimos las estructuras dentro de la plantilla identificadas por una etiqueta que serán buscadas dentro de la ontología para extraer la información necesaria.

Se optó por investigar estrategias para la eliminación de información redundante más sencillas que el refinamiento semántico, explicadas en el Apartado 4.4 .

Se emplearon los resultados obtenidos por el proyecto de código abierto *Expressing OWL axioms by English sentences* para conocer las funciones más utilizadas dentro de las ontologías con objeto de añadir las respectivas estructuras a las plantillas, de esta forma adaptamos las plantillas extrayendo la mayor cantidad de información posible.

Adicionalmente se añadieron nuevas funcionalidades a las plantillas que mejoran la calidad de la verbalización, explicadas en el Apartado 4.5 .

Capítulo 4

Diseño e implementación del método

En este apartado se hará un apunte sobre las principales partes del método y la labor de cada una de ellas para llegar a la verbalización final de la ontología.

Enlace al repositorio github del trabajo: https://github.com/Adrian-uni/OntVerbal.git

4.1. Entorno de desarrollo

Se ha empleado *Intellij* como entorno de desarrollo para Java, las dependencias del código han sido gestionadas mediante *Gradle*, adicionalmente se ha empleado la aplicación *Spyder* del programa *Anaconda Navigator* como entorno de desarrollo para el código escrito en Python.

4.2. Diseño del sistema

La ontología que se desea verbalizar se encuentra dentro de un fichero en formato *OWL/RDF*, una vez leído el contenido del fichero se guarda la ontología en una estructura orientada al manejo de ontologías, esta es la clase OntModel de *Jena*. Antes de comenzar a extraer la información aplicamos inferencia sobre el modelo empleando el razonador *Pellet*, esta acción nos permitirá acceder a información del modelo que de otra forma no podríamos utilizar, sin embargo también creará información redundante, esta será eliminada por un método antiredundancia conforme aparezca durante el proceso de extracción de la información sobre entidades, individuos y propiedades, este método es explicado en el apartado 4.3.

Una vez ha sido extraída la información es necesario corregir errores gramaticales, esto es mediante el uso de expresiones regulares que detectan el error y lo reemplazan por la cadena correspondiente como se explica en la parte final del apartado 4.5 .

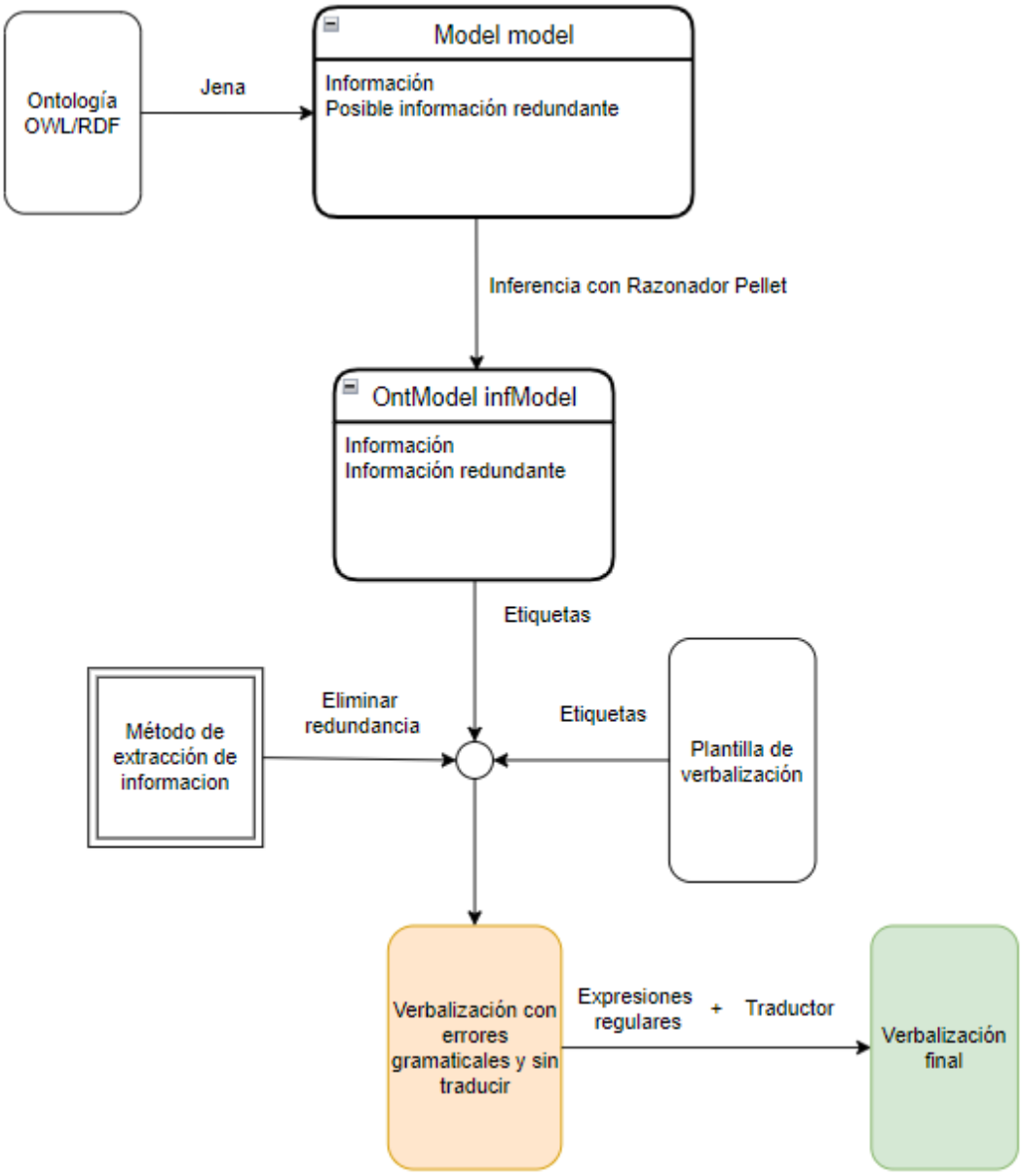

Figura 4.1: Diagrama de diseño del sistema implementado.

4.3. Extracción de la información

Los ficheros de ontologías se caracterizan por tener una sintaxis compuesta por una jerarquía de nodos, lo que facilita la extracción de información al recorrer estos ficheros de arriba a abajo. Durante la creación del método fue necesario atribuir un identificador a cada tipo de información (*SubClassOf*, *Disjoint*, *ClassAssertion*,...) para poder asignarle el contexto de verbalización que le corresponde, esto fue mediante el uso de etiquetas que generalmente coincide con el nombre del nodo dentro del fichero

del que extraen información.

Con objeto de guardar por separado la información de cada elemento se ha empleado una variable de tipo *java.io.Map* para cada elemento, esta es una estructura de datos dedicada a almacenar pares çlave/valor"donde en la clave se corresponde a las etiquetas nombradas anteriormente, mientras que el valor es la información del elemento extraída que corresponde a esa etiqueta.

El método recorre el fichero en busca de etiquetas que conozca, una vez encuentra una etiqueta conocida determina el elemento al que hace referencia la información contenida en la etiqueta y guarda esa información en la estructura *Map* del elemento después de haberse comprobado que no es redundante.

El método emplea dos métodos para extraer la información:

- Extracción empleando Jena: este método consiste en extraer información mediante consultas al modelo empleando la *API Jena*, estas consultas se hacen mediante funciones predefinidas que devuelven una lista de tripletas de las que se extrae la información que finalmente se verbaliza.

- Extracción recorriendo el fichero *XML*: este método a diferencia del anterior es automático, únicamente haciendo uso de la plantilla es capaz de identificar las etiquetas en el fichero *XML*, extraer su información y asignarle el contexto de verbalización que le corresponde. Mediante este método se extrae la mayor parte de la información que finalmente se verbaliza.

Existe información que debe ser verbalizada que no suele encontrarse en el fichero *XML* y por tanto empleando el método automático no es posible verbalizarla, por ejemplo las propiedades inversas. En una ontología en formato *XML* suele existir la información de una propiedad pero existen casos en que se obvia la información referente a su inversa, es decir, si existe la propiedad *purchasedPizza* que relaciona al cliente *Customer1* y a la pizza *AmericanaPizza3*:

Customer1 purchasedPizza AmericanaPizza3

Es posible que en el fichero *XML* no exista la información de su propiedad inversa (purchasedByCustomer):

AmericanaPizza3 purchasedByCustomer Customer1

Esto ocurre aun después de aplicar inferencia sobre el modelo, por ello cada vez

que se guarda la información de una propiedad se comprueba si existe una propiedad inversa a ella y se añade la información invirtiendo los roles de los objetos. Esto causaría redundancia en caso de que la información sobre la inversa si se encontrara en el modelo, por ello es de vital importancia el método para eliminar redundancias expuesto en el apartado 4.4 .

4.4. Eliminación de la redundancia

La redundancia se da cuando existe dentro de un texto información repetida de forma innecesaria, es decir no realiza ninguna aportación al objetivo del texto.

Emplear como entrada a un modelo de lenguaje un texto en lenguaje natural que tenga gran cantidad de redundancia es perjudicial para el modelo ya que dificulta la labor del tokenizador además de ocupar espacio de almacenamiento sin ninguna necesidad, por ello se estudiaron varios métodos para evitar la redundancia.

La fuente principal de redundancia en la verbalización se origina en que al emplear dos métodos de extracción de información independientes existirán casos en que la información extraída se repita, para acabar con la redundancia se tuvieron en cuenta las siguientes opciones:

- Distancia de Levenshtein: es una métrica que indica la distancia entre dos cadenas, corresponde al número mínimo de ediciones de un solo carácter (inserciones, eliminaciones o sustituciones) necesarias para pasar de la primera cadena a la segunda. Este método mostró un comportamiento aceptable, sin embargo ya que existen contextos de información muy similares entre si esto derivaba en la eliminación de información no redundante. [33]

- Cuando se quiere añadir nueva información para ser verbalizada se envía a este método, la nueva información está compuesta por palabras irrelevantes pertenecientes a su contexto de verbalización y por palabras relevantes rodeadas por corchetes ([ ]), estas palabras corresponden a con algún identificador de clase, individuo o propiedad. Para comprobar que la nueva información no es redundante para el elemento se comprueba si existe alguna clave (etiqueta) dentro de la estructura *Map* del elemento que contenga a la vez todas las palabras relevantes de la nueva información, si se da este caso entonces la información se clasifica como redundante y no se añade al *Map* del elemento.

Finalmente se decidió emplear el segundo método por haber mostrado un buen comportamiento al evitar la redundancia en la verbalización conservando la información relevante intacta.

4.5. Plantillas de verbalización

Las plantillas de verbalización son ficheros de texto compuestos por estructuras identificadas por etiquetas que contienen la información relativa a como extraer la información de su respectiva etiqueta y el contexto de verbalización correspondiente a ella, adicionalmente pueden guardar información como la lengua de la plantilla.

Se han desarrollado tres versiones de plantilla diferentes para cada idioma, la versión *base* tiene como objetivo ser fácil de entender por personas sin conocimientos sobre *RDF* u ontologías, la versión *formal* que está orientada a personas con conocimientos avanzados sobre ontologías y la versión *simple*, lo más sencilla posible que únicamente contiene las etiquetas identificadoras de los elementos de la ontología.

En la plantilla existen dos tipos de estructuras:

- Estructuras destinadas a la verbalización de información extraída usando Jena: este tipo de estructura solamente guarda el contexto de verbalización de la etiqueta que consiste en una máximo de tres frases, la primera para iniciar la verbalización *(Text)*, la segunda es empleada cuando existen varios fragmentos de información identificados por la misma etiqueta *(Loop)* y la última *(Finish)* se usa para hacer un apunte explicativo sobre la información mostrada en las dos anteriores frases (ver Figura 4.2).

```
<Restriccion id="Disjoint">
    <Text>A_An T[?name] can be a_an T[?0]</Text>
    <Loop> or a_an T[?0]</Loop>
    <Finish> but a_an T[?name] can not be more
            than one of these at a time.
    </Finish>
</Restriccion>
```

Figura 4.2: Fragmento de plantilla de verbalización versión *base* de la lengua inglesa correspondiente a la etiqueta *Disjoint*.

- Estructuras destinadas a la verbalización de información obtenida recorriendo el fichero *XML*: esta estructura al igual que la anterior contiene el contexto de verbalización de la etiqueta, adicionalmente contiene el método de extracción que debe seguir el programa principal para extraer la información correspondiente a la etiqueta (Ver Figura 4.3).

En la plantilla se rodean con corchetes las palabras correspondientes a identificadores de elementos marcándolos, de esta forma se les atribuye importancia

para indicar que deben ser traducidas (T[...]) o que se requiere identificar su género (G[...]) para elegir correctamente el artículo situado delante de la palabra.

```
<Restriccion id="SubClassOf">
    <Text>Un_Una G[?name] </Text>
    <Loop> es un tipo de T[?name]</Loop>
    <Finish> comparte sus propiedades.</Finish>
    <Extraccion>
        <Class>[?name]</Class>
    </Extraccion>
</Restriccion>
```

Figura 4.3: Fragmento de plantilla de verbalización versión *base* de la lengua española correspondiente a la etiqueta *SubClassOf*.

Cada plantilla adicionalmente puede incluir expresiones regulares que se emplean como mecanismo de corrección gramatical para la verbalización, para ello se especifica una expresión regular y seguidamente la cadena por la que se desea reemplazar esa expresión regular.

```
<Regular>
    <Antes>a_an '(?=[aeiou]);</Antes>
    <Despues>an ';</Despues>
</Regular>
```

Figura 4.4: Estructura perteneciente a una plantilla cuyo objetivo es arreglar errores gramaticales en lengua inglés, especifica la expresión que quiere reemplazarse (Antes) y la final (Después), en particular esta norma comprueba si la palabra inmediatamente después de un articulo empieza por vocal o no, en el primer caso el artículo será *an* y en el caso contrario *a*.

A modo de ejemplo se va a mostrar el proceso que se sigue para obtener desde el fichero que contiene una ontología un fragmento de ella verbalizado. Partimos del siguiente fragmento de ontología perteneciente al corpus sobre conferencias:

```
<SubClassOf>
    <Class IRI="#Paper"/>
    <ObjectMinCardinality cardinality="1">
        <ObjectProperty IRI="#readByReviewer"/>
    </ObjectMinCardinality>
</SubClassOf>
```

Las etiquetas que serán reconocidas son: *SubClassOf, Class, ObjectMinCardinality y ObjectProperty* en este orden. Las estructuras necesarias contenidas en la plantilla serán las identificadas por esas mismas etiquetas:

```
<Restriccion id="SubClassOf">
        <Text>A_An T[?name] </Text>
        <Loop> is a type of T[?name]</Loop>
        <Extraccion>
            <Class>[?name]</Class>
        </Extraccion>
</Restriccion>


<Restriccion id="ObjectMinCardinality">
    <Text> has T[?property] at least T[?content]
        T[?class]
    </Text>
    <Extraccion>
        <Class>[?class]</Class>
        <ObjectProperty>[?property]</ObjectProperty>
        <Content>[?content]</Content>
    </Extraccion>
</Restriccion>
```

En la plantilla de verbalización solamente se activa una estructura en caso de que el nodo al que hace referencia tenga algún nodo hijo, en este ejemplo únicamente los nodos *SubClassOf* y *ObjectMinCardinality* tienen nodos hijos, de los nodos que no tienen hijos solamente se extrae la información necesaria, en el caso del nodo *Class* extraemos su atributo *IRI* y para el nodo *ObjectProperty* también extraemos el valor de su atributo *IRI*.

El contexto de verbalización del fragmento sería el siguiente:

A_An T[?name] has T[?property] at least T[?content]

El método empleando la información dentro del nodo *Extraccion* de la *Restricion SubClassOf* sabe que debe sustituir *[?name]* por el contenido del atributo del nodo *Class*, en este caso es *Paper*, esta sustitución la realiza antes de entrar a explorar el nodo *ObjectMinCardinality*. Por último gracias a la información dentro del nodo *Extraccion* de la *Restriccion ObjectMinCardinality* el método sabe que debe sustituir *[?property]* por *readByReviewer* y [?content] por 1.

La verbalización actual seria la siguiente:

*A_An T[Paper] has T[readByReviewer] at least T[1]*c

Como se comentó anteriormente las expresiones *T[...]* indican que el contenido entre corchetes puede ser traducido y se empleará la siguiente expresión regular contenida en la plantilla para arreglar el error gramatical *A_An*:

```
<Regular>
    <Antes>A_An '(?=[^aeiou]);</Antes>
    <Despues>A ';</Despues>
</Regular>
```

Finalmente obtenemos como resultado final la siguiente verbalización del fragmento:

A Paper has read by reviewer at least 1

Capítulo 5

Evaluación del método y pruebas con modelos de lenguaje

Si bien el objetivo último es el uso de información ontológica como entrada a modelos de lenguaje, la verbalización también resulta útil para describir el conocimiento ontológico encerrado en una ontología de manera más accesible a humanos. Por ello empezaremos preguntándonos si la verbalización producida es de calidad (evaluación intrínseca) mediante la realización de cuestionario y en una segunda fase queremos averiguar si tiene un efecto beneficioso para alguna tarea concreta, en nuestro caso para el descubrimiento de equivalencias entre ontologías (evaluación extrínseca).

5.1. Encuestas

Se ha encuestado a seis personas integrantes del grupo *SID* (Sistemas de Información Distribuidos) con amplia formación y experiencia empleando ontologías. Una vez seleccionadas varias personas con conocimientos suficientes relacionados con ontologías se les han presentado fragmentos de una ontología junto a la verbalización de estos con objeto de que valoren cuan expresivas y coherentes son las verbalización.

Los fragmentos de la ontología mostrados a los encuestados se encontrarán en formato Turtle (tripletas) por ser un formato de representación de ontologías orientado a humanos, es decir, la sintaxis utilizada por el formato Turtle tiene como objetivo expresar el contenido semántico de la ontología de forma que un ser humano pueda comprenderlo fácilmente.

Han sido desarrollados cuatro cuestionarios compuestos por siete preguntas, todos ellos contienen las mismas preguntas en el mismo orden, se diferencian en la plantilla empleada para obtener las verbalizaciones, se han creado cuestionarios para las lenguas español e inglés y para las plantillas base y formal, cada pregunta puede contener varias verbalizaciones de elementos de la ontología, los cuestionarios desarrollados contienen 25 verbalizaciones en total cada uno.

Las verbalizaciones por valorar han sido divididas en:

- Verbalizaciones de clases: las valoraciones correspondientes a clases son *V1, V2, V3, V4, V5, V6, V13, V19, V20, V21, V22 y V24*.
- Verbalizaciones de individuos: las valoraciones correspondientes a individuos son *V10, V11, V14, V15, V16, V23 y V25*
- Verbalizaciones de propiedades: las valoraciones correspondientes a propiedades son *V7, V8, V9, V12, V17 y V18*

Los encuestados pueden realizar un comentario tras cada pregunta, de esta forma es posible identificar las partes de la verbalización peor valoradas, las razones detrás de las críticas recibidas y la forma de mejorar esas partes.

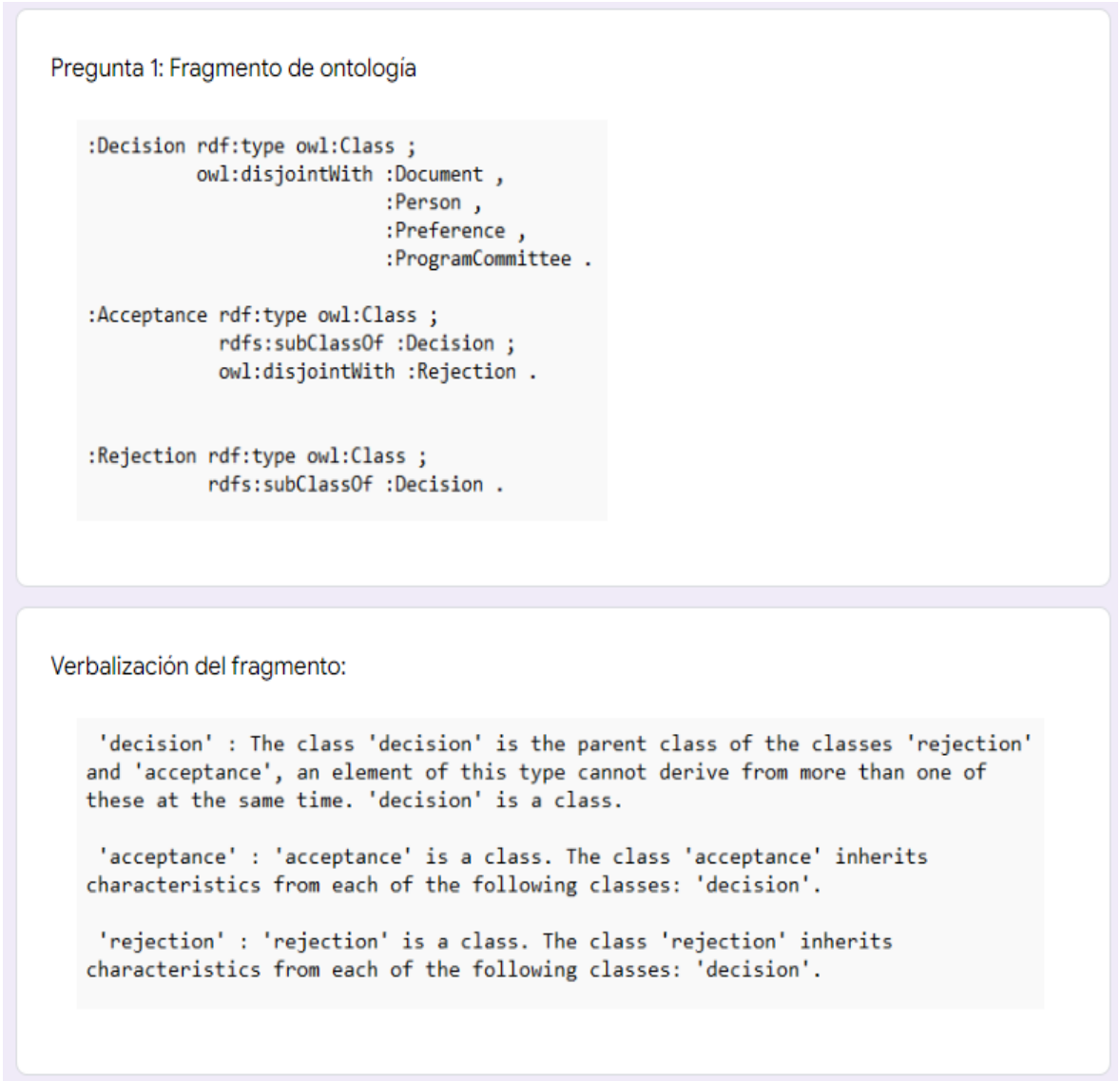

Figura 5.1: Fragmento de cuestionario correspondiente a la pregunta.

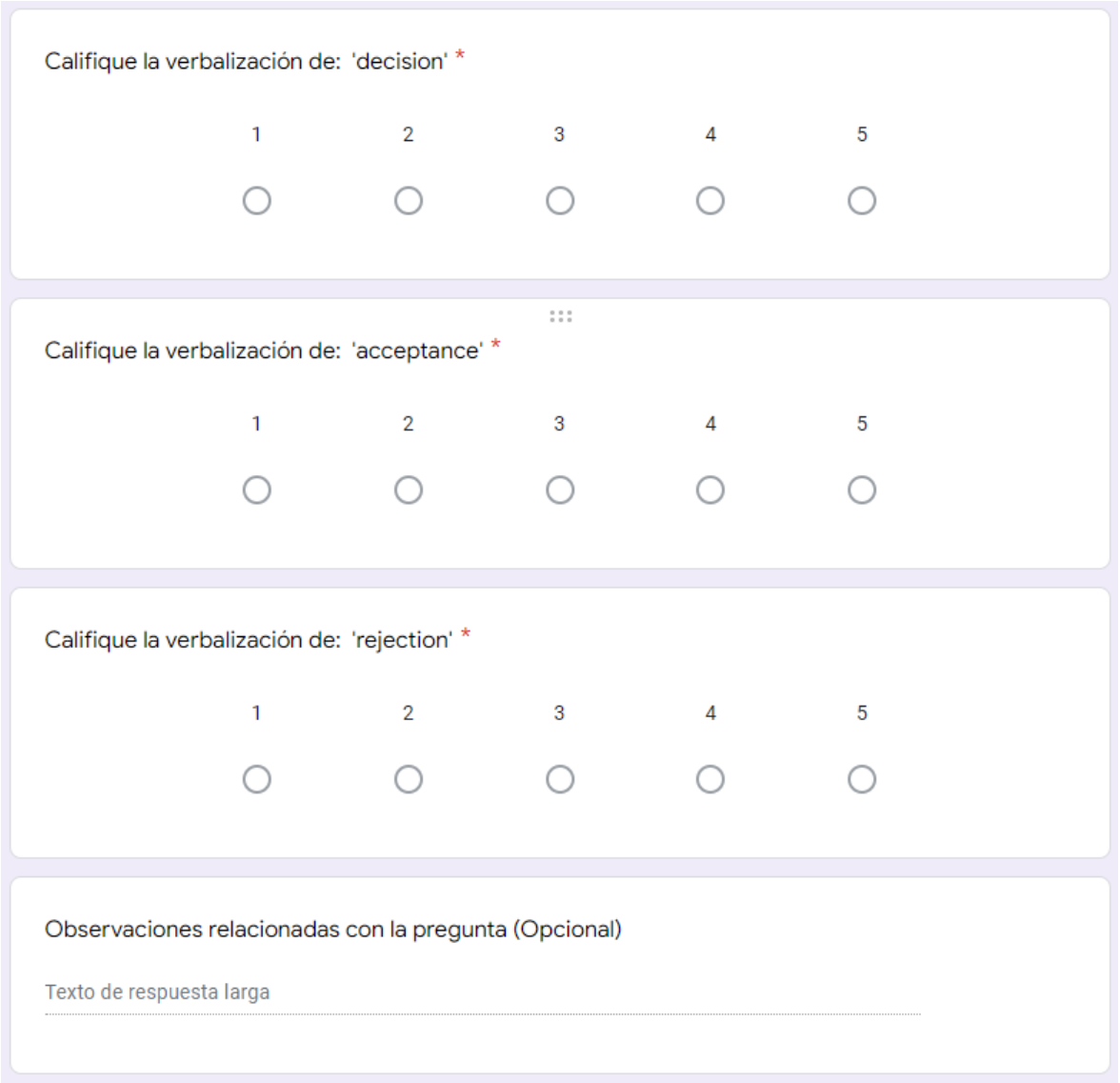

Figura 5.2: Fragmento de cuestionario correspondiente a las respuestas.

5.1.1. Valoraciones de los encuestados

Los encuestados pueden valorar cada verbalización del cuestionario por separado, asignando un valor entre uno y cinco, donde valorar con uno significa que la verbalización del elemento no expresa adecuadamente en lenguaje natural el elemento verbalizado y cinco significa que la verbalización ha cumplido perfectamente con su objetivo.

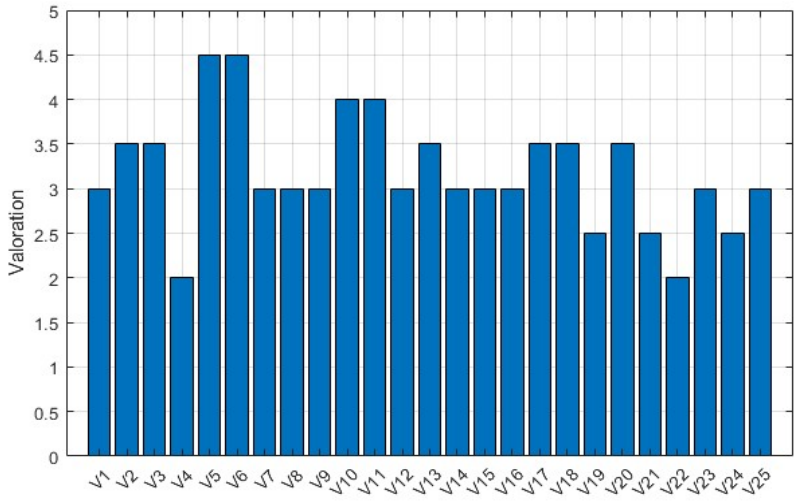

Figura 5.3: Resultados del cuestionario sobre verbalizaciones base en lengua española.

Se ha obtenido para la verbalización en español no formal una puntuación media para las clases de **3.12** sobre 5, una media para los individuos de **3.28** sobre 5 y una media para las de las propiedades de **3.16** sobre 5.

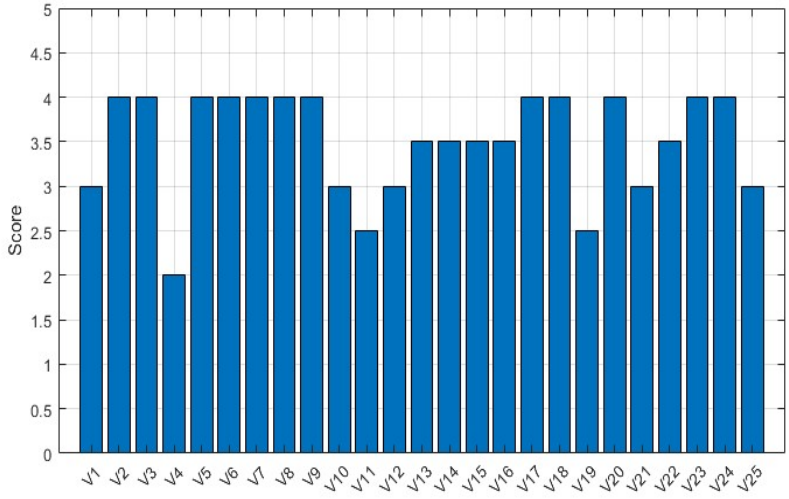

Figura 5.4: Resultados del cuestionario sobre verbalizaciones formales en lengua española.

Por otro lado, para la verbalización en español formal una puntuación media para las clases de **3.45** sobre 5, una media para los individuos de **3.28** sobre 5 y una media para las propiedades de **3.83** sobre 5.

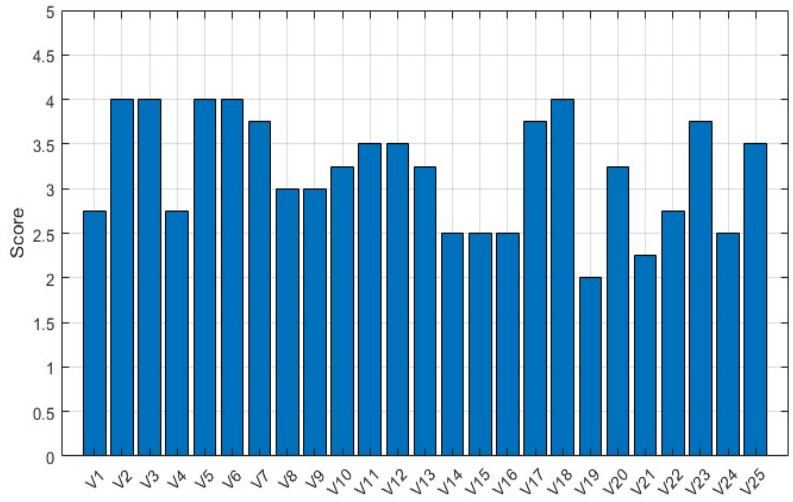

Figura 5.5: Resultados del cuestionario sobre verbalizaciones formales en lengua inglesa.

Por otro lado, para la verbalización en ingles formal una puntuación media para las clases de **3.12** sobre 5, una media para los individuos de **3.07** sobre 5 y una media para las propiedades de **3.5** sobre 5.

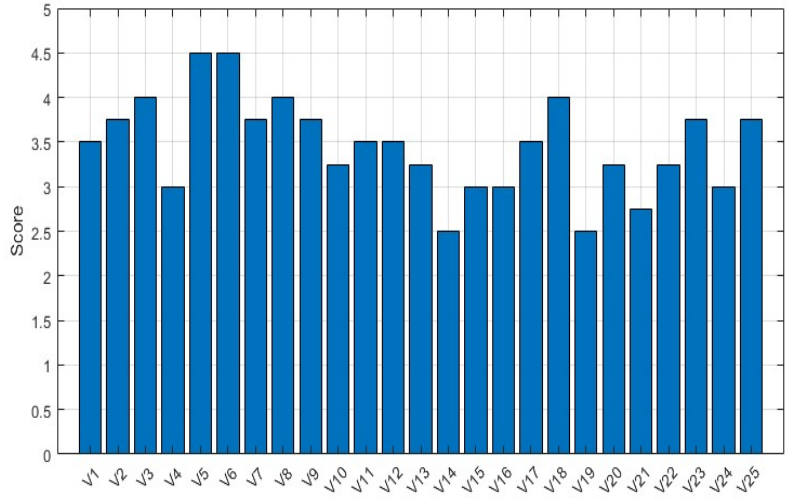

Figura 5.6: Resultados del cuestionario sobre verbalizaciones base en lengua inglesa.

Por otro lado, para la verbalización en ingles no formal una puntuación media para las clases de **3.43** sobre 5, una media para los individuos de **3.2** sobre 5 y una media para las propiedades de **3.7** sobre 5.

La verbalización de individuos ha sido la más criticada, las principales razones son el orden en que se muestra la información y presentación de la información mejorable.

No existe diferencia destacable en lo referente a la valoración media de las plantillas base y formal, tampoco respecto al idioma del cuestionario.

Los resultados obtenidos son en general positivos pero mejorables. La mayoría de comentarios recibidos coinciden en que las expresiones empleadas para ambos idiomas son muy mejorables, varios han comentado que sería buena idea agregar expresiones prediseñadas para los casos en que existe mucha información seguida separada solo por *y* o por *,* se tendrán en cuenta sus comentarios en el apartado de futuras mejoras.

5.2. Pruebas con Modelos de lenguaje

Se ha empleado un conjunto de datos sobre conferencias compuesto por siete ontologías distintas [4]. También ha sido empleado un conjunto de datos de 21 ficheros de texto [34] que contienen información sobre equivalencias entre conceptos contenidos en las diferentes ontologías contenidas en el conjunto de datos de conferencias, las ontologías disponibles son: *cmt.owl, conference.owl, confof.owl, edas.owl, ekaw.owl, iasted.owl* y *sigkdd.owl*.

Para entrenar un modelo del lenguaje además de necesitar el conjunto de datos de verbalizaciones [35] también es necesaria una tripleta de ficheros *CSV* que contengan información sobre la relación de equivalencias sobre parejas de conceptos pertenecientes a las ontologías, un fichero se empleará para entrenar el modelo, otro para validación durante el entrenamiento y el último para ser usado como datos de test. La estructura de estos ficheros es la siguiente:

```
source,target,rel
conference,conference volume,1
negative review,program committee member,0
preference,review preference,1
pending paper,submit,0
author,regular author,1
session chair,personal history,0
email,has an email,1
research institute,has authors,0
```

Disponemos de cinco tripleta de ficheros *CSV* (*train, val, test*) diferentes que pueden ser empleadas en el segundo entrenamiento, estos ficheros se han creado de forma que la mitad de parejas de conceptos son equivalentes y la otra mitad no lo son, además de que para cada tripleta al menos una de las ontologías del conjunto de datos no se encuentra en ninguno de los ficheros de entrenamiento *(train)* o valoración *(val)* pero si en el de test *(test)*, en cada tripleta la elección de esta ontología cambia. Puesto

que los 21 ficheros de equivalencia originales no tienen el mismo número de filas, la cantidad de elementos que se dedican a entrenamiento, validación y test variará en cada tripleta.

- Tripleta 1: la ontología elegida para los datos de test ha sido *edas.owl*, la tripleta cuenta con **140** filas de datos de entrenamiento, **32** filas de datos de validación y **92** filas de datos de test.
- Tripleta 2: la ontología elegida para los datos de test ha sido *ekaw.owl*, la tripleta cuenta con **124** filas de datos de entrenamiento, **58** filas de datos de validación y **82** filas de datos de test.
- Tripleta 3: la ontología elegida para los datos de test ha sido *cmt.owl*, la tripleta cuenta con **154** filas de datos de entrenamiento, **50** filas de datos de validación y **60** filas de datos de test.
- Tripleta 4: la ontología elegida para los datos de test ha sido *conference.owl*, la tripleta cuenta con **126** filas de datos de entrenamiento, **48** filas de datos de validación y **90** filas de datos de test.
- Tripleta 5: la ontología elegida para los datos de test ha sido *sigkdd.owl*, la tripleta cuenta con **154** filas de datos de entrenamiento, **66** filas de datos de validación y **44** filas de datos de test.

Las medidas a utilizar para valorar los modelos son la precisión, el recall y el F1-score:

$$\text{Precisión} = \frac{\text{Número de verdaderos equivalentes clasificados por el modelo}}{\text{Número de equivalentes clasificados por el modelo}} \tag{5.1}$$

$$Recall = \frac{\text{Número de equivalentes clasificados por el modelo}}{\text{Número real de equivalentes}} \tag{5.2}$$

$$F1 - score = 2 * \frac{\text{Precisión * Recall}}{\text{Precisión + Recall}} \tag{5.3}$$

En particular el F1-score se define como la media armónica entre la precisión y el recall, es útil ya que combina las medidas de precisión y recall en un solo valor, lo que permite hacer una comparación entre modelos muy cómoda.

Dado que la cantidad de datos disponibles para entrenar y testear el modelo es muy limitada se ha optado por realizar diez iteraciones de cada prueba pero cambiando los datos empleados para entrenamiento, validación y test, del conjunto de resultados correspondiente a cada prueba calcularemos la media de la *precision, recall* y *F1-score* para hacer la comparación con el resto de modelos. Se han tenido en cuenta cuatro variables durante la creación de las pruebas:

- Número de épocas a ejecutar durante el primer entrenamiento correspondiente al que emplea las verbalizaciones, en particular vamos ha usar de 3 a 5 épocas.
- Tipo de verbalización a utilizar durante el primer entrenamiento, contamos con tres verbalizaciones distintas: base, formal y simple.
- Tripleta de ficheros CSV (train, val, test) a utilizar durante el segundo entrenamiento correspondiente al que emplea los CSV, contamos con 5 tripletas distintas.
- Modelos a utilizar: vamos a hacer uso de cuatro modelos diferentes: *bert-base-uncased*, *bert-large-uncased-whole-word-masking*, *roberta-base* y *roberta-large*.

En primer lugar nos interesa conocer el impacto de utilizar los tipos de plantillas disponibles en las medidas de evaluación.

Results				
Model	Type of verbalization	Precision	Recall	F1
bert-base-uncased	X	0.61863	0.74782	0.66278
bert-base-uncased	BASE	0.88314	0.65652	0.74214
bert-base-uncased	FORMAL	0.87481	0.66739	0.73826
bert-base-uncased	SIMPLE	0.77516	0.73260	0.74639
roberta-base	X	0.75673	0.77391	0.75124
roberta-base	BASE	0.74487	0.80652	0.75518
roberta-base	FORMAL	0.79220	0.73043	0.72427
roberta-base	SIMPLE	0.75520	0.75434	0.70547

Tabla 5.1: Tabla que recoge los resultados de emplear o no emplear los diferentes tipos de verbalizaciones en el primer entrenamiento durante el proceso de entrenamiento del modelo.

La tabla 5.1 muestra una clara mejoría en los modelos BERT al emplear verbalizaciones durante el primer entrenamiento, inclusive haciendo uso de la verbalización más simple se ha visto una mejoría de ocho puntos en la medida

F1-score respecto a las previsiones del modelo sin verbalizaciones. El modelo Roberta ha obtenido resultados iguales o peores a no utilizar ninguna verbalización, interesa realizar más pruebas para comprobar la consistencia de estos resultados.

Results				
Model	Type of verbalization	Precision	Recall	F1
bert-large-uncased-whole-word-masking	X	0.77998	0.82608	0.78740
bert-large-uncased-whole-word-masking	BASE	0.84359	0.85652	0.82935
bert-large-uncased-whole-word-masking	FORMAL	0.88340	0.82173	0.83240
bert-large-uncased-whole-word-masking	SIMPLE	0.67323	0.76956	0.69707
roberta-large	X	0.88659	0.74565	0.78992
roberta-large	BASE	0.60709	0.88695	0.70400
roberta-large	FORMAL	0.71341	0.75652	0.72109
roberta-large	SIMPLE	0.64286	0.85434	0.71179

Tabla 5.2: Tabla que recoge los resultados de emplear los diferentes tipos de verbalizaciones para entrenar los modelos *large* de *BERT* y *Roberta*, el primer entrenamiento ha sido de tres épocas y se ha empleado la primera tripleta CSV de entre las cinco disponibles.

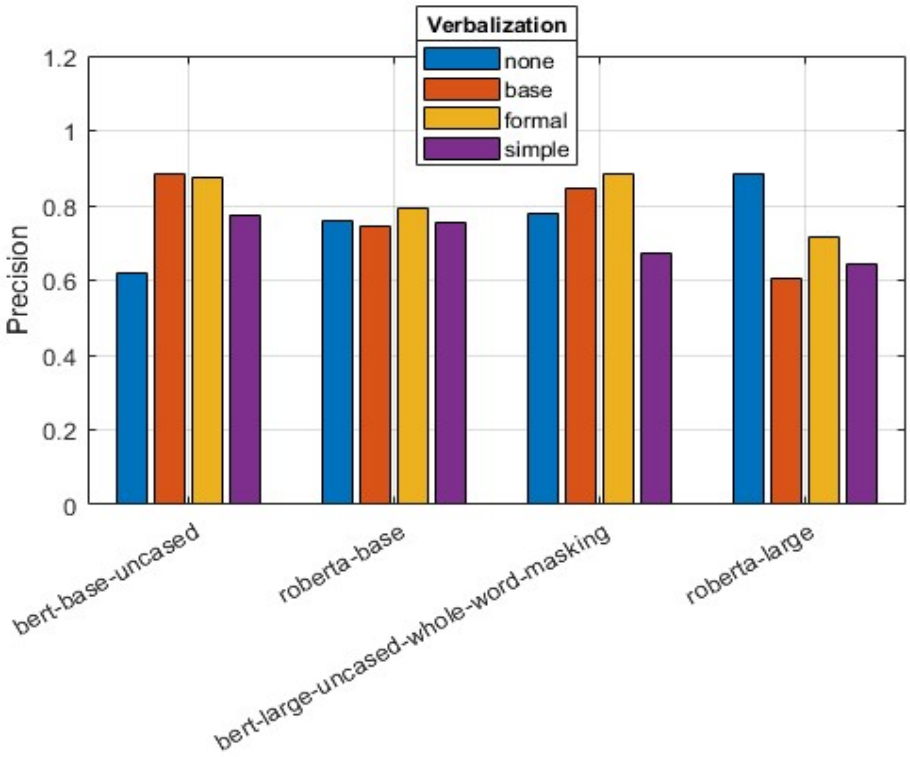

Figura 5.7: Gráfica auxiliar a la tabla 5.2. Resultados de precisión para los cuatro modelos empleando los cuatro tipos de verbalizaciones disponibles.

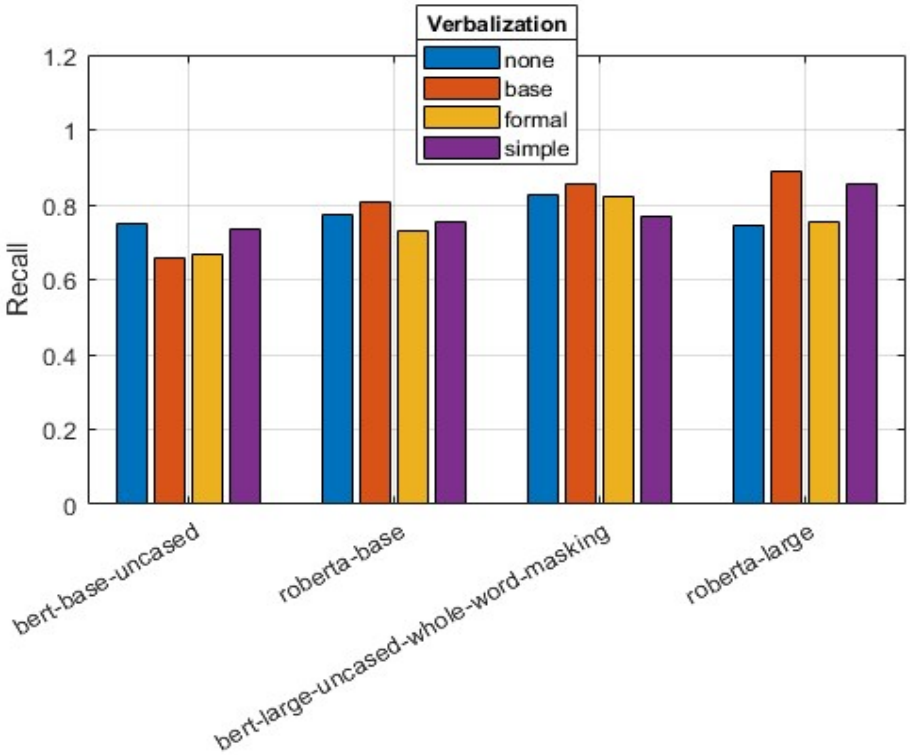

Figura 5.8: Gráfica auxiliar a la tabla 5.2. Resultados de recall para los cuatro modelos empleando los cuatro tipos de verbalizaciones.

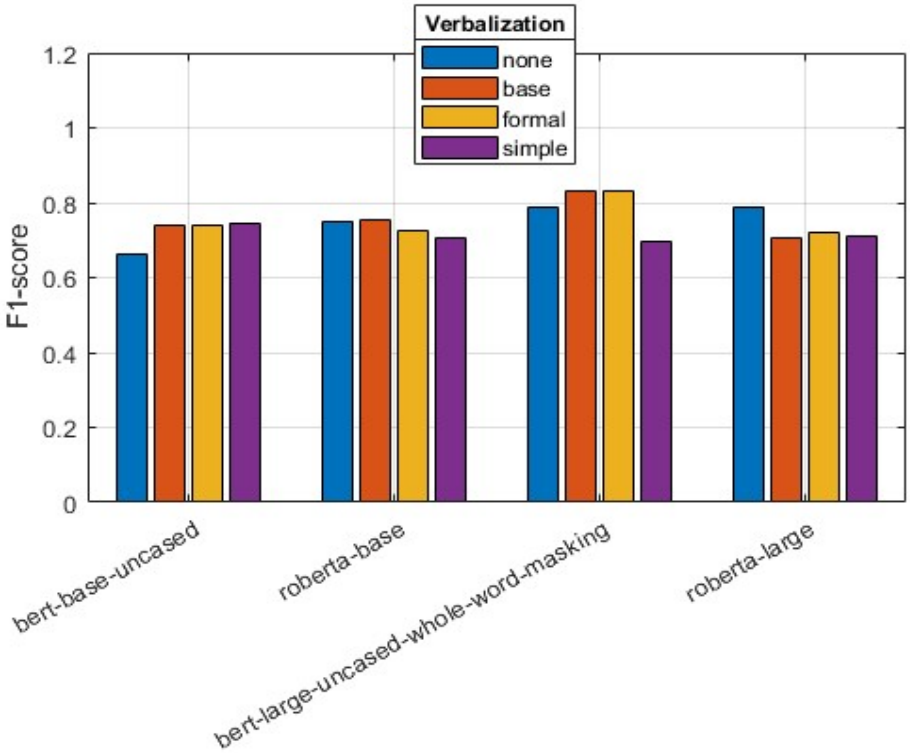

Figura 5.9: Gráfica auxiliar a la tabla 5.2. Resultados de la medida F1-score para los cuatro modelos empleando los cuatro tipos de verbalizaciones.

La tabla 5.2 junto a las gráficas auxiliares 5.7, 5.8 y 5.9, dan a entender que los modelos BERT son los que mejor reaccionan a ser entrenados por verbalizaciones, se puede apreciar que la única plantilla que se mantiene constante para cualquier modelo es la simple, esta plantilla presenta unos resultados de recall destacables, sin embargo cuenta con unas medidas de precisión muy bajas.

Hasta el momento hemos empleado la misma tripleta CSV para obtener los datos usados en entrenamiento, validación y test, por tanto para comprobar que los resultados hasta el momento son consistentes vamos a emplear el resto de tripletas en busca de alguna variación en los resultados.

Results				
Model	CSV Triplet	Precision	Recall	F1
bert-base-uncased	1	0.87481	0.66739	0.73826
roberta-base	1	0.79220	0.73043	0.72427
bert-base-uncased	2	0.71588	0.90243	0.79690
roberta-base	2	0.70685	0.64390	0.63881
bert-base-uncased	3	0.70331	0.77	0.73164
roberta-base	3	0.74436	0.67333	0.69846
bert-base-uncased	4	0.72593	0.88444	0.79498
roberta-base	4	0.66430	0.69555	0.62878
bert-base-uncased	5	0.76722	0.88636	0.81917
roberta-base	5	0.86147	0.73181	0.75932

Tabla 5.3: Tabla que recoge los resultados de emplear las cinco tripletas CSV disponibles en el segundo entrenamiento para los modelos base de *BERT* y *Roberta*, en este caso se han utilizado las verbalizaciones formales y el primer entrenamiento consta de cuatro épocas.

En la tabla 5.3 se observa una variación considerable en los resultados obtenidos para las cinco tripletas *CSV*, esto pueden indicar que con únicamente 3 épocas el modelo no ha tenido suficiente tiempo para converger y entrenar sus parámetros correctamente, también puede estar relacionado con la distinta distribución que tienen las tripletas de filas dedicadas a entrenamiento, validación y test.

Nos interesa saber cómo reaccionan los modelos a modificar el número de épocas que se ejecutan durante el primer entrenamiento, ya que al contar con mayor número de épocas es posible que el modelo sea capaz de asignar mejores pesos a sus parámetros, la diferencia en el número de datos empleados para entrenamiento, validación y test en las tripletas también puede estar afectando a los resultados.

Results				
Model	Number of epochs	Precision	Recall	F1
bert-base-uncased	3	0.88314	0.65652	0.74214
roberta-base	3	0.74487	0.80652	0.75518
bert-base-uncased	4	0.87280	0.63478	0.72249
roberta-base	4	0.78827	0.71739	0.69896
bert-base-uncased	5	0.86359	0.61521	0.70286
roberta-base	5	0.72866	0.75434	0.68354

Tabla 5.4: Tabla que recoge los resultados de variar el número de épocas a realizar durante el primer entrenamiento correspondiente a las verbalizaciones para los modelos base de *BERT* y *Roberta*, se ha empleado la verbalización base y la primera tripleta CSV de entre las cinco disponibles.

Los resultados reflejados en la tabla 5.4 parecen mostrar que aumentar el número de épocas del primer entrenamiento no garantiza obtener mejores resultados, ambos modelos han empeorado sus resultados con cada época añadida al primer entrenamiento, es posible que la causa radique en las pocas filas de validación disponibles en la tripleta *CSV* empleada.

Igualmente nos interesa saber si alguno de los tipos de verbalización presenta comportamientos extraños en caso de aumentar el número de épocas durante el primer entrenamiento.

Results					
Model	Type of verbalization	Number of epochs	Precision	Recall	F1
bert-base-uncased	BASE	3	0.76594	0.89090	0.81675
	FORMAL	3	0.76722	0.88636	0.81917
	SIMPLE	3	0.74208	0.89090	0.80006
	BASE	4	0.73600	0.90909	0.80735
	FORMAL	4	0.75771	0.89545	0.81574
	SIMPLE	4	0.73927	0.90909	0.80977
	BASE	5	0.75913	0.88181	0.81182
	FORMAL	5	0.75914	0.91818	0.82429
	SIMPLE	5	0.74850	0.86818	0.80140

Tabla 5.5: Tabla que recoge los resultados de modificar el número de épocas para cada una de las tres verbalizaciones disponibles durante el primer entrenamiento para el modelo base de BERT, se ha empleado la quinta de las cinco tripletas CSV disponibles.

Los resultados obtenidos no muestran variaciones considerables en ninguna medida que dependan del número de épocas o de la verbalización empleada, los resultados obtenidos son mejores a los obtenidos en la prueba anterior debido a que la tripleta CSV empleada en esta prueba parece beneficiar al recall habiendo disparado el valor

de esta medida en todas las subpruebas. La tripleta 1 es la tripleta de entre todas las disponibles con más filas dedicadas a test, tiene más del doble de filas dedicadas a test que la tripleta 5.

5.2.1. Discusión

El conjunto de datos de resultados se encuentra dentro del repositorio github del trabajo. [36]

Es evidente el impacto que tiene la tripleta *CSV* empleada durante el segundo entrenamiento, como se puede apreciar en las tablas 5.3 y 5.5, la principal razón de tener tanta relevancia sobre los resultados es debido a que el conjunto de datos disponibles es muy reducido. Modificar el número de filas dedicadas a entrenamiento, validación y test causa cambios notables en los resultados.

BERT tanto es su modelo base como en su modelo *large* ha demostrado mejorar considerablemente sus previsiones de equivalencia al emplear verbalizaciones durante el primer entrenamiento como puede observarse en las tablas 5.1 y 5.2.

En cuanto a las pruebas empleando Roberta, en su modelo base no se ha notado mucha variación, sin embargo su modelo *large* ha sufrido un aumento considerable en el recall del modelo al emplear verbalizaciones durante el primer entrenamiento, sin embrago la precisión del modelo ha descendido bastante, una media de 23 puntos (23 %) como se puede apreciar en las tablas 5.1 y 5.2.

Se creó la plantilla simple con objeto de disponer de una base sobre la que partir esperando que no fuera muy útil para los modelos del lenguaje, sin embrago ha logrado resultados muy superiores a los esperados, se cree que es debido a que las verbalizaciones generadas por esta plantilla ocupan menos tokens, luego el modelo requiere menos esfuerzo para entrenarse.

Capítulo 6

Conclusiones y trabajo futuro

En este trabajo se ha comprobado la eficacia de emplear un conjunto de datos de ontologías verbalizadas como entrada a un modelo del lenguaje con objeto de identificar relaciones de equivalencia entre conceptos contenidos dentro de dichas ontologías. Se ha implementado con éxito un método capaz de extraer el contenido explicito en ontologías obteniendo una descripción escrita en lenguaje ingles natural y más variedad de idiomas como el español, francés y portugués.

Las pruebas referentes a la evaluación por parte de expertos han resultado en una valoración positiva, aunque mejorable en algunos aspectos, sobre todo menciones a las expresiones empleadas y el orden en que se muestran los fragmentos de información.

Las pruebas con modelos del lenguaje han reflejado la utilidad del método desarrollado, logrando un aumento en la calidad de las previsiones sobre equivalencias entre conceptos de distintas ontologías en la mayoría de modelos empleados. La pruebas realizadas con BERT han concluido en un aumento medio del 8.361 % de *F1-score* en modelos BERT base y de un 4.5 % de *F1-score* en modelos BERT *large*.

En conjunto, los resultados obtenidos muestran que a pesar de contar con un conjunto de datos mucho más reducido que el deseado, se ha logrado el objetivo de mejorar las previsiones realizadas por los modelos del lenguaje.

Las principales limitaciones encontradas han sido la reducida cantidad de información disponible. Contando con un único repositorio de ontologías muy reducido y escrito en su totalidad en inglés, volviendo innecesaria la funcionalidad multilingüe del método.

En trabajos futuros convendría abordar el problema relacionadas con la reducida cantidad de datos disponibles accediendo a numerosos repositorios de ontologías si es posible de diferentes lenguas con temáticas variadas

Bibliografía

[1] Dimitrios Galanis Ion Androutsopoulos Gerasimos Lampouras. «Generating Natural Language Descriptions from OWL Ontologies: the Natural OWL System». En: *Artificial Intelligence Research* (2014). DOI: https://doi.org/10.1613/jair.4017. URL: https://arxiv.org/ftp/arxiv/papers/1405/1405.6164.pdf.

[2] Paolo Dongilli Mustafa Jarrar C. Maria Keet. «Multilingual verbalization of ORM conceptual models and axiomatized ontologies». En: (2014). URL: http://hdl.handle.net/20.500.11889/4200.

[3] Protégé. «File named pizza.owl». En: (2016). URL: https://protege.stanford.edu/ontologies/pizza/pizza.owl.

[4] Ontology Alignment Evaluation Initiative. «Conference track». En: (2021). URL: http://oaei.ontologymatching.org/2021/conference/index.html.

[5] *Jena Ontology API*. URL: https://jena.apache.org/documentation/ontology/.

[6] *Protégé*. URL: https://protege.stanford.edu/.

[7] *Pellet Reasoner*. URL: https://www.w3.org/2001/sw/wiki/Pellet.

[8] *OWL API Semantic Web*. URL: https://www.w3.org/2001/sw/wiki/OWLAPI.

[9] *Deep Translator python library*. URL: https://pypi.org/project/deep-translator/.

[10] *Langid python library*. URL: https://pypi.org/project/langid/1.1.2dev/.

[11] *Datasets python library*. URL: https://pypi.org/project/datasets/.

[12] *Transformers python library*. URL: https://www.analyticsvidhya.com/blog/2021/09/a-deep-dive-into-transformers-library/.

[13] *Kaggle*. URL: https://www.kaggle.com/.

[14] Asunción Gómez-Pérez, Mariano Fernández-López y Oscar Corcho-Garcia. *Ontological Engineering with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*. ISBN: 1852335513. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2003.

[15] *RDF 1.1 Primer*. Inf. téc. Feb. de 2014. URL: http://www.w3.org/TR/rdf11-primer/.

[16] «RDF Schema». En: (2022). URL: https://es.wikipedia.org/wiki/RDF_Schema.

[17] *Lenguaje de Ontologías Web (OWL)*. Inf. téc. 2007. URL: https://www.w3.org/2007/09/OWL-Overview-es.html.

[18] *RDF 1.1 Turtle*. Inf. téc. Feb. de 2014. URL: https://www.w3.org/TR/turtle/.

[19] *OWL 2 Web Ontology Language: Primer (second edition)*. Inf. téc. Dic. de 2012. URL: http://www.w3.org/TR/owl2-primer/.

[20] Jason Brownlee. *What Are Word Embeddings for Text?* Inf. téc. 2017. URL: https://machinelearningmastery.com/what-are-word-embeddings/.

[21] «Distributed Representations of Words and Phrases and their Compositionality». En: (2013). DOI: "https://doi.org/10.48550/arXiv.1310.4546". URL: https://arxiv.org/pdf/1310.4546.pdf.

[22] «Word 2 vector». En: (2022). URL: https://es.wikipedia.org/wiki/Word2vec.

[23] Yinhan Liu y col. «RoBERTa: A Robustly Optimized BERT Pretraining Approach». En: *CoRR* abs/1907.11692 (2019). arXiv: 1907.11692. URL: http://arxiv.org/abs/1907.11692.

[24] Yoshua Bengio, Réjean Ducharme y Pascal Vincent. «A Neural Probabilistic Language Model». En: *Advances in Neural Information Processing Systems*. Ed. por T. Leen, T. Dietterich y V. Tresp. Vol. 13. MIT Press, 2000. URL: https://proceedings.neurips.cc/paper/2000/file/728f206c2a01bf572b5940d7d9a8fa4c-Paper.pdf.

[25] «Efficient Estimation of Word Representations in Vector Space». En: (2013). DOI: "https://doi.org/10.48550/arXiv.1301.3781". URL: https://arxiv.org/pdf/1301.3781.pdf.

[26] Ashish Vaswani y col. «Attention is All you Need». En: *Advances in Neural Information Processing Systems*. Ed. por I. Guyon y col. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[27] «GPT-1, GPT-2 and GPT-3». En: (2021). URL: https://360digitmg.com/types-of-gpt-in-artificial-intelligence.

[28] Jacob Devlin y col. «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding». En: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[29] Zuhaib Akhtar. «BERT base vs BERT large». En: (2019). URL: https://iq.opengenus.org/bert-base-vs-bert-large/.

[30] Ph.D. Suleiman Khan. «BERT, RoBERTa, DistilBERT, XLNet — which one to use?» En: (2019). URL: https://towardsdatascience.com/bert-roberta-distilbert-xlnet-which-one-to-use-3d5ab82ba5f8.

[31] P Sreenivasa Kumar Vinu E.V. «Ontology Verbalization using Semantic-Refinement». En: *Semantic Web Journal* (2016). DOI: https://doi.org/10.48550/arXiv.1610.09964. URL: https://arxiv.org/pdf/1610.09964.pdf.

[32] Richard Power y Allan Third. «Expressing OWL axioms by English sentences: Dubious in theory, feasible in practice». En: *Coling 2010 - 23rd International Conference on Computational Linguistics, Proceedings of the Conference* 2 (ene. de 2010). URL: https://www.researchgate.net/publication/45515002_Expressing_OWL_axioms_by_English_sentences_Dubious_in_theory_feasible_in_practice.

[33] Daniel Rodríguez. *La distancia de Levenshtein*. 2020. URL: https://www.analyticslane.com/2020/06/17/la-distancia-de-levenshtein/.

[34] Adrián Espino Candalija. «Conjunto de datos de tripletas CSV». En: (2022). URL: https://github.com/Adrian-uni/OntVerbal/tree/main/pruebasKaggle/tripletas.

[35] Adrián Espino Candalija. «Conjunto de datos de verbalizaciones obtenidas usando OntVerbal». En: (2022). URL: https://github.com/Adrian-uni/OntVerbal/tree/main/pruebasKaggle/datos.

[36] Adrián Espino Candalija. «Conjunto de datos de resultados obtenidos en Kaggle». En: (2022). URL: https://github.com/Adrian-uni/OntVerbal/tree/main/pruebasKaggle/resultados.

[37] Yoshua Bengio, Réjean Ducharme y Pascal Vincent. «A Neural Probabilistic Language Model». En: *Advances in Neural Information Processing Systems*. Ed. por T. Leen, T. Dietterich y V. Tresp. Vol. 13. MIT Press, 2000. URL: https://proceedings.neurips.cc/paper/2000/file/728f206c2a01bf572b5940d7d9a8fa4c-Paper.pdf.

[38] Srinivas Chakravarthy. «Tokenization for Natural Language Processing». En: (2020). URL: https://towardsdatascience.com/tokenization-for-natural-language-processing-a179a891bad4.

[39] Nasser Zalmout y Nizar Habash. «Utilizing Subword Entities in Character-Level Sequence-to-Sequence Lemmatization Models». En: *Proceedings of the 28th International Conference on Computational Linguistics*. Barcelona, Spain (Online): International Committee on Computational Linguistics, dic. de 2020, págs. 4676-4682. DOI: 10.18653/v1/2020.coling-main.412. URL: https://aclanthology.org/2020.coling-main.412.

Lista de Figuras

Lista de Tablas

Anexos

Anexos A

Dedicación

En este apartado se expone mediante un diagrama de Gantt la organización y desarrollo seguida durante la realización del proyecto, las horas dedicadas a cada tarea del diagrama de Gantt de la figura D.1 son las siguientes:

Tiempo dedicado	
Tarea	Tiempo Dedicado(h)
Estudio Previo	65
Método de verbalización	142.5
Corrección de la verbalización	32.5
Pruebas de evaluación	44.5
Escritura de la Memoria	69.5
Reuniones	24
Total	378

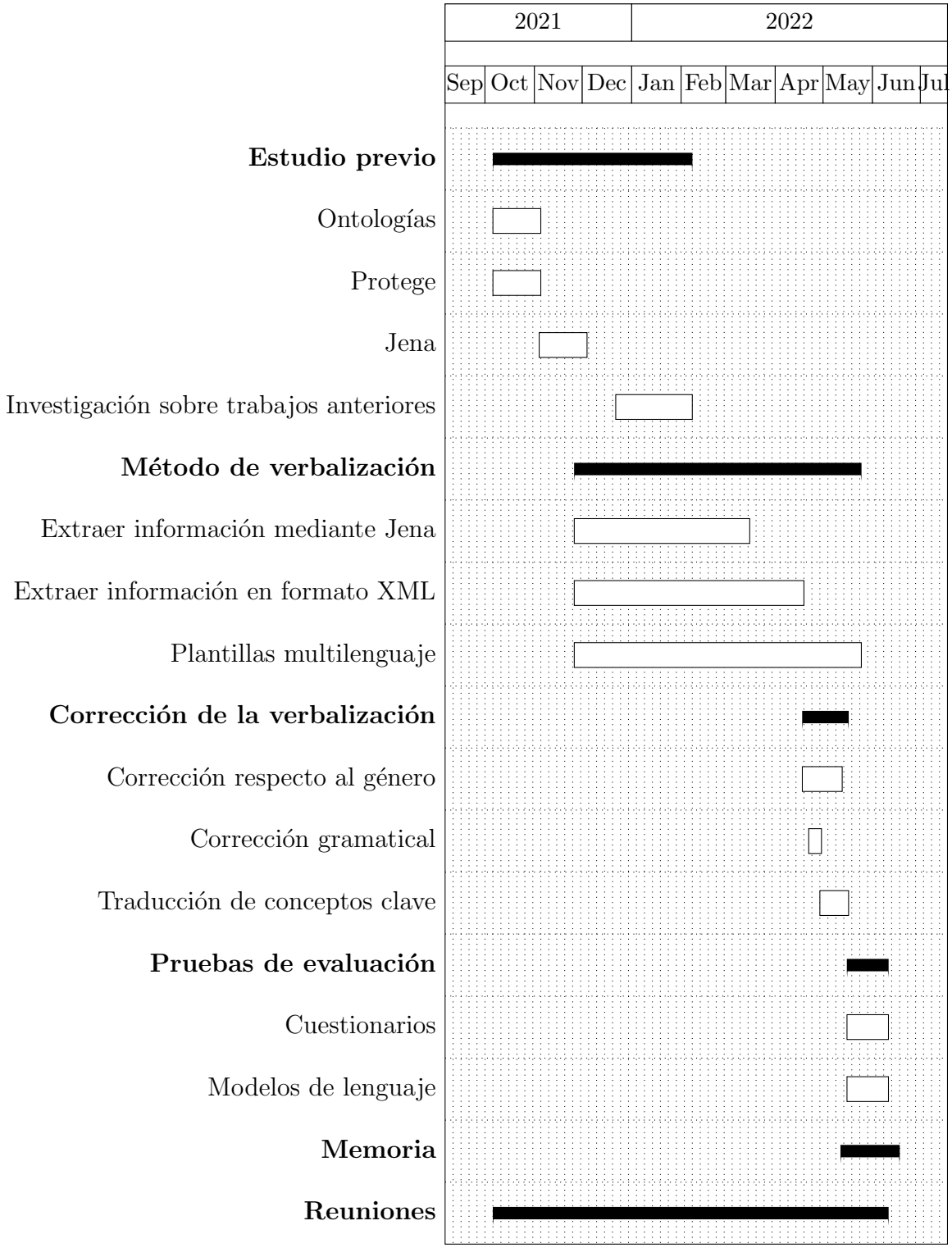

Figura A.1: Diagrama de Gantt.