Universidad Zaragoza
1542

Trabajo Fin de Grado

Sistema inteligente para el descubrimiento de relaciones de sinonimia e hiponimia en ontologías

An intelligent system for the discovery of synonymy and hyponymy relationships across ontologies

Autor

Miguel Bolsa Marquina

Director

Fernando Bobillo Ortega

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2019

AGRADECIMIENTOS

En primer lugar agradecer a Fernando, mi director, por su orientación, experiencia y ayuda a lo largo del desarrollo del proyecto.

Me gustaría dar las gracias a Blanca por estar a mi lado, apoyarme, creer en mí y aguantarme en todo momento, ya que sin ella no hubiera acabado el proyecto. También, a mi madre, mi padre y mi hermana por ayudarme a ser la persona que soy hoy. Y a mis amigos, por el apoyo y comprensión que me han brindado durante el transcurso de este proyecto.

Sistema inteligente para el descubrimiento de relaciones de sinonimia e hiponimia en ontologías

RESUMEN

Las ontologías se han convertido en un estándar para la representación del conocimiento en muchas aplicaciones del mundo real. En la práctica, es muy frecuente encontrar varias ontologías que tratan de modelar el mismo dominio de aplicación o, al menos, existe un solapamiento entre la información representada. En estos casos es generalmente necesaria una fase de integración de información que permita reconciliar ambas visiones del mundo. Una aproximación para resolver este problema consiste en alinear ambas ontologías, buscando relaciones semánticas entre ambas.

Típicamente, es habitual buscar relaciones de sinonimia (dos términos con el mismo significado) o hiponimia (un término con un significado más general que otro). Sin embargo, la mayoría de las propuestas existentes (con alguna excepción) se restringen a una de las dos: por ejemplo, un indicio de que dos términos son sinónimos es que tienen los mismos padres y los mismos hijos, pero los sistemas existentes se limitan a las relaciones de hiponimia conocidas (las de la ontología original).

Este trabajo aborda ambos tipos de relaciones de forma simultánea utilizando algoritmos de aprendizaje automático y muestra las relaciones encontradas en una aplicación Android.

Índice

Capítulo 1

Introducción

La evolución y popularización del uso de la red han derivado en la proliferación de contenidos dispersos que dificultan la extracción de información relevante para el usuario. Con el objetivo de que compartir información resulte más fácil y eficiente, actualmente podemos encontrar un gran volumen de contenidos disponibles en la red, estructurados según los estándares de la Web semántica y relacionados con una ontología, desarrollada para facilitar la extracción de contenidos relacionados entre sí.

No obstante, surge el problema de que, cuando diversas fuentes crean su ontología propia y la dotan de estructura, se originan conflictos para compartir la información. Cada fuente puede darle relevancia a relaciones distintas y estructurar la información con más o menos acierto a la vista de otro usuario. Esto puede suponer distintos problemas como puede ser la duplicidad, solapamiento o redundancia de la información.

Para dar respuesta a estos problemas, encontramos muchos trabajos sobre alineamiento e integración de ontologías. Para realizar dicha integración o alineamiento, se buscan relaciones entre ontologías identificando relaciones semánticas. La mayoría de estos trabajos se limitan a buscar únicamente relaciones de sinonimia o de hiponimia/hiperonimia, pero no una identificación simultánea de todas ellas.

Actualmente, se ha logrado un alto grado de estandarización en la definición de ontologías gracias al uso de un lenguaje estándar como OWL (vea Sección 2.1.2). Sin embargo, todavía se trabaja en el desarrollo de plataformas de integración de ontologías y, en el hecho de que la diferencia en el idioma de definición no constituya una barrera. Para ello, el alineamiento de ontologías, es decir, la identificación de las relaciones de coincidencia semántica y estructural entre distintas ontologías, constituye una tarea fundamental.

1.1. Objetivos

El objetivo de este Trabajo Fin de Grado (en adelante TFG o proyecto) es crear una plataforma inteligente para el descubrimiento automático de relaciones de sinonimia e hiponimia/hiperonimia entre ontologías. Las nuevas relaciones semánticas que se vayan descubriendo deberán poderse utilizar para la inferencia de otras relaciones. Para el descubrimiento, se consideran y comparan diferentes técnicas heurísticas y algoritmos de aprendizaje automático, para poder seleccionar los más adecuados. Además la plataforma podrá ejecutarse en dispositivos móviles Android.

El desarrollo del trabajo se ha realizado cubriendo objetivos paso a paso:

- Exploración eficiente de las ontologías utilizando un algoritmo de recorrido de grafos.
- Extracción de relaciones de sinonimia e hiponimia/hiperonimia, utilizando diferentes heurísticas.
- Utilización de las relaciones que se van descubriendo para que puedan inducir otras relaciones.
- Aprendizaje de los datos utilizando diferentes algoritmos de aprendizaje automático.
- Experimentación y validación de los modelos con datos nuevos.
- Desarrollo en Android, para que pueda ser utilizado por dispositivos móviles.

1.2. Estructura del documento

El contenido de este documento se desarrolla a través de los siguientes capítulos:

- El **capítulo 2** muestra el contexto técnico en el que se ha desarrollado el proyecto, a modo de somera introducción, con ánimo de ser poco exhaustivo, que permita familiarizarse con algunos de los conceptos y técnicas utilizadas a lo largo del documento, a posibles lectores poco iniciados.
- El **capítulo 3** comenta otros trabajos relaciones con el tema y los compara con el desarrollado.
- El **capítulo 4** se dedica a la descripción del proceso seguido para la extracción de las relaciones semánticas objeto de interés. Se muestra la arquitectura del sistema, se describe su funcionamiento, y la naturaleza de los resultados perseguidos.

– El **capítulo 5** describe la implementación multilingüe del sistema.

– El **capítulo 6** ofrece la descripción y valoración de los resultados obtenidos en diversas pruebas realizadas con el sistema definido.

– El **capítulo 7** explica el funcionamiento de la aplicación móvil.

– El **capítulo 8** está dedicado a mostrar las conclusiones obtenidas, identificar el posible trabajo a futuro y a la descripción de la gestión del proyecto.

Capítulo 2

Contexto

Este capítulo describe el contexto tecnológico en que se desarrolla el presente TFG, introduciendo una serie de conceptos sobre las tecnologías y técnicas utilizadas.

Se describen brevemente una serie de conceptos previos para familiarizarse con la naturaleza del entorno del trabajo, así como las técnicas fundamentales que conforman la base para obtener resultados y las herramientas utilizadas para su desarrollo.

2.1. Conceptos previos

2.1.1. Ontologías

La definición de ontología que, con mayor frecuencia, encontramos en la literatura, es la dada por Thomas Gruber en 1993, que la define como "una especificación de una conceptualización"[1].

Simplificando mucho y ciñéndonos al mínimo conocimiento que se requiere para la comprensión de este trabajo, podemos decir que, en el contexto de la Ciencia y Tecnología de la Información, una ontología es la especificación de un área de conocimiento a la que se dota de una estructura lógica, a partir de la realidad, para que resulte utilizable informáticamente para el intercambio de información de forma eficiente.

En este contexto, una ontología está constituida por multitud de elementos que conforman una estructura jerarquizada del área de conocimiento que se quiere plasmar. Entre ellos se encuentran los siguientes, que son los que se han utilizado en el desarrollo del trabajo que nos ocupa:

- Clases: representan conceptos reales relacionados con el área de conocimiento en cuestión y se encuentran relacionadas jerárquicamente con *"Super Classes"* (conceptos del mismo tipo en nivel superior) y *"Sub Classes"*.

– Propiedades: representan atributos o características que concretan la definición de la clase con la que se relacionan.

– Individuos: representan entes u objetos que pertenecen a unas clases determinadas.

2.1.2. OWL 2

El World Wide Web Consortium (*W3C*[1]), comunidad internacional que desarrolla estándares abiertos para asegurar el crecimiento de la Web, definió en 1999 el Marco de Descripción de Recursos (RDF, por sus siglas en inglés), consistente en un conjunto de especificaciones relacionadas con la definición digital de conocimiento[2].

Con posterioridad, en respuesta a las muchas limitaciones del RDF y RDFS (RDF Schema) el *W3C* impuso como recomendación el lenguaje *OWL, Web Ontology Language*, para desarrollar ontologías, diseñadas de manera que puedan ser utilizadas por aplicaciones que necesitan procesar el contenido de la información. De hecho este lenguaje se ha convertido en un estándar para la definición de ontologías.

OWL 2[2] es una revisión y extensión de *OWL* que dispone de tres versiones diferentes, que varían según su expresividad, la cual es restada a cambio de una mejora en el coste computacional: *OWL 2 EL*, *OWL 2 QL* y *OWL 2 RL*.

2.1.3. OWL API

OWL API [3] es una interfaz que soporta la creación y manipulación de ontologías escritas en OWL y el uso de razonadores semánticos (vea Sección 2.1.4). Está estrechamente alineada con la especificación estructural de OWL 2 (Sección 2.1.2), incluyendo validadores para sus tres versiones. Está implementada en Java. Permite parsear y serializar ontologías en una variedad se sintaxis y tiene un uso generalizado en una variedad de herramientas y aplicaciones.

2.1.4. Razonador semántico

Un razonador semántico [4] es un software que puede inferir consecuencias lógicas a partir de un conjunto de axiomas de una ontología. Implementa o da soporte a multitud de tareas, de las cuales muchas se han utilizado a lo largo de este proyecto, como por ejemplo: extracción de la estructura de una entidad, identificación del tipo de entidad, entre otras.

[1] https://www.w3.org/
[2] https://www.w3.org/TR/owl2-overview/

Existen varios razonadores compatibles con OWL API: HermiT, FaCT, Pellet, FaCT++. De entre ellos, se ha escogido HermiT [5] debido a su eficiencia y a que, gracias al grupo de *Sistemas de Información Distribuidos (SID)* de la *Universidad de Zaragoza* se dispone de una versión que funciona en Android, para el cual no está pensado originalmente.

2.1.5. Relaciones semánticas

Por relación semántica entendemos la correspondencia existente entre dos elementos que pertenecen a una ontología. Existen multitud de tipos de relaciones semánticas [6], pero este proyecto se centra en trabajar solo con tres de ellas (vea Figura 2.1):

- Sinonimia (*synonymy*): dos elementos e_1 y e_2 son sinónimos si tienen el mismo significado o son equivalentes (por ejemplo, mom $\equiv$ mother).

- Hiponimia (*hyponymy*): un elemento e_1 es hipónimo de otro e_2 cuando el significado de e_1 es más específico que el de e_2 (por ejemplo, man $\sqsubseteq$ person).

- Hiperonimia (*hypernymy*): un elemento e_1 es hiperónimo de otro e_2 cuando el significado de e_1 es más general que el de e_2 (por ejemplo, person $\sqsupseteq$ man).

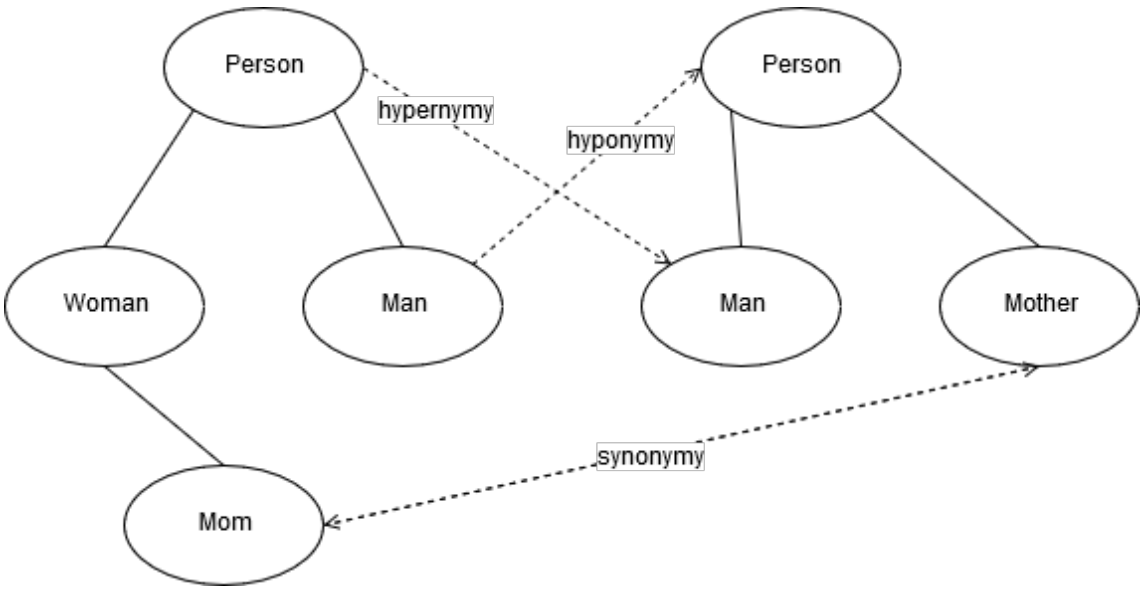


Figura 2.1: Ejemplo de relaciones semánticas

2.2. Técnicas de aprendizaje

2.2.1. Redes Neuronales

Intuitivamente, una red neuronal [7] es un modelo matemático que se inspira en el funcionamiento biológico del cerebro. Los elementos aritméticos simples de computación representan las neuronas y la red, en su conjunto, corresponde a la interconexión entre un conjunto de éstas.

Una red neuronal está compuesta por un conjunto de nodos conectados por enlaces. Cada enlace tiene asociado un peso. La actualización de estos pesos, pretende una mejora en el ajuste del comportamiento de la red, ya que son el medio principal de almacenamiento a largo plazo. Cada nodo tiene un conjunto de enlaces de entrada y salida a otros nodos y un nivel de actuación, que es calculado a partir de un método en el siguiente paso en el tiempo, a partir de sus entradas y pesos. De esta manera se consigue que cada nodo realice un cálculo local basado en las aportaciones de sus enlaces de entrada, pero sin necesidad de ningún control global sobre el resto de nodos.

2.2.2. Random Forests

Es un método de clasificación o regresión (en este proyecto de clasificación) que se creó para reducir la varianza y mejorar la predictibilidad de los árboles de decisión [8]. Un árbol de decisión es un método de clasificación o regresión cuyo objetivo es dividir el conjunto de predictores en regiones disjuntas. El funcionamiento del método consiste en los siguientes pasos:

1. Crear un conjunto de árboles de decisión con el conjunto de datos de entrenamiento.

2. Cada uno de estos árboles, entrena un subconjunto aleatorio de datos. Para entrenar los datos se eligen al azar un subconjunto de predictores, para así dar una oportunidad a los predictores con menos relevancia.

3. Cada uno de estos árboles aporta una clasificación de los datos. Estas clasificaciones se combinan de tal manera, que los datos serán clasificados en el grupo que más árboles lo han clasificado.

2.3. Herramientas utilizadas

Para el desarrollo de este proyecto se han utilizado las siguientes herramientas, ordenadas por funcionalidad dentro del proyecto.

Herramientas para trabajar con ontologías

- HermiT[3]: razonador semántico para ontologías escritas en *Web Ontology Language (OWL).*
- OWL API[4]: es una API de Java para trabajar con ontologías escritas en OWL.
- Protégé[5]: editor de código abierto que permite editar y visualizar la estructura de las ontologías en forma de árbol.

Lenguajes de programación

- Java[6]: lenguaje de programación que, en el contexto de este proyecto se ha utilizado en multitud de tareas como: extraer las características de las entidades de las ontologías, generar el algoritmo de de recorrido de grafos, buscar relaciones en WordNet, etc.
- R[7]: lenguaje de programación para crear *Random Forests.*
- Android[8]: lenguaje de programación móvil.

Entornos de desarrollo

- Eclipse[9]: entorno de desarrollo de Java.
- RStudio[10]: entorno de desarrollo de R.
- Android Studio[11]: entorno de desarrollo de Android.

Bases de datos externas

- WordNet[12]: base de datos en inglés que contiene nombres, adjetivos, adverbios y verbos, que están vinculados mediante relaciones léxicas y semánticas.
- JWI[13]: librería para interactuar con WordNet.

[3] http://www.hermit-reasoner.com/
[4] http://owlapi.sourceforge.net/
[5] https://protege.stanford.edu/
[6] https://www.java.com/es/
[7] https://cran.r-project.org/
[8] https://www.android.com/
[9] https://www.eclipse.org/
[10] https://www.rstudio.com/
[11] https://developer.android.com/studio
[12] https://wordnet.princeton.edu/
[13] https://projects.csail.mit.edu/jwi/

Librerías para la utilización de algoritmos de aprendizaje automático

- Neuroph Studio[14]: librería de código abierto para crear *redes neuronales*.
- Rserve[15]: librería para ejecutar código en R desde Java.

Traductores

- Yandex[16]: traductor de cadenas formadas por una o más palabras.
- ConcepNet[17]: traductor de cadenas formadas por una palabra.

Documentación y control de versiones

- Git + GitHub[18]: software y plataforma de desarrollo para el control de versiones.
- Overleaf[19]: plataforma para la redacción de este documento utilizando el sistema de composición de documentos llamado LaTeX[20].

[14] http://neuroph.sourceforge.net/
[15] https://www.rforge.net/Rserve/index.html
[16] https://translate.yandex.com/
[17] http://conceptnet.io/
[18] https://github.com/
[19] https://es.overleaf.com/
[20] https://es.wikipedia.org/wiki/LaTeX

Capítulo 3

Estado del arte

Como se ha comentado en el Capítulo 1, existen multitud de trabajos sobre alineamiento e integración de ontologías, pero que apenas existen trabajos que extraen los dos tipos de relación (sinonimia e hiponimia/hiperonimia). A continuación, se explica el funcionamiento de algunos de estos trabajos según el tipo o tipos de relaciones que extraen.

3.1. Extraen relaciones de sinonimia

Existen multitud de trabajos que tratan este aspecto, en [9] hacen referencia a muchos de ellos. A continuación, se va a describir un trabajo a partir del cual se basa el proyecto que se explica en este documento.

3.1.1. CIDER

CIDER (*Context and Inference baseD alignER*) [10] es un sistema para el alineamiento entre términos de dos ontologías. Trata de extraer relaciones de sinonimia extrayendo su contexto ontológico hasta una cierta profundidad y posteriormente combinando diferentes técnicas de emparejamiento ontológico. Estas técnicas son de dos tipos: similitud lingüística entre etiquetas utilizando Levenstein y similitud estructural a partir del contexto ontológico y también involucra fuentes externas como WordNet y otras ontologías indexadas por Swoogle. En su versión actual, utiliza redes neuronales para combinar estas técnicas.

3.2. Extraen relaciones de hiponimia/hiperonimia

3.2.1. SCARLET

SCARLET (*SemantiC relAtion discoveRy by harvesting onLinE onTologies*) [11] es un sistema que recoge información de la Web Semántica, explorando muchas ontologías

heterogéneas que están online. Trata de deducir la relación entre dos clases utilizando dos técnicas. La primera deriva de una relación entre dos clases si esta relación está definida dentro de una única ontología online. La segunda estrategia se utiliza cuando la primera no ha sido exitosa extendiendo su funcionamiento a dos o más ontologías.

3.2.2. PFC Enrique Solano

El sistema implementado por Enrique Solano consiste en la extracción de relaciones de hiponimia e hiperonimia entre clases de dos ontologías. Utiliza técnicas de similitud lingüística entre clases y similitud estructural a partir de la comparación entre todas las propiedades que existen en ambas ontologías. Utiliza una técnica de agrupación (*clustering*) para descartar relaciones a partir de un umbral dinámico, agrupándolas en tres según el nivel de confianza: alto, medio y bajo.

3.3. Extraen ambos tipos de relación

3.3.1. STROMA

STROMA (*SemanTic Refinement of Ontology MAppings*) [12] es un sistema que determina mapeos semánticos entre ontologías. Utilizan un enfoque de enriquecimiento de dos pasos para determinar estos mapeos. Estos dos pasos son: emparejamiento (utiliza otras herramientas de mapeo ontológico como *COMA*) y enriquecimiento semántico. Utilizan cinco estrategias (análisis de cadenas compuestas, conocimientos previos, itemización, estructura y relaciones encadenadas) para determinar relaciones de sinonimia, hiponimia/hiperonimia y partes de relaciones.

3.4. Comparación

El sistema presentado tiene un funcionamiento similar a CIDER, ya que las extracciones de relaciones se hacen a partir de la información de las ontologías a comparar y también de fuentes externas. La principal deferencia del sistema que se presenta es que mediante la aplicación o sustitución de métricas utilizadas (por ejemplo, en CIDER únicamente se utiliza Levenstein para encontrar similitudes lingüísticas, mientras que en el sistema desarrollado, también se utiliza Stoilos), se extraen además relaciones de hiponimia e hiperonimia. Además, también se ha utilizado el algoritmo de aprendizaje Random Forest para detectar relaciones.

Respecto a SCARLET, PFC y a STROMA, el sistema es capaz de extraer relaciones de hiponimia/hipernimia no solo entre clases, sino también entre propiedades, relaciones que inducen nuevas posibles relaciones entre clases.

Con respecto a STROMA, el sistema desarrollado implementa todas las estrategias de STROMA, a excepción de la itemización, porque se utilizan otras métricas que calculan su resultado, como es el uso de afijos.

Por último, el sistema se diferencia del PFC de Enrique Solano en que utiliza algoritmos de aprendizaje para sustituir el uso de un componente de juicio, así como una modificación y adición respecto a sus métricas. Igualmente, el sistema de Enrique Solano necesita recorrer las ontologías dos veces: una para encontrar hipónimos en una ontología y otra para encontrar hipónimos en la otra ontología, mientras que en el sistema desarrollado se encuentran simultáneamente, disminuyendo el tiempo de ejecución.

Capítulo 4

Extracción de relaciones semánticas

Este capítulo introduce el proceso seguido para la extracción de relaciones semánticas, tanto de sinonimia como de hiponimia e hiperonimia. Se describe la arquitectura del sistema implementado y la forma de realizar la comparación entre entidades, teniendo en cuenta las peculiaridades de sus características, el entrenamiento de los modelos, la extracción de candidatos y el filtrado de relaciones identificadas.

4.1. Arquitectura del sistema

Dadas dos ontologías (O_1 y O_2) el sistema trata de descubrir las posibles relaciones semánticas entre sus entidades[1] (vea Figura 2.1), mediante los siguientes pasos:

1. Exploración de ontologías: para poder encontrar relaciones semánticas entre O_1 y O_2, lo primero de todo se recorren las ontologías de abajo a arriba utilizando el algoritmo de grafos DFS (*Depth First Search*, en español búsqueda en profundidad). Además, se establecen restricciones entre las comparaciones (vea Sección 4.2).

2. Comparación entre pares de entidades: para cada par de entidades (e_1 y e_2), donde $e_1 \in O_1$ y $e_2 \in O_2$, se efectúa la comparación entre un conjunto de características predefinidas que poseen dichas entidades, según la tipología a la que pertenezcan (vea Sección 4.3). El resultado es un vector cuyas componentes, todas ellas pertenecientes al intervalo [0, 1], constituyen un indicador del grado de similitud que presentan las entidades comparadas, para cada característica considerada. Las comparaciones se establecen en cuanto a relación léxica, semántica y de estructura inherente a la ontología tratada.

[1] Las entidades pueden ser clases, propiedades o individuos.

3. Entrenamiento de los modelos[2]: el conjunto de resultados obtenidos en el paso anterior, es decir, todas las comparaciones posibles entre parejas de entidades de un conjunto de ontologías, se utilizan como fuente de aprendizaje mediante redes neuronales o *Random Forests* (vea Sección 4.4) y se determina un umbral para filtrar las posibles relaciones semánticas maximizando la relación entre *precision* (precisión) y *recall* (exhaustividad).

4. Extracción de candidatos de relaciones semánticas: una vez entrenados los modelos, se vuelve a ejecutar el paso 2, con dos ontologías que no se han utilizado en el entrenamiento. El vector obtenido se utilizará como dato de entrada de los modelos obtenidos en el paso 3, los cuáles generarán una salida que indica la posible relación semántica entre las parejas comparadas (vea Sección 4.5).

5. Filtrado de relaciones[3]: mediante la comparación con los umbrales establecidos, se filtran las posibles relaciones extraídas en el punto anterior, seleccionando la de mayor grado de confianza. (vea Sección 4.6).

La salida del sistema contiene una lista ordenada según el grado de confianza de todas las posibles relaciones identificadas como relevantes para cada tipo de relación. La Figura 4.1 muestra de forma gráfica la estructura del sistema.

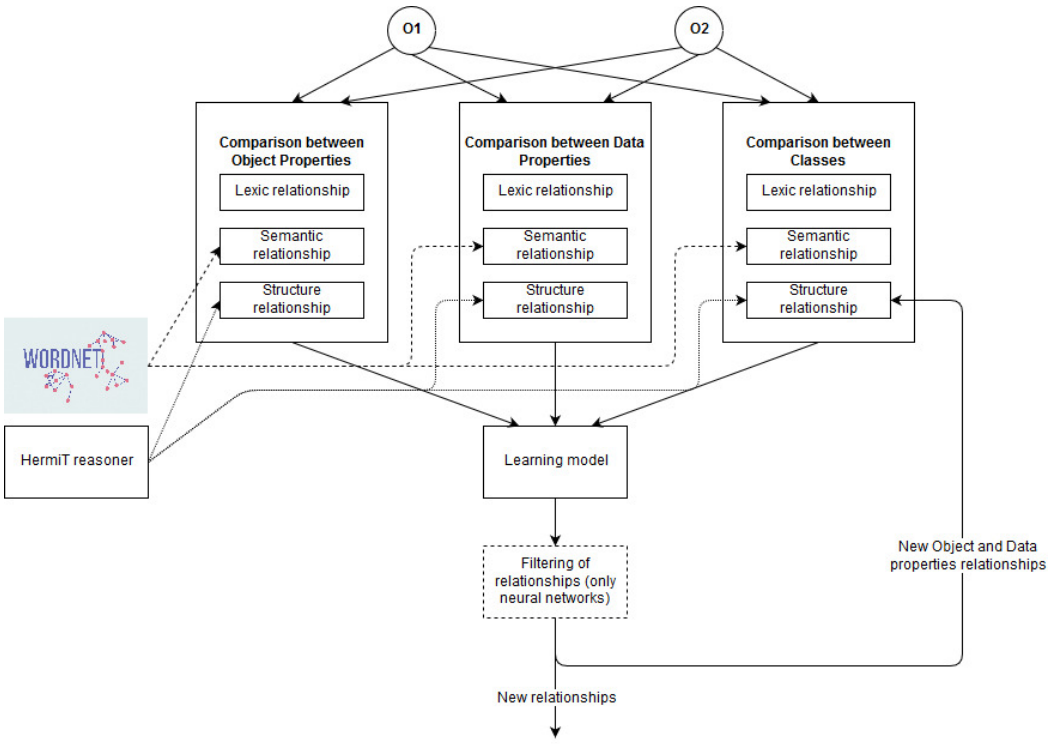


Figura 4.1: Arquitectura del sistema

[2]Un modelo para las clases, dos para propiedades (uno para Object Properties y otro para Data Properties) y otro para individuos.

[3]Únicamente si se utilizan redes neuronales como algoritmo de aprendizaje.

4.2. Exploración de ontologías

Antes de comenzar a comparar entidades entre sí, lo primero que hay que hacer es desarrollar un algoritmo que recorra las dos ontologías a comparar. Se utiliza el algoritmo DFS empezando por abajo hasta llegar al elemento raíz.

Se ha decidido así, porque una ontología tiene una estructura de grafo o árbol y por la naturaleza de las características que se van a usar para extraer relaciones (vea Sección 4.3). Por ejemplo, se elige el algoritmo DFS porque las características de estructura únicamente requieren de información que existe en la rama observada y dirección de abajo a arriba porque alguna característica necesita la información obtenida en la comparación con elementos que están por debajo.

El algoritmo no permite la comparación entre entidades de distinto tipo, es decir, únicamente se pueden comparar clase - clase, object property - object property, data property - data property e individuo - individuo. Además, tampoco permite realizar comparaciones entre entidades cuya IRI no es la misma que la de la ontología. Por ejemplo, muchas ontologías tienen la clase *"http://www.w3.org/1999/02/22-rdf-syntax-ns#List"*.

Se comparan todas las entidades de un tipo de una ontología con todas las entidades del mismo tipo de otra ontología, lo que implica que el coste temporal del algoritmo es 4 x O(n x m), siendo n y m el número de entidades del tipo que se está comparando de cada una de las dos ontologías en tratamiento. Por último, el orden de extracción de relaciones será el siguiente:

1. Propiedades
2. Clases
3. Individuos

4.3. Comparación entre pares de entidades

Para cada par de entidades (e_1 y e_2), donde $e_1 \in O_1$ y $e_2 \in O_2$, se efectúa la comparación con respecto a un conjunto de características predefinidas que poseen dichas entidades, según la tipología a la que pertenezcan (vea Sección 2.1.1).

Algunas características, como puede ser *comments*, resultan comunes a todas las entidades (aunque pueden no tener contenido en la ontología) pero otras, son típicas de determinados tipos de entidades (por ejemplo *domain* solo aparece en las propiedades). Por ello, el vector será diferente según el modelo, debido a que no se tienen en cuenta las mismas características para clases, que para propiedades o individuos (por ejemplo,

las características para evaluar la relación de hiponimia/hiperonimia no se aplican a los individuos debido a que no tiene sentido, pues se encuentran todos en la base de la jerarquía definida).

Para identificar la posible relación semántica se recurre a distintos mecanismos:

- Métricas de similitud de cadenas.
- Extracción de relaciones conocidas a través de fuentes externas: WordNet.
- Características basadas en la estructura de la ontología tratada.

Métricas de similitud de cadenas

Se han utilizado tres tipos de métricas diferentes, de los indicados en [13, 14].

- Levenstein: devuelve el número mínimo de cambios que se requiere hacer en una cadena para formar otra (por ejemplo, para formar *back* a partir de *book* habría que realizar dos cambios). Para que el resultado se encuentre en el intervalo [0,1] se normaliza mediante la siguiente fórmula [15]:

$$levenstein(x, y) = \text{máx}(0, \frac{\text{mín}(|x|, |y|) - movements(x, y)}{\text{mín}(|x|, |y|)})$$

  donde *movements(x, y)* devuelve el citado número mínimo de cambios entre dos cadenas, $|x|$ devuelve el tamaño de la cadena x y la función *mín($|x|$, $|y|$)* devuelve el tamaño de la cadena menor.

- Stoilos: considera tanto los caracteres coincidentes como las diferencias entre las dos cadenas a comparar. El resultado se calcula a partir de la fórmula [16]:

$$sim(x, y) = comm(x, y) - diff(x, y) + winkler(x, y)$$

  donde *comm(x,y)*, proporciona una medida de similitud a partir de las subcadenas compartidas más largas que encuentra (por encima de dos caracteres) de las entidades comparadas; *diff(x,y)*, calcula una medida de disimilitud a partir de *comm(x,y)*; y *winkler(x,y)* mejora el resultado utilizando el método introducido en [17], el cual da preferencia a aquellas cadenas que tienen un prefijo en común. Como el resultado pertenece al intervalo [-1, 1], se normaliza con la siguiente fórmula:

$$stoilos(x, y) = \frac{sim(x, y) + 1}{2}$$

– Afijos[4]: consiste en indicar si una de las cadenas es prefijo o sujifo respecto a la otra:

$$isAffix(x,y) = \begin{cases} 0 & \text{si } x \text{ ni es un sufijo ni un prefijo de } y \text{ o son iguales} \\ 1 & \text{si } x \text{ es un sufijo o un prefijo de } y \end{cases}$$

Es interesante utilizar esta métrica debido a que a veces el *ID* o la *label* de una entidad, que es hipónimo, es la especificación de su entidad hiperónimo (por ejemplo, *technicalReport* y *report*).

Extracción de relaciones conocidas a través de fuentes externas: WordNet

La base de datos WordNet proporciona el conjunto de significados de un término en inglés. La importancia de utilizar un recurso como WordNet para extraer relaciones semánticas es que contiene sinónimos, hipónimos e hiperónimos para cada uno de dichos significados. Esto facilita que dos entidades que no se pueden relacionar entre sí utilizando una métrica de las mencionadas antes [18], se puedan vincular (por ejemplo, *lorry* y *truck*) en base a su relación semántica.

Para el proyecto se ha utilizado la librería JWI de Java debido a su facilidad de uso, como se comenta en [19]. Además, esta librería permite cargar en memoria su base de datos que, aunque implica una pérdida de tiempo inicial en la carga, se compensa por la mejora del tiempo de búsqueda de un elemento.

A partir de los datos que se obtienen de WordNet, se implementa el siguiente conjunto de operaciones:

$$isSynonym(y,x) = \begin{cases} 1 & \text{si } y \in synonyms(x) \\ 0 & \text{si } y \notin synonyms(x) \end{cases} \tag{4.1}$$

$$isHypernym(y,x) = \begin{cases} 1 & \text{si } y \in hypernyms(x) \\ 0 & \text{si } y \notin hypernyms(x) \end{cases}$$

$$isHyponym(y,x) = \begin{cases} 1 & \text{si } y \in hyponyms(x) \\ 0 & \text{si } y \notin hyponyms(x) \end{cases}$$

donde las funciones *synonyms(x), hypernyms(x) e hyponyms(x)* devuelve el conjunto de sinónimos, hiperónimos e hipónimos, respectivamente, de x a partir de WordNet.

[4]isAffix se aplica en dos direcciones, es decir, isAffix(x, y) e isAffix(y, x) para detectar posibles relaciones de hiponimia/hiperonimia.

Características basadas en la estructura de una ontología.

La existencia de palabras de gran similitud léxica, pero con significados totalmente diferentes (por ejemplo, *knight* y *night*), aconseja considerar otro tipo de características para extraer relaciones semánticas (vea Sección 4.3.2). Por ejemplo, explorar las subclases de una clase, el dominio de una propiedad o el tipo de individuo. Se trata, en definitiva, de explorar las relaciones jerárquicas o estructurales definidas en la ontología.

4.3.1. Características comunes

Las características *label*, *ID* y *comment* [9] son comunes a todas las entidades con independencia de que tengan contenido o no, como se puede apreciar en la Figura 4.2 (aplicable tanto a clases como a propiedades e individuos).

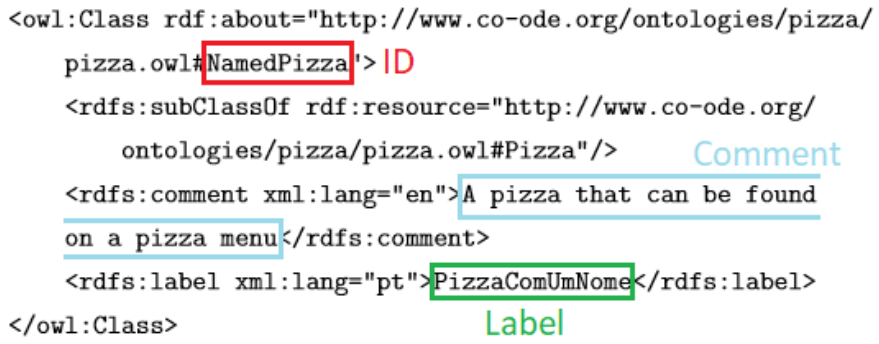


Figura 4.2: Ejemplo de clase en OWL

A estas características, se le aplicarán una serie de métricas de las definidas en los párrafos anteriores (Tabla 4.1). Pero previamente, tienen que ser normalizadas (eliminando guiones, reescribiendo en minúsculas, etc).

Característica	Métricas
Label	levenstein, stoilos, isSynonym, isHypernym, isHyponym e isAffix
ID	levenstein, stoilos, isSynonym, isHypernym, isHyponym e isAffix
Comment	levenstein, stoilos

Tabla 4.1: Características comunes de las entidades

4.3.2. Características no comunes

Como ya se ha introducido anteriormente, además de las características comunes, se analizan otras que dependen del tipo de entidades que se comparan: clases, propiedades e individuos.

Además, el tratamiento difiere, ya que mientras para las comunes las métricas se calculan comparando dos cadenas, para las específicas del tipo de entidad, la comparación se establece entre conjuntos de cadenas.

Para estimar la similitud entre los conjuntos de cadenas[5] que representan las características de cada entidad (en las Secciones 4.3.2.1, 4.3.2.2 y 4.3.2.3, se detallan listados de características de las clases, propiedades e individuos, respectivamente), se utiliza una variante de la intersección de conjuntos. Cada cadena del conjunto de menor tamaño se compara con cada una de las cadenas del otro conjunto guardando el mayor valor de similitud obtenido.

En cuanto a las métricas (o fórmulas) utilizadas para evaluar el grado de similitud, dependerán del tipo de relación semántica que se intenta detectar: sinonimia o hiponimia/hiperonimia.

– Sinonimia: para extraer relaciones semánticas de este tipo ($e_1 \equiv e_2$); en un mundo ideal, e_1 debe tener exactamente las mismas características que e_2 para que tengan el mismo significado. En cambio, en el mundo real se encuentran entidades que no son idénticas, pero sí muy similares [20]. Tras comparar los conjuntos de cadenas, la similitud global entre las entidades se calcula aplicando la siguiente fórmula:

$$sim(e_1, e_2) = \frac{\sum_{x \in m_i(e_1), y \in m_i(e_2)} lexSim(id(x), id(y))}{|m_i(e_1) \cup m_i(e_2)|}$$

$$lexSim(x, y) = \begin{cases} 1 & \text{si } isSynonym(x, y) = 1 \\ \frac{levenstein(x,y)+stoilos(x,y)}{2} & \text{si } isSynonym(x, y) = 0 \end{cases} \quad (4.2)$$

$$|m_i(e_1) \cup m_i(e_2)| = |m_i(e_1)| + |m_i(e_2)| - \sum_{x \in m_i(e_1), y \in m_i(e_2)} lexSim(id(x), id(y))$$

donde $m_i(e_1)$ representa el conjunto de elementos que devuelve la característica i-ésima, $|m_i(e_1)|$ representa la cardinalidad de dicho conjunto, *id(x)* devuelve el *fragment* de la entidad x, es decir, su ID, que es el texto que va seguido de # en *rdf:about* (vea Figura 4.2) y la función *isSynonym(x, y)* es la fórmula 4.1, mencionada anteriormente.

– Hiponimia/hiperonimia: para extraer relaciones semánticas de este tipo ($e_1 \sqsubseteq e_2$, para hiponimia; o $e_1 \sqsupseteq e_2$, para hiperonimia); en un mundo ideal, si e_1 es hipónimo

[5] En ningún caso ni *"owl:Thing"*, ni *"owl:topObjectProperty"*, ni *"owl:topDataProperty"* ni *"owl:Nothing"* pertenecerán a estos conjuntos.

de e_2, entonces e_1 debe tener una serie de características[6] coincidentes con e_2 (hiperónimo) [21]. Mientras que en el mundo real [14]:

- e_1 puede tener definida una característica que no tiene e_2.
- e_1 puede no tener definida una característica que tiene e_2.
- e_1 puede tener definidas todas las características que tiene e_2 e incluso alguna más, pero no ser un hipónimo de e_2.

Dentro de una misma ontología, si $e_1 \sqsubseteq e_2$ entonces $e_2 \sqsupseteq e_1$, esto es la hiponimia implicaría hiperonimia en sentido contrario entre las dos mismas entidades. Sin embargo, cuando se está comparando dos ontologías diferentes considerar esta implicación supondría asumir que ambas entidades se encuentran definidas en ambas ontologías, lo cual puede no ser cierto. Para evitar este problema se nos plantean dos alternativas: realizar el mismo trabajo de O_1 hacia O_2 y de O_2 hacia O_1, lo cual duplicaría el tiempo de computación, o bien, las fórmulas relacionadas con hiponimia realizarlas en los dos sentidos posibles, que es la opción elegida.

Para extraer este tipo de relaciones semánticas utilizaremos la siguiente fórmula:

$$simHyponomy(e_1, e_2) = \frac{\sum_{x \in m_i(e_1), y \in m_i(e_2)} lexSim(id(x), id(y))}{|m_i(e_1)|}$$

donde *simHyponomy(e_1, e_2)* devuelve la proporción de características que e_1 comparte con e_2, *lexSim(id(x),id(y))* es la fórmula 4.2 y $|m_i(e_1)|$ representa la cardinalidad del conjunto de características totales de e_1.

Y, debido a la relación transitiva ($e_1 \sqsubseteq e_2 \wedge e_2 \sqsubseteq e_3 \rightarrow e_1 \sqsubseteq e_3$), hay que tener en cuenta a las sub-entidades (co-hipónimos) de e_2, ya que pueden tener características que tiene e_1[21]. Para calcular la proporción se utilizará la siguiente fórmula[7]:

$$simCohyponyms(e_1, e_2) = \frac{\sum_{i \in childs(e_2)} \left(\frac{simHyponomy(e_1, i) + simHyponomy(i, e_1)}{2} \right)}{|childs(e_2)|}$$

donde $childs(e_2)$ representa el conjunto de co-hipónimos de e_2, i representa al co-hipónimo i-ésimo y $|childs(e_2)|$ indica el número de co-hipónimos que tiene e_2.

Introducidas las distintas fórmulas a utilizar para la extracción de relaciones semánticas, sobre la base de un ejemplo de ontología[8] (vea Figura 4.3), se exponen

[6] En el caso de las clases e_1 debe tener todas las propiedades de e_2; y en el caso de las propiedades, todas las clases que conforman el dominio y rango de e_2.

[7] Tanto simHyponomy como simCohyponyms se aplica en dos direcciones, del mismo modo que se ha explicado para isAffix.

[8] https://protege.stanford.edu/ontologies/pizza/pizza.owl

las características que se ha considerado pertinente utilizar, a partir de [6, 9], así como las métricas, mostrando las salidas que surgen del esquema, y fórmulas a aplicar a cada una de ellas.

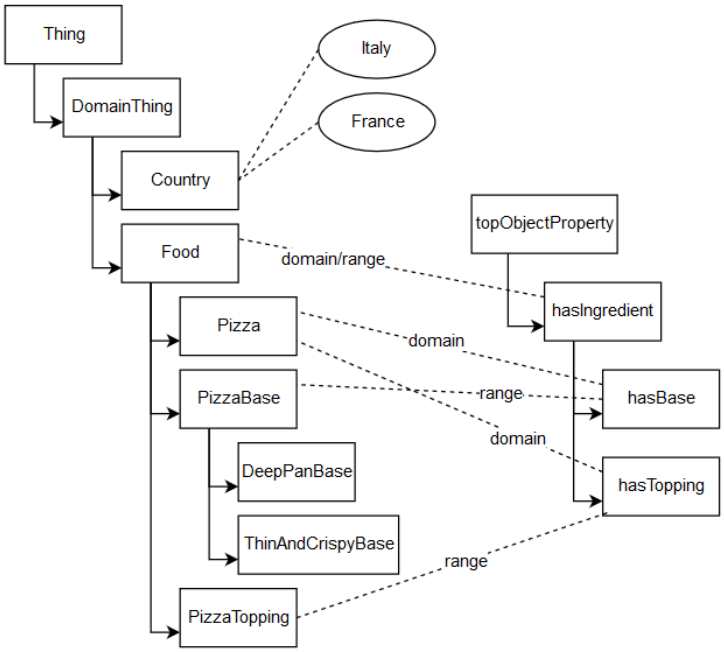


Figura 4.3: Parte de la ontología *Pizza*

4.3.2.1. Características específicas de las clases

Para evaluar el grado de similitud entre clases, se extrae el conjunto de cadenas que verifican cada una de las siguientes características para las clases a comparar de cada ontología:

- Equivalent Classes: devuelve el conjunto de clases definidas en su ontología respectiva como equivalentes. Dentro de la ontología está definido en la clase como *owl:sameClassAs*.

- Direct Properties: devuelve el conjunto de propiedades cuyo dominio directo es la clase en comparación, es decir, indica que esa propiedad es requerida para especificar la clase. Está definido en las propiedades como *rdfs:domain*.

- Ultimate Properties: devuelve el conjunto de propiedades que son hoja de la clase (no tienen sub propiedades) y que tienen definida a ésta como dominio directo o heredado. Está definido en las propiedades y en las propiedades de sus ancestros como *rdfs:domain*.

– Properties: devuelve las propiedades que tienen a la clase en cuestión como dominio directo o heredado de su padre, sin la restricción de ser hoja. Está definido en las propiedades y en las propiedades de sus ancestros como *rdfs:domain*.

– Direct Super Classes: devuelve las clases padre de la comparada. Está definido en la clase como *rdfs:subClassOf*.

– Super Classes: devuelve las clases padre y ancestros. Está definido en la clase y ancestros como *rdfs:subClassOf*.

– Direct Sub Classes: devuelve las clases hijo. Está definido en los hijos como *rdfs:subClassOf*.

– Sub Classes: devuelve las clases hijo y descendientes. Está definido en los hijos y descendientes como *rdfs:subClassOf*.

– Ultimate Sub Classes: devuelve las clases hijo y descendientes que son hoja. Está definido en los hijos y descendientes como *rdfs:subClassOf*, incorporando la restricción de no tener sub clases.

– Direct Super Class Axioms: devuelve las propiedades que forman un axioma que representa a la clase como padre. En la Figura 4.4a se puede apreciar como se representa en *OWL*. Mientras que en la Figura 4.4b ofrece una representación más visual (recuadro azul).

– Super Class Axioms: devuelve las propiedades que forman un axioma que representa una superClass y sus herederos. Al igual que la característica Direct Super Class Axioms está representada en las Figuras 4.4a y 4.4b.

– Direct Individuals: devuelve los individuos de la clase. Se define en los individuos como *rdf:type*.

– Individuals: devuelve los individuos de la clase y los que son de alguna de sus sub clases. Está definido en los individuos y en los individuos de sus descendientes como *rdf:type*.

```
<owl:Class rdf:about="http://www.co-ode.org/ontologies/pizza/pizza.owl
#Pizza">
    <rdfs:subClassOf rdf:resource="http://www.co-ode.org/ontologies/
    pizza/pizza.owl#Food"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.co-ode.org/
            ontologies/pizza/pizza.owl#hasBase"/>
            <owl:someValuesFrom rdf:resource="http://www.co-ode.org/
            ontologies/pizza/pizza.owl#PizzaBase"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:label xml:lang="en">Pizza</rdfs:label>
</owl:Class>
```

(a) Clase *Pizza* en *OWL*

(b) Super clases de *Pizza* en *Protégé*

Figura 4.4: Representación de un super class axiom para la clase *Pizza*

Los conjuntos de cadenas que dependen de propiedades (*Direct Properties, Ultimate Properties, Properties, Direct Super Class Axioms y Super Class Axioms*) se extraen a partir de las relaciones existentes en los ficheros de alineamiento[9]; las reales en el entrenamiento (vea Sección 4.4), y las relaciones obtenidas, en la validación (vea Sección 4.5).

En este caso, para calcular el grado de similitud, se utiliza una variante de la fórmula 4.2:

$$lexSim(x, y, C) = \begin{cases} 1 & \text{si } detectSynonym(x, y, C) = 1 \\ 0 & \text{si } detectSynonym(x, y, C) = 0 \end{cases}$$

donde *detectSynonym(x, y, C)* indica si la pareja de propiedades comparada está identificada como sinonimia en el conjunto C, formado por las relaciones obtenidas a partir de los ficheros de alineamiento o de las relaciones obtenidas en la fase de validación, según el caso.

Para un mejor entendimiento, se expone un ejemplo de las características que se acaban de definir en la Tabla 4.2, a partir de la Figura 4.3.

[9]En estos ficheros, llamados *"refalign.rdf"* en los datasets considerados en la Sección 4.4.1, se encuentran las relaciones reales que hay entre dos ontologías. Se utilizan para comprobar el nivel de acierto del sistema entre las dos ontologías comparadas.

Característica	Ejemplo	Métricas
Equivalent Classes	m_i(Pizza) = {}	sim
Direct Properties	m_i(Pizza) = {hasBase, hasTopping}	sim
Ultimate Properties	m_i(Pizza) = {hasBase, hasTopping}	sim
Properties	m_i(Pizza) = {hasIngredient, hasBase, hasTopping}	sim, simHyponomy y simCohyponyms
Direct Super Classes	m_i(Pizza) = {Food}	sim
Super Classes	m_i(Pizza) = {DomainThing, Food}	sim
Direct Sub Classes	m_i(Food) = {PizzaBase, Pizza, PizzaTopping}	sim
Sub Classes	m_i(Food) = {PizzaBase, ThinAndCrispyPizza, PizzaTopping, Pizza, DeepPanBase}	sim
Ultimate Sub Classes	m_i(Food) = {Pizza, ThinAndCrispyPizza, PizzaTopping, DeepPanBase}	sim
Direct Super Class Axioms	m_i(Pizza) = {hasBase}	sim
Super Class Axioms	m_i(Pizza) = {hasBase}	sim, simHyponomy y simCohyponyms
Direct Individuals	$m_i(Country)$ = {Italy, France}	sim
Individuals	$m_i(Country)$ = {Italy, France}	sim

Tabla 4.2: Características para detectar relaciones en las clases

4.3.2.2. Características específicas de las propiedades

Con respecto a las propiedades (existen dos tipos *Object Properties* y *Data Properties*), para cada par de propiedades en comparación, se analizan las siguientes características:

- Equivalent Properties: devuelve el conjunto de propiedades equivalentes a las comparadas, en sus respectivas ontologías. Está definido en cada propiedad como *owl:samePropertyAs*.

- Direct Domains: devuelve el conjunto de clases que conforman el dominio directo de cada propiedad. Está definido como *rdfs:domain*.

– Domains: devuelve el conjunto de clases que conforman el dominio de la propiedad y sus super clases.

– Direct Ranges: devuelve las clases (si es de tipo *Object Property*) o tipo de dato (si es de tipo *Data Property*) que conforman el rango directo de la propiedad. Está definido como *rdfs:range*.

– Ranges[10]: devuelve las clases que conforman el rango y sus super clases.

– Direct Super Properties: devuelve las propiedades padre dy está definido como *rdfs:subPropertyOf*.

– Super Properties: devuelve las propiedades padre y ancestros. Está definido en la propiedad y ancestros como *rdfs:subPropertyOf*.

– Direct Sub Properties: devuelve las propiedades hijo. Está definido en los hijos como *rdfs:subPropertyOf*.

– Sub Properties: devuelve las propiedades hijo y descendientes. Está definido en los hijos y descendientes como *rdfs:subPropertyOf*.

De la misma manera que con las clases, se presenta un ejemplo de las características que se acaban de definir en la Tabla 4.3, a partir de la Figura 4.3.

Característica	Ejemplo	Métricas
Equivalent Properties		sim
Direct Domains	m_i(hasBase) = {Pizza}	sim
Domains	m_i(hasBase) = {DomainThing, Food, Pizza}	sim, simHyponomy y simCohyponyms
Direct Ranges	m_i(hasBase) = {PizzaBase}	sim, simHyponomy y simCohyponyms
Ranges	m_i(hasBase) = {DomainThing, PizzaBase, Food}	sim, simHyponomy y simCohyponyms
Direct Super Properties	m_i(hasBase) = {hasIngredient}	sim
Super Properties	m_i(hasBase) = {hasIngredient}	sim
Direct Sub Properties	m_i(hasIngredient) = {hasBase, hasTopping}	sim
Sub Properties	m_i(hasIngredient) = {hasBase, hasTopping}	sim

Tabla 4.3: Características para detectar relaciones en las propiedades

[10]Únicamente se calcula para las *Object Properties*

4.3.2.3. Características específicas de los individuos

De forma análoga, para cada par de individuos en comparación, las características analizadas son las siguientes:

- Equivalent Individuals: devuelve el conjunto de individuos equivalentes a los comparados. Está definido como *owl:sameAs*.
- Direct Classes: devuelve las clases directas a las que pertenece el individuo. Está definido como *rdf:type*.
- Classes: devuelve el conjunto de clases directas y super clases a las que pertenece el individuo.
- Properties: devuelve el conjunto de propiedades que tiene cada individuo en comparación. Está definido como el dominio de sus clases directas.

De nuevo, esta vez para los individuos, se muestra un ejemplo de las características que se acaban de definir en la Tabla 4.4, a partir de la Figura 4.3.

Característica	Ejemplo	Métricas
Equivalent Individuals	m_i(Italy) = {}	sim
Direct Classes	m_i(Italy) = {Country}	sim
Classes	m_i(Italy) = {DomainThing, Country}	sim
Properties	m_i(Italy) = {}	sim

Tabla 4.4: Características para detectar relaciones en los individuos

4.4. Entrenamiento de los modelos

Se han aplicado dos modelos de aprendizaje: *Redes Neuronales* y *Random Forests*. Como se ha comentado en la Sección 4.1, se necesita un modelo para cada tipo de entidad, es decir, se necesitarán cuatro redes neuronales y cuatro *Random Forests*. Para crear los distintos modelos, se han seguido los siguientes pasos:

1. Creación de los conjuntos de entrenamiento y test.
2. Establecimiento de un umbral de filtrado.

4.4.1. Creación de los conjuntos de entrenamiento y test

El entrenamiento de los modelos requiere crear un dataset (en este caso cuatro, uno para cada modelo, para que funcionen correctamente). Estos datasets

han sido establecidos a partir de las comparaciones entre distintas ontologías, extraídas de *Ontology Alignment Evaluation Initiative*[11] *(OAEI)*, utilizando todas las características, comunes y no comunes, que se han explicado en la Sección 4.3. Únicamente se han utilizado ontologías de esta página Web ya que dispone de alineamientos de referencia que permiten crear datasets con salidas conocidas y, posteriormente, evaluar los resultados obtenidos. En concreto, para el entrenamiento, se utilizó la información proporcionada bajo la propuesta "oriented matching"de 2009, en la que a partir de las ontologías que utilizaron en 2006, se introducen resultados de hiponimia e hiperonimia a los de sinonimia ya incluidos desde ese año. En la fase de evaluación, también se hizo uso de la misma propuesta del año 2011 (ver Capítulo 6). En años posteriores los ejercicios planteados se dirigían más a la detección de sinonimia en grandes ontologías y a probar diferentes herramientas, centrándose únicamente en las relaciones entre clases, sin proporcionar alineamiento de propiedades ni abordar la comparación entre individuos.

Puesto que, tanto las redes neuronales como los Random Forests, son técnicas supervisadas, es decir, se dispone datos de entrada y de salida, se tiene que asignar una salida a cada entrada del dataset. Para codificar estas salidas, se utiliza un fichero de alineamiento que contiene las relaciones reales entre las ontologías comparadas. El código asignado a las salidas depende del algoritmo utilizado (vea Tabla 4.5).

Salida	Neural Network	Random Forest
Synonym	$[1, 0, 0]$	synonymy
Hyponym	$[0, 1, 0]$	hyponymy
Hypernym	$[0, 0, 1]$	hypernymy
No relation	$[0, 0, 0]$	no relation

Tabla 4.5: Tipos de salida en función del algoritmo de aprendizaje

Como limitaciones, derivadas de la fuente de ontologías utilizadas, no se ha podido crear un dataset para individuos, debido a que no se contemplan relaciones entre ellos y, por otra parte, los datasets de propiedades no contemplan relaciones de hiponimia/hiperonimia.

La creación de los conjuntos de entrenamiento y test, se realiza de distinta manera según el algoritmo de aprendizaje.

4.4.1.1. Redes neuronales

El conjunto de entrenamiento se crea a partir del 80 % del dataset, y el conjunto de test a partir del 20 %. No se ha utilizado *cross-validation* para entrenar el modelo,

[11] http://oaei.ontologymatching.org

debido a que la versión utilizada de Neuroph Studio, que es la única que funciona en Android, no dispone de esta función.

La red neuronal se entrena a partir del conjunto de entrenamiento, generando unos pesos que su aplicación indicará si tiende o no a la existencia de relación. En caso afirmativo, también indica el tipo de relación. Se utiliza *backpropagation with momentum* como algoritmo de entrenamiento y con los parámetros *learning rate* y *momentum* asignados a 0.2 y 0.7, respectivamente.

En un principio se consideró la posibilidad de utilizar una red neuronal con cuatro posibles salidas, una para cada tipo de resultado (Tabla 4.5). Sin embargo, debido a que la salida *No relation* carecía de interés para el proyecto, y además se podía asumir por defecto, fijados unos umbrales adecuados (Sección 4.6) se desechó la idea. Y como los datasets no se encuentran balanceados y existe una mayor presencia de *No relations* las predicciones tienden hacia ese resultado, aconsejando que la estructura de las redes no sea compleja, porque puede suponer que haya sobre entrenamiento, que además podría ser erróneo.

Las distintas redes neuronales, presentan la estructura de la Tabla 4.6:

Tipo de entidad	Entradas	Neuronas (1 capa interna)	Salidas
Classes	37	3	3
Object Properties	33	3	3
Data Properties	32	3	3

Tabla 4.6: Estructura de las redes neuronales

Una vez entrenada la red neuronal, como producto del conjunto de test, devuelve para cada par en comparación una estimación de la similitud encontrada para cada tipo de relación. Esta estimación se utiliza para calcular el umbral a partir del cual se estima relevante el tipo de relación.

4.4.1.2. Random Forest

En este caso, se utiliza cross-validation para entrenar el modelo. La estructura del Random Forest está compuesta por 500 árboles y el dataset se ha dividido en 5 partes.

Una vez entrenados los 5 modelos, se selecciona el mejor. El criterio utilizado, para elegir el mejor es el que maximiza el valor de *F-measure*. Se decide utilizar este valor, que es una combinación de la *precision* y el *recall* (fórmula 4.3), ya que primar la precision (maximizar el porcentaje de aciertos sobre detectados) provoca una reducción en exceso del número de relaciones detectadas admisibles con la consiguiente disminución del

recall (número de relaciones detectadas que son reales/correctas), y al contrario.

$$F - measure = 2 * \frac{precision * recall}{precision + recall} \tag{4.3}$$

A la hora de calcular dicho valor, únicamente se tiene en cuenta de los datos de test, aquellas relaciones que han sido detectados como relación y/o son relación, es decir, no se tiene en cuenta la precision y el recall de las comparaciones que no tienen relación. Se ha decidido calcular así, para primar la correcta identificación de relaciones semánticas, ya que si se utilizaran las comparaciones que no tienen relación el cálculo saldría descompensado, porque el dataset no está balanceado: apenas el 10 % de los datos representan relaciones. Se intentó balancear el dataset utilizando diferentes técnicas como *oversampling, undersampling o SMOTE*, pero aparentemente tendían a producir sobre entrenamiento al eliminar o duplicar información relevante.

Por último, a diferencia que con las redes neuronales, no se necesita calcular un umbral ya que devuelve el tipo de relación directamente. Cuando se utiliza el algoritmo Random Forest el sistema presenta un pequeño cambio respecto a la estructura presentada en la Figura 4.1, no necesita el componente *"filtering of relationship"*.

4.4.2. Establecimiento de un umbral de filtrado

Como se ha comentado, este paso a la hora de entrenar un modelo, únicamente se utiliza en el caso de que el algoritmo de aprendizaje elegido sea una red neuronal.

Una vez, introducidos los datos de test en la red neuronal, se utilizan las salidas que genera para calcular un umbral, a partir del cual el tipo de relación identificada se considera susceptible de resultar relevante.

Para cada tipo de relación, el umbral se calcula de la siguiente manera:

- Se consideran todas las salidas de una columna o tipo de relación[12].
- Las relaciones identificadas distintas a la de la columna seleccionada, se consideran como ausencia de relación porque son resultados negativos respecto a la columna elegida.
- Se calcula a partir de qué valor de similitud se maximiza el valor de *F-measure* para los datos de test. Al igual que se ha comentado en el Random Forest, existe el problema de que el dataset no está balanceado, con lo que tampoco se tendrá en cuenta aquellas comparaciones que no tienen relación.

Los umbrales obtenidos son los siquientes:

[12]Existen tres columnas que indican si hay sinonimia, hiponimia o hiperonimia.

Tipo de entidad	Sinonimia	Hiponimia	hiperonimia
Classes	0.24	0.24	0.21
Object Properties	0.41	-	-
Data Properties	0.39	-	-

Tabla 4.7: Umbrales de filtrado

Cabe señalar que, el hecho de que los umbrales obtenidos se sitúen como máximo en torno a 0.4 parece indicar que el sistema diseñado distingue bastante claramente la posible existencia de relación respecto a la ausencia de la misma.

4.5. Extracción de candidatos de relaciones semánticas

Una vez se han entrenado los modelos, se pasa a la fase de validación, en la que se va a comprobar la bondad de los modelos generados en la sección 4.4.

Para ello, se calcula el vector de comparaciones entre dos ontologías que no se hayan incluido en la fase de entrenamiento (vea Sección 4.3) de la misma forma que se generaron los de los datasets de los modelos. Estos vectores, se introducen en los modelos ya entrenados, generando una salida. En el caso de que la comparación se haya producido entre propiedades, tanto Object Properties como Data Properties, se guarda si se ha detectado como relación. Previo paso por el filtrado de relaciones (vea Sección 4.6) en el caso de utilizar una red neuronal.

Las comparaciones guardadas se utilizan para calcular la similitud entre las características que dependen de propiedades entre las clases (vea Sección 4.3.2.1).

4.6. Filtrado de relaciones

Después de haber calculado el umbral con los datos de test (vea Sección 4.4.2) y de haber realizado la comparación de cada característica entre dos entidades (vea Sección 4.5), se procede a indicar si dicha comparación se descarta o no.

Para admitir una relación entre dos entidades, se siguen los siguientes pasos:

1. Se introduce el vector de comparaciones en la red neuronal que se ha generado en la sección 4.4.1, que devuelve una salida.

2. La salida generada tiene 3 componentes (synonymy, hyponymy, hypernymy), y se selecciona la de mayor valor.

3. Si el máximo seleccionado supera el umbral que tiene asignado esa componente, se guarda el resultado obtenido y, en el caso contrario se clasifica esa comparación como “nothing”, es decir, no hay relación.

Capítulo 5

Implementación multilingüe

El sistema no funciona únicamente en inglés, también funciona en otros lenguajes como por ejemplo el francés.

5.1. Extracción de traducciones

El primer paso, a la hora de traducir términos es elegir uno o varios traductores. En este caso, se han elegido dos (vea sección 2.3). En un primer momento, el principal punto que se tuvo en cuenta a la hora de elegir los traductores fue que devolviera una lista con las diferentes traducciones posibles, debido a la posible falta de contexto para elegir la traducción. Es por eso, que se escogió *ConceptNet*[1].

ConceptNet devuelve correctamente las traducciones de términos formados por una palabra (por ejemplo, banco = {bank, bench}). Se descartó utilizar *BabelNet*[2], aunque funciona mejor que *ConceptNet*, debido a que los recursos gratuitos que ofrecen son muy limitados. Apenas se podría ejecutar una vez la extracción de relaciones semánticas e incluso ni una dependiendo del tamaño de la ontología. Puesto que *ConceptNet* únicamente devuelve correctamente términos de una palabra, se decidió añadir otro traductor *Yandex*[3].

Yandex a diferencia de *ConceptNet* únicamente devuelve una traducción, pero traduce correctamente términos con más de una palabra. En el caso de términos con más de una palabra no se suele necesitar varias traducciones porque ese conjunto de palabras añade contexto.

[1] http://conceptnet.io/
[2] https://babelnet.org/
[3] https://translate.yandex.com/

5.2. Proceso

Para que el sistema funcione con otros lenguajes, se deben seguir los siguientes pasos:

1. Traducir los términos a inglés. Aunque, los elementos a comparar estén en un mismo idioma distinto al inglés, se traducen siempre al inglés, aunque suponga un incremento del coste de ejecución y pudiera ser que se perdiera información[4]. Se ha tomado esta decisión, debido a que WordNet únicamente funciona en inglés y si no se tradujera a este idioma, no se podría utilizar este recurso.

2. Almacenar el vector obtenido (una o varias componentes, según el traductor) en una tabla *Hash* para evitar la búsqueda del mismo término, puesto que la petición a los traductores, conlleva un tiempo muy costoso. Por ejemplo, se realizó una prueba con dos ontologías[5]: cuando se evitaba la repetición de búsquedas, la comparación costó alrededor de 5 minutos; mientras que, si no se evitaba la repetición, costó más de 30 minutos.

3. En último lugar, una vez obtenidas las traducciones de los términos a comparar, se ha decidido utilizar una solución optimista. Es decir, utilizando la fórmula 4.2 se ha decidido quedarse con el mejor resultado devuelto (por ejemplo, si se compara *camión* con *truck*, la fórmula 4.2 devuelve 1, porque una de las posibles traducciones de camión es *truck*).

5.3. Problemas

Los principales problemas de la implementación multilingüe, tal como se plantea en el proyecto, son los siguientes:

- Dependencia de traductores externos
- Ausencia del atributo *xml:lang* en las entidades

Dependencia de traductores externos

El correcto funcionamiento del sistema multilingüe depende de la disponibilidad del idioma en el que está definida la ontología en los traductores utilizados, y la completitud de los mismos. Debido a que los traductores utilizados son fuentes externas, no se puede

[4]No se sabe la completitud de los distintos traductores utilizados
[5]Ontologías 101 (inglés) y 207 (francés) de http://oaei.ontologymatching.org/2006/

garantizar que devuelvan todas las traducciones posibles, ni que funcionase con otros posibles idiomas.

Ausencia del atributo *xml:lang* en las entidades

Todas las entidades de una ontología pueden tener declarado un atributo que indica el idioma (*xml:lang*) de dicha entidad. Este problema puede ser solucionado utilizando una petición *http* (*detect*) de *Yandex*, que detecta el idioma. Pero, tiene un problema, el coste temporal de la petición (1 segundo). El coste individual, puede no ser muy grande, pero en el peor de los casos, habría que ejecutar esta petición para cada una de las entidades. Por ejemplo, se efectuó una prueba entre dos ontologías[6] escritas en inglés: sin la llamada de detección, el proceso no llega a los 2 minutos; con la llamada, casi una hora. En el caso de ontologías en idiomas diferentes al inglés, este problema se acentúa, ya que además de este coste, se le tiene que añadir el tiempo de traducción. Por este motivo, se ha decidido que aquellas entidades que no tienen declarado este atributo, se asume por defecto que están escritas en inglés, aunque no sea cierto.

[6]Ontologías 101 y 103 de http://oaei.ontologymatching.org/2006/benchmarks/

Capítulo 6

Evaluación

Una vez diseñado el sistema se ha procedido a comprobar su funcionamiento mediante la alineación de diversas ontologías, procedentes de la misma fuente que los datos de entrenamiento[1], no incluidas en los datasets utilizados en el diseño, y para las cuales se disponía de resultados, para evaluar la fiabilidad del sistema implementado.

Para el caso de las clases el sistema proporciona como salida tanto sinónimos como hipónimos e hiperónimos. Los resultados obtenidos se muestran en las Tablas 6.1 a 6.3.

Ontologías	Synonyms						
		Red Neuronal			Random Forest		
	Nº Rel.	Prec.	Recall	F1 score	Prec.	Recall	F1 score
101 - 207	33	0.94	0.91	0.92	1	0.97	0.98
101 - 223	33	0.86	0.97	0.91	0.91	0.97	0.94
101 - 224	33	1	1	1	1	1	1
101 - 231	33	1	1	1	1	1	1
101 - 238	33	0.86	0.97	0.91	0.91	0.97	0.94
	media	0.93	0.97	0.95	0.97	0.98	0.97
	dv. est.	0.07	0.04	0.05	0.05	0.02	0.03

Tabla 6.1: Resultados de sinonimia en las clases

donde *Nº Rel.* indica el número de relaciones de tipo sinonimia existentes, *Prec.* la precisión del modelo y *dv. est.* es la desviación estándar.

A partir de los datos se puede concluir que el sistema detecta la relación de sinonimia con un grado de fiabilidad elevado. Los resultados más bajos se producen cuando la ontología comparada está dotada de una jerarquía expandida, introduciendo un mayor número de clases intermedias, es el caso de las ontologías 223 y 238.

[1]Los sinónimos se han extraído de http://oaei.ontologymatching.org/2006/benchmarks/ y la hiponimia e hiperonimia de http://oaei.ontologymatching.org/2009/oriented/ como se comenta en la Sección 4.4.1.

Ontologías	Hyponyms				Hypernyms			
	Nº Rel.	Prec.	Recall	F1 score	Nº rel.	Prec.	Recall	F1 score
101 - 207	25	1	0.88	0.94	25	0.81	0.84	0.82
101 - 223	27	0.47	0.59	0.52	41	0.58	0.44	0.50
101 - 224	25	1	0.92	0.96	25	1	0.92	0.96
101 - 231	25	1	0.92	0.96	25	1	0.92	0.96
101 - 238	27	0.47	0.59	0.52	41	0.58	0.44	0.50
	media	0.79	0.78	0.78	media	0.79	0.71	0.75
	dv. est.	0.29	0.17	0.23	dv. est.	0.21	0.25	0.23

Tabla 6.2: Resultados de hiponimia/hiperonimia en las clases (red neuronal)

Ontologías	Hyponyms				Hypernyms			
	Nº Rel.	Prec.	Recall	F1 score	Nº rel.	Prec.	Recall	F1 score
101 - 207	25	1	0.36	0.53	25	1	0.24	0.39
101 - 223	27	0.91	0.74	0.82	41	0.86	0.59	0.70
101 - 224	25	1	1	1	25	1	1	1
101 - 231	25	1	1	1	25	1	1	1
101 - 238	27	0.91	0.74	0.82	41	0.86	0.59	0.70
	media	0.96	0.77	0.83	media	0.94	0.68	0.76
	dv. est.	0.05	0.26	0.19	dv. est.	0.08	0.32	0.26

Tabla 6.3: Resultados de hiponimia/hiperonimia en las clases (Random Forest)

Por lo que respecta a las relaciones de hiponimia/hiperonimia, los resultados presentan mayor variabilidad y algunos son más bajos. No obstante, pueden considerarse aceptables. Los peores resultados en red neuronal se obtienen en las comparaciones con las ontologías 223 y 238 que tienen una jerarquía más compleja. Probablemente de haber incorporado en los datasets de entrenamiento resultados procedentes de ontologías más complejas los resultados serían mejores, aunque esto no impide concluir favorablemente sobre la bondad del sistema para detectar las relaciones. En Random Forest el peor resultado se produce al comparar la 207 que está en francés. Pese a estas diferencias, en media, ambos algoritmos presentan un F-score similar.

Las Tablas 6.4 y 6.5 muestran los resultados obtenidos para la extracción de sinonimia entre parejas de propiedades (Object Properties y Data Properties). Se puede observar unos resultados para sinonimia mejorados con respecto a las clases. Esto puede deberse a que las parejas de sinónimos en propiedades muestran mayor tendencia a la coincidencia léxica.

De hecho, la comparación que arroja un resultado ligeramente inferior es la que compara la ontología 101 con la 207 que es la que las etiquetas o ID traducidas al francés.

Ontologías	Synonyms						
	Red Neuronal				Random Forest		
	Nº Rel.	Prec.	Recall	F1 score	Prec.	Recall	F1 score
101 - 207	24	0.91	0.88	0.89	1	0.96	0.98
101 - 223	24	1	1	1	1	1	1
101 - 224	24	1	1	1	1	1	1
101 - 231	24	1	1	1	1	1	1
101 - 238	24	1	1	1	1	1	1
	media	0.98	0.98	0.98	1	0.99	1
	dv. est.	0.04	0.06	0.05	0	0.02	0.01

Tabla 6.4: Resultados de sinonimia en las Object Properties

Ontologías	Synonyms						
	Red Neuronal				Random Forest		
	Nº Rel.	Prec.	Recall	F1 score	Prec.	Recall	F1 score
101 - 207	40	0.91	0.78	0.84	0.91	1	0.95
101 - 223	40	1	1	1	1	1	1
101 - 224	40	1	1	1	1	1	1
101 - 231	40	1	1	1	1	1	1
101 - 238	40	1	1	1	1	1	1
	media	0.98	0.96	0.97	0.98	1	0.99
	dv. est.	0.04	0.1	0.07	0.04	0	0.02

Tabla 6.5: Resultados de sinonimia en las Data Properties

También, se han realizado pruebas con otras ontologías que no pertenecen a la misma fuente que los datos de entrenamiento[2]. Únicamente se han obtenido resultados de sinonimia entre clases (Tabla 6.6). Esto es debido a que estas ontologías carecen de relaciones entre propiedades, cosa que es primordial para la extracción de relaciones de hiponimia e hiperonimia, ya que las funciones que se utilizan para extraer estos tipos de relaciones son escasas, motivo por el cual sí que se encuentran relaciones de sinonimia, y dependen casi en exclusividad de que existan relaciones de sinonimia entre propiedades (Sección 4.3.2.1).

Ontologías	Synonyms			
	Nº Rel.	Prec.	Recall	F1 score
301 - 302	10	0.88	0.7	0.78
301 - 303	10	1	0.5	0.67
302 - 303	7	0.7	1	0.83

Tabla 6.6: Resultados de sinonimia en las clases

[2] http://oaei.ontologymatching.org/2011/oriented/

No se ha podido probar comparaciones con la ontología 304 debido a que salta siempre la siguiente excepción, parece un error sintáctico de la ontología, ya que viola alguna de las conocidas restricciones del lenguaje OWL 2 (o bien es un bug del parseador), al intentar leerla *Exception: java.lang.IllegalArgumentException: Non-simple property '<http://oaei.ontologymatching.org/2009/benchmarks/304/onto.rdf#isPartOf>' or its inverse appears in the cardinality restriction 'ObjectMaxCardinality(1 <http://oaei.ontologymatching.org/2009/benchmarks/304/onto.rdf#isPartOf> owl:Thing)'.*

Capítulo 7

Aplicación móvil

7.1. Problemas en la migración

Una vez comprobado el correcto funcionamiento del sistema en PC, se procede a migrarlo a Android. Para una completa y correcta migración se han encontrado varios problemas, principalmente con librerías:

- Neuroph Studio: uso de una versión modificada, debido a que la forma de almacenamiento de los ficheros que contienen las redes neuronales no se podían leer correctamente.
- HermiT y OWL API: uso de una versión modificada de HermiT, no puede ser convertido directamente a Dalvik, la máquina virtual utilizada en dispositivos Android [22], y cambio de versión de OWL API que sea compatible con esta versión de HermiT[1].
- Lectura de ficheros: la lectura de los ficheros que conforman la base de datos de WordNet y las redes neuronales supuso el mayor problema, ya que es necesario cargar estos ficheros en el móvil para poder acceder a ellos.

7.2. Tiempo de procesamiento

En este apartado se pretende comparar el coste de procesamiento o de ejecución del sistema entre el PC y un dispositivo móvil. Para el experimento, se van a utilizar las ontologías 101 y 224 que se han utilizado en la Sección 6. El PC utilizado es un Asus R510VX-DM010T que tiene un procesador Intel® Core™ i7-6700HQ Quad-Core (6M Cache, 2.6GHz hasta 3.5GHz) y 8GB de memoria RAM. Respecto al móvil, se ha utilizado es un Xiaomi Redmi Note 5 con Android 9 y con Snapdragon 636 Octa Core Kryo a 1,8 Ghz y 4GB de memoria RAM. En la tabla 7.1 se puede apreciar la

[1] http://sid.cps.unizar.es/ANDROIDSEMANTIC/REASONERS/hermit.html

diferencia de tiempo entre PC y móvil, la cual es 22 veces mayor en el móvil respecto al PC.

Ontologías	PC	Móvil
101 - 224	30 segundos	10 - 11 minutos

Tabla 7.1: Tiempo de procesamiento

Este considerable cambio pude ser causado debido a diferentes factores (a parte de por las capacidades del hardware de cada uno):

- Carga en memoria de los ficheros que contienen las redes neuronales y los ficheros necesarios para que funcione WordNet.
- HermiT no está pensado para funcione en Android.

7.3. Diseño de la aplicación

En este apartado se explican los requisitos funcionales y no funcionales, restricciones y la implementación de la aplicación desarrollada.

7.3.1. Requisitos

Código	Descripción
RF-1	El usuario puede elegir dos ontologías a las que se aplicará el proceso
RF-2	El usuario puede visualizar las relaciones de sinonimia detectadas entre clases
RF-3	El usuario puede visualizar las relaciones de hiponimia detectadas entre clases
RF-4	El usuario puede visualizar las relaciones de hiperonimia detectadas entre clases
RF-5	El usuario puede visualizar las relaciones de sinonimia detectadas entre Object Properties
RF-6	El usuario puede visualizar las relaciones de sinonimia detectadas entre Data Properties

Tabla 7.2: Requisitos funcionales

Código	Descripción
RFN-1	Cada relación detectada consta de dos entidades y de grado de confianza

Tabla 7.3: Requisitos no funcionales

7.3.2. Restricciones

La aplicación muestra una serie de restricciones importantes a la hora de utilizarla. Además, alguna de ellas no está presente en el funcionamiento en PC:

- Necesidad de conexión a Internet (*android.permission.INTERNET*) para poder utilizar los traductores. Los traductores utilizan peticiones *HTTP* para convertir a inglés los términos.

- Necesidad de conceder permisos para acceder al almacenamiento local (*android.permission.READ_EXTERNAL_STORAGE*) del dispositivo.

- *SDK 24* mínimo[2].

- Únicamente funcionan las redes neuronales como algoritmo de aprendizaje, ya que Random Forest funciona con R y no se ha conseguido instalar en un móvil Android.

- No se carga en memoria la base de datos de JWI para WordNet, ya que funciona más lento de esa manera, al contrario que en PC (vea Sección 4.3).

- El sistema está preparado para mostrar relaciones de hiponimia e hiperonimia entre propiedades, sean de tipo Object o Data. Sin embargo, las pantallas aparecen vacías debido a que los alineamientos disponibles para crear los conjuntos de entrenamiento de los modelos contemplaban exclusivamente relaciones de sinonimia. Lo mismo sucede con individuos para sinonimia (vea Sección 4.4.1).

7.4. Implementación

En este apartado se va a explicar el funcionamiento de la aplicación a través de sus pantallas, indicando su contenido y cómo llegar a ellas.

En primer lugar, se seleccionan dos ontologías de la carpeta de descargas del dispositivo, pulsando los botones *"SEARCH ONTOLOGY"* (vea Figura 7.1). Al pulsar por primera vez cualquiera de ellos, el sistema pide permiso para poder acceder a los ficheros locales del móvil. También, está contemplado introducir ontologías disponibles en Internet introduciendo su IRI.

Una vez escogidas las ontologías, se pulsa el botón *"play"*, para que comience el proceso de extracción de relaciones.

[2]Se utilizan funciones *lambda* que necesitan tener al menos esa *SDK*

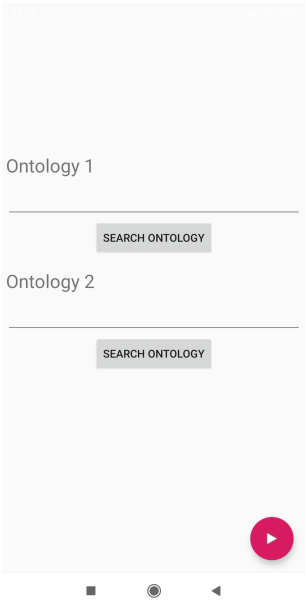
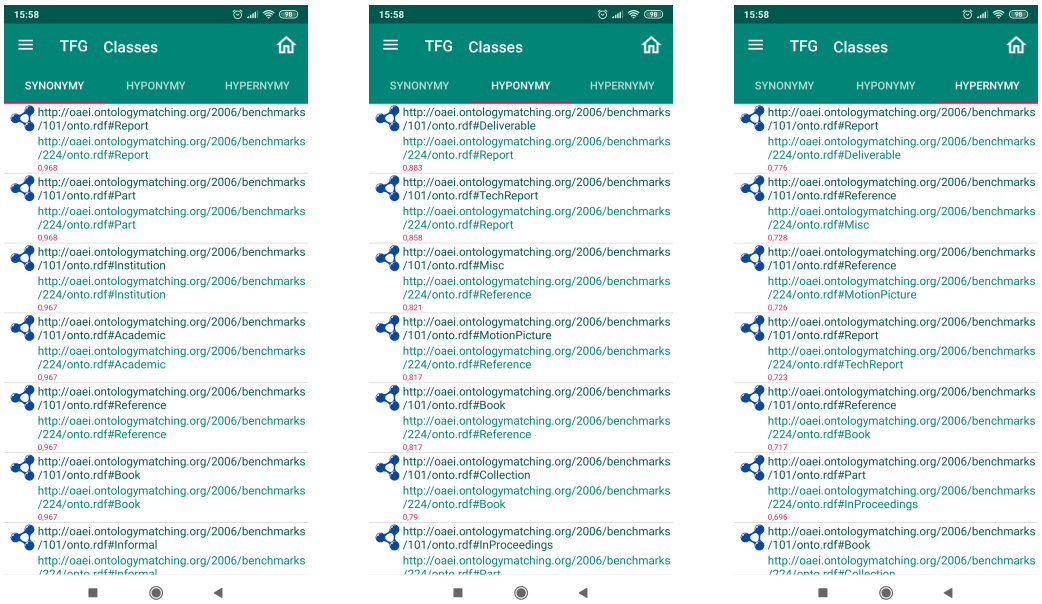


Figura 7.1: Pantalla principal

Una vez termina el proceso de extracción, se muestran las relaciones de sinonimia entre clases (Figura 7.2a) que se han encontrado. Desde esta pantalla se puede acceder a las vistas de hiponimia (Figura 7.2b) e hiperonimia (Figura 7.2c) entre clases pulsando sobre el nombre de las relaciones semánticas en el menú superior.



(a) Sinonimia entre clases (b) Hiponimia entre clases (c) Hiperonimia entre clases

Figura 7.2: Pantallas classes

Además, si se pulsa el icono ≡ se abre un menú (Figura 7.3) a partir del cual se puede acceder a la vista de los otros tipos de entidades.

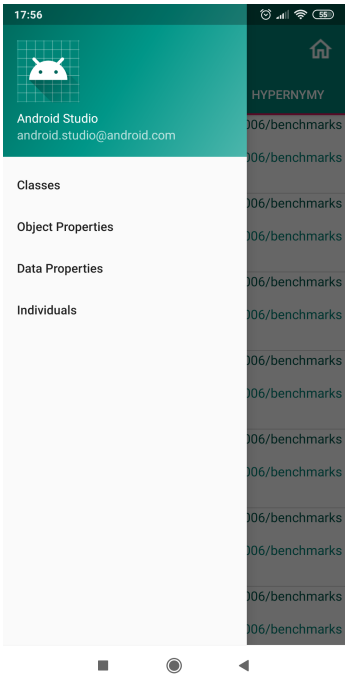
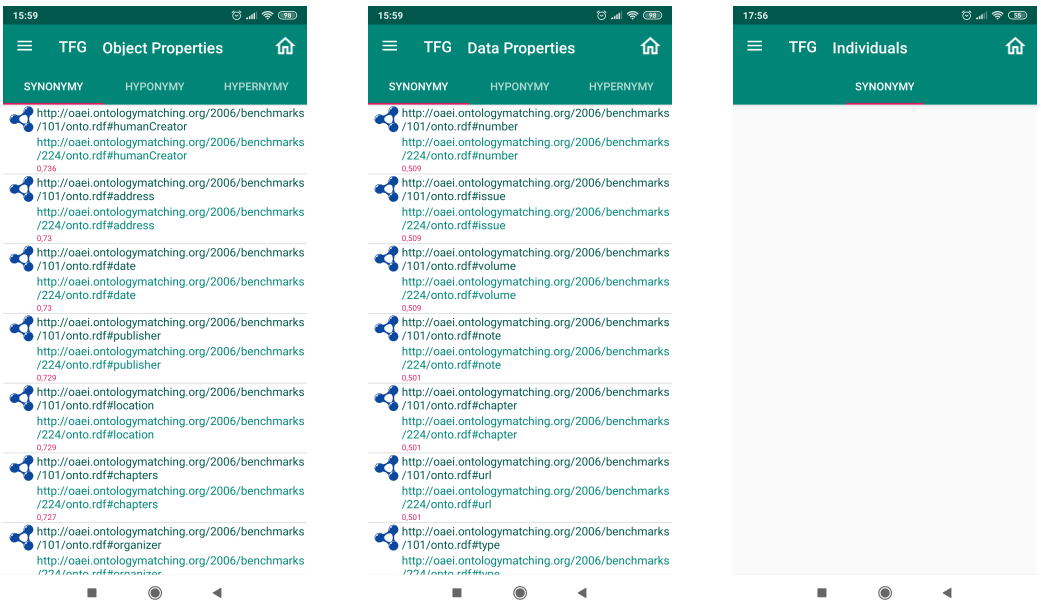


Figura 7.3: Menú de navegación

Según el tipo de entidad elegido, se mostrará la vista con las relaciones de sinonimia entre Object Properties (Figura 7.4a), Data Properties (Figura 7.4b) o individuos (Figura 7.4c).



(a) Sinonimia entre object properties

(b) Sinonimia entre data properties

(c) Sinonimia entre individuos

Figura 7.4: Pantallas Object, Data Properties e individuos

Desde estas vistas se puede acceder a los otros tipos de relaciones. Por último, desde

el icono de la casa de cualquier pantalla, se puede volver a la pantalla principal para volver a ejecutar el proceso con otras dos ontologías (Figura 7.1).

Capítulo 8

Conclusiones

8.1. Conclusiones generales

Este trabajo diseña e implementa un método de alineamiento de ontologías, buscando a la vez hiponimia e hiperonimia, en el que el problema de la ponderación del conjunto de medidas de similitud utilizadas se soluciona mediante el uso de redes neuronales y de Random Forest. El uso de estas técnicas permite eliminar el componente de juicio del investigador que, como se observa en otros trabajos, a menudo se introduce para adaptar los resultados a las salidas esperadas, ya que el sistema se encarga de optimizar los pesos de las distintas variables analizadas.

Por otra parte, la implementación multilingüe (vea Capítulo 5) permite comprobar el funcionamiento del método diseñado incluso aun cuando las dos ontologías que se quiere alinear se encuentren definidas en idiomas diferentes. Y además, tanto en la versión para ordenadores de sobremesa como en la versión para dispositivos móviles Android.

La dificultad para disponer de ontologías que contemplen tanto sinonimia como hiponimia/hiperonimia, ha supuesto un lastre a la hora de entrenar los modelos, ya que con más datos los resultados obtenidos podrían haber sido mejores.

Algunas características, que son fundamentales para extraer estos tipos de relaciones en otros trabajos, no funcionan o no desempeñan bien su labor en este proyecto, utilizando las mismas ontologías. Esto supone que el camino para obtener resultados correctos no es único.

Debido a que el sistema siempre termina realizando comparaciones léxicas entre cadenas, aunque utilice características basadas en la estructura de la ontología (compara conjuntos de cadenas, vea Sección 4.3.2), no funcionará correctamente si alguna de las ontologías no utiliza "ids" con sentido. No obstante, es de esperar que la mayoría de ontologías se encuentren definidas de forma lógica con lo que esta restricción resulta poco probable con ontologías reales.

8.2. Trabajo futuro

A partir de lo aprendido y desarrollado durante el proyecto, se ha llegado a la conclusión de que existen distintas vertientes sobre las que mejorar o investigar:

- Creación de datasets con ontologías que estén pensadas para la extracción de relaciones de hiponimia/hiperonimia entre propiedades. Y también, la extracción de relaciones de sinonimia entre individuos.

- Introducción de nuevas características o métricas para mejorar la extracción de relaciones de hiponimia/hiperonimia entre clases, porque para que muestre buenos resultados entre dos ontologías, las clases deben contener propiedades.

- Investigar si es interesante la comparación entre Object y Data Properties, cuando dos ontologías que han sido escritas por diferentes personas han decidido declarar las propiedades de diferente manera.

- Solucionar el problema de la ausencia del parámetro *xml:lang* (vea Sección 5.3).

- Mejorar el coste temporal del algoritmo de recorrido de ontologías, para que no implique tanto tiempo de ejecución para que muestre resultados en el móvil.

- Identificar aquellas características que no tienen importancia a la hora de determinar si existe o no una relación semántica. Y así, acelerar el coste computacional del sistema. Para ello, se puede utilizar el *índice Gini* (Anexo A.1), que indica la relevancia de las entradas en un Random Forest (cuánto más alto mejor).

8.3. Gestión del proyecto

El desarrollo del trabajo se ha compaginado con el curso y con otras actividades extracurriculares como estudiar inglés en una academia y participar en un equipo de balonmano. La mayor parte del desarrollo del proyecto (vea Tabla 8.1) se divide en dos partes: extracción de relaciones y entrenar los modelos. Estas partes conllevan tanto la búsqueda de ontologías para crear los datasets como la modificación de las características para que los modelos pudieran ser entrenados.

Actividad	Horas
Estudio previo	40
Algoritmo (explorar ontologías)	25
Extracción de relaciones	190
Implementación multilingüe	35
Entrenar modelos	70
Aplicación	20
Documentación	45
Total	435

Tabla 8.1: Horas dedicadas por actividad

Para seguir el proceso de implementación del proyecto (vea Anexo B.1), se parte de un estudio previo sobre ontologías, métricas relevantes y estudio del estado del arte. Durante el proceso, se presentan varios saltos, debido al período de exámenes de enero y a cambios o correcciones de ciertas actividades que se daban por finalizadas.

Bibliografía

[1] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220, June 1993.

[2] Graham Klyne and Jeremy Carroll. Resource description framework (RDF): Concepts and abstract syntax. *World Wide Web Consortium*, 10, January 2006.

[3] Matthew Horridge and Sean Bechhofer. The OWL API: A java API for OWL ontologies. *Semantic Web*, 2:11–21, January 2011.

[4] Jorge Bobed Lisbona. Visualizador de ontologías basado en un razonador de lógica descriptiva. Trabajo fin de carrera, Universidad de Zaragoza, 2012.

[5] Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. Hermit: An OWL 2 reasoner. *Journal of Automated Reasoning*, 53(3):245–269, October 2014.

[6] Jorge Gracia. *Integration and Disambiguation Techniques for Semantic Heterogeneity Reduction on the Web*. PhD thesis, University of Zaragoza, October 2009.

[7] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.

[8] Jordi Bernad and Sergio Ilarri. Métodos no lineales. Apuntes de la asignatura Almacenes y Minería de Datos. Universidad de Zaragoza, curso 2019/20.

[9] Jérôme Euzenat and Pavel Shvaiko. *Ontology Matching*. Springer-Verlag, Berlin, Heidelberg, 2007.

[10] Jorge Gracia, Jordi Bernad, and Eduardo Mena. Ontology matching with CIDER: Evaluation report for OAEI 2011. In *Proc. of 6th Ontology Matching Workshop (OM'11), at 10th International Semantic Web Conference (ISWC'11), Bonn (Germany)*, volume 814. CEUR-WS, ISSN-1613-0073, October 2011.

[11] Marta Sabou, Mathieu d'Aquin, and Enrico Motta. Scarlet: Semantic relation discovery by harvesting online ontologies. January 1970.

[12] Patrick Arnold and Erhard Rahm. Enriching ontology mappings with semantic relations. *Data Knowledge Engineering*, 93:1 -- 18, 2014. Selected Papers from the 17th East--European Conference on Advances in Databases and Information Systems.

[13] Michelle Cheatham and Pascal Hitzler. String similarity metrics for ontology alignment. In *International Semantic Web Conference*, pages 294–309, 2013.

[14] Ignacio Huitzil, Fernando Bobillo, Eduardo Mena, Carlos Bobed, and Jesús Bermúdez. Some reflections on the discovery of hyponyms between ontologies. In *Proceedings of the 21st International Conference on Enterprise Information Systems (ICEIS 2019)*, pages 130–140, 2019.

[15] Alexander Maedche and Steffen Staab. Measuring similarity between ontologies. volume 2473, pages 15–21, October 2002.

[16] Giorgos Stoilos, Giorgos Stamou, and Stefanos Kollias. A string metric for ontology alignment. volume 3729, pages 624–637, November 2005.

[17] William Winkler. The state of record linkage and current research problems. *Statist. Med.*, 14, October 1999.

[18] Raquel Trillo, Jorge Gracia, Mauricio Espinoza, and Eduardo Mena. Discovering the semantics of user keywords. *Journal on Universal Computer Science (JUCS). Special Issue: Ontologies and their Applications, ISSN 0948-695X*, 13(12):1908–1935, December 2007.

[19] Mark Alan Finlayson. Java libraries for accessing the princeton wordnet: Comparison and evaluation. In *In Proceedings of the 7th Global Wordnet Conference*, 2014.

[20] Fernando Bobillo, Carlos Bobed, and Eduardo Mena. On the generalization of the discovery of subsumption relationships to the fuzzy case. In *Proceedings of the 26th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2017), Naples (Italy)*, pages 1–6. IEEE Press, ISBN 978-1-5090-6034-4, July 2017.

[21] Enrique Solano Bes. Descubrimiento de relaciones semánticas en la integración de conocimiento en entornos móviles. Trabajo fin de carrera, Universidad de Zaragoza, 2014.

[22] Marta Lanau Coronas. Sistema inteligente para la optimización del razonamiento semántico en dispositivos móviles. Trabajo fin de grado, Universidad de Zaragoza, 2018.

Lista de Figuras

Lista de Tablas

Anexos

Anexo A

Índice de Gini

El índice de Gini se utiliza en problemas de clasificación porque permite conocer la relevancia de los predictores que forman el modelo. Cuanto más alto es el valor del predictor, más importancia tiene. En la Figura A.1 se puede apreciar que *"coSuperClassAxiomsHyponyms2"*, representa la similitud entre la característica *Super Class Axioms* (vea Sección 4.3.2.1) entre una clase y los co-hipónimos de otra, es el predictor que tiene mayor relevancia. Mientras que *"isAffixLabel2"*, que indica si hay un afijo entre las label de dos clases, es la de menor relevancia.

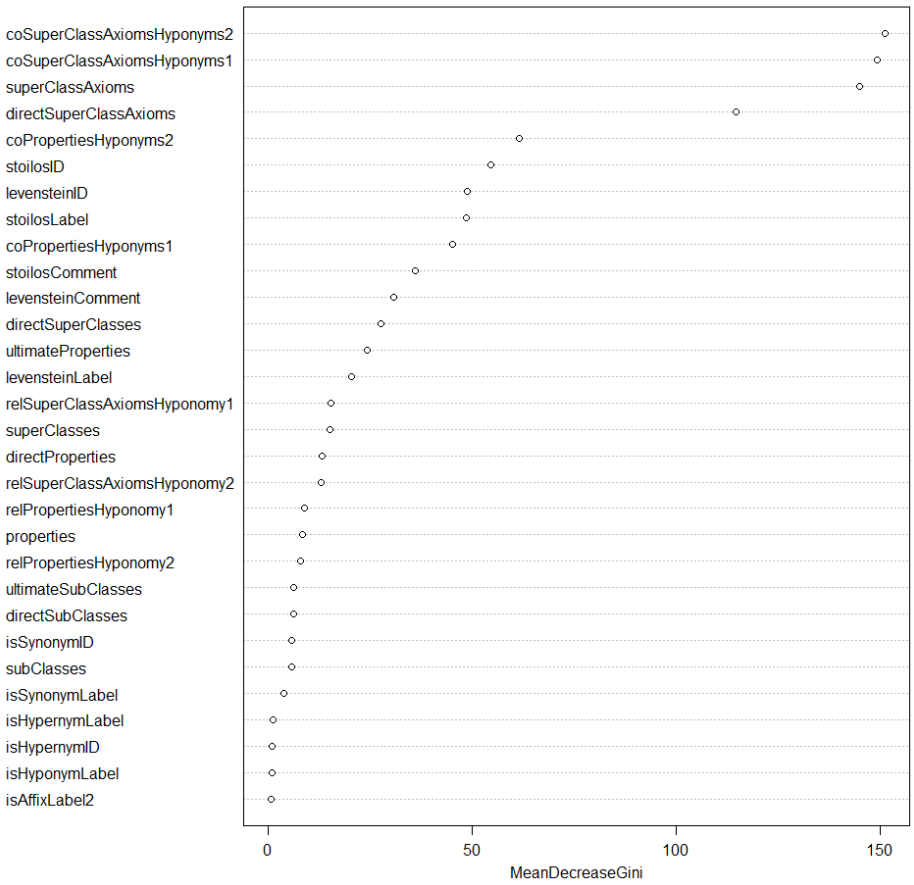


Figura A.1: Índice Gini en el Random Forest generado para clases

Anexo B

Diagrama de Gantt

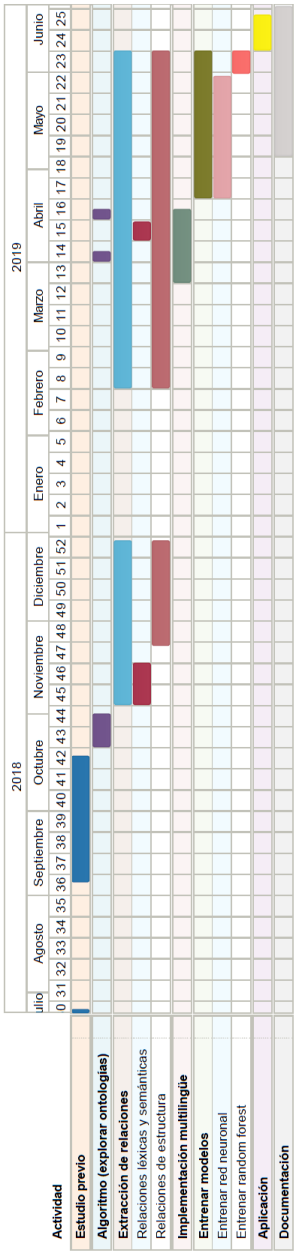


Figura B.1: Diagrama de Gantt del proyecto