Trabajo de Fin de Grado

Grado en Ingeniería Informática

Machine Learning aplicado a terminologías médicas

Machine Learning applied to medical terminologies

Ángel Alberto Hamilton López

La Laguna, 30 de junio de 2018

D. **Dagoberto Castellanos Nieves**, con N.I.F. 79234766-L profesor Contratado Doctor adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor.

D. **Manuel Quesada Martínez**, con N.I.F. 48452813-R profesor Ayudante Doctor adscrito al Departamento de Estadística, Matemáticas e Informática de la Universidad Miguel Hernández, como cotutor.

C E R T I F I C A N

Que la presente memoria titulada:

"Machine Learning aplicado a terminologías médicas"

ha sido realizada bajo su dirección por D. **Ángel Alberto Hamilton López**, con N.I.F. 54111145-B.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 30 de junio de 2018

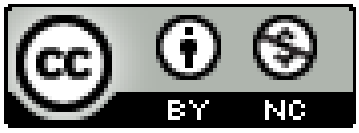
Agradecimientos

A mis padres por haberme apoyado y ayudado en todo momento.

A mis tutores Dagoberto y Manuel por haber tenido una paciencia inhumana conmigo.

A mi novia por asegurarse de que no moría en el intento.

Licencia



Resumen

El objetivo de este trabajo ha sido investigar y evaluar la aplicación de tecnologías de Machine Learning en terminologías médicas.

Las terminologías médicas nacen de la necesidad de mejorar la comunicación entre profesionales del sector y unificar criterios. La terminología sobre la que hemos trabajado en este proyecto es SNOMED CT, la mayor terminología médica del mundo y la más ampliamente utilizada. SNOMED CT contiene tanto lenguaje natural, para que los humanos podamos entenderla, como relaciones lógicas entre conceptos, para que los ordenadores puedan entenderla. Lo ideal sería que ambas formas de representar la información describieran las mismas relaciones entre conceptos, pero no siempre es así.

Las etiquetas del lenguaje natural dentro de SNOMED CT siguen su propia estructura y muchas palabras se repiten. Nosotros pretendemos explotar el lenguaje natural de las clases de SNOMED CT para detectar regularidades léxicas y combinarlas con técnicas de Machine Learning. Esto nos permitirá analizar y visualizar las clases de la ontología desde el punto de vista de las reglas de asociación creadas a partir del contenido expresado en lenguaje natural.

Palabras clave: Aprendizaje automático, SNOMED CT, Terminologías médicas, Ontologías, Regularidades Léxicas.

Abstract

The objective of this project has been to research and evaluate the utility and possible applications of Machine Learning in medical terminologies.

The medical terminologies were born due to the need of improving the communication between professionals and to unify standards. The terminology we have worked on is SNOMED CT, the biggest and most used medical terminology in the world. This ontology contains both natural language for the humans to understand and logical connections between items for the computers to understand. In an ideal scenario the information stored on the ontology should be present in both the language and the logical connections, but it is not always the ideal case.

The labels written in natural language in SNOMED CT follow their own structure and have a lot of repeated words. We pretend to exploit the natural language from the classes of SNOMED CT to detect lexical regularities and combine them with Machine Learning. This will allow us to analyze and visualize the classes of the ontology from the association rules created from the knowledge expressed in the natural language.

Keywords: Machine Learning, SNOMED CT, Medical terminologies, Ontologies, Lexical regularities.

Índice general

Capítulo 1

Introducción

"Toda rama del conocimiento humano o ciencia necesita crear su propia terminología adecuada a sus necesidades de comunicación y expresión. La terminología médica tiene el propósito de expresar en términos precisos los complejos conceptos e ideas del mundo de la medicina. También tiene como propósito la unificación de criterios. Cada término debe poseer un significado único aceptado por la comunidad científica, facilitando, así, el intercambio de información a nivel internacional."[30]

Para responder a la necesidad de una terminología estándar internacional que permitiera comunicarse de forma precisa con profesionales de cualquier parte del mundo se creó SNOMED CT, una terminología con alcance global y usada (entre otros) por el gobierno de España para integrarla en el sistema de Historia Clínica Digital del Sistema Nacional de Salud (HCDSNS).

Antecedentes y estado actual de tema

La terminología de SNOMED CT se constituye, de forma básica, por conceptos, descripciones y relaciones [9]. Esto nos permite la representación con precisión de los datos y el conocimiento clínico en el ámbito sanitario. Además, nos facilita la codificación, la recuperación, la comunicación y el análisis de los datos sanitarios de forma precisa e inequívoca. Las historias clínicas tanto en el ámbito nacional como internacional representadas con SNOMED CT pueden procesarse y presentarse en diversas formas para facilitar la atención directa de los pacientes, la investigación, la epidemiología, o la planificación de los servicios. Sus innumerables ventajas avalan su uso en más de 50 países.

Para facilitar la automatización y el desarrollo de aplicaciones relacionadas con el sector biosanitario, SNOMED CT también es distribuida como una ontología de la cual se puede extraer toda la información necesaria sobre la terminología. Esta ontología trata de representar mediante axiomas lógicos entre clases las relaciones que para los humanos son obvias en el lenguaje

natural. Sin embargo, debido a su tamaño es complicado supervisar que todas las relaciones existan y sean correctas. Para solucionar este problema planteamos el uso de tecnologías de Machine Learning (“aprendizaje automático” en español) que faciliten la detección de reglas de asociación entre diferentes conceptos/clases de la ontología que exploten el contenido codificado en lenguaje natural. Estas reglas podrían ayudar a detectar patrones que permitan corregir o enriquecer la ontología desde el punto de vista semántico.

En el ámbito del proyecto diversos autores han presentado propuestas para mejorar el tanto el uso clínico de SNOMED, como la calidad de la información clínica o el incremento de la información contenida. En [10] se presentan nuevas medidas de similitud semántica y se evalúan con la ontología biomédica estándar SNOMED. Bona y Ceusters proponen en [18] que deben de tratarse las etiquetas semánticas en SNOMED como parte de la estructura formal, pudiendo mejorar la sincronización de las sub-jerarquías con las etiquetas semánticas. En [41] los autores presentan algoritmos que pueden ayudar a los desarrolladores de las terminologías y a los usuarios a identificar posibles errores en SNOMED. En [22] se describe una herramienta que identifica, en SNOMED CT, la semántica oculta a partir del contenido descrito en lenguaje natural (apto para humanos), y cómo se enriquece la ontología creando nuevos axiomas lógicos (aptos para máquinas).

Objetivos

El objetivo principal de este proyecto consiste en identificar patrones léxicos que puedan representar conexiones entre clases de SNOMED CT con tecnologías de Machine Learning. Entre los objetivos secundarios tenemos los siguientes:

- Evaluar la viabilidad de las tecnologías del Machine Learning para identificar patrones léxicos en el ámbito sanitario, en general, y en las terminologías clínicas en particular.
- Evaluar la viabilidad de integración de las tecnologías de la web semántica (ontologías, etiquetas...), el procesamiento del lenguaje natural y el aprendizaje automático (algoritmos no supervisados) para identificar patrones léxicos-semánticos en SNOMED CT.

Asimismo, este proyecto puede ofrecer unas soluciones tecnológicas a un sector que cada vez hace más uso de tecnologías de la información. Se intenta dar una respuesta a una necesidad del sector sanitario como es el análisis de grandes volúmenes de información con anotaciones en lenguaje natural.

Capítulo 2

Tecnologías

En este capítulo discutiremos los distintos software, librerías y lenguajes de programación utilizados en este trabajo.

2.1 Ontologías

En el ámbito de la inteligencia artificial, una ontología [24] “es una forma de modelar un sistema estableciendo las entidades relevantes y las relaciones que emergen por su observación” o también la “especificación formal, explícita, de una conceptualización compartida de un dominio” [5]. Las ontologías se usan para estructurar la información y permiten compartir el conocimiento y que este pueda reutilizarse. Las ontologías, debido a su estructura y a su combinación de etiquetas descritas en lenguaje natural y relaciones lógicas, facilitan la comunicación entre usuarios y sistemas informáticos y su comprensión. Existen lenguajes que estandarizan la codificación de ontologías en programación tales como RDF [32], OWL [43] y SPARQL [34]. Las ontologías se utilizan para estructurar la información. La semántica de los lenguajes ontológicos facilita el procesamiento automático de la información representada y la inferencia de nuevo conocimiento por medio de técnicas de razonamiento [26].

2.1.1 SNOMED CT

Como se comentaba anteriormente, SNOMED CT [46] es la principal terminología clínica de referencia seleccionada para la Historia Clínica Digital del Sistema Nacional de Salud. Nosotros utilizaremos la International Edition 20180131.

SNOMED CT contiene más de 310.000 conceptos, estructurados en múltiples jerarquías (a través de más de 947.000 relaciones lógicas/semánticas), y a los que se le asocian descripciones en lenguaje natural. Por ejemplo, dados los conceptos “`Left ear`” y “`Left knee`” hay una relación

léxica entre ellos que se formaliza a través de la relación lógica de "`laterality`" representado como un axioma lógico, como se puede ver en la figura 1. Dado el tamaño de la ontología, no siempre un patrón léxico (apropiado para humanos) está formalizado a través de relaciones lógicas a través de un patrón semántico (apropiado para máquinas/razonadores). SNOMED CT identifica estos conceptos utilizando un sistema de URIs (Identificador de Recurso Uniforme) [38]. Cada concepto tiene asignado un número único y la URI sirve como referencia a ese concepto de la ontología. Por ejemplo, el URI "http://snomed.info/id/321286001" hace referencia al objeto 321286001: "`Product containing only hydroxyzine hydrochloride 25 mg/ 1 each oral tablet (clinical drug)`".

Figura 1: Ejemplos de clases en SNOMED CT.

2.1.2 OWL

El Lenguaje Ontológico Web (OWL) según está definido en [43] es un lenguaje destinado a representar conocimiento complejo sobre cosas, grupos de cosas y las relaciones entre estas. Fue desarrollado por el World Wide Web Consortium (W3C) [42] como parte del grupo de Semántica Web en 2004, aunque existe una versión más reciente (2009) conocida como OWL2. OWL nos sirve para representar ontologías con sus clases, individuos y propiedades, pudiendo esta información estar codificada en formato XML [13] o RDF [32]. En [25] podemos encontrar una explicación más detallada del funcionamiento de OWL2 y en la Figura 2 podemos ver un diagrama de su estructura.

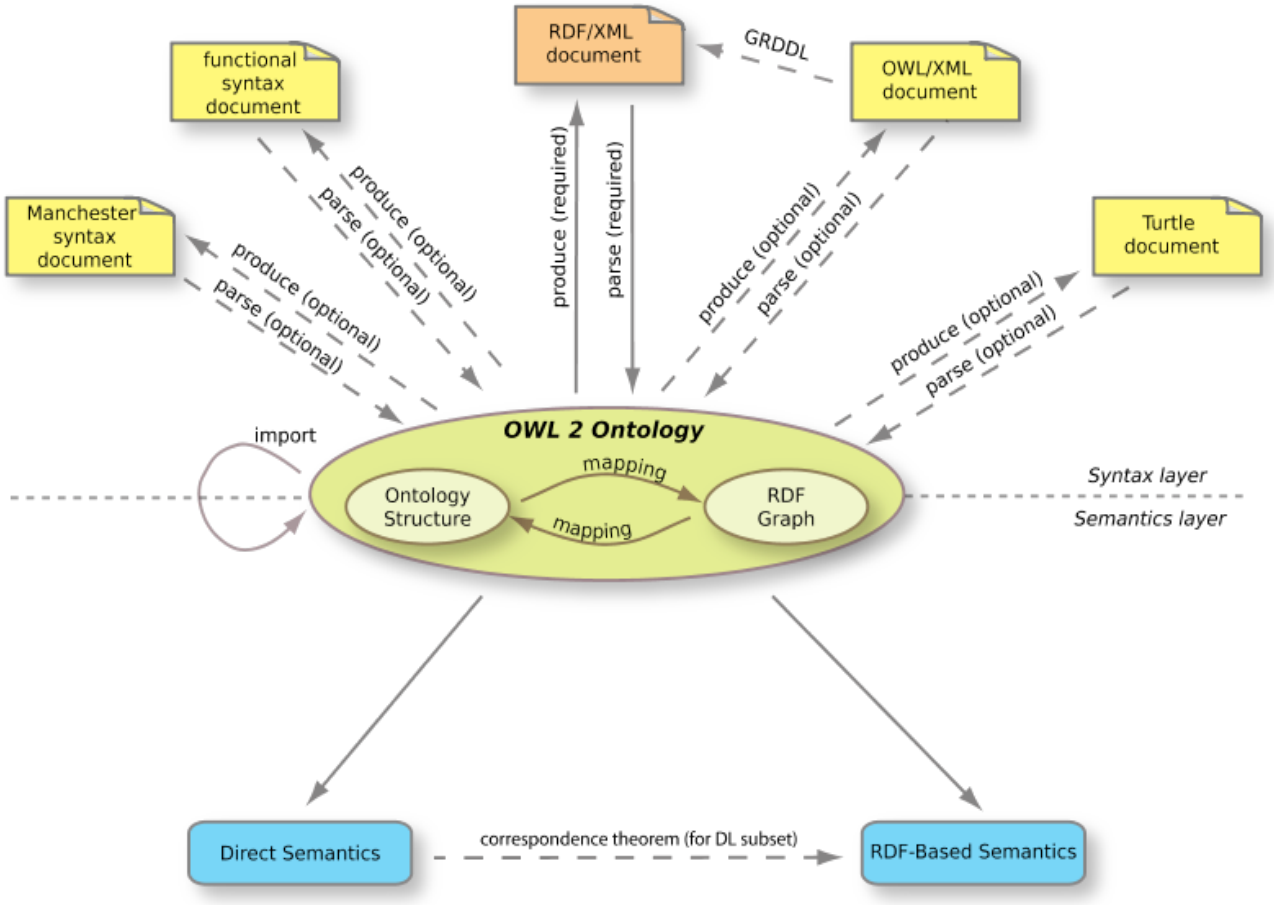


Figura 2: Estructura del OWL 2 [25].

2.1.3 Herramientas para explorar la ontología

2.1.3.1 Protégé

Protégé [1] empezó como un pequeño proyecto en la universidad de Stanford inicialmente diseñada para ayudar a construir bases de conocimiento en el campo de la medicina. La versión original de Protégé simplificaba el proceso de adquisición de conocimiento [14]. La versión más reciente de Protégé es Protégé 5 y tiene una base de usuarios a nivel mundial, convirtiéndose en el software más usado para abrir, explorar y trabajar con ontologías [23]. Actualmente Protégé cuenta con una aplicación web y una aplicación descargable (véase Figura 3) que podemos usar para crear o modificar ontologías. En este proyecto, hemos utilizado este software como apoyo para comprobar la estructura de la ontología y ayudarnos a interpretar los resultados.

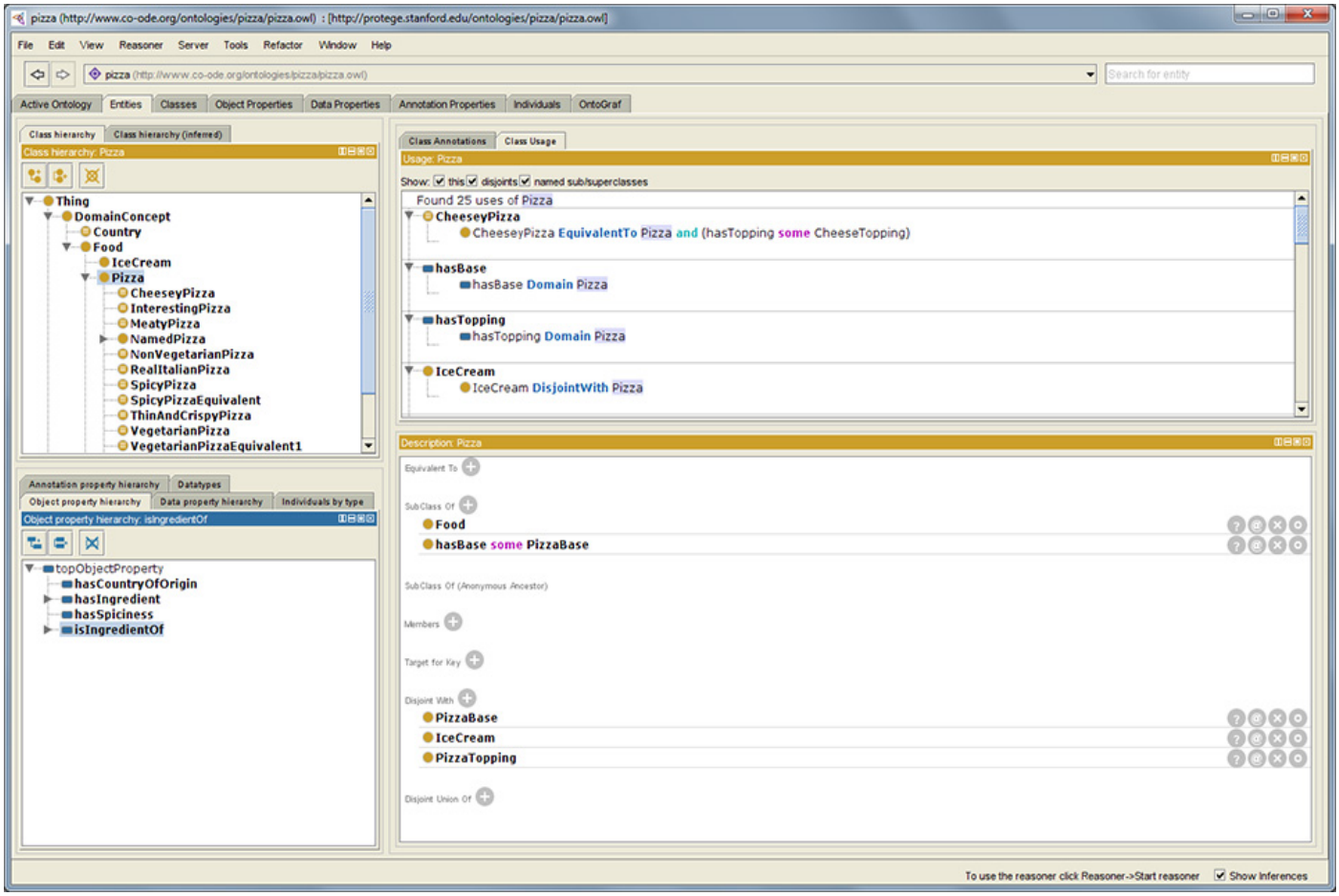


Figura 3: Pantallazo de la versión de Windows de Protégé. [1]

2.2 Procesamiento del lenguaje natural

Un lenguaje es una función que permite expresar pensamientos y comunicaciones entre los individuos. Un lenguaje natural se define como aquel que ha evolucionado con el tiempo para la comunicación humana, en el que el uso del lenguaje define las reglas y no al revés [40]. Los lenguajes naturales son muy útiles para comunicarse entre las personas porque tienen una capacidad expresiva enorme, pero a la hora de utilizarlo para comunicarse con las máquinas surgen muchos problemas debido a su dificultad intrínseca para formalizarlo. En el lenguaje natural existe información que no se expresa directamente mediante el lenguaje pero que los humanos de forma natural entendemos por el contexto, las formas o enlazándolo con información previa dentro de la misma conversación.

Se considera como procesamiento del lenguaje natural a cualquier tarea que consista en usar herramientas de computación para tratar con un lenguaje natural, sin importar la finalidad o el método. Para esto existen una multitud de herramientas y librerías, aunque cabe destacar dos de estas librerías para Java que evaluamos para usarlas en este trabajo: CoreNLP [7] y OpenNLP [45]. Estas

bibliotecas disponen de métodos y clases que nos ayudan a construir la pipeline del procesamiento del lenguaje natural. Una pipeline es un programa o sistema en la que cada paso recibe como entrada la salida del paso anterior. En el caso del procesamiento del lenguaje natural los pasos son los mostrados en la Figura 4, aunque no todos los sistemas implementan todos los pasos:

- Tokenization: separa el texto a analizar en tokens significativos que pueden ser palabras, conjuntos de palabras o símbolos.
- Sentence splitting: separa los enunciados dentro del texto. En ocasiones es complicado discernir donde empieza o acaba un enunciado.
- Part-of-speech Tagging: marca los tokens identificando su funcionalidad dentro del enunciado.
- Morphological Analysis: identifica el signifcado de cada token tratando de entender el contexto del que se extrae. Por ejemplo: “dicho” podría ser una conjugación del verbo “decir” o un sustantivo.
- Named Entity Recognition: identifica si los tokens hacen referencia a una entidad concreta como el nombre de una persona, de un lugar, o un año concreto.
- Syntactic Parsing: identifica los tokens como componentes de un enunciado identificándolos como, por ejemplo, sujeto, verbo o complemento directo.
- Coreference Resolution: identifica qué tokens hacen referencia al mismo concepto. Por ejemplo, me puedo referir a un mismo lugar como “mi casa”, “allí” o “ese sitio”.
- Other anotations: se añaden otras anotaciones a los tokens, cómo el sentimiento que expresan o el género que tienen.

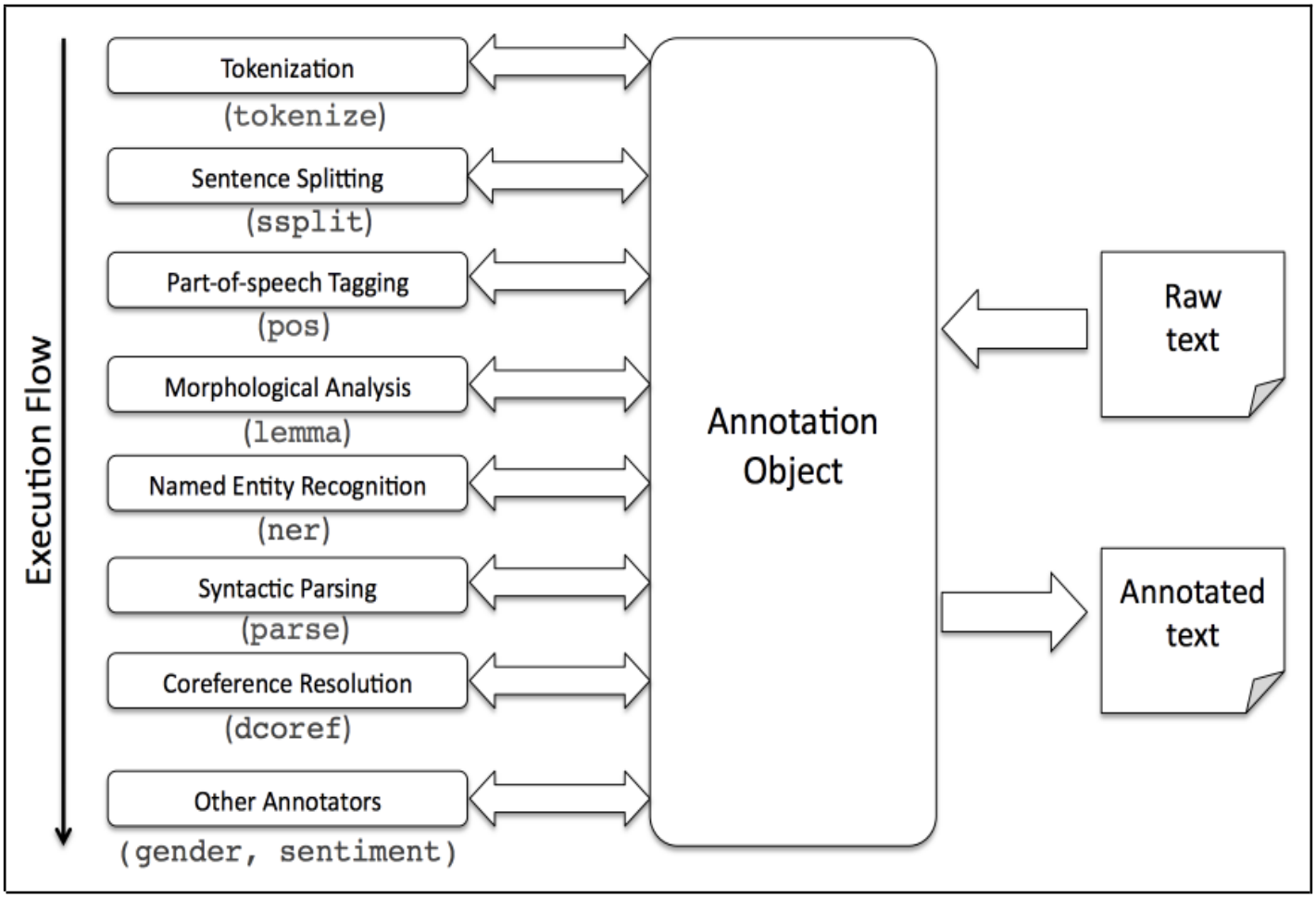


Figura 4: Ilustración de una pipeline para el procesamiento del lenguaje natural. [8]

Algunos ejemplos de procesamiento del lenguaje natural son la traducción automática, reconocimiento de voz o extracción de información. En nuestro caso, utilizaremos el procesamiento del lenguaje natural en el último ámbito mencionado para la extracción de información. Concretamente, proponemos la implementación de un método para el procesamiento automático de las etiquetas de texto asociadas a las clases de la ontología. Utilizaremos tan solo los pasos de Tokenization y Sentence Splitting utilizando las palabras a modo de tokens. Estas palabras las utilizamos después para detectar las regularidades léxicas así que no nos preocupa su significado o contexto, sólo la frecuencia con la que aparecen en la ontología.

2.3 Machine Learning

El Machine Learning [31] es una rama de la inteligencia artificial que busca crear sistemas capaces de aprender. Estos sistemas utilizan uno (o varios) algoritmos de aprendizaje cómo Reglas de Asociación, Clustering, Redes neuronales, Algoritmos Genéticos o una mezcla de varios. Los algoritmos de Machine Learning se suelen clasificar en dos tipos: algoritmos supervisados y no supervisados.

Los algoritmos supervisados [20] se entrenan dándoles una serie de casos llamados “training set” junto con el tipo de respuesta que se espera en cada caso. Luego, al dársele un caso fuera de este conjunto se espera que el sistema busque las similitudes con las situaciones a las que ya se ha enfrentado (training set) y proponga una solución. En este sentido trata de imitar el aprendizaje: buscar soluciones a situaciones nuevas basadas en la experiencia de situaciones similares ya superadas.

En el caso de los algoritmos no supervisados [16] no se les entrena. El algoritmo se diseña y se ejecuta directamente en un caso real en el que no se tiene una clasificación predefinida. En este tipo de algoritmos se pretende comprobar que conclusiones saca el sistema sin tener una idea de lo que queremos obtener. Nosotros utilizaremos este tipo de algoritmos puesto pretendemos visualizar las clases de SNOMED CT desde el punto de vista de las reglas de asociación.

2.3.1 Reglas de asociación

Las Reglas de Asociación están formadas por un grupo de atributos I y un grupo de transacciones D. Cada transacción en D consta de un conjunto de atributos existentes en I. Se busca obtener un conjunto de reglas de la forma $X => Y$ que nos indiquen que siempre que encontramos X en una transacción también nos encontraremos Y [2]. X e Y son conjuntos de atributos existentes en I que aparecen por lo menos en una transacción en D y cuya intersección es nula. Estas reglas tienen asociados dos atributos principales: el soporte y la confianza. El soporte nos indica en qué cantidad del total de transacciones aparecen X e Y juntos, llamándose “itemset frecuente” a este conjunto de atributos que tiene un soporte superior al umbral. Puede expresarse como una proporción o cómo un valor absoluto. La confianza nos indica en que proporción de las transacciones en las que aparece X también aparece Y. Esto se consigue dividiendo el total de transacciones en las que parece X entre el total de transacciones en las que aparece el itemset frecuente. Si estos valores son superiores a un cierto umbral previamente establecido se considera que la regla es válida [4].

En la Figura 5 podemos ver un ejemplo de una regla de asociación. En este ejemplo el Itemset frecuente encontrado está formado por los items “`body`” y “`structure`” dado que aparecen juntos en múltiples transacciones. De este itemset podemos sacar las reglas “{`body`} => {`structure`}” y “{`structure`} => {`body`}”. La segunda regla tiene menos confianza que la primera ya que hay transacciones en las que aparece “`structure`” pero no “`body`”, sin embargo, siempre que aparece “`body`” está también “`structure`”.

Nosotros utilizamos las reglas de asociación en este trabajo con el objetivo

de encontrar estructuras de palabras que se repitan y tengan cierta relación. Luego tendremos que comprobar que significan estas relaciones.

Transacción	Items
1	disease, vulva
2	structure, lumbar, part, diaphragm
3	phalanx, ring, finger, body, structure
3	accesroy, nerve, body, structure
4	multiple, body, structure
5	cord, body, structure

Itemset frecuente: {body, structure}
Regla: {body} => {structure}

Figura 5: Ejemplo de Reglas de Asociación.

2.4 Herramientas de desarrollo

2.4.1 Java

Java [19] es un lenguaje de programación de propósito general orientado a objetos que fue diseñado para ser implementado fácilmente en cualquier plataforma. De esta forma se facilitaba el trabajo de los programadores que tan solo tiene que escribir el programa una vez. Actualmente Java se usa principalmente en aplicaciones distribuidas. En su última versión “Java 8” mejora su capacidad de procesamiento en paralelo sin necesidad de usar threads gracias a los stream. También implementa expresiones lambda y añade la capacidad de pasarle código a los métodos. Estos cambios se explican con detalle en [39] junto con ejemplos de código.

Decidimos usar Java en este trabajo dado que nos permitía usar la OWLAPI de la que hablaremos más abajo. Usamos Java 8 en concreto porque es un requisito para usar la última versión de la API, más eficiente y sencilla. Usaremos estos componentes para extraer información de la ontología y obtener las regularidades léxicas.

2.4.2 Eclipse IDE

La Fundación Eclipse [36] es una organización sin ánimo de lucro que busca proveer a una comunidad global de usuarios con un ambiente centrado en la colaboración y la innovación. Cuenta entre sus productos con el entorno de desarrollo integrado del mismo nombre, Eclipse IDE (véase Figura 6). Este entorno provee al usuario con todas las herramientas que necesita para poder desarrollar código de forma rápida y segura. Entre estas herramientas se incluyen analizador sintáctico, compilación en tiempo real, refactorización y un modo depuración. También tiene soporte para plug-ins y una base de usuarios muy grande, lo que hace que encontrar ayuda sea fácil. Existen muchas versiones distintas del Eclipse IDE, pero la que hemos usado en este proyecto es Eclipse Oxigen para desarrollo en Java.

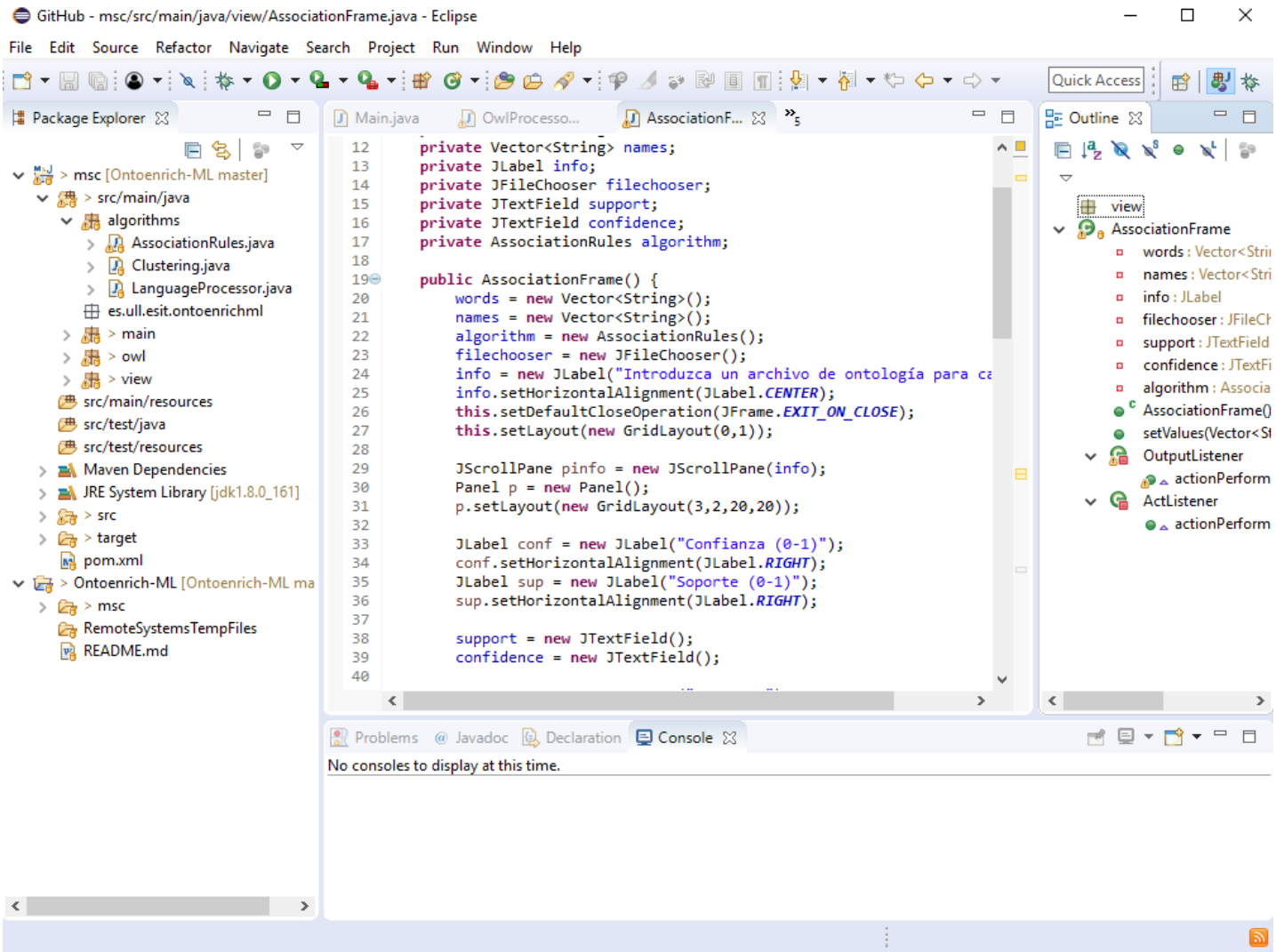


Figura 6: Pantallazo de Eclipse Oxigen, donde se pueden ver algunas de sus herramientas.

2.4.3 OWLAPI

OWLAPI [35][15] es una interfaz de desarrollo de aplicaciones (API) pensada para trabajar con ontologías OWL 2 y es compatible con todas las sintaxis definidas por la W3C. Se desarrolló con la intención de abstraer al programador

de la sintaxis OWL para que así fuera más accesible. Está basada en Java y lo hemos usado junto con de Apache Maven [44]. La versión que usamos para el trabajo es la 5.1.5, la última en el momento de empezar. La elegimos porque es la más eficiente ya que aprovecha todas las características de Java 8.

2.4.4 R

R [37] es tanto un entorno de desarrollo como un lenguaje de programación. Es un proyecto de software libre que provee una gran variedad de funciones centradas en el campo de la estadística y la representación gráfica. En muchos sentidos es una implementación en software libre de S [6].

El entorno de desarrollo mostrado en la figura 5 incluye las funciones necesarias para programar en R, incluidas consola y editor de texto, a demás de otras funcionalidades que facilitan el análisis de datos y representación gráfica. Además, el lenguaje puede ampliarse añadiendo nuevas funciones hechas en el propio R y librerías programadas en C, C++ o Fortran. Una descripción en profundidad de R y detalles de su diseño puede encontrarse en [17].

En este trabajo utilizamos R para procesar la información extraída de la ontología a través de las rutinas implementadas en Java. Lo elegimos porque existen librerías que permiten trabajar y visualizar reglas de asociación tales como arules [27] y arulesViz [28], las cuales comentaremos más adelante.

2.4.4.1 Rstudio

Rstudio [33] es un entorno de desarrollo integrado multiplataforma de código abierto con versión web que dispone de muchas utilidades para trabajar con R. Algunas de estas utilidades son: editor con resalto de sintaxis y ejecución directa, herramientas de depuración, visión de variables e histórico de ejecución (Figura 7).

A pesar de que R tiene su propio entorno de desarrollo nosotros utilizamos Rstudio porque añade muchas utilidades sobre el entorno por defecto, por lo que resulta mucho más cómodo trabajar con este.

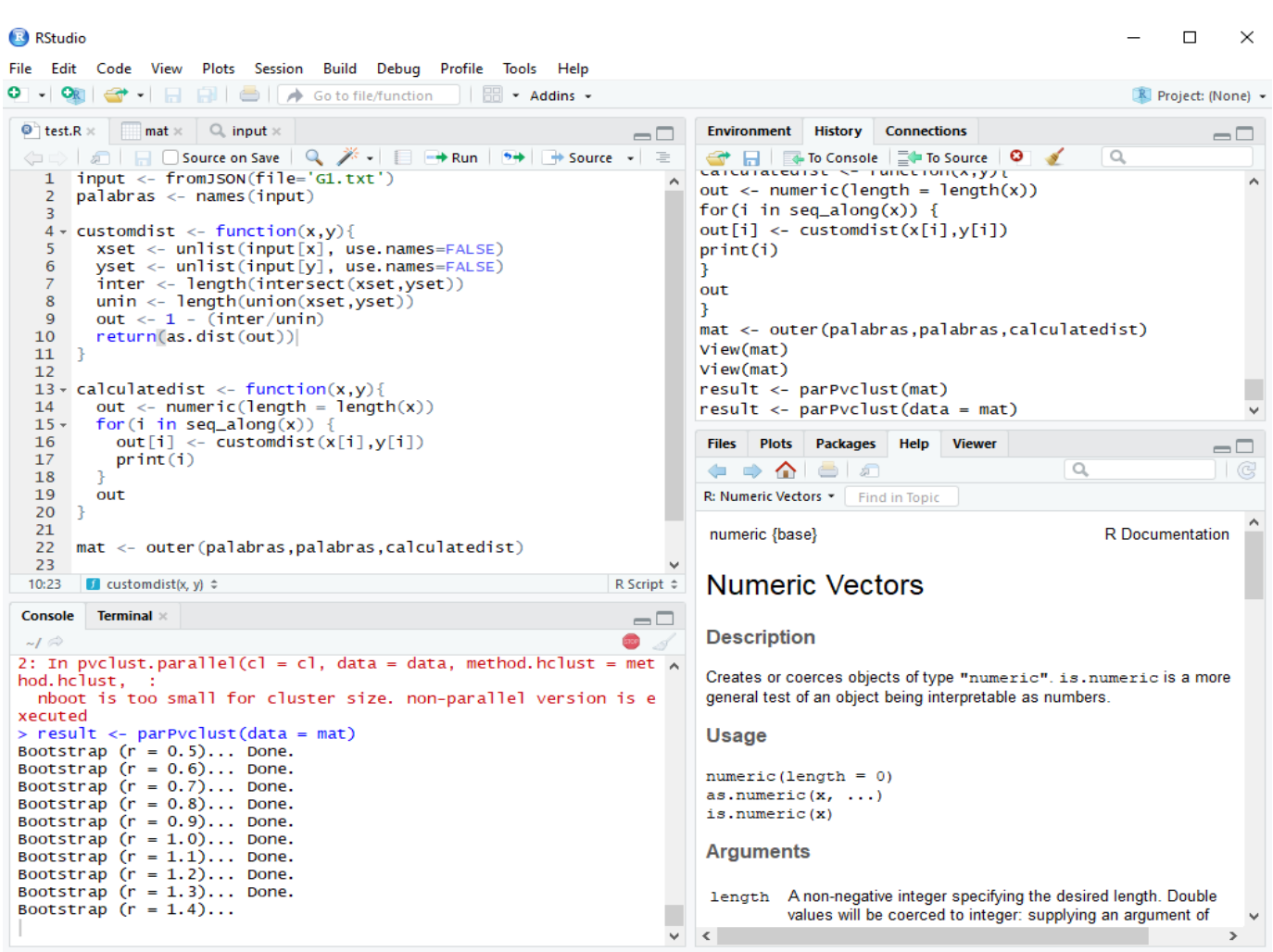


Figura 7: Pantallazo del entorno Rstudio. [12]

2.4.4.2 Arules y Arulesviz

Arules [27] y arulesViz [28] son dos librerías de R diseñadas para trabajar con reglas de asociación. Arules tiene funciones que permiten representar, manipular y analizar reglas de asociación, aunque nosotros la usaremos principalmente para construir las reglas. ArulesViz, por otro lado, está especializada en la representación gráfica de reglas de asociación e ItemSet frecuentes y está pensada para usarse junto con arules para añadir nuevas técnicas de visualización de los resultados.

Capítulo 3

Metodología

En este capítulo hablaremos de la metodología empleada en el trabajo y los distintos pasos que hemos seguido. En la figura 8 se encuentra un diagrama de funcionamiento del sistema desarrollado, que explicaremos a continuación. Los detalles de la implementación de los programas y análisis exhaustivos de los datos obtenidos en cada paso pueden encontrarse en el capítulo 4.

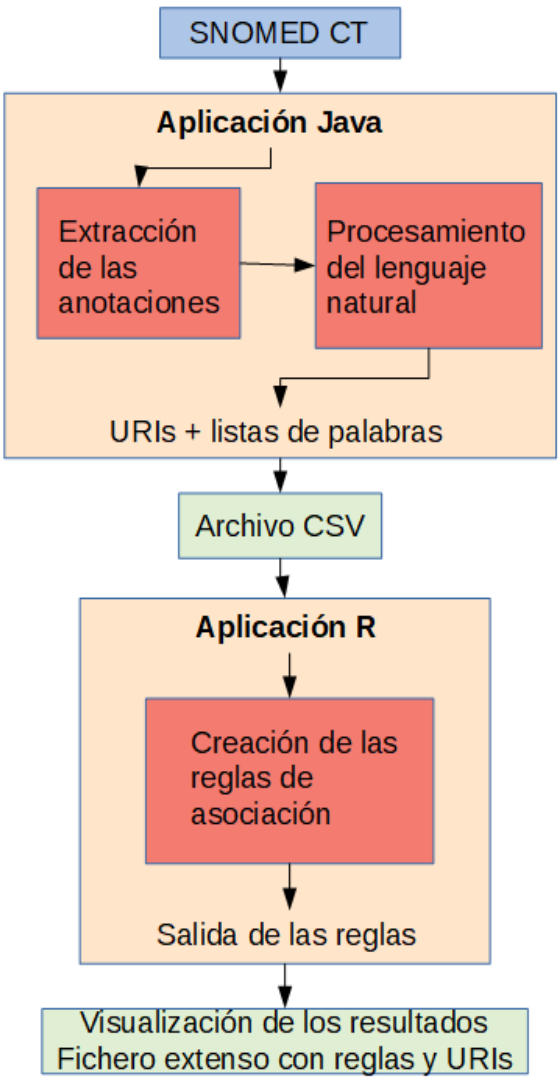


Figura 8: Diagrama del funcionamiento del sistema.

3.1 Entrada del sistema

Este sistema que hemos utilizado para el trabajo recibe como entrada el fichero OWL de la ontología de SNOMED CT. Este se introduce en una aplicación de desarrollo propio hecha en Java para ser procesada.

3.2 Aplicación Java

Esta aplicación utiliza las librerías de OWLAPI para extraer la información del fichero de la ontología. Recibe como entrada el fichero de la ontología en formato OWL y lo procesa en dos pasos.

- Primero obtiene los axiomas con anotaciones correspondientes a los nombres de las clases y almacena estas anotaciones junto con las URI de los axiomas.
- Luego utiliza procesamiento de lenguaje natural en estas anotaciones a nivel muy básico para separar las palabras y eliminar las palabras vacías [29].

El fichero de salida que produce este programa es un fichero CSV con las URIs de las anotaciones y las palabras extraídas de estas. Este fichero se introduce directamente la aplicación de R.

3.3 Aplicación R

El fichero CSV generado lo cargamos directamente en esta aplicación para generar y visualizar las reglas de asociación. Debido a limitaciones de hardware sólo podemos procesar parte de la información extraída cada vez. Una vez tenemos los datos cargados en el programa probamos a generar reglas con distintos umbrales de soporte y confianza. Una vez generadas usando las librerías descritas más arriba las visualizamos y a demás sacamos un fichero con las reglas de asociación generadas y las URIs en las que se apoyan.

3.4 Salida del sistema

La salida del sistema son las imágenes de visualización generadas por R y el fichero con todas las reglas y URIs. La visualización es necesaria para poder entender qué hemos obtenido de forma fácil e intuitiva, además de servir de una mejor forma de exposición. El archivo con las reglas y las URIs en las que se basan nos ayuda a poder contextualizar los resultados obtenidos y poder discernir si tienen o no sentido y comprobar si ya existen las relaciones que describen.

Capítulo 4

Implementación y resultados

4.1 Trabajo con la ontología y extracción de regularidades léxicas

Para extraer las regularidades léxicas de SNOMED CT con las clases que las contienen hemos utilizado la librería de OWLAPI versión 5.1.5, Java 8 y Eclipse Oxigen. Es un software muy simple que nos pide simplemente el archivo de la ontología, un archivo de salida y un número P entre 0 y 1. Este número P se refiere a la proporción del total de anotaciones en las que tiene que salir una palabra para sacarla en el fichero de salida. Es poco práctico extraer todas y cada una de las palabras puesto que la salida sería demasiado grande para poder después procesarla con el programa en R, así que utilizamos este parámetro para "purgar" los datos y obtener las palabras más frecuentes. Probando distintos valores se obtienen distintas salidas, que después exploraremos, y que influyen en cierta medida en el resultado final.

Lo primero que debemos hacer es buscar el archivo que contiene la ontología en formato OWL y cargarlo (figura 9). Luego elegimos un valor de P (figura 10) y finalmente exportamos los resultados. Podemos cambiar el valor de P y volver a exportar tantas veces como queramos.

Cuando el programa carga la ontología procesa los axiomas para obtener aquellos que tienen como componente el nombre de una clase. En el caso de que tengan, guardamos el nombre y el URI. Al nombre de la clase le aplicamos la Tokenization y Sentence splitting que vimos en el apartado sobre Procesamiento del lenguaje natural. Los tokens los obtenemos dividiendo el string del nombre en los espacios y eliminando los signos de puntuación como paréntesis o comas. Como cada nombre es un enunciado, el Sentence Splitting no hace falta realizarlo activamente. Tampoco procesamos sinónimos ni tenemos en cuenta el contexto de las palabras. El URI y las palabras obtenidas las almacenamos en dos estructuras distintas: un mapa de palabras con

vectores de String con las URI de las anotaciones en las que aparece y un mapa de las URI con vectores de String con las palabras que contienen. De esta manera, para saber el número de clases en las que aparece una palabra preguntamos al vector asociado su tamaño y para saber en qué clases exactamente sacamos el vector de mapa. También podemos recorrer fácilmente las URI para obtener sus palabras por orden. Con este sistema de almacenamiento se consume más memoria, pero ahorra tiempo a la hora de producir la salida. Como el uso de memoria no ha sido un problema hemos seguido usando este método. En la figura 11 se encuentra el código del procesamiento del lenguaje natural y en la Figura 12 el código para cargar la ontología procesando y guardando los axiomas y palabras.

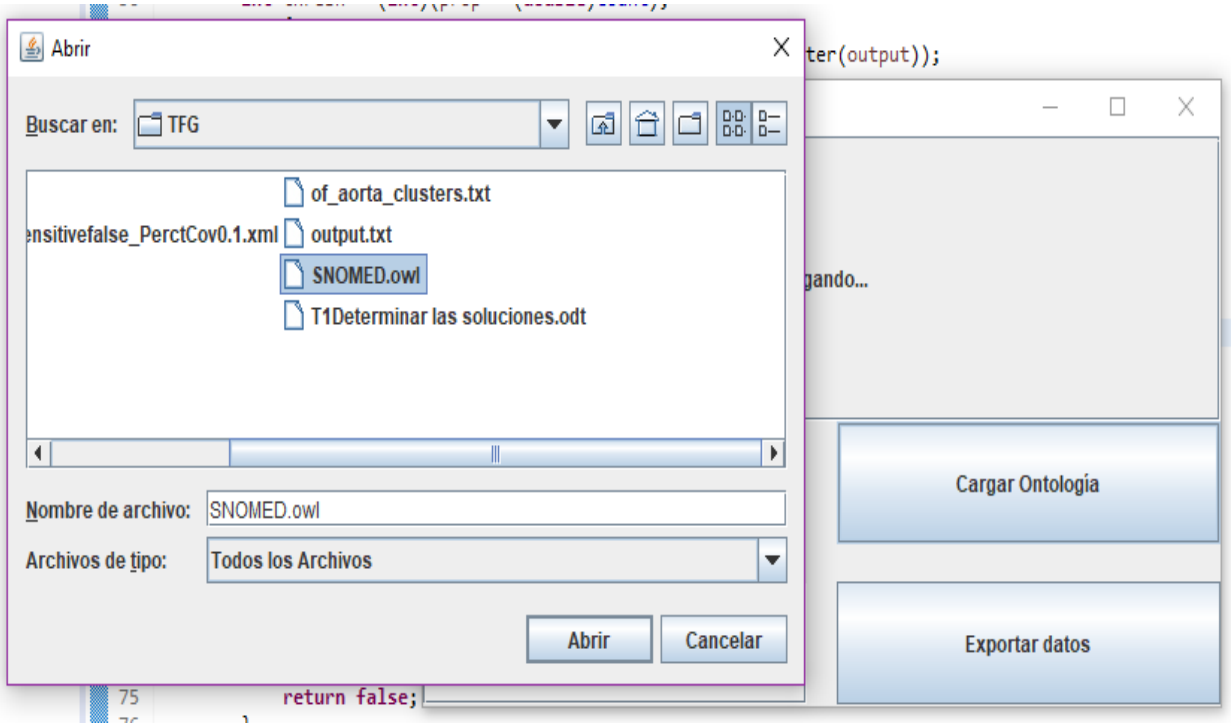


Figura 9: Pantallazo de la aplicación. Buscando el archivo de ontología.

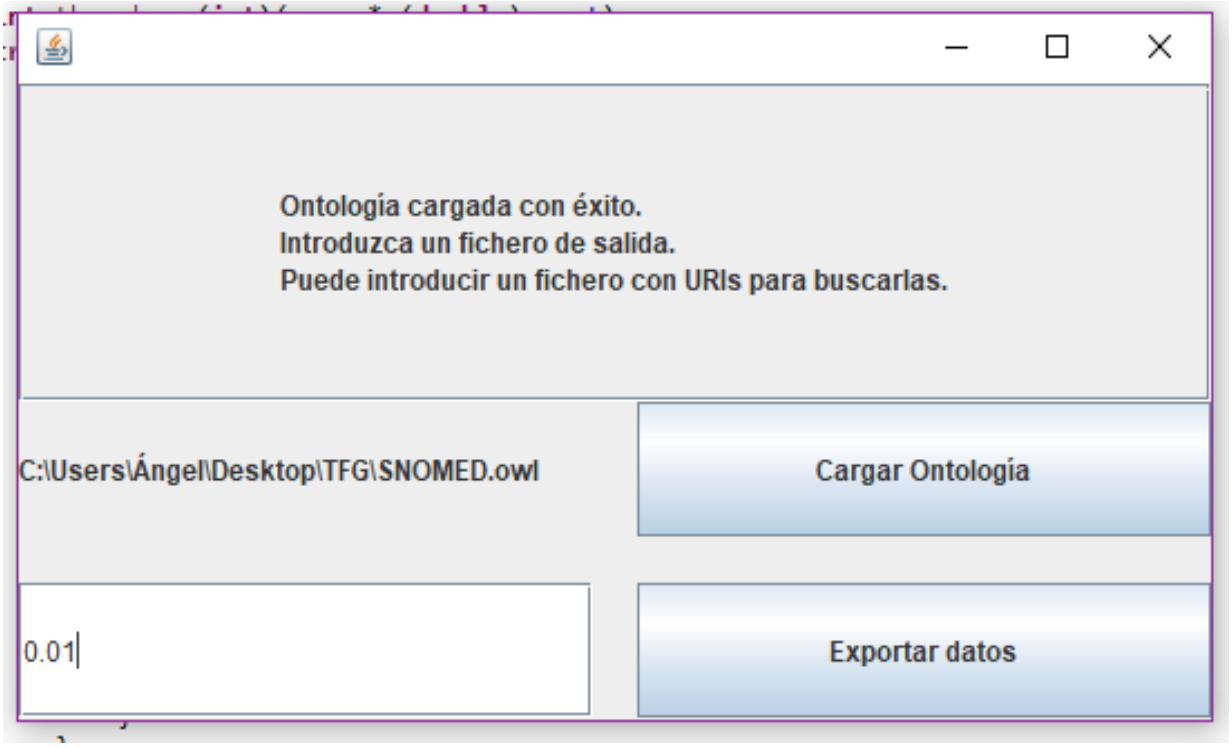


Figura 10: Pantallazo de la aplicación. Ajustamos P.

```java
public static Vector<String> obtainWords(String name){
    //Extrae las palabras de un nombre
    name = name.toLowerCase();
    Vector<String> output = new Vector<String>();
    //Divide usando espacios en blanco
    String[] words = name.split(" ");
    //Por cada palabra
    for(int i = 0; i < words.length; i++) {
        //Elimina la puntuación
        String word = removePunctuation(words[i]);
        //Si no esta en la salida ya, no es una palabra vacía
        //y no es una cadena nula la guardamos
        if(!output.contains(word) && !isStopword(word) && word != "")
            output.add(word);
    }
    return output;
}
```

```java
public static boolean isStopword(String word) {
    //Método para eliminar stopwords
    for(int i = 0; i < stopwords.length; i++)
        if(stopwords[i].equals(word))
            return true;
    //también elimina números sueltos
    boolean n = true;
    for(char c : word.toCharArray())
        if(!Character.isDigit(c))
            n = false;
    return n;
}
```

```java
public static String removePunctuation(String word) {
    String out = word;
    if(word.startsWith("("))
        out = word.substring(1);
    if(out.endsWith(")"))
        out = out.substring(0, out.length()-1);
    if(out.endsWith(","))
        out = out.substring(0, out.length()-1);
    if(out.endsWith("."))
        out = out.substring(0, out.length()-1);
    if(out.endsWith(":"))
        out = out.substring(0, out.length()-1);
    if(out.endsWith(";"))
        out = out.substring(0, out.length()-1);
    return out;
}
```

Figura 11: Fragmentos del código para procesar los nombres y obtener palabras clave.

```java
public OwlProcessor(File input) {//Método para cargar la ontología
    // Cargamos la ontología en el ontology Manager
    OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
    try {
        manager.loadOntologyFromOntologyDocument(input);
    } catch (OWLOntologyCreationException e) {
        e.printStackTrace();
    }

    // Guardamos esa ontología en concreto en una variable de clase
    // Es la primera del array porque es la única
    snomed = (OWLOntology) manager.ontologies().toArray()[0];

    //Creamos los mapas que guardará todas las palabras y clases
    wordtouri = new HashMap<String, Vector<String>>();
    uritoword = new HashMap<String, Vector<String>>();
    for(Object o :  snomed.axioms().toArray()) {
        //axioms se devuelve como un objeto. Hay que hacer typecast
        OWLAxiom axiom = (OWLAxiom)o;
        //Obtenemos la uri y el nombre
        String uri = getUriFrom(axiom);
        String name = getNameFrom(axiom);
        //Si ninguna es nula es una anotación, la guardamos.
        if(uri != "NULL" && name != "NULL"){
            //guardamos la uri en una nueva entrada en el mapa
            if(!uritoword.containsKey(name))
                uritoword.put(uri, new Vector<String>());

            for(String w : LanguageProcessor.obtainWords(name)) {
                //Añadimos la palabra al array de su URI
                uritoword.get(uri).addElement(w);

                //Si la palabra ya existe en el mapa de palabras
                //añadimos la clase al array
                if(wordtouri.containsKey(w)) {
                    wordtouri.get(w).add(uri);
                }
                //Si no creamos una nueva entrada
                else {
                    Vector<String> temp = new Vector<String>();
                    temp.add(uri);
                    wordtouri.put(w, temp);
                }
            }
        }
    }
}
```

Figura 12: Código de carga de la ontología, procesamiento de sus axiomas y obtención de las anotaciones asociadas con las clases.

Para producir una salida, la función utiliza como entrada el fichero seleccionado como output y el valor de P. Calcula en un número entero la cantidad de anotaciones en las que debería de salir una palabra para que su proporción de aparición en el total fuera mayor que P. Luego, por cada una de

las anotaciones del mapa de URIs obtiene su vector de palabras, por cada palabra dentro del vector obtenido comprueba si dicha palabra supera este requisito, consultando el tamaño del vector de anotaciones del mapa de palabras. Si lo supera entonces se añade la URI y la palabra a una nueva línea del fichero CSV de salida. Este proceso se repite por cada palabra de cada URI. La figura 13 es el código de esta función y la 14 es un ejemplo de salida.

```
public boolean printUriscsv(File output, double prop) {//Método de salida
    //Calculamos el umbral
    int thresh = (int)(prop * (double)uritoword.keySet().size());

    try {
        //buffer de salida
        BufferedWriter bw = new BufferedWriter(new FileWriter(output));
        //Primera línea del fichero
        bw.write("Word,URI\n");

        //Por cada URI
        for(String u : uritoword.keySet()) {
            //Por cada una de sus palabras
            for(String l : uritoword.get(u)) {
                //Si dicha no supera el umbral nos la saltamos
                if(wordtouri.get(l).size() < thresh)
                    continue;
                //Si lo supera la escribimos
                bw.write(l+","+u+"\n");
            }
        }
        bw.close(); //Cerramos el buffer
    } catch (IOException e) {
        e.printStackTrace();
        return false;
    }

    return true;
}
```

Figura 13: Código para generar la salida.

Word	URI
disease	http://snomed.info/id/402375009
vulva	http://snomed.info/id/402375009
structure	http://snomed.info/id/77305008
lumbar	http://snomed.info/id/77305008
part	http://snomed.info/id/77305008
diaphragm	http://snomed.info/id/77305008
family	http://snomed.info/id/160314003
history	http://snomed.info/id/160314003
situation	http://snomed.info/id/160314003
hb	http://snomed.info/id/84083005
arthropathy	http://snomed.info/id/53338001
reaction	http://snomed.info/id/53338001
congenital	http://snomed.info/id/719518004
type	http://snomed.info/id/719518004
breathing	http://snomed.info/id/229314005
childbirth	http://snomed.info/id/229314005
regime/therapy	http://snomed.info/id/229314005
language	http://snomed.info/id/297492005
value	http://snomed.info/id/297492005

Figura 14: Ejemplo de salida del programa.

Este algoritmo en dos partes primero procesa todas las palabras y URIs y luego produce una salida filtrada con los datos que nos interesan. Es muy rápido, sólo lleva tiempo el proceso de cargar la ontología usando la API. Una vez está cargada la ontología el proceso de buscar los axiomas, procesarlos y guardarlos no tarda poco más de 10 segundos. A la hora de producir la salida el tiempo varía en función del valor de P, entre menos de una décima a unos 5 segundos para $P = 0$.

Como ya hemos dicho, distintos valores de P dan como resultado distinta cantidad de datos de salida. Para ver qué valor o valores nos interesa usar como conjunto al que aplicar las reglas de asociación hicimos una serie de pruebas con distintos valores de P. En la tabla 1 se muestran los resultados de estas pruebas. En esta tabla están indicados los valores de P, el umbral mínimo de anotaciones en las que tiene que aparecer una palabra, el número de palabras de salida y el número total de líneas del fichero de salida.

P (0-1)	**Umbral**	**Número de palabras**	**Número total líneas del fichero de salida**
1	339520	0	1
0.2	67904	1	26999
0.1	33952	6	101271
0.08	27161	7	110711
0.07	23766	8	118860
0.04	13580	9	125442
0.03	10185	15	149977
0.02	6790	38	222698
0.01	3395	123	376285
0.009	3055	136	393137
0.008	2716	152	410706
0.007	2376	184	443038
0.006	2037	225	479929
0.005	1697	275	517013
0.004	1358	356	565234
0.003	1018	494	630202
0.002	679	748	712922
0.001	339	1423	840080
0.0001	33	8805	1120522
0.00001	3	56309	1263507
0	0	117049	1301002

Tabla 1: Análisis de los datos de los valores de P.

En los experimentos observamos lo siguiente:

- La primera palabra en aparecer es “`disorder`”.
- Cuando P es 0.001 aparecen menos del 1,22% de las palabras, sin embargo ya hay más del 64,57% de las líneas de salida.

- Más del 51,89% de las palabras aparecen menos de 3 veces, por lo que podrían ser despreciables.

Con esta información obtenemos varios grupos de interés, elegidos de forma arbitraria para realizar los experimentos, los cuales analizaremos individualmente. Nos referiremos a ellos de la siguiente manera:

- **G1**: $P = 0.006$. Tiene 225 palabras y 479929 líneas. Es nuestro principal grupo de trabajo, ya que es el mayor que puede ser procesado por el programa en R sin necesidad de trocearlo. Hablaremos de esta limitación en el próximo apartado.
- **G2**: $P = 0.02$. Con 38 palabras es un grupo en el que podemos controlar bien todas las reglas que salgan y comprobar si tienen o no sentido.
- **G3**: $0.00001 < P < 0.0001$. Este grupo podría ser interesante analizarlo para ver si estas palabras con baja frecuencia están relacionadas entre sí de alguna manera.
- **G4**: $P = 0.0001$. El grupo más grande que consideramos significativo. Sin embargo, debido a problemas de memoria no lo utilizamos en los análisis.

4.2 Implementación de las reglas de asociación

Para aplicar el algoritmo de reglas de asociación en nuestro caso los datos son los siguientes:

- I es el conjunto de palabras extraídas de la ontología.
- D es el conjunto de URIs de las anotaciones que procesamos en el paso anterior. Los elementos I de cada transición son las palabras sacadas de la URI en cuestión.
- Las reglas $X => Y$ nos indican que cuando aparece una palabra o palabras X también aparece Y. Si se produce una regla quiere decir que existe algún tipo de relación entre las palabras de X e Y o entre los conceptos que representan. En estos casos tendríamos que analizar si se requiere una relación nueva en la ontología o si no es necesaria o si ya existe una.

Como ya hemos comentado estas reglas de asociación las crearemos y

analizaremos utilizando un programa desarrollado en R con las librerías especializadas “arules” y “arulesViz”. Hemos utilizado como base el código de ejemplo obtenido de [11], modificándolo para adaptarlo a nuestro caso y añadiéndole nuevas funcionalidades.

Este programa recibe como entrada el archivo CSV producido por la aplicación Java y lo almacena transformándolo en una matriz y en un vector de transiciones. El paso de convertir la entrada en una lista es el proceso que más tiempo y recursos consume y hace de cuello de botella para el resto del programa. Utilizando un ordenador con un procesador Intel Core i5-4570 CPU de 3.20GHz con 4 núcleos y 8GB de memoria RAM sólo se pueden procesar archivos de entrada con menos de 500.000 elementos porque con más surgen problemas de memoria. Cuando nos acercamos a este límite el tiempo de ejecución de este proceso es de una media hora. En la figura 15 se encuentra el código en R de esta parte.

```
#http://apuntes-r.blogspot.com.es/2015/07/reglas-de-asociacion.html
# CARGA LIBRERIA Y DATOS
# ------------------------------------------------------------
library(arulesViz);library(arules) #Incluimos las librerías
input <- read.csv('G1.csv') #Entrada del archivo
trx    <- input #trx se convertirá en transiciones
mat <- as.matrix(input) #mat representa los mismos datos como una matriz


# TRANSFORMA Data.Frame en TRANSACCIONAL
# ------------------------------------------------------------
# Transforma  data.frame en transaccional
trx       <- split(trx$Word,trx$URI) # convierte datos en lista
trx       <- as(trx,"transactions") # convierte datos en transacciones
```

Figura 15 : Código para cargar los datos en R.

Tras cargar los datos procedemos a la creación propiamente dicha de las reglas de asociación, utilizando el método “apriori” con unos parámetros de confianza y soporte arbitrarios. Sin importar los valores suministrados al algoritmo no tarda más de 30 segundos en ningún caso, así que podemos producir muchos sets de reglas con distintos valores sin que suponga un problema el tiempo de procesado. Una vez creadas las reglas de asociación procedemos a mostrar los resultados como una lista de reglas, ItemSet frecuentes y distintos gráficos de visualización. La figura 16 muestra esta parte del código.

```
# CREA REGLAS
# ------------------------------------------------------------------
reglas <- apriori(trx, parameter=list(support=0.001, confidence = 0.9))

# PRINT
# ------------------------------------------------------------------
# cantidad de reglas creadas
print(reglas)
# ordena las reglas por confianza
reglas <-sort(reglas, by="confidence", decreasing=TRUE)
# imprime todas las reglas
inspect(reglas)

# data.frame con frecuencia porcentual de cada palabra
FreqWord <- data.frame(Palabra=names(itemFrequency(trx)),
                       Frecuencia=itemFrequency(trx), row.names=NULL)
FreqWord <- FreqProd[order(FreqProd$Frecuencia, decreasing = T),]
inspect(Freqword)

#Print del data set de transacciones
inspect(trx)

# PLOT
# ------------------------------------------------------------------
reglas <-sort(reglas, by="confidence", decreasing=TRUE) # ordena reglas
labels <- inspect(reglas)
# grafica los 20 productos mas frecuente
itemFrequencyPlot(trx,topN=20,type="absolute")
# Grafico de red 20 reglas con mayor confianza
plot(head(reglas,20), method="graph", control=list(type="items"))
# Grafico de matriz de 100 regla con mayor confianza
plot(head(reglas,100), method="graph", engine = "interactive")
# grafico de dispersion de todas las reglas
plot(reglas,  method = "iplots")
```

Figura 16 : Código para crear las reglas y mostrar los resultados.

Estos mismos métodos de “plot” producen imágenes de gráficos y estadísticas así que no nos tenemos que preocupar por eso. Lo que sí tenemos que implementar nosotros es la salida completa de los datos en un fichero CSV. Este fichero, un fragmento del cual se muestra en la figura 17, tiene 5 columnas: “lhs” (antecedente), “rhs” (consecuente), support, confidence, lift y una lista de URIs en las que se cumple la regla. Este fichero nos permite comprobar toda la información de golpe y gracias a la lista de URIs podemos contextualizar en qué casos aparece la regla dentro de la ontología.

lhs	rhs	support	confidence	lift	count	URIs
{enterica}	{salmonella}	0.001435496	1.0000000	105.42053	199	…
{mellitus}	{diabetes}	0.001204663	1.0000000	613.39823	167	…
{mg/1}	{each}	0.006585971	1.0000000	137.93831	913	…
{mg/1}	{oral}	0.006585971	1.0000000	67.16473	913	…
{dosage,parenteral}	{form}	0.001139741	1.0000000	168.03394	158	…
{ser,subsp}	{enterica}	0.001341720	1.0000000	696.62312	186	…
{enterica,ser}	{subsp}	0.001341720	1.0000000	660.13333	186	…
{ser,subsp}	{salmonella}	0.001341720	1.0000000	105.42053	186	…
{salmonella,ser}	{subsp}	0.001341720	1.0000000	660.13333	186	…
{enterica,ser}	{salmonella}	0.001341720	1.0000000	105.42053	186	…
{salmonella,ser}	{enterica}	0.001341720	1.0000000	696.62312	186	…
{enterica,subsp}	{salmonella}	0.001356147	1.0000000	105.42053	188	…
{salmonella,subsp}	{enterica}	0.001356147	1.0000000	696.62312	188	…
{diagnostic,extract}	{allergen}	0.001247944	1.0000000	495.10000	173	…
{allergen,diagnostic}	{extract}	0.001247944	1.0000000	563.52846	173	…
{behavior,uncertain}	{neoplasm}	0.001666330	1.0000000	77.88090	231	…
{behavior,neoplasm}	{uncertain}	0.001666330	1.0000000	529.11450	231	…
{e,specific}	{immunoglobulin}	0.001543700	1.0000000	243.20702	214	…
{dosage,medicinal}	{form}	0.001103673	1.0000000	168.03394	153	…

Figura 17: Ejemplo de salida de las reglas de asociación.

R ofrece muchas facilidades para imprimir modelos de datos como ficheros de texto, sin embargo la complicación al producir este fichero surge al añadir la lista de URIs a cada regla. Para conseguirlo lo primero que hacemos es crear una nueva matriz de salida que contiene la misma información que las reglas y luego le añadimos una nueva columna de URIs. Luego, por cada regla, creamos un vector de palabras que tenga las que aparecen en el antecedente y consecuente. Por cada palabra, utilizamos el método “which” que nos devuelve una lista de índices que indican en que fila se encontró una palabra igual, luego accedemos a esas filas y almacenamos los URIs en un vector. Tras eso, hacemos la intersección de este vector con el de URIs previamente encontradas para, de esta forma, quedarnos con los que siempre aparecieron al buscar cada palabra individualmente. Por último convertimos este vector en un string y lo añadimos a la columna de URIs en la regla correspondiente. En la figura 18 se encuentra el código de esta parte.

```
#Extract
#-----------------------------------------------------------
#creamos la matriz con la inforación de las reglas
output <- inspect(reglas)
output <- as.matrix(output)
#añadimos una nueva columna vacía
output <- cbind(output, URIs = "")

for(i in 1 : nrow(output)){#recorremos las reglas
  t <- output[i,"lhs"]
  l <- strsplit(t, ",")[[1]] #lado izquierdo
  t <- output[i,"rhs"]
  r <- strsplit(t, ",")[[1]] #lado izquierdo
  vec = vector() #vector para almacenar las palabras
  #Añadimos las palabras al vector
  #Eliminamos los corchetes si lo tienen
  for(j in 1 : length(l)){ #Lado izquierdo
    c <- l[[j]]
    if(startsWith(c, "{")){
      c <- substr(c, 2, nchar(c))
    }
    if(endsWith(c, "}")){
      c <- substr(c, 1, nchar(c)-1)
    }
    vec[j] <- c
  }
  for(j in 1 : length(r)){ #lado derecho
    c <- r[[j]]
    if(startsWith(c, "{")){
      c <- substr(c, 2, nchar(c))
    }
    if(endsWith(c, "}")){
      c <- substr(c, 1, nchar(c)-1)
    }
    vec[j+length(l)] <- c
  }
  #Añadimos al vector de resultados todas las URI
  result <- as.vector(mat[,2])
  result <- unique(result)
  for(j in 1 : length(vec)){
    x <- which(mat == vec[j]) #Devuelve las filas
    for(k in 1 : length(x)){
      #Sacamos las URIs de las filas
      x[k] <- mat[strtoi(x[k]),2]
    }
    result <- intersect(x, result) #Intersección
  }
 #Escribimos las URIs en la fila correspondiente
 output[i, "URIs"] <- paste(result, collapse = "  ")
}
#Salida a fichero
write.table(output,file="test.csv", quote = TRUE,
            sep = ",", row.names = FALSE)
```

Figura 18 : Código para crear las reglas y mostrar los resultados.

4.3 Análisis y resultados de las reglas de asociación

Lo primero que hay que destacar es que el grupo G3 que en un principio nos habíamos planteado analizar no ha podido ser manejado por el algoritmo. Esto es debido a que al haber una enorme cantidad de URIs con pocas palabras se genera un vector de transacciones demasiado grande. Lo dejamos fuera del análisis junto como el grupo G4.

4.3.1 Descripción de las regularidades léxicas obtenidas

Al realizar los análisis con los grupos G1 y G2 se observó, como era de esperar, que las palabras más frecuentes eran las mismas en ambos casos. La figura 17 muestra el histograma de la frecuencia de aparición de las palabras de G1. En G2 la cantidad de apariciones de cada palabra es menor, pero la proporción se mantiene. SNOMED CT organiza sus clases en jerarquías [3] y al definir la descripción en lenguaje natural asociada a una clase incluye entre paréntesis el nombre de la jerarquía. Algunas de estas jerarquías son "`disorder`", "`procedure`", "`finding`"... y este podría ser un motivo que explica que se mantenga la proporción entre los dos grupos.

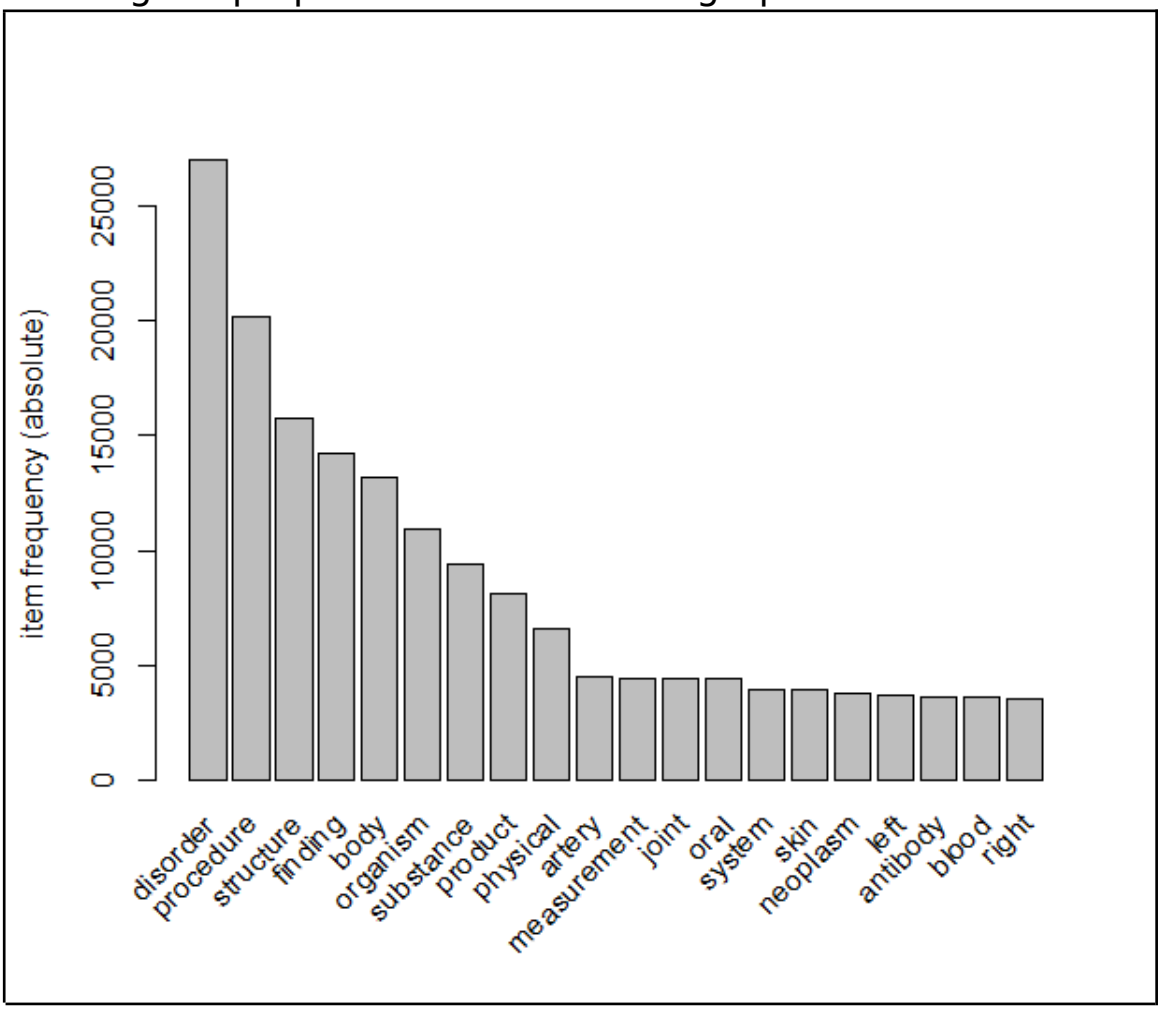


Figura 19: Número de apariciones de las 20 palabras más comunes en G1.

4.3.2 Análisis de las reglas de asociación basadas en las regularidades léxicas

En un primer análisis procesamos G1 con una confianza de 0.9 y un soporte de 0.001 obteniendo 444 reglas. Las 20 con mayor confianza y soporte se encuentran en la figura 20 y en la tabla 2 aparecen estas reglas con sus valores asociados. En el grafo de la figura 20, cada círculo verde representa la palabra que tiene escrita y cada círculo rosa o rojo representa una regla. Los círculos de reglas tienen flechas que entran desde las palabras que forma su antecedente y flechas que salen hacia las palabras de su consecuente. Por ejemplo, en la esquina superior derecha hay una regla en la que entran "`product`" y "`dosage`" y sale "`form`", así que la regla que representa es {`product`, `dosage`} => {`form`}.

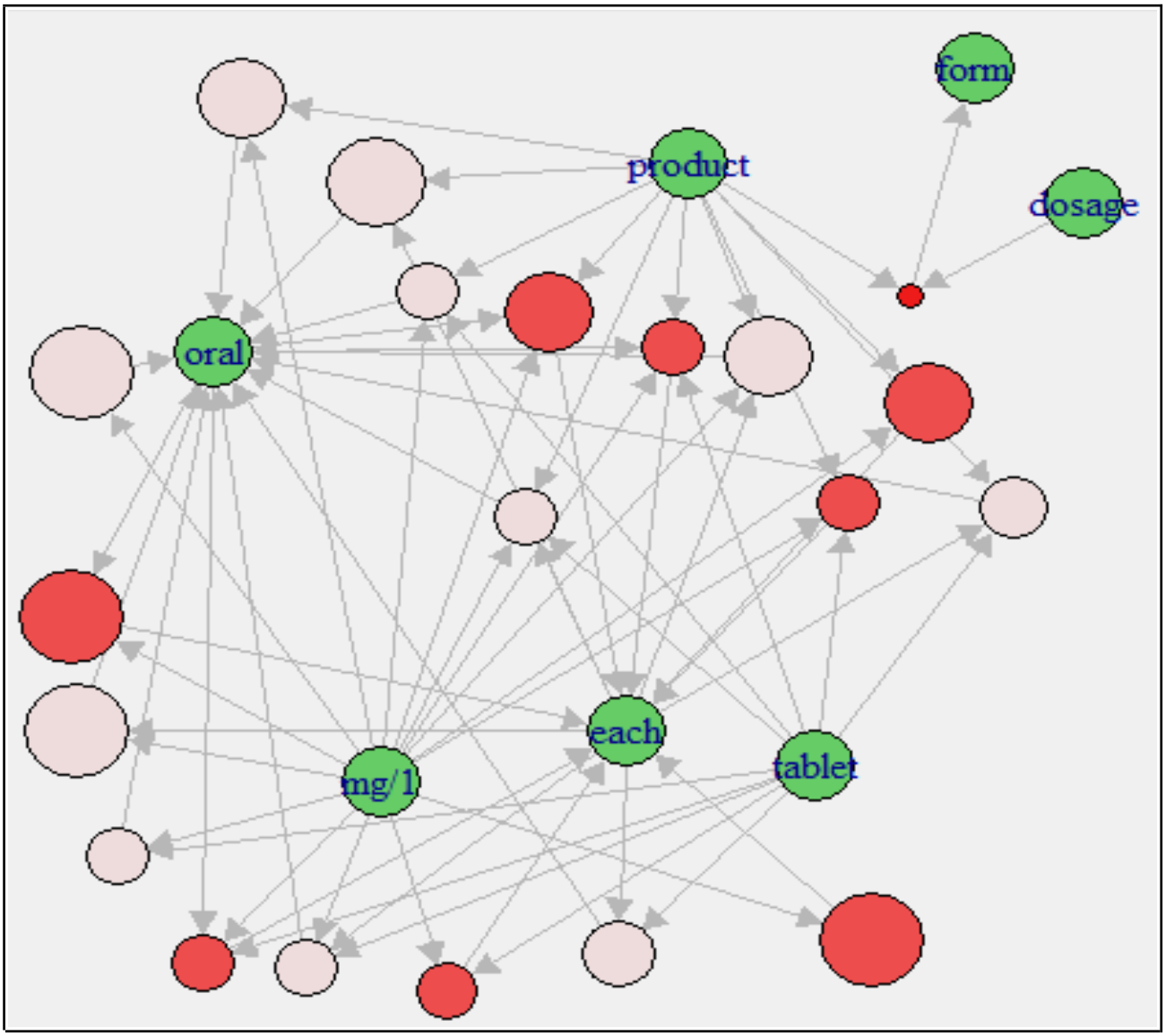


Figura 20: Grafo de las 20 mejores reglas de G1.

Podemos observar que estas reglas se forman a partir de las combinaciones de 5 palabras: "`product`", "`oral`", "`each`", "`tablet`", "`mg/1`". Además, el hecho de que la confianza sea 1 nos indica que siempre salen por lo que suponemos se trata de algún tipo de nombrado sistemático usado para

describir una clase en lenguaje natural. Al revisar los objetos de los que surgen estas reglas en SNOMED CT comprobamos que, efectivamente, muchos de las subclases de "`Pharmaceutical / biological product (product)`" contienen una estructura similar a "`Product containing` [...] `mg/1 each oral tablet (clinical drug)`" concretamente, todos aquellos en la jerarquía de "`Clinical drug`". Ejemplos de estos conceptos en SNOMED CT se encuentran en la figura 21.

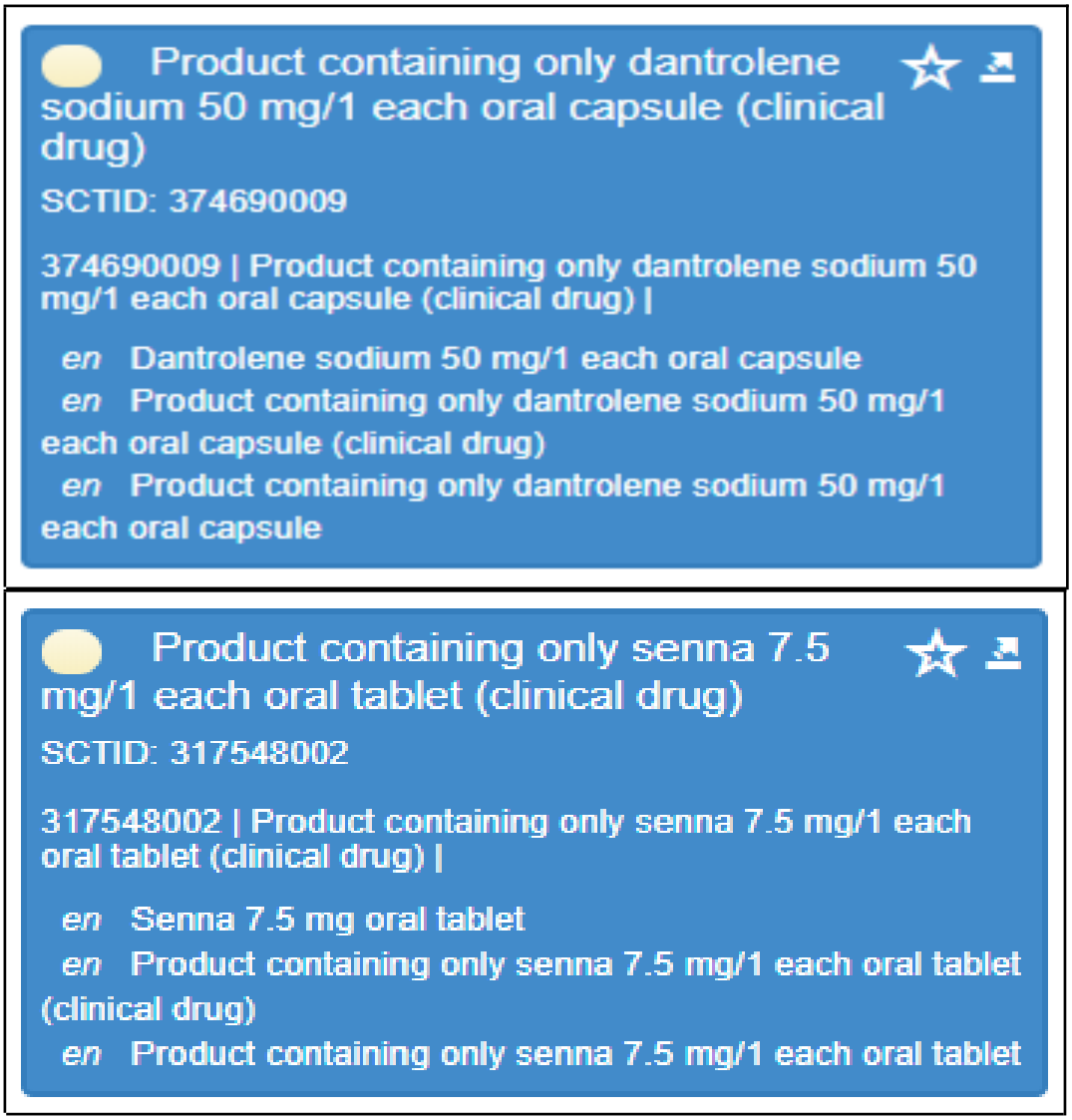


Figura 21: Dos clases que apoyan las reglas observadas en G1.

Además de estas palabras existen otras reglas con algo menos de soporte que están también relacionadas con este conjunto de clases. Estas reglas combinan las 5 palabras que ya hemos visto con otras como "`dosage`", "`form`" (que aparecen también en el grafo), "`clinical`", "`drug`" y "`hydrocloride`". Este conjunto de reglas con alta confianza que refleja un nombrado sistemático dentro de los productos farmacéuticos recogidos en SNOMED CT conforma 360 de las 444 reglas encontradas. Dentro G1 encontramos otros conjuntos de reglas de este estilo, formadas por combinaciones de palabras que siempre aparecen juntas en un nombrado sistemático tales como "`Antigen, blood, group`"y "`fluoroscopic, percutaneous, using, guidance`".

LHS	RHS	Support	Confidence	lift
{mg/1}	{each}	0.00778	1	116.988
{mg/1}	{oral}	0.00778	1	55.424
{each, mg/1}	{oral}	0.00778	1	55.424
{mg/1, oral}	{each}	0.00778	1	116.988
{each, product}	{oral}	0.00748	1	55.424
{mg/1, product}	{each}	0.0069	1	116.988
{mg/1, product}	{oral}	0.0069	1	55.424
{each, mg/1, product}	{oral}	0.0069	1	55.424
{mg/1, oral, product}	{each}	0.0069	1	116.988
{each, tablet}	{oral}	0.00579	1	55.424
{each, product, tablet}	{oral}	0.00568	1	55.424
{mg/1, tablet}	{each}	0.00524	1	116.988
{mg/1, tablet}	{oral}	0.00524	1	55.424
{each, mg/1, tablet}	{oral}	0.00524	1	55.424
{mg/1, oral, tablet}	{each}	0.00524	1	116.988
{mg/1, product, tablet}	{each}	0.00523	1	116.988
{mg/1, product, tablet}	{oral}	0.00523	1	55.424
{each, mg/1, product, tablet}	{oral}	0.00523	1	55.424
{mg/1, oral, product, tablet}	{each}	0.00523	1	116.988
{dosage, product}	{form}	0.00317	1	132.425

Tabla 2: 20 primeras reglas de G1.

Al aplicar los mismos umbrales de confianza (0.9) y soporte (0.001) en G2 obtenemos las 21 reglas mostradas en la tabla 3 y la figura 22. En este grupo ocurre el mismo fenómeno que en G1, se forman grupos de reglas con las mismas palabras. En la figura 22 se pueden apreciar claramente estos grupos. "`Observable`" y "`entity`" aparecen juntas en muchos casos, y tras comprobar las URIs vemos que hay muchas clases que pertencen a la jerarquía de "`(Observable entity)`" así que estas reglas no nos aportarían suficiente información. "`(Body structure)`" también es otra jerarquía que encontramos en algunas clases y de ahí nace el conjunto de reglas que se aprecia en el grafo. En G2 también está presente el mismo conjunto de palabras "`tablet`", "`each`", "`oral`" y "`product`" que en G1. El último conjunto de reglas que observamos en es el de "`substance`", "`group`", "`blood`" y "`antibody`". Estas reglas están asociadas a un conjunto de clases que describen todos los anticuerpos siguiendo la estructura "`Blood group antibody` [...] `(substance)`". Ejemplos de clases de estas reglas se pueden encontrar en la figura 23. Cabe destacar que al disminuir el soporte a 0.00001 las nuevas reglas que aparecen sólo expanden estos grupos.

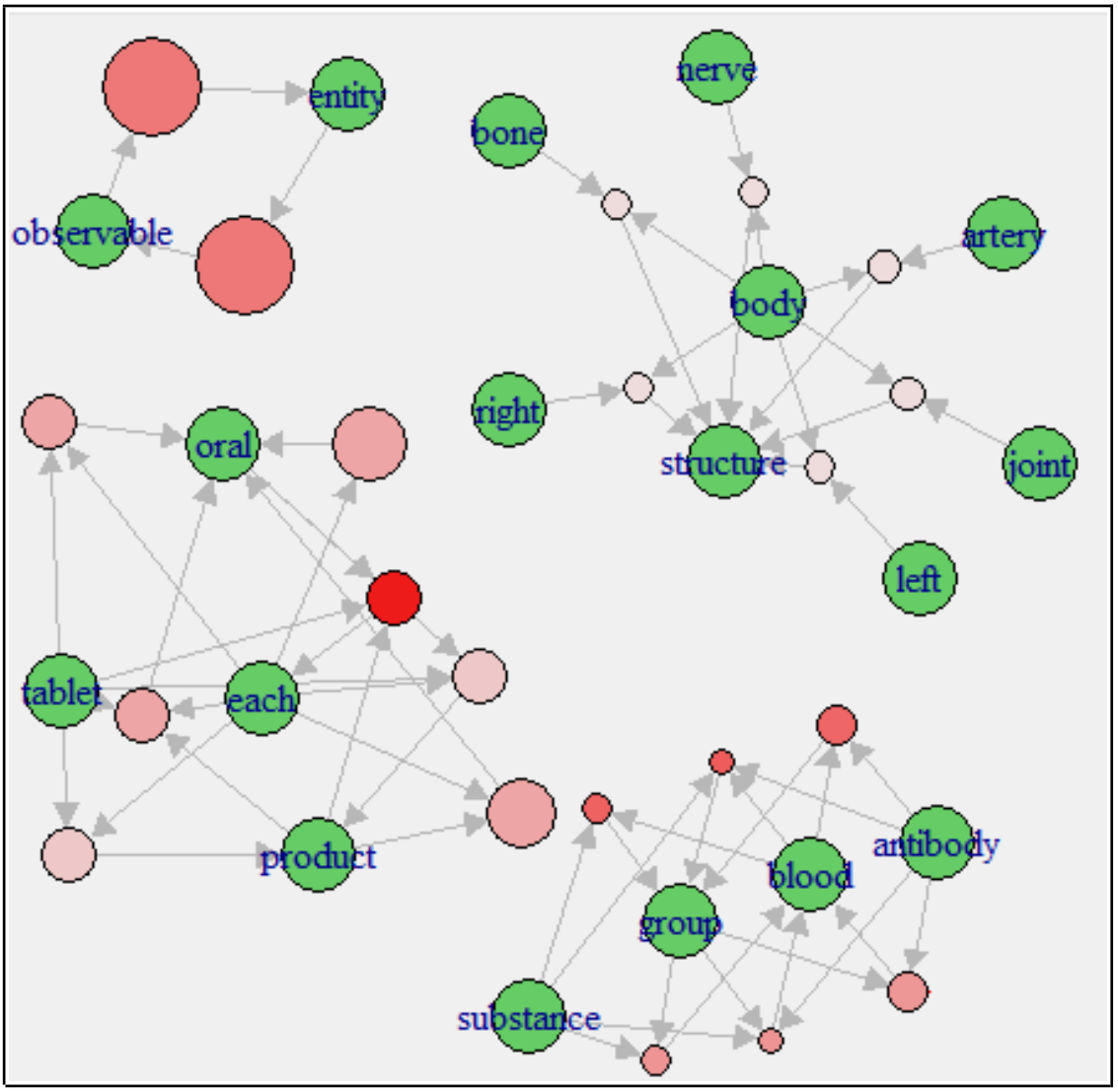


Figura 22: Grafo de las 21 reglas de G2.

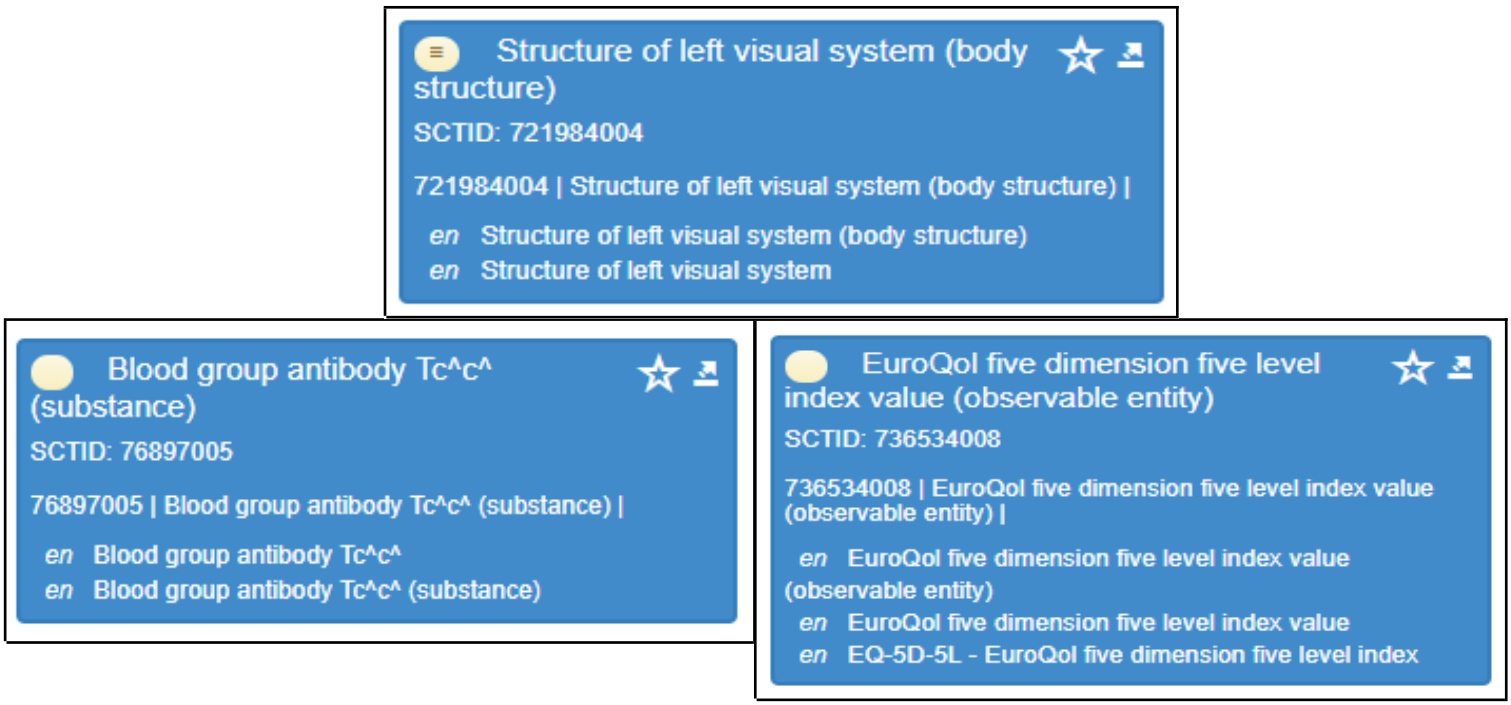


Figura 23: Ejemplos de URIs de las reglas de G2.

LHS	RHS	Support	Confidence	lift
{entity}	{observable}	0.0178	0.995	53.142
{observable}	{entity}	0.0178	0.952	53.142
{each}	{oral}	0.0125	0.996	37.372
{body, nerve}	{structure}	0.00273	0.976	10.174
{antibody, group}	{blood}	0.00442	0.974	44.628
{antibody, blood}	{group}	0.00442	0.952	58.297
{group, substance}	{blood}	0.00299	0.984	45.0631
{blood, substance}	{group}	0.00299	0.964	59.029
{body, bone}	{structure}	0.00277	0.962	10.028
{body, right}	{structure}	0.00277	0.972	10.134
{body, left}	{structure}	0.00296	0.907	9.454
{each, tablet}	{oral}	0.00855	1	37.517

LHS	RHS	Support	Confidence	lift
{each, tablet}	{product}	0.0084	0.981	19.801
{each, product}	{oral}	0.011	1	37.517
{body, joint}	{structure}	0.00321	0.913	9.522
{artery, body}	{structure}	0.0033	0.985	10.271
{antibody, group, substance}	{blood}	0.00152	0.976	44.721
{antibody, blood, substance}	{group}	0.00152	1	61.184
{each, oral, tablet}	{product}	0.0084	0.981	19.801
{each, product, tablet}	{oral}	0.0084	1	37.517
{oral, product, tablet}	{each}	0.0084	0.998	79.077

Tabla 3: Las 21 reglas de G2

En las tablas aparece señalado el atributo de “lift” y en los grafos la intensidad del color rojo también señala un lift más alto, pero todavía no hemos hablado de este valor. El lift, según [21], es un valor que señala aproximadamente lo descriptiva que es una regla en comparación con las otras. Esta aproximación se obtiene dividiendo la confianza entre la probabilidad de que aparezca el consecuente en una transacción, por lo que nos indica cómo de más probable es que nos encontremos el consecuente si en una transacción está el antecedente. La idea de este valor es usarlo comparándolo con el de las otras reglas obtenidas, y aquellas con un lift especialmente alto son muy descriptivas pues indican que el consecuente aparece casi exclusivamente cuando aparece el antecedente. Al ordenarlas por lift, lo que nos encontramos es que las reglas con un valor más alto son las que forman parte de uno de estos grupos de reglas nacidas del nombrado sistemático dentro de la ontología, puesto que muchas de las palabras que se usan son exclusivas de estos grupos de clases.

Conclusiones y líneas futuras

Conclusiones

Este trabajo propone un método en Java para procesar el lenguaje natural asociado a las clases de una ontología y obtener un conjunto de regularidades léxicas asociadas a las URIs de las clases, que pueden ser clasificadas según su frecuencia. Esta salida se enlaza con un método en R para obtener reglas de asociación a partir de las relaciones léxicas codificadas en la ontología. El análisis de estas reglas nos ha permitido detectar los grupos de clases que comparten un nombrado sistemático. En el caso de aquellas que incluyen los nombres de la jerarquía, los creadores de la ontología podrían conocer estas relaciones. Sin embargo, en otros casos no analizados podría ser interesante comparar las reglas con la representación formal a través de axiomas.

Volviendo a los objetivos del trabajo, hemos conseguido identificar con éxito patrones léxicos que representan conexiones entre clases de la ontología. Las reglas estudiadas en la realización del trabajo ya están representadas como axiomas dentro de la ontología, lo que nos indica que este sistema tiene capacidad para encontrar relaciones significativas. Por lo que hemos visto en este trabajo, podemos concluir que el Machine Learning ha demostrado ser de utilidad en el análisis de la información en la ontología.

Trabajo futuro

Como posible actividad futura se podría modificar la parte de procesamiento del lenguaje natural para que obtuviera conjuntos de palabras que se repitieran juntas como “`(body structure)`” o “`mg/1 each`”. Esto reduciría el número de tokens obtenido haciendo más fácil el procesamiento de las reglas y además eliminaría estos grupos de reglas redundantes. Otra posible mejora para el procesamiento del lenguaje natural sería incluir las palabras indicativas de una jerarquía, las primeras de la figura 17, en la lista de stopwords. Estas palabras normalmente aparecen entre paréntesis al final del nombre de una clase y sirven para clasificarlas por lo que podría argumentarse que no tienen un significado en sí mismas.

Otra actividad futura sería repetir estos análisis con un hardware más potente que pudiera manejar los grupos G3 y G4. Especialmente G3, que es un conjunto prometedor ya que incluye las palabras que no se repiten mucho por lo que no generará los grandes conjuntos de reglas consistentes en palabras reordenadas de distintas formas que se observan en G1 y G2. G4 estaría bien poder analizarlo para tener una visión general de toda la ontología.

Como última propuesta para futuro, una vez mejorado el procesamiento del lenguaje natural y con un hardware más potente, sería recurrir a alguien con experiencia clínica para confirmar nuestros descubrimientos. Una vez tengamos unas reglas sería el experto quien nos indicaría si los conceptos que relaciona están efectivamente conectados y debería formalizarse la relación o si es una mera coincidencia.

Conclusions and future activities

Conclusions

This work proposes a Java method to process the natural language associated with the classes of an ontology and obtaining a set of lexical regularities associated with the URIs of the classes, which can be classified according to their frequency. The output of this method is connected to another method in R to obtain association rules from the lexical relations encoded in the ontology. The analysis of these rules has allowed us to detect groups of classes that share a systematic naming system. In the case of those that include the names of the hierarchy, the creators of the ontology might know that these relationships exist. However, in other cases not analyzed it could be convenient to compare the rules with their formal representation through axioms.

Turning back to the objectives of this project, we have successfully identified lexical patterns that represent connections between classes in the ontology. The rules studied in the realization of this project are already represented as axioms within the ontology, which indicates that this system has the capability to find meaningful relationships. From what we have seen in, we can conclude that Machine Learning has proven to be useful in the analysis of information in ontology.

Future activities

As a possible future activity, the processing of the natural language could be modified to obtain sets of words that were repeated together such as "`(body structure)`" or "`mg/1 each`". This would reduce the number of tokens obtained by it, making it easier to process the rules and also eliminate these redundant rule groups. Another possible improvement for the processing of natural language would be to include the most used words of the ontology, the first ones of figure 17, in the list of stopwords. These words usually appear in brackets at the end of the name of a class and serve to classify them, so we could argue that they do not have a meaning for themselves.

Another future activity could be to repeat these analysis with better hardware that could handle the G3 and G4 groups. Especially G3, which is a promising set because it includes words that are not repeated too much so it will not generate large sets of rules consisting of reordered words in different ways that are observed in G1 and G2. Also would be good to be able to analyze G4 to have an overview of the ontology as a whole.

As a last proposal for the future, once the processing of the natural language has been improved and with a more powerful hardware, we should appeal to someone with clinical experience to confirm our discoveries. Once we have some rules the expert should indicate us if the concepts that are related in them are really connected and the relationship should be formalized or if it is a mere coincidence.

Presupuesto

En este capítulo se realiza una estimación aproximada del proyecto, en función de las horas dedicadas y de los recursos utilizados para su elaboración.

Horas de trabajo

Cada hora tiene un valor estimado de 10 euros.

Tipos	**Descripción**	**Horas**	**Costo(€)**
Tarea 1	Determinar las soluciones concretas de cada tecnología a incorporar en el proyecto.	120	1200
Tarea 2	Procesar los datos mediante las tecnologías seleccionadas.	80	800
Tarea 3	Analizar y visualizar los patrones obtenidos.	100	1000
Tarea 4	Desarrollar prueba de concepto (Proof of Concept - POC).	60	600
Tarea 5	Elaboración de la memoria final del Trabajo Final de Grado.	60	600
Total		320	3200

Tabla 4: Presupuesto de las horas

Software utilizado

Software	Costo(€)
Protégé 5	0
Eclipse Oxigen	0
OntoEnrich	0
Windows 10 Home	145*
Total:	0

Tabla 5: Presupuesto de software.

Hardware utilizado

Software	Costo(€)
Ordenador de sobremesa	450*
Total:	**0**

Tabla 6: Presupuesto de hardware

Presupuesto total

Concepto	Costo(€)
Software	0
Hardware	0
Horas de trabajo	3200
Total:	**3200**

Tabla 7: Presupuesto total

**Materiales de uso personal utilizados para la realización del trabajo. Como ya se disponía de ellos no hizo falta invertir en comprarlos y no se incluyen en el total.*

Referencias

[1] A free, open-source ontology editor and framework for building intelligent systems. Visto el 12/06/18 en https://protege.stanford.edu/

[2] Agrawal, R., Imieliński, T., & Swami, A. (1993, June). Mining association rules between sets of items in large databases. In Acm sigmod record (Vol. 22, No. 2, pp. 207-216). ACM.

[3] Aspectos básicos de SNOMED CT. Visto el 28/06/18 en https://confluence.ihtsdotools.org/pages/viewpage.action?pageId=47687570

[4] Berzal, F. (2015). Reglas de asociación. Departamento de Ciencias de la Computación e IA, Universidad de Granada.

[5] Borst, W. N., & Borst, W. N. (1997). Construction of Engineering Ontologies for Knowledge Sharing and Reuse Enschede: Centre for Telematics and Information Technology (CTIT)

[6] Chambers, J. M. Programming with data: A Guide to the S Language., 1998.

[7] ¿Conoces la librería CoreNLP de Stanford de procesamiento de lenguaje natural? Visto el 28/06/18 en http://data.cervantesvirtual.com/blog/2017/07/17/libreria-corenlp-de-stanford-de-procesamiento-lenguage-natural-reconocimiento-entidades/

[8] CoreNLP. Visto el 28/06/18 en https://stanfordnlp.github.io/CoreNLP/pipelines.html

[9] D. Lee, R. Cornet, F. Lau y N. De Keizer, «A survey of SNOMED CT implementations,» Journal of biomedical informatics, vol. 46, pp. 87-96, 2013.

[10] D. Sánchez y M. Batet, «Semantic similarity estimation in the biomedical domain: An ontology-based information-theoretic perspective,» Journal of biomedical informatics, vol. 44, pp. 749-759, 2011.

[11] Data Mining con R. Visto el 20/06/18 en http://apuntes-r.blogspot.com/2015/07/reglas-de-asociacion.html

[12] De Sr. Alvaro - Trabajo propio, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=43736646

[13] Extensible Markup Language. Visto el 12/06/18 en https://www.w3.org/XML/

[14] Gennari, J. H., Musen, M. A., Fergerson, R. W., Grosso, W. E., Crubézy, M., Eriksson, H., ... & Tu, S. W. (2003). The evolution of Protégé: an environment for knowledge-based systems development. International Journal of Human-computer studies, 58(1), 89-123.

[15] Horridge, M., & Bechhofer, S. (2011). The owl api: A java api for owl ontologies. Semantic Web, 2(1), 11-21.

[16] Hyvärinen, A., Gutmann, M., & Entner, D. (2010). Unsupervised Machine Learning.

[17] Ihaka, R., & Gentleman, R. (1996). R: a language for data analysis and graphics. Journal of computational and graphical statistics, 5(3), 299-314.

[18] J. P. Bona y W. Ceusters, «Mismatches between major subhierarchies and semantic tags in SNOMED CT,» Journal of biomedical informatics, vol. 81, pp. 1-15, 2018.

[19] Java. Visto el 12/06/18 en https://java.com/es/

[20] Kotsiantis, S. B., Zaharakis, I., & Pintelas, P. (2007). Supervised Machine Learning: A review of classification techniques. Emerging artificial intelligence applications in computer engineering, 160, 3-24.

[21] Lift (data mining). Visto el 21/06/18 en https://en.wikipedia.org/wiki/Lift_(data_mining)

[22] M. Quesada-Martínez, D. Castellanos-Nieves y J. T. Fernández-Breis, «OntoEnrich: Una plataforma para el análisis léxico de ontologías orientado a su enriquecimiento axiomático», Procesamiento del Lenguaje Natural, vol. 59, pp. 171-174, 2017.

[23] Musen, M. A. (2015). The protégé project: a look back and a look forward. AI matters, 1(4), 4-12.

[24] Oberle, D., Guarino, N., & Staab, S. (2009) What is an ontology? In: "Handbook on Ontologies". Springer, 2nd edition, 2009.

[25] OWL 2 Web Ontology Language Document Overview (Second Edition). Visto el 12/06/18 en https://www.w3.org/TR/2012/REC-owl2-overview-20121211/

[26] OWL Reasoning Examples. Visto el 28/06/18 en http://owl.man.ac.uk/2003/why/latest/

[27] Package ‘arules’. Visto el 20/06/18 en https://cran.r-project.org/web/packages/arules/arules.pdf

[28] Package ‘arulesViz’. Visto el 20/06/18 en https://cran.r-project.org/web/packages/arulesViz/arulesViz.pdf

[29] Palabra vacía. Visto el 20/06/18 en https://es.wikipedia.org/wiki/Palabra_vacía

[30] Prof. Edwin Saldaña Ambulódegui «Manual de Terminología médica» 2012.

[31] R.S. Michalski, J.G. Carbonell and T.M. Mitchell «Machine Learning: An artificial intelligence approach» 1983.

[32] Resource Description Language. Visto el 12/06/18 en https://www.w3.org/RDF/

[33] Rstudio. Visto el 16/06/18 en https://www.rstudio.com/products/rstudio/

[34] SPARQL Query Language for RDF. Visto el 28/06/18 en https://www.w3.org/TR/rdf-sparql-query/

[35] The OWLAPI. Visto el 12/06/18 en http://owlapi.sourceforge.net/

[36] The platform for open innovation and colaboration. Visto el 12/06/18 en https://www.eclipse.org/

[37] The R project for Statistical Computing. Visto el 16/06/18 en https://www.r-project.org

[38] Uri Standard. Visto el 28/06/18 en https://confluence.ihtsdotools.org/display/DOCURI/URI+Standard

[39] Urma, R. G., Fusco, M., & Mycroft, A. (2014). Java 8 in action: lambdas, streams, and functional-style programming. Manning Publications Co.

[40] Vásquez, A. C., Quispe, J. P., & Huayna, A. M. (2009). Procesamiento de lenguaje natural. Revista de investigación de Sistemas e Informática, 6(2), 45-54.

[41] W. Ceusters, B. Smith, A. Kumar y C. Dhaen, «Ontology-based error detection in SNOMED-CT,» 2004.

[42] W3C. Visto el 12/06/18 en https://www.w3.org

[43] Web Ontology Language (OWL). Visto el 12/06/18 en https://www.w3.org/OWL/

[44] Welcome to Apache Maven. Visto el 12/06/18 en https://maven.apache.org/

[45] Welcome to Apache OpenNLP. Visto el 28/06/18 en https://opennlp.apache.org/

[46] Welcome to Snomed International. Visto el 12/06/18 en http://www.snomed.org/