La Web Semántica como plataforma para sistemas de recomendación

Guido Zuccarelli

17 de abril de 2015

Índice general

Capítulo 1

Introducción

1.1. Evaluaciones y opiniones

Llamaremos ítem, a un elemento del mundo real que es aprovechado por las personas, dicho ítem puede ser, un objeto, un servicio, una idea, un programa, etc. Además, debería poder ser abstraído a un modelo representado por una computadora, de manera tal, que esa representación describa con precisamente a qué se referie ante la interpretación del lector.
Para ello, el modelo del ítem contendrá un conjunto de atributos que lo describen. Algunos más importantes que otros.

Por ejemplo, imaginemos que se debe representar una bicicleta mediante un conjunto de pares atributo;valor, tendríamos algo como esto.

```
Tipo: Bicicleta
Sexo: Unisex
Talle: 51
Cuadro-Material: Aluminio
Cuadro-color: Rojo
Cuadro-Tipo: Ruta
Horquilla-Material: Fibra de carbono
Horquilla-Color: Rojo
Asiento-tipo: Adamo
Asiento-Color: Blanco
```

La cantidad de atributos que pueden utilizarse para describir el ítem puede ampliarse prácticamente hasta el cansancio.

Se pueden optar también por una representación como ésta:

```
Tipo: Bicicleta
Talle: 51
Marca: Merida
```

Figura 1.1: Relación entre usuario, review e ítem

```
Modelo: Reacto 500
```

Puede aprovecharse el hecho de que es muy probable que ya existan definiciones del ítem que intentamos describir, por lo que si se utilizan sólo los atributos que lo identifican, será suficiente para que el lector tenga la interpretación correcta del mismo.

Cada ítem es de alguna manera utilizado por uno o más usuarios. Y cada usuario puede de la misma manera ser representado en la computadora mediante pares de atributo;valor. Y al igual que los ítems algunos atributos servirán para identificar al usuario.

Definiremos a una reseña (de aquí en adelante review) a la representación mediante la computadora, de la medida de conformidad con respecto a dicha relación entre usuario e ítem.

Esta relación de uso entre usuario e ítem, contiene un grado de conformidad entre el primero y el segundo, si el ítem cumplió o no con las expectativas del usuario debería poder modelarse también para ser representado computacionalmente.

El objetivo de un review, es que un usuario pueda reflejar el sentimiento que le generó utilizar el ítem e informarlo a otros usuarios.

También los reviews dispondrán de un conjunto de atributos de los cuales algunos serán indispensables y otros que enriquecerán no sólo su valor intrínseco, sino también su utilidad para el contexto en el que será planteado.

Imaginemos un usuario que realiza un review sobre la bicicleta, podría generar el siguiente review:

```
Puntuacion:4
T tulo:``Buena opcion''
Texto:``Liviana y comoda, pero un poco rigida para
       doblar, y no frena adecuadamente ''
Fecha:15/10/2012
```

La existencia de los reviews en la web es muy importante, ya que dan un panorama socialmente perfeccionado (dado que permiten que no sólo una persona opine sobre un ítem, lo que da a lugar a muchas evaluaciones y provoca que se acerque al valor real) sobre un ítem, lo que ayuda a quienes necesiten

una descripción y valoración de los mismos que no provenga de quien los quiere promocionar.

> Al terminar de leer esta sección, el lector debe entender a que te referís con reviews (en términos generales), que muchas de ellas son publicadas en la web en redes sociales, en sitios de productos etc, y que sirven para ??. Podés agregar algunos ejemplos e incluso imágenes. No queda claro por que en el titulo de la sección dice "entrecruzadas", ¿es importante o fue solo una elección al azar?. Para que este se conecte con el que sigue, podés dejar picando algo como "muchos han intentado procesar automáticamente las opiniones de los usuarios para ... pero eso es muy difícil porque..."

1.2. Sistemas de Recomendación

Los sistemas de recomendación son herramientas de software que, en base a un conjunto de ítems (películas, libros, productos, hoteles, etc) e información sobre estos, y un conjunto de usuarios, intentan sugerir ítems apropiados a dichos usuarios. Los sistemas de recomendación se han vuelto una de las herramientas más poderosas para múltiples tipos de aplicaciones web, como comercio electrónico o páginas de noticias.

El desarrollo de estas herramientas, involucra conocimiento en múltiples áreas, como inteligencia artificial, minería de datos, estadística, etc.

Los sistemas de recomendación poseen dos enfoques, el de filtrado colaborativo, y el basado en contenido.

Los sistemas de recomendación basados en contenido utilizan un conjunto de informaciones y descripciones de los ítems previamente valorados por un usuario para poder construir un perfil del mismo de manera tal de poder determinar, cuales son sus intereses.

Una vez construido el perfil, pueden procesarse las características de distintos ítems que potencialmente pueden ser recomendados a dicho usuario para determinar si alguno de ellos va acorde al perfil.

Los sistemas de recomendación de filtrado colaborativo utilizan la información histórica de cada usuario para intentar encontrar y generar grupos de usuarios con gustos similares. Para lograrlo se compara cada usuario con otro observando qué ítems evaluaron y qué puntajes fueron otorgados.

De esta forma, si se requiere predecir el interés de un usuario en un ítem, se podrá buscar en dicho grupo de usuarios con gustos similares e inspeccionar aquellos usuarios que hayan realizado un review sobre el ítem.

Los sistemas de recomendación de filtrado colaborativo tienen una eficacia su-

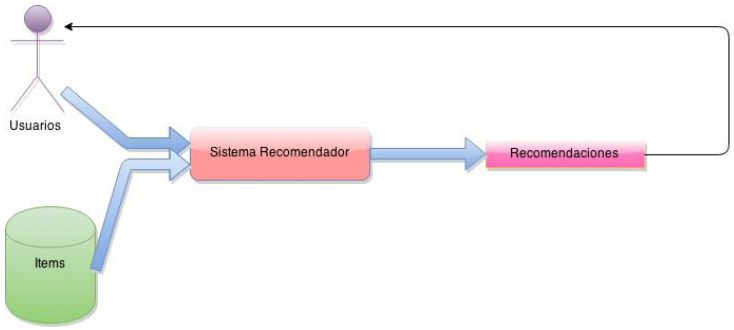

Figura 1.2: Flujo de un Sistema de Recomendación

perior a los basados en contenido, pero cuentan con una gran desventaja, no permiten predecir intereses para aquellos ítems que aún no poseen evualuaciones, o también para aquellos usuarios que no han evualuado ningún ítem. A esto se lo llama "arranque en frío".

1.3. La web semántica

En sus comienzos en los 90, la web podía verse como un conjunto de sitios web que ofrecían una colección de documentos web interconectados mediante la hipermedia, con el objetivo de comunicar información a los usuarios. El contenido de esos documentos sólo era generado por el mismo creador y publicador del documento, y los usuarios se limitaban a consumirlo. Por otro lado, era a su vez estático, es decir, se publicaba en la misma forma que se almacenaba y no cambiaba.

Hacia fines de los 90 los sitios web comenzaron a implementar una serie de herramientas (que si bien ya se encontraban disponibles anteriormente no se utilizaban por un problema de performance) que permitieron a los usuarios finalmente participar de la producción del contenido web. Lo que produjo notorios cambios en cuanto a la cantidad de información disponible y proveyó diferentes formas de uso de la web (blogs, redes sociales, canales rss, etc). Más adelante ante la apreciación del pasaje de web estática a una web dinámica, se acuño a esa actual web como web 2.0 y retrónimamente 1.0 a la anterior.

Ese cambio provocó un aumento en el tamaño de la web, que se volvió inmenzamente grande, y llevó a la necesidad de implementar tecnologías que ayuden al aprovechamiento de esa cantidad de información. Se comenzó entonces a utilizar una serie de frameworks y estándares que permitieron enriquecer (mediante metadatos semánticos y ontológicos dentor de los estándares de la W3C) los datos contenidos en los documentos de manera tal que estos puedan ser consumidos, interpretados y utilizados directamente no sólo por las personas, sino

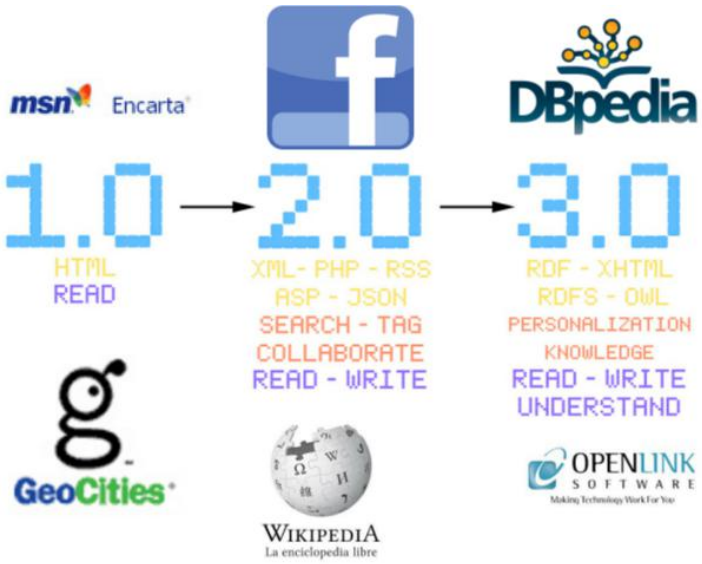

Figura 1.3: Evolución de la web

también por las computadoras. Esto generó que los datos también puedan ser relacionados entre sí, de la misma manera que los documentos son interconectados formando una web de documentos, los datos interconectados forman una web de datos parelela. Todo este conjunto de actividades frameworks y herramientas forman la "Web semántica" que es el puntapié incial para una web mucho más interoperable, lo que permite facilidades para el uso de la web por parte de las aplicaciones y da lugar a otro paso en la evolución de la web, la web 3.0. La Web Semántica promete facilitar el desarrollo de la web social y la inteligencia colectiva, a través de mecanismos de clasificación, relación y descripción del contenido de la información publicada mejorando su interoperabilidad. Promete también mejorar la recolección, agregación e integración de datos específicos mediante el uso de agentes de búsqueda automáticos que aprovechan las ventajas.

Al terminar de leer esta sección, el lector debe entender que es la web semántica y a que apunta. Si dejaste picando el tema de automatización en el párrafo anterior, acá se va a imaginar porque hablás de ws. Habla muy bremente de tripletas y meciona RDF. También mencioná linked open data para darle una idea de que la web semantica es una web de datos interconectada. Podés tomar lo que ya escribiste en la propuesta. Ejemplifica con dbpedia o algo asi... En un capitulo mas adelante vas a entrar en detalle en web semantica, rdf, etc. Acá contás solo lo suficiente para que se entienda lo que vas a proponer en la próxima sección

1.4. Reviews en la web semántica

La web 2.0 dio la posibilidad a los usuarios consumidores de la web de generar y publicar contenido en la misma, lo cual cumple con los requerimientos de una plataforma para reviews. Proporciona un entorno para crearlos y publicarlos, lo cual en el caso es una tarea muy sensilla. Pero la dificultad de encontrarlos y explotarlos parece ser inversamente proporcional a generarlos. Dado que a mayor facilidad de publicar contenido en la web, mayor se vuelve la inmensidad de la misma y mayor se vuelve la dificultad de encontrar algo dentro de ella.

Veamos dos ejemplos que requieren encontrar reviews en la web:

Imagínense que son ingenieros de Mérida y lanzaron al mercado la bicicleta Reacto 500. Como buenos ingenieros necesitan conocer qué opinan sus clientes, por lo que buscarán reviews en algún motor de búsqueda y luego leerán uno por uno cada review con el objetivo de resumir las opiniones. Humanamente esta tarea demandará mucho tiempo y con un límite corto de cantidad de reviews. Ahora peor aún, imagínense que necesitan saber que opina la gente de determinada región (por ejemplo el norte de europa), la tarea de buscar los reviews necesarios se volvería aún más complicada. Con la posibilidad de contar con una web en la cual los datos son interpretados por las aplicaciones de software y estos a su vez están interconectados, podría crearse una aplicación que pueda automáticamente buscar, clasificar y procesar los reviews para generar automáticamente el resumen.

Ahora bien, si en lugar de requerir reviews de un ítem en particular, se necesitan reviews para una aplicación que recomiende ítems a usuarios, ya no sería una tarea realizable humanamente. Para ello haría falta una aplicación que haga crawling en la web y de alguna manera identifique reviews y a su vez identifique el ítem al cual el review hace referencia. Parece algo muy difícil de lograr, salvo que se trabaje sobre sitios conocidos con documentos estructuralmente dominados los cuales se pueda recorrer el DOM automáticamente y acceder a los reviews. De nuevo, la Web Semántica promete solucionar este problema, con el uso de metadatos que dan información sobre los datos, haciendo que una aplicación pueda fácilmente identificar reviews y navegar por la hipermedia de los datos para conseguir información sobre el ítem y usuario referenciados.

Acá es donde presentas el problema a resolver. Ya contaste que son los reviews y por que alguien querría integrarlos. Ya contaste que es la web semántica y linked open data. Ahora tenés que contar que ha habido iniciativas para darle semántica a los reviews y que existen datos; ahora hay una posibilidad de aprovechar esa información. Y ahí decís algo como lo que dijiste al final de la propuesta: *El objetivo principal de esta tesis es evaluar la viabilidad, y entender los desafíos de la utilización de la información contenida en la web semántica en la construcción de sistemas de recomendación. Para eso, y con foco en el caso particular de opiniones de usuarios sobre distintos tipos de recursos se buscará: capturar, extraer, val-*

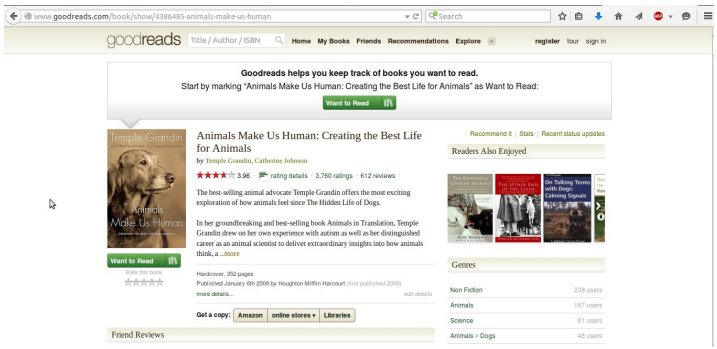

Figura 1.4: Ejemplo de review en la web semántica

idar calidad, curar, integrar, publicar y explotar los datos disponibles.

1.5. Organización

ya saben lo que vas a contar; acá les das una idea de como te vas a organizar para contarlo - puede ser algo como lo que está a continuación. Para el caso de estudio podés tomar el texto que escribimos en el articulo

El capítulo 2 presenta una aplicación de ejemplo que muestra claramente el problema que se quiere resolver y que servirá como referencia a lo largo de esta tesis. El capítulo 3 introduce los conceptos de Web Semantica, sus principios y tecnologías y presenta la estrategia de trabajo en términos generales. Los capítulos 4 a 11 discuten en detalle cada uno de pasos de la estrategia elegida, aplicándolos al caso de estudio: los reviews en la web semántica. Finalmente, el capítulo 12 resumen los resultados observados, saca conclusiones al respecto, y plantea trabajo futuro. El anexo 13 presenta una publicación que fue resultado del trabajo efectuado en este trabajo de tesis.

Capítulo 2

Caso de Estudio

Capítulo 3

Enfoque General

Lo que vamos a construir es un caso de aplicación de la web semantica....

3.1. Aplicaciones en la Web Semantica

3.1.1. DBpedia

La aplicación más importante de la Web Semántica, que tiene como objetivo extraer información de wikipedia, y crear una versión semántica de la información. Para ello creó una serie de ontologías, donde modelan y relacionan la información extraída de wikipedia.

Dbpedia en la actualidad es reconocido como el sitio autoritativo de recursos de la web semántica, siendo las URIs de estos recursos la mejor forma de identificarlos. Es por eso que la gran mayoría de las aplicaciones de la web semántica utilizan la información de dbpedia para enriquecer sus datasets. En Septiembre de 2013, se contaron mas de 45 millones de links entre DBpedia y datasets externos como Freebase, OpenCyc, UMBEL, GeoNames, Musicbrainz, etc.

3.1.2. Revyu

Es una aplicación que se utiliza para publicar reviews de cualquier tipo en RDF. Alentando al usuario a generar reviews en la Web Semántica sin la necesidad de conocimiento sobre la misma.

Tiene el objetivo de proveer información reusable para otras aplicaciones, y generar datos de confianza y recomendación en las redes sociales.

La información es publicada tanto en documentos HTML como RDF además de su SPARQL Endpoint, siendo así de utilidad para los distintos tipos de usuarios. También alienta el uso de links a distintos dataset semánticos, como el de DBpedia, RDF Book Mashup, etc.

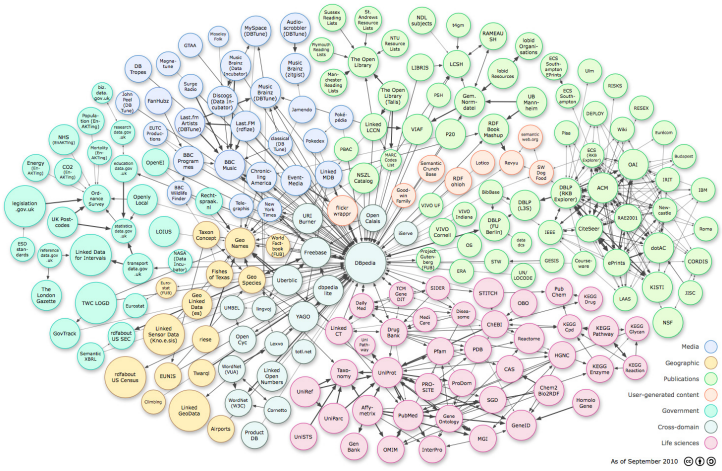

Figura 3.1: La utilización de DBpedia como fuente de datos

3.2. Principios

3.2.1. Resource Description Framework (RDF)

Es una familia de especificaciones de WC3 diseñada para modelar datos intercambiables en la web. Consta de expresiones hechas por tripletas. Cada tripleta es considerada una sentencia para este lenguaje. Una tripleta es un conjunto de un valor de cada uno de los siguientes tipos:

Recurso (sujeto) Todo aquello descripto por RDF se lo denomina recurso, este último podría ser por ejemplo una página web entera (por ejemplo un documento HTML "http://www.w3.org/Overview.html") o una parte de ella. Un recurso puede ser también un objeto que no se puede acceder directamente a través de la Web; por ejemplo un libro impreso. Los recursos son siempre nombrados por las URIs más identificadores de anclaje opcionales (ver [URI]). Cualquier cosa puede tener un URI; la extensibilidad de URIs permite la introducción de identificadores para cualquier entidad imaginable.

Propiedad (predicado) Una propiedad es un aspecto específico, característica, atributo o relación se utiliza para describir un recurso. Cada propiedad tiene un significado específico, define sus valores permitidos, los tipos de recursos que se puede describir, y su relación con otras propiedades.

Objeto Puede ser otro recurso o puede ser un literal; es decir, un recurso (especificado por un URI) o una cadena simple o de otro tipo de datos primitivo definido por XML. En términos RDF, un literal puede tener contenido que es el formato XML, pero no se evalúa aún más por el procesador de RDF.

Esta estructura forma una vinculación dirigida de un gráfico etiquetado, donde las aristas representan el enlace entre dos recursos (propiedad), representados por los nodos del grafo.

Su representación puede ser asociada tanto a un modelo entidad-relación como a un diagrama de clases de objetos. Las Propiedades RDF pueden ser considerados como atributos de los recursos y en este sentido corresponden a pares atributo-valor tradicionales. Las Propiedades RDF también representan las relaciones entre los recursos y un modelo RDF, por tanto, pueden parecerse a los de un diagrama entidad-relación. En la terminología de diseño orientado a objetos, los recursos corresponden a los objetos y propiedades corresponden a las variables de instancia.

3.2.2. RDF Schema (RDFS)

Es una extensión semántica de RDF, que provee elementos básicos para la descripción de ontologías (también llamadas vocabularios RDF) con el objetivo de estructurar los recursos RDF. Proporciona la manera de definir las clases y las propiedades específicas de la aplicación en RDF. RDFS no proporciona las clases sino que proporciona el marco necesario para poder definirlas.

Clases en RDFS se parecen mucho a las clases en lenguajes orientados a objetos. Esto permite que los recursos se definan como instancias de clases y subclases de las clases.

Elementos básicos

- rdfs:Class permite declarar recursos como clases para otros recursos.

  Atravez de la propiedad rdf:type, se puede asignar un tipo al recurso, un rucurso de tipo rdfs:Class puede entonces ser tipo de otro recurso. Por ejemplo, podemos definir el recurso Auto como un rdfs:Class, y a su vez podemos definir el recurso Volkswagen Gol con la propiedad rdf:type y siendo el objeto de esa propiedad el recurso Auto, entonces el tipo de Volkswagen Gol será auto.

  La defición de rdfs:Class es recursiva: rdfs:Class es la rdfs:Class de cualquier rdfs:Class.

- rdfs:Resource es la clase a la que pertenecen todos los recursos.

- rdfs:Literal es la clase de todos los valores literales, cadenas y enteros.

- rdfs:Datatype es la clase que abarca los tipos de datos definidos en el modelo RDF.

- rdfs:Property es la clase que abarca las propiedades.

Elementos que definen relaciones

- rdfs:subClassOf es una instancia de rdf:Property (osea una propiedad) que permite definir jerarquías. Relaciona una clase con sus superclases.
- rdfs:subPropertyOf es una instancia de rdf:Property que permite definir jerarquías de propiedades.

Restricciones de Propiedades

- rdfs:domain es una instancia de rdf:Property que especifica el dominio de una propiedad P. Esto es, la clase de los recursos que aparecen como sujetos en las tripletas donde P es predicado.
- rdfs:range es una instancia de rdf:Property que especifica el rango de una propiedad P. Esto es, la clase de los recursos que aparecen como objetos en las tripletas donde P es predicado.

3.2.3. Vocabularios

Son una colección de clases y propiedades con sus respectivas URIs, que son utilizadas para describir significado. Dichos vocabularios pueden ser fácilmente creados por cualquier usuario, sólo es necesario darle uso a la URI y documentarlo en algún lado. Por lo general, son estándares de uso no restrictivos de propiedades y clases que poseén una documentación. El objetivo de los vocabularios es poder describir el significado de los recursos web y del mundo real, modelándolos en clases y propiedades que se utilizarán para describir al objeto en cuestión.

3.2.4. OWL

Web Ontology Languaje: Es un vocabulario que funciona como lenguaje para crear ontologías en la web. Fue creado en el 2006 y aceptado por WC3 como estándar en 2007 en su versión OWL 1.1. Tiene por objetivo proveer reglas a los vocabualrios para aumentar su expresividad. Estas reglas incluyen: jerarquías de clasess, cardinalidad, relaciones de conjunto, tipologías de propiedades más complejas, etc. Lo que proporciona la posibilidad de crear ontologías que pueden representarse en un paradigma muy parecido a la orientación a objetos. Cabe destacar que OWL no es el único vocabulario con este objetivo, ya que RDFS es también un conjunto de estándares mucho más básico que permite estas características. En la actualidad existe la versión 2 de OWL que es mucho más completa.

3.2.5. Ontologías

La aparición de OWL permitió darle mucha más expresividad y restricciones a los vocabularios, que ahora no sólo eran un conjunto de propiedades y clases,

sino que podían tener una definición mucho más completa, que permitía describir taxonomías y relaciones entre clases que pueden formar axiomas. “An ontology is a formal specification of a shared conceptualization” Tom Grubber (cita requerida) Las ontologías se han convertido en uno de los pilares de la web semántica, esto se debe a que es uno de los principales componentes que permiten cumplir con el objetivo de ésta, que fue planteado en la introducción: “La Web Semántica promete facilitar el desarrollo de la web social y la inteligencia colectiva, a través de mecanismos de clasificación, relación y descripción del contenido de la información publicada mejorando su interoperabilidad.”

3.2.6. Motores de inferencias.

Son componentes de software que pertimente establecer consecuencias lógicas en base a los axiomas definidos tanto por OWL como por RDFS Estos componentes tienen como objetivo enriquecer las ontologías, creando reglas de inferencia. Dichas inferencias pueden ser más o menos complejas dependiendo de la capacidad y la configuración del motor utilizado. Existen tanto gratuitos como pagos.

3.2.7. SPARQL

Se trata de un lenguaje estandarizado de consulta de grafos RDF. SPARQL se puede utilizar para expresar consultas que permiten interrogar diversas fuentes de datos (si es que los datos fueron almacenados de forma nativa como RDF o fueron definidos mediante vistas a RDF a travez de algún middleware). Tiene capacidades para la consulta de los patrones obligatorios y opciones de grafo, junto con sus conjunciones y disyunciones. SPARQL también soporta la ampliación o restricción del ámbito de las consultas indicando los grafos sobre los que opera (grafos con nombre). Los resultados de las consultas SPARQL pueden ser conjuntos de resultados o grafos RDF.

Términos de conjuntos de colecciones de datos semánticos

Model: Colección de sentencias.

DataSource: Colección de Modelos. Siendo uno de ellos el modelo por defecto y el resto modelos con nombre. El Datasource es de lectura y escritura, por lo que se pueden agregar tripletas.

Dataset: Lo mismo que el DataSource pero estático, por lo que es de solo lectura.

Graph: Colección de tripletas. Cualquier modelo puede ser transformado a un grafo, y así tener una representación más cercana de RDF.

DatasetGraph: Un contenedor de Grafos, similar al DataSource (También de lectura y escritura) que provee una estructura para un grafo por defecto y grafos con nombre.

3.2.8. HTML semántico

El uso y participación en la Web Semántica a travez de documentos RDF no es atractivo para la mayoría, porque no parece proveer un beneficio directo, ya que esto implica, aprender un lenguaje nuevo para generar documentos que sólo le será de utilidad a usuarios con un grado de conocimiento muy alto.

Para resolver este problema, existen etiquetas de marcado en HTML que le proporcionan información semántica al documento, que copmo se verá más adelante puede ser extraída para generar documentos RDF. Existen distintos tipos:

Microdatos: Son un grupo de pares nombre-valor anidados dentro del documento, en paralelo con el contenido existente. Dichos grupo son también llamados ítems y cada par nombre-valor es la propiedad. Los atributos más utilizados son:

- itemscope: Se utiliza para crear un ítem
- itemprop: Se utiliza para crear una propiedad
- a, href, src, img: Se utilizan para definir valores URL
- value: Se utilizan para definir valores String
- itemtype: Se utiliza para establecer el tipo a un ítem.

Microformatos: Nacen con la intención acercar más personas al uso de la web semántica a travez de una forma sensilla de utilizar una serie de atributos de las etiquetas de XHTML:

- class: especifica el nombre de la clase
- rel: Se utiliza en los enlaces para indicar relaciones en los documentos.
- rev: Igual al rev pero en sentido inverso.

Con el uso del atributo class se puede entonces especificar un microformato a utilizar. Existen muchos tipos como por ejemplo: hCard (para describir personas), hreview (para describir reviews), geo (para describir posiciones), etc. Cada uno de ellos con distintas propiedades.

RDF-a: Permite directamente embeber tripletas (sujeto-predicado-objeto) utilizando namespaces de XML. Modela la información en forma de grafo, a diferencia de microdatos y microformatos que lo hacen en forma de árbol, lo que hace que el mapeo a RDF sea claro, ya que en los otros casos puede ser problemático. También tiene la opción de darle tipo a los literales. Como contrapartida tiene la desventaja de ser mucho más complejo de utilizar.

3.2.9. Extractores

Son herramientas que permiten extraer tripletas de los lenguajes de metadatos semánticos embebidos en HTML vistos anteriormente. Utilizan estándares como GRDDL, parseando todo el documento HTML y buscan sentencias para traducir a RDF y generar un nuevo documento que contiene sólo las tripletas de información semántica. Muchas veces (sobre todo con microformatos) los extractores tienen que tomar decisiones sobre cómo traducir a RDF, por lo que un mismo documento HTML puede ser extraído de distintas maneras según el extractor. Esto no ocurre con RDFa, ya que la equivalencia a RDF es mucho más clara.

3.2.10. Dublin core

Es un vocabulario ampliamente utilizado para describir tanto recursos web como recursos físicos creado en 2012. Comenzó siento un conjunto de 15 propiedades

Title	Publisher	Format
Creator	Contributor	Identifier
Subject	Date	Source
Description	Type	Language
Relation	Coverage	Rights

Consta también de Dublin Core Metadata Initiative que provee un foro de desarrollo de estándares de metadatos interoperable. En la actualidad, se ha extendido a 55 propiedades y 22 clases.

3.3. Tecnologías

Triplestores, frameworks, extractores, crawlers, motores de busqueda semanticos...

3.4. Estrategia propuesta

Con el fin de crear una aplicación que satisfaga los requerimientos mensionados anteriormente, se debe encontrar un procedimiento que incluya desde obtener los datos relevantes de la web hasta llevarlos a un estado que permita una explotación satisfactoria.

El procedimiento debería incluir los siguientes pasos

Selección de vocabularios Tanto la recolección como la publicación de datos en la web semántica involucra la elección del o de los vocabulario/s que mejor modelen el dominio de problema, con la excepción que para su publicación existe la posibilidad de desarrollar uno propio que se ajuste correctamente en caso de que no se encuentre uno existente.

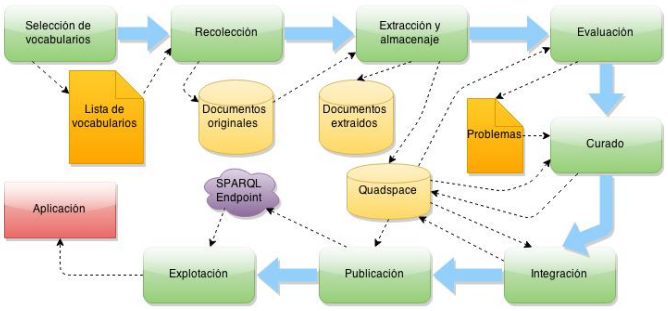

Figura 3.2: Flujo completo del procedimiento

Recolección Proceso que comprende dos tareas (Búsqueda y Obtención) en el cual se intenta conseguir los datos deseados, los cuales se encuentran dispersos en la web y para ello pueden utilizarse múltiples herramientas y servicios web.

Extracción y almacenaje de los datos Se le aplica estos dos pasos a los datos crudos obtenidos en la web para poder tener la información semántica y descartar el resto en un formato mucho mejor procesable.

Evaluación de Calidad de los Datos Se evalúa si los datos poseen la calidad mínima necesaria para crear una aplicación que satisfaga a los requerimientos.

Curado de los Datos Intentar corregir los problemas de calidad de los datos encontrados en el paso anterior. Siempre y cuando sea posible y viable.

Integración de los datos Este puede ser el proceso más costoso pero también el más importante, en el cual se intenta unificar e interconectar la información para que pueda ser aprovechable. Existen distintos niveles de integración más simples o más complejos.

Publicación del Dataset Curado El dataset ya con una calidad teóricamente superior y un estado mucho más aprovechable que el que se encontraba en un primer momento se deja a disposición de todos de forma online.

Explotación del Dataset Se construye una aplicación que en base a los requerimientos utilice el dataset.

acá sería bueno incluir un diagrama que muestre el pipeline. En la lista de arriba con un párrafo se explica cada paso. Luego, en las secciones que sigue se analiza mas cada paso, pero se lo plantea como un problema.. se

analizan los retos; entonces los capítulos 4 a 9, explican como los resolviste. Alternativamente, se puede hablar menos acá, solo lo suficiente para que se entienda la estrategia general, y luego en los capítulos 4 a 9 se analiza el problema en detalle y se propone la solución. Con esta ultima forma, todo lo que se refiere a los vocabularios quedaría en el capitulo 4 y no acá. Puede ser mejor.

3.5. Selección de vocabularios

Seleccionar un vocabulario implica analizar varios aspectos del mismo, no sólo su definición e implementación, sino también el uso práctico dado por sus usuarios. En primer lugar se debe comprobar que los nombres de las propiedades que posee sean correctamente auto-explicativas. Supongamos por ejemplo que existe un ítem con un rating agregado modelado por una ontología que posee las siguientes propiedades:

1. minrating
2. ratingValue
3. countRating

Las dos últimas propiedades resultan fácilmente identificables, ratingValue se trata del promedio de puntaje, y ratingCount la cantidad de puntajes que le fueron otorgados, pero la propiedad minRating podría generar distintas formas de interpretación, alguien podría suponer que se trata del valor mínimo que fue adquirido por un usuario, o el valor mínimo que un usuario puede otorgar. Y muchas veces la documentación de la ontología (si es que existe) no es suficiente.

Explicar que en realidad uno no elige "un" vocabulario sino que la información podría expresarse combinando términos de varios vocabularios. Dejar claro que no es lo mismo elegir el/los vocabulario/s en el que uno va a publicar que decidir cuales son todos los vocabularios que uno va a considerar al buscar información publicada por otros - dar un ejemplo

Este párrafo que sigue no se entiende; aclarar que sería cubrir las necesidades mínimas de los casos de uso.

Luego se deberá analizar si existen las propiedades para cubrir las necesidades mínimas de los casos de uso. Esto significa que pueda modelar toda la información que más adelante será requerida. Por ejemplo, si se quiere construir una

aplicación que compare autos, al momento de elegir un vocabulario que modele autos, es necesario que contenga ciertas propiedades que puedan expresar información para determinados casos de uso, como comparar cantidad de puertas, lo que requeriría que el vocabulario elegido contenga una propiedad que exprese la cantidad de puertas del auto.

Es posible que ante una aplicación muy extensa, con casos de uso muy complejos, no se consiga un vocabulario con propiedades tan abarcativas que puedan cubrir todos los casos de uso.

Para esto, se puede aprovechar que las propiedades de un vocabulario, no necesariamente tienen que ser utilizadas estrictamente dentro del mismo. Lo único que debe respetarse es el dominio y rango de cada una, si no se explicita, puede utilizarse en otros vocabularios.

Goodrelations es un ejemplo del uso de propiedades de múltiples vocabulairos para crear uno nuevo.

Y por último se debería intentar buscar ejemplos reales que muestren el uso que le dieron los usuarios a la ontología, para determinar qué propiedades están incorrectamente interpretadas o también para los casos donde las propiedades que se encuentren en desuso.

Con estas precauciones en mente se puede emprender la búsqueda, que podría tener como comienzo búsquedas en search engines. Existen dos buscadores específicos para esta tarea:

3.5.1. Linked Open Vocabulary (LOV)

Proporciona una plataforma técnica de búsqueda y evaluación de calidad sobre un dataset extraído de linked data cloud que contiene descripciones de vocabularios RDFS y ontologías OWL. Esas descripciones están en forma de metadatos y pueden ser generados tanto por los autores de los vocabularios como por curadores de LOV. Posee además de la búsqueda las funciones de estadística o sugerencia.

Actualmente el dataset está integrado por 475 namespaces distintos que contienen una media de 10 clases y 20 propiedades, siendo schema.org el más grande de ellos.

3.5.2. Vocab.cc

Vocabcc es un proyecto opensource que permite a los desarolladores de RDF realizar búsquedas de vocabularios de Linked Data.

Para facilitar la decisión de seleccionar un vocabulario deseado, proporciona además información estadística de cada uno sobre el dataset Billon Triple Challenge (BTCD). Esta información incluye el número de apariciones globales de la URI dada en el BTCD, así como el número de documentos dentro de la BTCD, que contiene el URI dado. Estos números permiten una clasificación de

propiedades y clases, respectivamente, con respecto a su uso. También se proporciona información acerca de la posición de una URI dada en estos rankings.

Los desarrolladores pueden buscar las URIs con queries arbitrarias o búsquedas de URIs específicos (prefijos comunes se resuelven automáticamente con datos de prefix.cc).

Para permitir una fácil integración de la funcionalidad vocab.cc, toda la información está disponible como RDF y se puede acceder como Servicio Vinculado.

Una vez definidos el/los vocabularios a utilizar, se debe proceder a recuperar información disponible en la web; a eso llamamos "Recolección"

3.6. Recolección

Como se mencionó antes, la web contiene grandes cantidades de documentos publicados con información semántica. Pero la tarea de encontrarlos, con el agregado de que sólo una pequeña porción de ellos será relevante para los requerimientos no es trivial en lo absoluto debido a la inmensidad del universo en el que se encuentran. La forma de llevar a cabo este objetivo está atada al hardware disponible, tanto para almacenar los datos, como para el tiempo que va a emplear la ejecución de esta tarea.

Dado que las bases de datos semánticas sólo almacenan información en forma de tripletas o cuadrupletas, los documentos encontrados deberán someterse a un proceso de extracción que seleccione las sentencias HTML y las convierta a alguno de los lenguajes que soportan tripletas o cuadrupletas. Para esto existen múltiples herramientas.

Una vez encontrados, descargados y transformados los documentos HTML a documentos semánticos puede construirse la base de datos semántica con la información recolectada.

Estos son los cuatro pasos necesarios para lograr tener el dataset semántico con el cual se puede empezar a trabajar. Cada uno de ellos posee distintas alternativas para su realización, algunas se describirán a continuación

3.6.1. Búsqueda

OJO: ESTO ESTABA EN EL ARCHIVO busqueda.tex no en el opciones-Busqueda

La forma de ejecución de esta tarea dependerá de algunos aspectos:

Recursos de hardware disponibles

Cantidad y calidad de la información requerida

El grado de atemporalidad mínima tolerable en los datos

El primer paso para realizar la recolección es intentar responder la siguiente pregunta: ¿Dónde encuentro la informanción?

Una vez seleccionados los vocabularios, se necesitará obtener fuentes de datos que contengan sus datos publicados en los vocabularios previamente seleccionados.

Para lograrlo se podrá utilizar como punto de partida:

Sitios indexadores: Son algunos sitios que disponen de un dataset muy grande procesado con documentos indexados, que ofrecen consultar dicho dataset a travez servicios web. Generalmente proveen una API donde se pueden consutlar los datos mediante distintos grados de flexibilidad.

Sindice, LOD cloud cache y UriBurner son algunos ejemplos de estos sitios. Se puede utilizar entocnes, dichos servicios web a fin de obtener una lista de URLs en las cuales se cree que contendrán la información necesaria.

Sindice: Es una herramienta creada en conjunto por la Universidad de Deri, Fondazione Bruno Kessler y Openlink Software que propociona múltiples tipos de API ofreciendo acceso a su dataset de la web de datos. Este dataset contiene información recolectada de la web en múltiples formatos de la web semántica y puede ser accedido a travez de un search engine, una API restful o un SPARQL endpoint. En la actualidad posee indexados 708.26 millones de documentos.

Link Open Data Cloud Cache (LOD): Al igual que sindice proporciona acceso a un dataset recolectado de la web de datos pero de una manera mucho más restringida. Se dispone de un text search (funciona como un search engine) o de un SPARQL Endpoint muy poco flexible y limitado en cuanto a la cantidad de información ofrecida comparada con la que podría ofrecer. Sólo se puede acceder al dataset entero mediante federated queries, en caso de querer buscar sobre todo el dataset, el endpoint limita la consulta sólo a una parte del mismo. Tambien posee muchas restricciones respecto a los time outs, por lo que las consultas no pueden ser muy flexibles. Para poder disponer de la funcionalidad completa del endpoint y así poder aprovechar tanto el dataset como las consultas SPARQL deberá comrprarse una licencia.

Sitios autoritativos: Son sitios conocidos que generan información relevante y la publican en las ontologías y vocabularios seleccionados. Ejemplos aplicados al caso de estudio podrían ser IMDB (que publica reviews de películas bajo el vocabulario schema), o Rottentomatoes (que hace lo mismo pero no solo con películas). Se podrán utilizar entonces estos sitios como punto base para un posible crawling.

Catálogos de enpoints: Son catálogos provistos por algunos sitios que mantienen una lista actualizada de SPARQL endpoints junto con el estado de disponibilidad en el que se encuentran. Por ejemplo los sitios http://labs.mondeca.com/sparqlendpointsstatus/ y http://www.w3.org/wiki/SparqlEndpoints que este último además provee detalles sobre la información de los mismos.

Consultar estos sitios entonces generará una lista de endpoints SPARQL los cuales se podrían utilizar para pedir la información necesaria.

Volcados de datos: Son datasets muy grandes que fueron el resultado de un web crawling, los cuales están disponibles para su descarga y se pueden utilizar para procesarlos y obtener los datos requeridos. El más importante es http://challenge.semanticweb.org/2014/ .

3.6.2. Obtención

Este paso está directamente relacionado con el anterior, dado que la estrategia de obtención fue planeada con anterioridad, según cuál sea la seleccionada, habrá sido la opción ejecutada en el paso anterior.

Para este paso se tienen las siguientes estrategias:

Web Crawling: Realizar un crawling partiendo de unas pocas URLs seleccionadas en el paso anteiror. Obtenidas de sitios autoritativos. Existen formas distintas de hacer web craling, dependiendo del nivel de profundidad en el cuál se recorrerá la web, podría también utilizarse los sitios indexadores, SPARQL queries, o motores de búsqueda para obtener una lista de URLs que contienen la información deseada, y luego realizar un crawling de profundidad uno sobre cada url. Hacer crawling con profundidades muy grandes requiere de una algoritmia mucho más compleja para evitar loops, y también de muchos más recursos de hardware, pero se podrá obtener mucha más información.

De-referenciamiento de URIs: Igual al web crawling, pero exclusivo de documentos semanticos NO embebidos, ya escritos en un formato nativo de la web semántica como RDF, turtle, n-Triples, etc. Luego la profundidad puede continuarse realizando lo que se llama un "follow your nose", que se realiza dereferenciando los documentos que son objeto de la propiedad owl:sameAs, o rdf:seeAlso.

Descarga de las respuestas de las APIs: Este es otro caso del uso de lso sitios indexadores, SPARQL queries, o motores de búsqueda. Pero en lugar de que la información utilizada sean simpelmente las URL de los documentos, se utilizará también la ifnromación cacheada de dichos documentos y evitarse un crawling. Como desventaja se puede tener información desactualizada, pero como ventaja se podrá obtener información que ya no esté disponible. Todo dependerá de lo que se busque obtener. La información obtenida de estas APIs ya debería encontrarse en un formato de la web semántica por l oque no requerirá tampoco una futura extracción.

Procesamiento de grandes volcados: Este paso requiere de la utilización de recursos minímos de hardware, ya que estos volcados suelen tener volúmenes de varios teras de información, y para procesarlos se requerirá de mucha RAM, memoria secundaria y microprocesador. La idea de este paso es inspeccionar el volcado o bienpara obtener la URL de los documentos con la información deseada, para luego realizar un crawling de profundidad 1, o directamente utilizar la información contenida en el volcado. Esta opción enrealidad es una suma de un paso intermedio con las demás opciones de este paso ya que se estaría realizando el trabajo que hacen los indexadores, que es procesar grandes volúmenes de datos.

3.7. Extracción y almacenaje

Una vez que se logró encontrar la información requerida en la web y contener una copia local, se necesita darle un mínimo procesamiento a determina-

dos datos. Específicamente a aquellos que estén en un formato de información semántica embebida, que requeriran un proceso de extracción para luego ser almacenados en un triplestore.

3.7.1. Extracción

Esto significa, parsear el documento HTML, que como mensionamos anteriormente puede contener información semántica en formato RDF-a, microformatos o microdatos, buscando la información semántica para luego generar un documento RDF (o de otro lenguaje de tripletas puro) que represente esa información semántica encontrada en tripletas que puedan ser almacenadas bajo un triplestore.

Si bien podría escribirse manualmente un algoritmo que realice esta tarea, existen muchas herramientas que lo hacen, por lo que es recomendable la utilización de alguna de ellas.

Los extractores más utilizados son los siguientes:

getSchema: Es una herramienta online que se utiliza a travez de una API Restful para extraer documentos con microdatos. Como respuesta se obtendrán documentos semánticos con la información extraída en formatos N-Triples, JSON, o N3. Es una herramienta rápida y eficaz pero sólo sirve para microdatos.

RDF Translator: Funciona también a travez de una API restful, y además de soportar también docuemntos con formato RDF-a, proveé la funcionalidad de transformar los documentos en sentido inverso, (de tripletas a información semántica embebida en HTML). También está la posibilidad de descargarse la librería y correr el algoritmo localmente.

Any23: Una herramienta muy potente y completa que cuenta con una comunidad que lo actualiza constantemente. Al igual que las anteriores se puede utilizar de forma online a travez de una API restful. Tiene como ventaja que soporta también microformatos. Es el extractor utilizado por Sindice. También está disponible en forma de librería para utilizar localmente.

Data Linter:Es una herramienta online que parsea documentos RDF-a, microdatos y JSON-LD. Muy útil para analizar documentos, ya que dispone de varias ventajas:

Presenta la información organizada en tablas anidadas que son mucho más amigables para un análisis humano.

Posee un mecanismo de inferencia que alerta cuando se viola la sintaxis de los vocabularios.

Como contrapartida, la información retornada por la herramienta es en texto plano. Por lo que no puede ser utilizado en un triplestore.

3.7.2. almacenaje

Con la información recolectada de internet en forma de tripletas (o cuadrupletas) ya puede ser almacenada conjuntamente en un triplestore.

Se disponen de muchas opciones de triplestores en la actualidad, algunos pagos y otros gratuitos. Cada uno con ventajas en algunos aspectos y desventajas

en otros.

Muchos son librerías que forman parte de un framework, además algunos mapean tripletas a otros motores de base de datos como por ejemplo MYSQL o POSGRE.

Algunos posibles triplestores pueden ser

TDB: Componente del framework Jena, que puede ser accedido mediante linea de comandos o a travez de la API de jena. Almacena tanto tripletas como cuadrupletas, y las almacena en varios archivos donde cada uno indexa los datos de distintas formas. Tiene una muy aceptable performance pero pequeños errores en su uso pueden corromper la información facilmente.

SDB: Al igual que TDB es un componente de Jena, pero que no almacena la información como archivos en el disco, sino que mapea los datos a un motor de bases de datos subyacente como MYSQL y POSGRE. Su performance está atada a dicho motor de bbdd subyacente pero por lo general no suele dar buenos resultados.

Sesame triplestore: Componente del framework Sesame, que tiene la opción de almacenar la información en disco o en memoria. Lo que puede significar un improtante beneficio para quienes requieran performance elevada en queries. Soporta no solo SPARQL sino también SeRQL. Tiene además la ventaja de poder integrar Alibaba, que es un mapeador de calses de Java a ontologías RDF.

Aquí hay un par de datos extraídos del paper Chris Bizer, Andreas Schultz. The Berlin SPARQL Benchmark. Int. J. Semantic Web Inf. Syst. 5(2): 1-24 (2009) que habla sobre un benchmark que compara triplestores:

```
To load 100m triples of evaluation data:
    Sesame: 3 days 6 hours
    Jena TDB: 1.5 hours
    Jena SDB: 1 day 15 hours
Complex Query Mixes per Hour (rate of query answering -- higher = better) fo
    Sesame: 18,094
    Jena TDB: 4,450
    Jena SDB: 10,421
Complex Query Mixes per Hour for 25M triples:
    Sesame: 1,343
    Jena TDB: 353
    Jena SDB: 968
Complex Query Mixes per Hour for 100M triples:
    Sesame: 254
    Jena TDB: 81
    Jena SDB: 211
Simple Query Mixes per Hour for 1M triples:
    Sesame: 38,727
    Jena TDB: 15,842
    Jena SDB: 15,692
Simple Query Mixes per Hour for 25M triples:
```

```
        Sesame: 39,059
        Jena TDB: 1,856
        Jena SDB: 4,877
    Simple Query Mixes per Hour for 100M triples:
        Sesame: 3,116
        Jena TDB: 459
        Jena SDB: 584
```

No sólo se debería elegir el triplestore en cuanto a su performance, su facilidad de uso y flexibilidad también es un aspecto muy importante a evaluar.

3.8. Evaluación de Calidad de los Datos

Como sabemos los reviews son creados por usuarios que en su mayoría no están familiarizados con el desarrollo de una aplicación, esto causa que una parte muy significativa del contenido publicado no esté de la forma adecuada para ser procesado. La falta de calidad en el contenido publicado puede deberse tanto a errores por parte del usuario como por parte del publicador.

El publicador deberá asegurarse que los datos respeten rigurosamente las ontologías en las cuales se publican.

El sitio web en el cual el usuario se encuentre realizando el review deberá guiarlo en todo lo posible para lograr que la evaluación quede en un formato adecuado.

Aunque también existen muchos problemas que no dependen del sitio web, generalmente errores semánticos de calidad, donde lo redactado esta hecho de forma inconsistente o insuficiente.

Este paso entonces tiene por objetivo encontrar todos los problemas de calidad que pueda haber en el dataset que acaba de ser descargado y extraído y que generen inconvenientes para una posterior integración/explotación.

La estrategia se divide en dos partes, una formal y otra informal.

La formal se encuentra ligada a la detección de errores sintácticos en el dataset, que son generalmente causados por una mala utilización del vocabulario (Aunque también suelen aparecer problemas causados por malas definiciones de los vocabularios)

El ejemplo típico de este tipo de error es el uso incorrecto del dominio o rango de una propiedad.

Encontrar este tipo de errores no es una tarea demasiado compleja, disponiendo de los vocabularios se puede hacer inferencia sobre el dataset y determinar automáticamente los errores sintácticos.

Y luego se encuentra la estrategia informal, que intenta encontrar los problemas semánticos del dataset, para los cuales resulta extremadamente difícil su detección automatizada.

Un ejemplo de este caso es la falta de información precisa necesaria para identificar un ítem, por ejemplo poner “batman” como nombre, siendo esta la única forma de identificarlo. Como sabemos batman puede referirse a muchos

ítem distintos y se necesita un nombre más preciso para su identificación. Este problema haría más dificultosa una posible integración, ya que no sería tan simple identificar cuales reviews hablan de los mismos ítems.

Otro ejemplo es el de la inconsistencia en la información provista, si por ejemplo se establece schema:Book como tipo de ítem pero el nombre del ítem es "Samsung SyncMaster 753s" claramente es el caso de un tipo incorrecto, ya que hay una propiedad que sólo tendría sentido si se tratara de un schema:Product. Este último error modificaría el correcto funcionamiento de una posible aplicación resultante además de interferir en la integración. Ya que si esa aplicación por ejemplo, lista reviews según el tipo de ítem, cuando un usuario busque reviews de libros, se encontraría con el review de un monitor.

Estos últimos errores suelen ser muy difíciles de encontrar y prácticamente imposibles de detectar mediante búsquedas automáticas.

Para lograrlo se requiere observar humanamente el vocabulario y reflexionar sobre el mismo, deduciendo errores que podrían aparecer, también ayudaría mucho observar resultados de estadísticas sobre el uso de propiedades en el dataset.

Existen frameworks para realizar estas operaciones.

3.9. Curado de los Datos

En el paso anterior se describieron los problemas en la calidad del dataset que tendrán un impacto en la aplicación resultante, o que, podrían limitar o imposibilitar la realización de la misma. Dichos problemas pueden ser muchos, y algunos muy difíciles de solucionar, será parte entonces del proceso, reconocer aquellos que su resolución sea viable y además encontrar la forma de implementarla.

Cabe destacar que el proceso de evaluar el dataset en búsqueda de problemas y implementar soluciones es iterativo, debido a que una mejora puede conllevar a la detección de nuevos problemas.

3.10. Integración de los datos

Esto va acá o es parte de Explotación?

Los datos recolectados fueron generados por distintos usuarios, en múltiples sitios y bajo distintas ontologías y estándares. Estos aún curados, necesitan un último paso para poder realizar una explotación, y es la integración.

Este gran y dificultoso proceso abarca cualquier operación que intente dar una visión más unificada de la información. Este proceso puede tener distintos niveles y aspectos a integrar, como podría ser por ejemplo, unificar las ontologías de los reviews, o bien en una nueva ontología de review, o bien en una ya existente, para lograr tener información semánticamente más parecida. Ya que este

proceso puede ser muy dificultoso, habrá que ver en base a los requerimientos, qué aspectos de los datos se integrarán.

Una vez realizada la integración mínima necesaria para los requerimientos, el dataset ya estará listo para su explotación.

3.11. Publicación del Dataset Curado

Existen varias maneras de publicar el dataset, y la elección de una en particular dependerá tanto de para qué se podría utilizar el dataset en el futuro como de los recursos de tiempo/hardware/económicos que se esté dispuesto a gastar.

Una posible solución muy barata y simple es generar un volcado de la base de datos, comprimirlo y subirlo a un hosting de archivos como Mega. Cualquier usuario podría descargarse entonces la base de datos y utilizarla de la forma que más le convenga. Tendría el problema en ese caso de no serle útil a usuarios con pocos conocimientos sobre esta tecnología, o que no posean demasiados recursos de hardware, o tiempo.

Una solución un poco más costosa es utilizar un servidor (ya sea propio, pago, o simplemente una pc), y crear un SPARQL Endpoint accesible mediante la web. Esta última opción puede implementarse utilizando algúnas herramientas:

Fuseki: SPARQL endpoint que utiliza el protocolo SPARQL mediante HTTP con estilo REST. Tiene una performance aceptable y la ventaja de ser muy sensilla su instalación y puesta en marcha. Se puede directamente utilizar un volcado de TDB.

Virtuoso Server: Middleware que incluye no solo un endpoint, sino también un motor de base de datos, un server, una aplicación web y todo lo necesario para publicar los datos en la web. Puede ser muy útil pero su instalación y puesta en marcha no es trivial. Y requiere la adquisición de una licencia para su uso.

ARQ: Componente de Jena que funciona como query engine, y puede ser utilizado para armar el endpoint manualmente, eso singifica realizar a mano la aplicación web.

3.12. Explotación del Dataset

Acá es donde efectivamente se discute que implica cubrir los requerimientos de la aplicación . Puede ser que acá tambien te refieras a como la forma en la que querés explotar los datos impacta en las fases anteriores. Incluso, habría que pensarlo, en este capitulo tal vez conviene primero hablar de explotación y luego de las otras fases - porque explotación determina mucho que es lo que vas a mirar en las fases anteriores, ¿no?

Capítulo 4

Selección de Vocabularios

En la actualidad existen cuatro vocabularios que cumplen los requerimientos mínimos para modelar el dominio de problema planteado.

4.1. RDF Review Vocabulary

También conocido como Review Ontology, es una de las ontologías más antiguas de review, que fue pensada para uso del lenguaje

RDF y está definido bajo el namespace http://purl.org/stuff/rev# .

Fue utilizado para la construcción Revyu , y sirvió como guía para otras ontologías.

Consta de tres clases y trece propiedades.

Clases

Comment: Un comentario sobre el review.

Feedback: Expresa la utilidad del review.

Review: El review mismo.

Propiedades

commenter: Especifica el usuario que realizó el comentario del review. Tiene dominio Feedback o Comment y rango foaf:Agent.

Actualmente se encuentra en desuso.

hasReview: Enlaza el ítem evaluado con el review. Su dominio es rdfs:Resource (siendo ésta la clase del ítem evaluado) y su rango es Review. Es una de las propiedades principales.

hasComment: Idem anterior pero con el comentario en lugar del review.

Se encuentra actualmente en desuso.

hasFeedback: Idem anterior pero con feedback en lugar de comentario.

Se encuentra actualmente en desuso.

maxRating: Establece el puntaje máximo que es posible otorgar por un usuario a travez de la propiedad rating. Tiene como dominio Review y como rango Literal siendo este último un número positivo. Su ausencia en un review

asume su valor por defecto (5). Por lo que si bien no es indispensable que esté, se deberá respetar la convención ala hora de generar el rating.

minRating: De la misma forma que el anterior, sólo que establece el puntaje mínimo y su valor por defecto es (1).

positiveVotes: Se refiere a la cantidad de votos positivos que tuvo el review, otorgados por usuarios que lo leyeron y lo encontraron útil. SU domino es Review y su rango Literal siendo este último un número positivo.

Se encuentra actualmente en desuso.

rating: Una de las propiedades principales, indica el valor numérico otorgado por el creador del review sobre el ítem evaluado. Su domino es Review y su rango es Literal siendo este último un número entre los valores de minRating y maxRating.

reviewer: Especifica el usuario que realizó el review. Tiene dominio Review y rango foaf:Person.

text: Otra de las propiedades principales, define el texto que describe el sentimento del usuario hacia el ítem. Tiene como domino Review y como rango Literal.

totalVotes: Exactamente igual a positiveVotes.

title: El título del review . Tiene dominio Review y rango Literal. Subclase de dc:title

No tiene demasiada utilidad para el caso de estudio y además se encuentra en desuso.

type: Enuncia el tipo de ítem que clasifica taxonómicamente al ítem evaluado. Su domino es rdfs:Resource (siendo ésta la clase del ítem evaluado) y su rango no se encuentra especificado.

Es una propiedad muy útil pero actualmente se encuentra en desuso.

date: Si bien no está definida dentro del vocabulario, es correcto utilizar http://purl.org/dc/terms/date, implica la fecha en la que se realizó el review.

4.2. hReview

Como se mencionó anteriormente, está establecido por convención, que los microformatos son una forma de publicar información en la web semántica, pero al no disponer de namespaces, no pueden ser representados por ninguna ontología o ningún otro lenguaje de la misma.

Al no poder utilizar ontologías que modelen reviews surgió la necesidad de crear un estándar específico para embeberlos dentro de HTML utilizando microformatos.

Dicho estándar se encuentra actualmente en la versión 0.4 y propone el uso de 10 propiedades, que se supone, deberían ser suficientes para cubrir todas las necesidades a la hora de generar un review.

summary (opcional): Puede ser el título o nombre del review, o es posible también hacer una pequeña sinopsis del mismo. Se encuentra en desuso.

type (opcional): Representa el tipo de ítem evaluado, pero se encuentra acotado a alguno de estos product — business — event — person — place —

website — url .

También está en desuso.

item: Esta propiedad enuncia toda la información que se crea necesaria para identificar al ítem, mínimamente los atributos nombre, url y foto.

Para lograr bajo una propiedad cubrir todos los atributos, el rango de la misma debería ser un hCard, que luego contendrá las propiedades mínimas necesarias: fn, url, photo y cualquier otra que se quiera adicional.

reviewer (opcional): Al igual que ítem, indica todo lo necesario para identificar a la persona autora del review, para lo cual también deberá representarse con un hCard.

dtreviewed (opcional): Se refiere a la fecha en la que fue creado el review.

rating: El valor numérico con el cual el usuario expresa su satisfacción con el ítem, y está formado por un entero con un solo decimal de precisión, que se encuentra dentro del rango 1.0 a 5.0. Dicho rango puede ser alterado con la presencia de las propiedades worst y best que restringen el valor mínimo y máximo respectivamente.

descripción (opcional): Establece el valor textual con el cual el usuario expresa su satisfacción con el ítem, creando una sinopsis detallada del mismo.

tags (opcional): Una etiqueta intenta establecer una idea acerque de qué se trata el contenido en una sola palabra para una rápida identificación o para mejorar las búsquedas.

permalink (opcional): Genera una URI que identificará al review creándole una especie de ID, que será útil para los casos donde se podría repetir la publicación del mismo.

license (opcional): Expresa la licencia del review.

El indicador "(opcional)" se refiere a si es indispensable para conformar un hReview o no, de manera tal que si no se encuentra una propiedad que no está marcada como opcional no podrá ser considerado un hReview.

Cabe destacar que muchas de las propiedades opcionales, podrían ser necesarias para el caso de estudio.

El problema con este vocabulario surge a la hora de trabajar con la información obtenida, que no puede ser representada por ningún otro lenguaje de la web semántica, por lo que se vió la necesidad de mapear las propiedades de hReview, a otro vocabulario que sí pueda.

Esto llevó a que en la web donde definen hReview lo consideren compatible con RDF Review Vocabulary, por lo que en Noviembre de 2007 se creó una herramienta que transforma de uno al otro hreview2rdfxml.xsl, pero existe un problema con el rango de algunas propiedades, por ejemplo reviewer (propiedad homónima en ambos vocabularios, pero una con rango hCard y otra con rango foaf:Person).

En general si bien este vocabulario bien utilizado puede ser efectivo, resulta poco flexible y complciado de manejar por parte de quien quiera explotarlos, el motivo por el cual se ha vuelto muy popular es su facilidad para generarlos, dado que microformatos es un lenguaje muy sencillo y cualquier persona con un relativamente mínimo ocnocimiento de HTML puede generar sin problemas un hReview, teniendo además herramientas online como op-

ción, que generan el código a travez de un formulario. Como es el caso de http://microformats.org/code/hreview/creator.

4.3. RDF Data Vocabulary

En Mayo de 2009 Google anuncia la introdución de los llamados "Google Rich Snippets", estos fragmentos enriquecidos son una convención de etiquetas (con soporte para RDFa-lite y microdata) que permitían agregar información útil a los SERP del buscador de Google. De manera tal que los datos que contenían estos fragmentos, recibían un tratamento especial.

Sin embargo en el anuncio, Google revela que el soporte se limitó al uso de las clases y propiedades del vocabulario definido en una página notoriamente improvisada llamada http://rdf.datavocabulary.org/ .

En ella se establecían modelos de clases para varios tipos de ítems, tales como Persona, Organización o Producto y también para los Reviews.

La clase Review, quedó definida bajo el namespace http://data-vocabulary.org/Review e incluía las siguientes propiedades:

itemreviewed: Enlace al ítem que está siendo evaluado.

rating: El valor numérico con el cual el usuario expresa su satisfacción con el ítem, tiene como rango un valor numérico bajo la clase xsd:string o Rating, y los valores posibles se encuentran en escala de 1 a 5, pudiendo la misma ser alterada con la presencia de las propiedades worst y best que restringen el valor mínimo y máximo respectivamente.

reviewer: El autor del review, su rango es dvocab:Person o xsd:string.

dtreviewed: La fecha en la que se realizó el review, no contiene un rango específico pero aclara que debe respetar el formato ISO para las fechas.

description: El cuerpo del review que representa el valor textual de satisfacción del usuario con el ítem.

summary: Un resumen corto del review.

Se puede notar la excesiva similitud de este vocabulario con hReview, queda claro que no hubo una intención de innovar algo, sino de representar el hReview en microdatos, probablemente por el apuro en la que data vocabulary fue creado.

Vale aclarar que limitar las clases y propiedades posibles en RDFa es básicamente hacerle perder el sentido al lenguaje (la decentralización de los vocabularios sobre los términos) haciendo que el lenguaje se utilice como si fuese microformatos, pero perdiendo su valor más improtante (la simplicidad) de manera tal que tomó la inflexibilidad de microformatos y la complejidad de RDFa.

Más adelante los Google Rich Snippets incluyeron también soporte para microformatos (lo que incluía hReview).

4.4. Schema.org

Luego de los fragmentos enriquecidos y el improvisado vocabulario data-vocabulary creado para soportar los fragmentos, Google en conjunto con Bing y

Yahoo crearon en el 2011 Schema.org, con el objetivo de obtener un vocabulario más completo y organizado para la implementación de los snippets con RDFa y microdata.

Su lanzamiento provocó la inmediata obsoletización de data-vocabulary cuyas clases fueron todas reemplazadas por equivalentes dentro de la nueva ontología:

http://www.data-vocabulary.org/Address -¿http://schema.org/PostalAddress

http://www.data-vocabulary.org/Geo -¿http://schema.org/GeoCoordinates

http://www.data-vocabulary.org/Organization -¿http://schema.org/Organization

http://www.data-vocabulary.org/Person -¿http://schema.org/Person

http://www.data-vocabulary.org/Event -¿http://schema.org/Event

http://www.data-vocabulary.org/Product -¿http://schema.org/Product

http://www.data-vocabulary.org/Review -¿http://schema.org/Review

http://www.data-vocabulary.org/Offer -¿http://schema.org/Offer

Este nuevo vocabulario, recibe constantes actualizaciones, al punto que, al día de la fecha, la ontología cuenta con 946 clases.

En particular, la clase review, definida bajo el namespace http://schema.org/Review y es subclase de schema:CreativeWork que a su vez es subclase de schema:Thing .

Propiedades de Review

itemReviewed El ítem que está siendo evaluado, tiene rango schema:Thing

reviewBody El valor textual del review de la evaluación, que tiene rango schema:Text

reviewRating El valor numérico del review de la evaluación, que tiene rango schema:Rating

Propiedades de CreativeWork

about El tema del contenido. Rango schema:Thing.

aggregateRating El promedio de la acumulación de uno o más ratings dentro de un review. Tiene rango schema:AggregateRating

author El autor del contenido. Tiene rango schema:Person o schema:Organization

comment Comentarios sobre el contenido. De rango schema:Comment o schema:UserComments.

commentCount Cantidad de comentarios. Rango schema:Integer .

creator El autor del contenido. Tiene rango schema:Person o schema:Organization

dateCreated Fecha en la que fue creado. Rango schema:Date .

dateModified Fecha en la que fue modificado por última vez. Rango schema:Date .

datePublished Fecha en de la primer publicación. Rango schema:Date .

publisher El publicador. Tiene rango schema:Organization

review El review sobre el contenido. Tiene rango schema:Review .

text Contenido textual. Tiene rango schema:Text .

Properties de Thing

additionalType Tipos adicionales para el ítem que en general son utilizados para especificar tipos externos, como podría ser por ejemplo una película de clase http://schema.org/Movie con el tipo adicional http://dbpedia.org/ontology/ . Tiene rango schema:URL .

alternateName Especifica un alias. Rango schema:Text .

description Una descripción corta del ítem. Rango schema:Text

name Establece el nombre. Rango schema:Text .

sameAs Una referencia a una url que desambiguadamente represente al ítem, como podría ser una URL de wikipedia, dbpedia, freebase, etc . Rango schema:URL.

url URL del ítem. Rango schema:URL

Unas 60 propiedades de CreativeWork fueron omitidas por no iban al caso ya que sólo tienen razón de ser para otras clases que también heredan de CreativeWork. Lo importante aquí es remarcar un par de situaciones interesantes:

=Las propiedades Review:reviewBody, CreativeWork:text y Thing:description podrían contener el valor textual de la evaluación del review semánticamente correcta por la definición de cada una.

=Las propiedades CreativeWork:author y CreativeWork:creator son exactamente iguales.

=Los valores CreativeWork:dateCreated CreativeWork:dateModified y CreativeWork:datePublished podrían generar confusión.

=Sería sintácticamente correcto que un Review utilice la propiedad Review .

Más adelante se mostrarán y enumerarán los problemas que estas cuestiones (causadas por el afán de hacer uso de la herencia lo más posible) generan.

Capítulo 5

Recolección de Datos

5.1. Objetivo

Como se indicó anteriormente la información semántica que va a ser necesaria para construir se encuentra en la web en forma de documentos HTML que tienen la particularidad de ser muy efímeros, de manera tal que un documento que poseía datos relevantes a la fecha, puede al día siguiente, o dejar de estar disponible on-line, o haber cambiado de forma tal que la información de éste ya no es relevante, o ya no la posee.

En [] sección 4.2 Challenges for the selection of data sources se generó una estadística de este caso, donde se estableció que en promedio 62 % de los documentos entoncontrados, continuaban on-line luego de un año, y de estos, sólo un 56 % aún poseían datos relevantes.

Si bien armar un dataset con sólo información extraída de los documentos sin descargar estos últimos es posible, la situación anterior genera la necesidad de mantenerlos en una copia local para evitar una posible pérdida de los mismos.

El objetivo entonces será armar un repositorio local con los documentos on-line descargados que se cree que tienen la información necesaria.

5.2. Estrategia

Se optó por la utilización de Sindice como fuente de datos. Dado que contiene indexados suficiente cantidad de documentos de las ontologías a utilizar para hacer una prueba del caso. La consulta arrojó: 10.216.632 documentos que contenían la clase purl:Review y 394.533 que contenían la clase schema:Review Luego en base a que a comienzos del año sindice limitó la paginación de sus consulas a 100. De manera que sólo se puede acceder a 5000 resultados se planificó la siguiente estrategia:

Primero se realizó una consulta de los documentos para cada ontología con los reusltados agrupados por dominio (el sitio dueño del documento) ordenada en orden descendiente por cantidad de docuemntos

Luego a mano se inspeccionaron y seleccionaron los cuarenta dominios con mayor cantidad de documentos para la ontología que aún conservaban intactos sus documentos

Y luego se ejecutó una consulta para recuperar hasta 5000 URLs de documentos por cada uno de esos 40 dominios.

Esto generó una lista de URLs a la cuál se le realizó un crawling, para descargar el documento actual de la web por cada uno. Para realizar la descarga se utilizó la librería fluent.

5.3. Resultados

Se obtuvieron de las consultas a Sindice 254950 urls de documentos potencialmente contenedores de reviews.

De las 254950 existían 236697 accesibles.

De los 236697 51 documentos tenían una url de más de 255 caracteres por lo que no se pudo almacenar en el disco utilizando la url como nombre del archivo. Y 6 documentos malformados.

La estadística fue la siguiente:

Respuesta HTTP	Cantidad de documentos
200	236697
Error sin código	10026
408	3519
500	2963
400	77
403	25
Connection reset	19
Premature EOF	8
Server redirected too many times	8
504	7
Total	254950

Capítulo 6

Extracción y Almacenamiento de los Datos

Objetivo

Convertir los documentos con información semántica embebida en documentos HTML en documentos RDF, que luego puedan ser almacenados en un triplestore y almacenarlos.

La necesidad de tener la información en un triplestore surge de varios puntos:

Tener la información centralizada, así, por cada paso siguiente a realizar, resulte mucho más simple aplicar un mismo proceso a todos los datos.

Tener la información en un mismo lenguaje, por la misma razón que el punto anterior.

Poder realizar queries SPARQL tanto para generar estadísticas como para realizar un curado de la información.

Poder utilizar el motor de inferencias para detectar errores en los documentos.

Estrategia

El extractor utilizado fue any23 con su librería para java. El motivo es que es el único de los extractores que provee librería para java, y además soporta todos los lenguajes de RDF embebido en HTML. Los documentos se guardaron en formato n-Quads ya que son mucho más eficientemente procesables y además se puede conservar el grafo del cual provienen.

Como siguiente paso, con el objetivo de tener un backup ligero de los datos que pueda ser levantado a una base de datos fácilmente, se realizó un merge de todos los documentos extraídos en uno solo. Para ello utilizó la herramienta RIOT (disponible en la librería any23).

Y por último se determinó utilizar una base de datos TDB, por múltiples razones:

Es gratuita y opensource.

Es muy eficiente, ya que no trabaja sobre una base de datos MySQL.

Se encuentra incorporado a jena por lo que se dispone de su utilización en Java.

Para levantar la base de datos, a partir de ese documento nq se utilizó la operación bulkloader2 que la manera más rapida. Con lo que se obtuvo la base de datos TDB.

Resultados

Any23

Extractores utilizados más importantes:

Extractor	Documentos	Porcentaje
html-head-title	233.743	98,77
html-rdfa11	156.096	65,96
html-microdata	110.325	46,62
html-mf-hreview	109.096	46,10
html-hcard	73.197	30,93
html-hreview-aggregated	48.771	20,60
html-mf-adf	44.788	18,92

Errores por documentos mal descargados:

null 4661

invalid property name "95

Error while retrieving mime type 1

Documentos obtenidos:

Documentos con review	187.088
Documentos sin review	49.552
Documentos sin tripletas	2.878
Tripletas totales	16.614.727
Tripletas promedio por documento	71.66
Tripletas promedio en documentos con tripletas	72.56
Tripletas promedio sobre documentos con review	84.02
Tripletas totales de documentos con review	15719392
Tripletas promedio sobre documentos sin review	20.01
Total Documentos	236640

Cabe destacar que, ese procentaje de documentos con review 79 % implica que el 21 % restante fueron documentos que sindice tiene indexados como contenedores de alguna de las clases de review mencionadas y ya no los tiene en la actualidad. Ya sea porque:

El documento no existe más

El documento cambió y no habla sobre reviews

El documento dejó de utilizar la web semántica para publicar reviews

Pero ese porcentaje no refleja una estadística sobre un muestreo válido de sindice ya que, com ose mencionó anteriormente, los dominios fueron seleccionados dependiendo de si aún existen y además aún tenían publicados documentos de reviews en la web semántica.

Riot:

Múltiples errores detectados en el merge de los documentos extraídos. No queda claro si causados por any23 en la extracción, o el documento ya de origen mal confeccionado. Fueron errores no detectados por any23:

PORT_SHOULD_NOT_BE_WELL_KNOWN in PORT: Ports under 1024 should be accessed using the appropieate scheme name.

ILLEGAL_CHARACTER in FRAGMENT: The character violates the grammar rules for URIs/IRIs.

DEFAULT_PORT_SHOULD_BE_OMITTED in PORT: If the port is the default one for the scheme it should be omitted.

REQUIRED_COMPONENT_MISSING in HOST: A component that is required by the scheme is missing.

DOUBLE_WHITESPACE in QUERY: Either two or more consecutive whitespace characters, or leading or trailing whitespace. These match no grammar rules of URIs/IRIs. These characters are permitted in RDF URI References, XML system identifiers, but not XML Schema anyURIs.

WHITESPACE in PATH: A single whitespace character. These match no grammar rules of URIs/IRIs. These characters are permitted in RDF URI References, XML system identifiers, and XML Schema anyURIs.

ILLEGAL_PERCENT_ENCODING in QUERY: The host component a percent occurred without two following hexadecimal digits.

NOT_DNS_NAME in HOST: The host component did not meet the restrictions on DNS names.

DNS_LABEL_DASH_START_OR_END in HOST: A DNS name had a - at the beginning or end.

PROHIBITED_COMPONENT_PRESENT in USER: A component that is prohibited by the scheme is present.

WHITESPACE in FRAGMENT: A single whitespace character. These match no grammar rules of URIs/IRIs. These characters are permitted in RDF URI References, XML system identifiers, and XML Schema anyURIs.

WHITESPACE in QUERY: A single whitespace character. These match no grammar rules of URIs/IRIs. These characters are permitted in RDF URI References, XML system identifiers, and XML Schema anyURIs.

TDB Bulk load

La base de datos resultante contiene:

Cantidad de grafos: 182153

Cantidad de grafos con schema/review: 50955

Cantidad de grafos con schema/aggregate rating: 52022

Cantidad de grafos con purl/review: 104316

Cantidad de grafos con purl/aggregate review: 43991

La búsqueda en sindice se realizó para schema/review y purl/review. Sin embargo, como se puede apreciar se obtuvieron muchos resultados con reviews agregados.

Capítulo 7

Evaluación de Calidad de los Datos

Objetivo: Encontrar problemas en el dataset que dificulten una posible integración y modifiquen el correcto resultado de una posible aplicación derivada.

7.1. Evaluación nº 1 - Vocabularios

Objetivo: Buscar aquellas propiedades y clases ligadas a recursos relevantes (que estén relacionados con algún review mediante property paths) para las cuales no se encuentre representada en ninguna ontología. Y luego verificar que genere algún problema para la aplicación a construir.

Estrategia: Este primer paso es bastante simple, básicamente se deben conseguir todos los vocabularios utilizados (para saber cuales son estos basta con revisar el namespace de los prefijos) y cargarlos en el dataset. Una vez con los vocabularios cargados se puede realizar una consulta en SPARQL para las propiedades y leugo otra para las clases:

```
SELECT ?c (count (distinct ?o) as ?cantidad)
WHERE{
{
?s <http://local.org/id> ?id .
?s (:|!:)+ ?o .
}UNION{
?s <http://local.org/itemId> ?id .
?s (:|!:)+ ?o .
}
?o a ?c .
FILTER NOT EXISTS{
?c a rdfs:Class.
}
```

```
}GROUP BY ?c ORDER BY DESC(?cantidad)
```

Luego se puede hacer una consulta igual pero buscando por propiedades.

Resultados: Los resultados de esta evaluación fueron interesantes. Además de algunas propiedades y clases mal escritas, ya sea por una letra faltante o una minúscula en lugar de una mayúscula, se encontró que todas las propiedades utilizadas de la ontología schema estaban incorrectas.

Ninguna de las propiedades encontradas bajo el namespace http://schema.org/ pertenecían a la ontología schema. Esto se debe a que cada propiedad incluía bajo su path su clase dominio de la siguiente manera: http://schema.org/CLASE/Propiedad . Por ejemplo: http://schema.org/Product/aggregateRating en lugar de http://schema.org/aggregateRating

Situaciones como ésta generan problemas cuando se intenta realizar consultas generales. Si por ejemplo se requiriera encontrar todos los aggregateRatings de todos los recursos no alcanzaría con una simple querie, habría que realizar tantas queries como clases de la ontología schema existan.

Haciendo un recorrido sobre los datos desde su origen (los documentos html publicados) hasta este punto, se encontró que el problema se originó en el proceso de extracción. Any23 cuando extrae las propiedades de la ontología schema, modifica las mismas adicionando el nombre de la clase de su correspondiente dominio, provocando que éstas queden incorrectas.

7.2. Evaluación nº 2 - Duplicados

Objetivo: Disponiendo de un gran dataset de información recolectada de la web, un paso muy útil es limpiar aquellos datos innecesarios. Este paso consiste en detectar aquellos reviews en el dataset que se encuentren duplicados.

Estrategia: La forma más sencilla que a cualquiera normalmente se le ocurriría es comparar los reviews todos con todos. Cuyo algoritmo tiene orden cuadrático que para el tamaño del dataset resulta demasiado denso. El algoritmo que se utilizó fue el siguiente: 1)Por cada ontología de review se escogió arbitrariamente las propiedades más utilizadas y representativas de cada una:

Purl-Review dcterms:date rev:text rev:title rev:rating

Schema-Review sorg:datePublished sorg:description sorg:name sorg:author sorg:reviewBody

Purl-ReviewAggregate revagg:average revagg:count dcterms:date rev:title

Schema-AggregateRating sorgagg:ratingCount sorgagg:ratingValue sorgagg:reviewCount

Luego se arma en cada ontología un mapa para cada propiedad escogida, que contiene como clave un valor posible existente para esa propiedad, y como valor un arreglo con todos los reviews que contiene ese valor para dicha propiedad. En otras palabras se separaron por cada propiedad, los reviews según su valor para esa propiedad. Luego se buscaron qué grupos de reviews se encontraban juntos para más de una propiedad mendiante obtener la intersección de ambos conjuntos. Por ejemplo:

Para el valor 4 de la propiedad rev:rating se encuentran review1, review2, review3 Y para el valor "buena película" de la propiedad rev:text se encuentran review2, review3, review4 Entonces la intersección entre el conjunto de reviews para el primer conjunto con el segundo conjunto es review2, review3. Ahora para marcar ese conjunto como posible grupo de duplicados deben cumplir la condición de que, el ítem sobre el cuál el review habla tiene el mismo nombre en cada uno de los elementos.

Esta comparación se reliza entre todos los conjuntos de reviews de los valores de una propiedad con el resto de los conjuntos de los valores de las restantes propiedades de esa ontología.

Una vez obtenidos los conjuntos de los posibles duplicados se procede ahora sí a analizar minuciosamente los reviews todos con todos pero dentro del conjunto marcado como posible grupo de duplicados para corroborar.

Resultados:

Se encontraron en total 17343 reviews replicados en un total de 78705 reviews de los cuales.

1009 estaban duplicados dentro de un mismo documento. 64991 estaban duplicados entre distintos documento pero dentro de un mismo dominio 12709 estaban duplicados en documentos pertenecientes a distintos dominios

Ésta es la lista con los dominios a los cuales se le encontraron más cantidad de reviews duplicados:

Dominio	Cantidad de reviews duplicados
4outof10.com	21582
www.superpages.com	18756
www.kollermedia.at	9864
www.realtruck.com	8372
ormigo.com	8133
www.reptilecentre.com	3440
www.querfood.de	3275
www.carsurvey.org	2766
www.chip.de	1526
www.thewinecellarinsider.com	1491

7.3. Evaluación nº 3 - RDFUnit-Automatizado

Objetivo: Encontrar todos los problemas sintácticos de los datos relevantes del dataset e identificar cuales pueden representar un problema para construir la aplicación.

Estrategia: El framework crea test automaticamente a partir de las ontologías, buscando una amplia variedad de problemas posibles. Por lo que se instaló el framework, se lo proveyó de las ontologías relevantes y se corrieron los test sobre el dataset.

Resultados:

Total test cases	2116
Succeeded	2067
Failed	49

Los tests fallados más relevantes fueron para schema:

http://schema.org/datePublished does not have datatype: http://www.w3.org/2001/XMLSchema#date
http://schema.org/ratingCount does not have datatype: http://www.w3.org/2001/XMLSchema#decima
http://schema.org/ratingValue has rdfs:domain different from: http://schema.org/Rating
http://schema.org/worstRating has rdfs:domain different from: http://schema.org/Rating
http://schema.org/bestRating has rdfs:domain different from: http://schema.org/Rating
http://schema.org/aggregateRating is missing proper range
http://schema.org/reviewRating has different range from: http://schema.org/Rating
http://schema.org/name does not contain a literal value (http://www.w3.org/2001/XMLSchema#string
http://schema.org/itemReviewed is missing proper range
http://schema.org/reviewRating has rdfs:domain different from: http://schema.org/Review
http://schema.org/email does not contain a literal value (http://www.w3.org/2001/XMLSchema#string

Para purl:

http://purl.org/stuff/rev#reviewer is missing proper range	8334
http://purl.org/stuff/rev#reviewer has different range from: http://xmlns.com/foaf/0.1/Person	7366
http://purl.org/stuff/rev#hasReview has different range from: http://purl.org/stuff/rev#Review	7239
http://purl.org/stuff/rev#title has rdfs:domain different from: http://purl.org/stuff/rev#Review	1470

7.4. Evaluación nº 4 - RDFUnit-Manual

Objetivo: Correr test manuales en RDFUnit para evaluar el dataset en busca de errores sintácticos no contemplados por los test automáticos de la evaluación anterior.

Estrategia: Se configuró RDFUnit utilizando patrones del framework para crear tests manuales. Luego se corrió para obtener resultados de la misma manera que en la evaluación anterior

Resultados:

http://purl.org/dc/terms/date has a format different to YYYY-MM-DD	92580	207130
http://schema.org/datePublished has a format different to YYYY-MM-DD	53740	144594
http://schema.org/publishDate has a format different to YYYY-MM-DD	50070	59825
http://schema.org/dtreviewed has a format different to YYYY-MM-DD	26903	26903
http://purl.org/stuff/rev#rating is not in the expected range (1-5)	15319	298458
http://purl.org/stuff/rev#rating is not a natural number	2866	301322
http://schema.org/ratingValue is smaller than http://schema.org/worstRating	41	108088
http://schema.org/ratingValue is not a natural number	33	60682
http://schema.org/ratingValue is not in the expected range (1-5)	1	73090

7.5. Evaluación nº 5 - Análisis experto

Objetivo: Encontrar problemas a grandes razgos que son muy difícil de detectar por algoritmos. También indicios indicadores de situaciones que lleven a pensar en otras búsquedas de problemas. Esto debería ayudar a entender mejor el dataset para mejorar el manejo de los datos en posteriores paso, como el de integración e implementación.

Estrategia: Este último paso de evaluación se comienza con un pantallazo general de cómo está la situación de los reviews. Para esto se pueden analizar las propiedades que contienen los reviews, y luego el valor objeto de las mismas. Dado que los problemas sintácticos deberían estar cubiertos en las evaluaciones anteriores, este paso sólo dejaría establecidos algunos problemas semánticos del dataset, que al ser detectados en base al punto de vista racional humano, resultaría improbable poder encontrar una solución algoritmica automatizada para el problema.

Primera query:

```
SELECT ?p (count (distinct ?review) as ?cantidad)
WHERE{
?review a {ReviewClass}.
?review ?p ?o .
}GROUP BY ?p ORDER BY DESC(?cantidad)
```

Esta query debería devolver las propiedades que poseen los reviews y la cantidad de reviews que la utilizan para una clase de review dada.

Resultados más improtantes:

Para la clase schema:Review

Propiedad	Cantidad
http://schema.org/author	146243
http://schema.org/datePublished	144594
http://schema.org/reviewRating	143908
http://schema.org/description	122519
http://schema.org/name	106219
http://schema.org/reviewBody	93634
http://schema.org/publishDate	59825
http://schema.org/url	44905
http://schema.org/dtreviewed	26903
http://schema.org/reviewer	26903
http://schema.org/about	14693
http://schema.org/keywords	7119
http://schema.org/review	2292
http://schema.org/itemReviewed	2071
http://schema.org/itemreviewed	67

Mirando la tabla de resultados, el primer error semántico que salta a la vista son las 2292 propiedades review dentro de un review. Es claramente un error en la producción del documento. Una rápida búsqueda de los documentos con este problema mostró que son 2 los dominios que repiten el error a lo largo de sus documentos http://www.ilgiardinodeilibri.it/ y http://www.iplaysoft.com/ Más en detalle la intención de los autores del documento fue realizar comentarios para los reviews (situación que tiene una solución específica y es el uso de la propiedad ¡http://schema.org/comments¿). La causa principal de que existan errores de este tipo es la ontología schema que aunque no tenga sentido, propone que sea sintácticamente válido escribir un review de un review. Dado que la clase Review es subclase de Creative Work, y esta última contiene la propiedad review.

Por otro lado otra propiedad en el listado que llama la atención es description. Analizando la propiedad semánticamente, significaría describir el review en sí mismo, y no el ítem, dado que si ese fuese el objetivo, dicha propiedad estaría en el ítem y no en el review. Observando varios de los valores otorgados a esa propiedad en el contexto del review, muestran que se trata de la opinión textual del ítem, para lo cual al igual que en la situación anterior existe una propiedad específica para eso: reviewBody. La causa de que se utilice description en lugar de reviewBody (incluso con mayor frecuencia) proviene de la ontología de data vocabulary, en la cuál como se explicó al principio de la tesis utiliza la propiedad description como propiedad específica para realizar el review textual. Puede suceder que sean dominios que migraron sus reviews desde data vocabulary hacia schema y usaron como propiedad equivalente la que era literalmente igual (lo cual es incorrecto). También puede deberse a la custumbre de haber utilizado aquella ontología que hizo que no se le preste atención a este detalle, o simplemente que hayan visto que es también sintácticamente correcto realizar una descripción de un review y creyeron que era correcto utilizarlo para plantear el review textual. De cualquier manera la ontología vuelve a generar inconvenientes debido a la jerarquía propuesta (Review hereda description de la clase Thing). Vale la pena aclarar que también existe otra propiedad proveniente de Creative Work que es ¡http://schema.org/text¿cuya descripción de la ontología es ‘ “El contenido textual de este trabajo creativo” la cuál sería sintáctica y semánticamente correcto utilizarla en lugar de reviewBody, aunque no sería la más específica.

La propiedad name, es ampliamente utilizada en los reviews, la descripción de la propiedad en la ontología declara que se refiere al nombre del objeto en cuestión. Claramente no está pensada para un review (Al igual que la anterior se hereda de schema). Esta propiedad tiene sentido para algún objeto con un nombre real, como Teclado Genius L3, o Batman Begins. Un análisis en el uso de la propiedad bajo este contexto, demuestra que la intención fue utilizar esta propiedad para proveer un título al review. Y al no haber una propiedad provista por parte de la ontología para titular un review el uso de la propiedad name en su lugar si bien es semánticamente incorrecto, no es una mala idea implementarlo.

Puede observarse tambien que se utiliza la propiedad itemreviewed, la cual no existe en la ontología schema, pero claramente es un error de tipeo en un intento de utilizar la propiedad itemReviewed. Éste al parecer es un error sintáctico no cubierto en las evaluaciones anteriores.

Una propiedad curiosa que aparece en el listado es ¡http://schema.org/about¿. Esta propiedad debería describir el tema en el cual se está tratando en el review. Pero un simple vistazo a los valores que toma la propiedad mediante una consulta sparql muestran que el único valor es “This is required”

Otra propiedad que vale la pena analizar es ¡http://schema.org/keywords¿. Es una propiedad que debería contener palabras que describan al recurso, el recurso en cuestión es el review y no el ítem, por lo que nuevamente nos encontramos con un error semántico en el uso de la propiedad (ya que es sintácticamente correcto utilizarla en un review porque forma parte de la clase CreativeWork), pero no tiene sentido utilizar palabras para describir un review,

claramente son palabras que utilizaron para describir al ítem. Una observación de los valores verifican lo descripto anteriormente y además muestran que esta propiedad es sólo utilizada en lso reviews por el dominio http://www.kennstdueinen.de/ que además utiliza el idioma danés.

También hay que mencionar la superpoblación de propiedades que establecen la fecha del review: datePublished, publishDate y dtreviewed. El motivo por el cual aparecen tres propiedades distintas para referirse a lo mismo es básicamente porque publishDate y dtreviewed no existen bajo la ontología schema. Es el mismo caso que itemreviewed. Es interesante también mencionar que tanto itemreviewed como dtreviewed son propiedades existentes bajo la ontología data vocabulary, por lo que la causa es muy probable que sea a un intento erróneo de migrar el vocabulario de una a otra ontología, o de venir acostumbrado a la ontología anterior y no prestar atención a este detalle. En cambio el motivo por el cual muchos dominios optaron por utilizar publishDate en lugar de datePublished, es que hasta fines de 2014, era una propiedad válida que además se encontraba como ejemplo de un correcto uso de la clase Review provisto por parte de la documentación de la ontología. También la misma ontología en su documentación aclaraba que tanto datePublished como publishDate eran indistintas. Exactamente esto mismo sucede con la propiedad reviewer, la cual ya no está disponible.

Por último, un vistazo a la propiedad más popular de la clase Review, author, mostró que todos los valores de dicha propiedad están compuestos por un string o más específicamente un Literal. Esto contradice la definición de la ontología schema, tanto la documentación como la implementación, que establecen que le rango de la propiedad author es Organization o Person. El asunto más relevante en este caso es entender porqué este error sintáctico de rango de propiedades no fue contemplado en los test automáticos generados por el framework RDFUnit. Y la respuesta parece estar en la complejidad con la que establecieron el rango en la definición de la ontología. El extracto de las tripletas relevantes muestran lo siguiente:

```
schema:author rdfs:domain schema:CreativeWork .
_:node1 rdf:type owl:Class .
_:node2 rdf:first schema:Organization .
_:node2 rdf:rest _:node3 .
_:node3 rdf:first schema:Person .
_:node3 rdf:rest rdf:nil .
_:node1 owl:unionOf _:node2 .
schema:author rdfs:range _:node1 .
```

Al parecer la definición del rango de la propiedad author utiliza estructuras de lista y conjuntos que parecen requerir más inferencia de la que el framework RDFUnit puede realizar.

Conclusión: La mayoría de los problemas son ocacionados por la inconsistencia, inestabilidad e incoherencia de la ontología. Principalmente en el afán de hacerla elegante, crearon a Review como subclase de CreativeWork que normal-

mente suele ser suplerclase de clases que son ítems de reviews lo que concluye en una ontología demasiado larga y redundante.

Para la clase purl:Review

Propiedad	Cantidad
`http://purl.org/stuff/rev#text`	207259
`http://purl.org/stuff/rev#rating`	195005
`http://purl.org/stuff/rev#reviewer`	154663
`http://purl.org/dc/terms/date`	146473
`http://purl.org/stuff/rev#title`	88148
`http://purl.org/stuff/rev#hasReview`	6424
`http://www.w3.org/2006/vcard/ns#n`	6424
`http://www.w3.org/2006/vcard/ns#fn`	6424
`http://www.w3.org/2006/vcard/ns#url`	6420
`http://www.w3.org/2006/vcard/ns#geo`	6123

Los resultados más llamativos son la aparición de propiedades que normalmente se encuentran en el ítem, dado que no tienen sentido para un review que son hasReview (un review dentro de un review), n (nombre para un review), fn (idem n), geo (posición geográfica de un review). El hecho de que todas se encuentren en cantidades similares es el gran indicador de que es un dominio repitiendo algún problema a lo largo de sus documentos. Una inspección de los dominios donde se encuentra cada uan muestra que todas están incluídas en http://hoteles.hotelopia.es y http://hotels.hotelopia.it/ Básicamente el problema es que le asignaron el tipo purl:Review a ítems.

Capítulo 8

Curado de los Datos

Objetivo: Corregir todos los problemas viables encontrados en el paso anterior que se pueda.

8.1. Curado nº 1 - Vocabularios

Estrategia: Con una consulta SPARQL, se eliminó la propiedad incorrecta y se agregó la misma en la forma que corresponde. Todo se realizó con una simple consulta SPARQL

```
DELETE{
  GRAPH ?g {
    ?s ?t ?o .
  }
}
INSERT{
  GRAPH ?g {
    ?s ?p ?o .
  }
}
WHERE{
  SELECT ?g ?s (IRI(CONCAT("http://schema.org/", ?prop)) AS ?p) ?o
  WHERE{
    GRAPH ?g{
      ?s ?t ?o .
      FILTER(REGEX(str(?t), "http://schema.org/*+", "i")).
      BIND(REPLACE(str(?t), '^.*(\#|/)', "") AS ?prop)
    }
  }
}
```

Resultados: Se crearon 2498221 propiedades nuevas de forma correcta correspondientes a cada una de las propiedades incorrectas de la ontología schema.

8.2. Curado nº 2 - Duplicados

Estrategia:

Se recorrieron todos los grupos de duplicados y se procedió a realizar la siguiente operación: Se le asignó a cada grupo un id único, al cual llamaremos groupId. Lueog por cada grupo de duplicados se seleccionó de manera arbitraria un representante con un criterio que se describirá a continuación: Si dentro del conjunto de documentos entre los cuales se encuentran todos los integrantes del grupo de duplicados no existe ninguno que contenga un recurso ya elegido anteriormente como representante, se selecciona cualquier recurso de la lista de duplicados al azar. Caso contrario, se selecciona el primer recurso encontrado en la lista, que pertenezca al documento que ya contiene algún recurso escogido como representante. Esto se realizó de esa forma, para que todos los representantes se concentren en los mismos documentos y no suceda que dados 2 documentos idénticos con 2 reviews cada uno, no terminen con un representante cada uno. De esta forma se minimizan la cantidad de documentos con reviews en el dataset. Luego de elegir un representante para el grupo de duplicados, a ese recurso se le asignó una propiedad http://local.org/representantOf y como objeto de la propiedad el groupId. Y para todos los recursos restantes del grupo de duplicados, se les asignó una propiedad http://local.org/duplicatedOf con objeto el groupId.

Resultados:

Se asignaron 78705 propiedades duplicatedOf y 17343 representantOf

8.3. Curado nº 3 - RDFUnit-Automatizado

Muchos de los problemas detectados por el framework, no otorgaban demasiada información, como por ejemplo http://schema.org/datePublished does not have datatype: http://www.w3.org/2001/XMLSchema#date el cual sólo especifica que le falta el tipo al recurso objeto, pero no necesariamente significa que esté mal, esto da un indicio de que algo puede andar mal y será verificado en la consulta manual, en este caso por ejemplo, lo importante es que la fecha tenga el formato apropiado. Se decidió entonces atacar los problemas más específicos:

http://schema.org/ratingValue has rdfs:domain different from: http://schema.org/Rating

Estrategia: Primero hubo que investigar, cuáles son todos los dominios para esa propiedad que aparecen en el dataset, para ver si se pueden acomodar. Esto se realizó con la siguiente consulta sparql:

```
select distinct ?dominio (count (?s) as ?cantidad)
where{
?s <http://schema.org/ratingValue> ?value .
?s a ?dominio .
```

```
}GROUP BY ?dominio
```

Y los resultados fueron

¡http://schema.org/Rating¿	120165
¡http://schema.org/AggregateRating¿	54815
¡http://schema.org/Review¿	5867
¡http://schema.org/Product¿	331

Se puede apreciar que en la mayor parte de los casos de error, se dieron porque se incluyó la propiedad ratingValue directamente en el Review. Ahora surge la necesidad de saber si esos casos donde el Review tiene la propiedad ratingValue, existe una propiedad reviewRating con su respectivo Rating. Se relizó la consulta:

```
select (count (?s) as ?cantidad)
where{
?s a <http://schema.org/Review> .
?s <http://schema.org/ratingValue> ?value .
?s <http:schema.org/reviewRating> ?rating .
}
```

El resultado fue 0.

Por lo tanto ese problema se solucionó con la siguiente consulta:

```
DELETE {
GRAPH ?g{
?s <http://schema.org/ratingValue> ?value .
}
}
INSERT{
GRAPH ?g{
?s <http://schema.org/reviewRating> ?rating .
?rating a <http://schema.org/Rating> .
?rating <http://schema.org/ratingValue> ?value .
}
}
WHERE{
GRAPH ?g {
?s a <http://schema.org/Review> .
?s <http://schema.org/ratingValue> ?value .
}
}
```

schema:worstRating and schema:bestRating has rdfs:domain different from: http://schema.org/Rating Estrategia: El proceso es igual al anterior y los dominios disintos de rating eran Review, la misma cantidad que en el anterior 5867

Resultados: Idem los anteriores.

http://schema.org/itemReviewed is missing proper range

Estrategia: Esta evaluación se refiere a que existen recursos objeto de dicha propeidad, que no tienen un tipo específico. Para ver si puede ser resuelto, primer hubo que evaluar qué clase de recursos contienen como objeto, esas propiedades que presentaron problemas.

Resultado: La evaluación encontró que, todos esos recursos eran URIs de un dominio web 411ca.com para ser derreferenciados. La solución podría haber sido descargar dichos recursos y anexarlos a la propiedad, pero lamentablemente dichos recursos no se encontraban disponibles de forma online.

http://purl.org/stuff/rev#hasReview has different range from: http://purl.org/stuff/rev#Review En este caso no hay error, el inconvieniente surge porque los recursos de tipo purl:ReviewAggregate utilizan también la propiedad hasReview que no esta contemplado en la ontología de purl.

http://purl.org/stuff/rev#reviewer is missing proper range Este caso se da porque se genera un recurso vació, específicamente el reviewer. El reviewer no fue provisto en lso documentos orginales donde se derivó este problema, simplemente establecían el metadato reviewer dejándolo vacío.

http://purl.org/stuff/rev#reviewer has different range from: http://xmlns.com/foaf/0.1/Person En estos casos analizando qué clase contenian si no era foaf:Person, se observó que la clase era http://www.w3.org/2006/vcard/ns#VCard. En todos los casos donde el reviewer no se encontraba vacío, se utilizó un v:card en lugar de un foaf:Person, y esto tiene una explicación, y es que al haberse utilizado hreview con microformatos es imposible crear un recurso de esa clase, dado que no se posee un namespace, para que eso sea posible, la clase foaf debería encontrarse dentro de los estándares de los microformatos. En este caso al no haber alternativa por parte del creador del review, se considera correcto el uso del v-card, en reemplazo de foaf.

8.4. Curado nº 4 - RDFUnit-Manual

DateProperty has a format different to YYYY-MM-DD: Para todas las propiedades que indiquen una fecha, los pasos para reparar este caso son los mismos.

Cuando nos encontramos con este caso, se requiere de otra evaluación para entender cómo encarar el problema. Es una evaluación cuyo resultado pueda responder la siguiente pregunta:

Si no tiene formato YYYY-MM-DD, ¿Qué formato tiene?

La idea en esta evaluación entonces, es encontrar qué formatos distintos contienen los datos, para poder trasladarlos al correcto.

Estrategia:

Comenzando con la siguiente consulta:

```
SELECT distinct ?date
WHERE{
?review <http://local.org/id> ?id .
?review {DateProperty} ?date .
```

```
FILTER (!REGEX(str(?date), "^[0-9]{4}-[0-9]{2}-[0-9]{2}$", "i")) .
FILTER NOT EXISTS{
?review <http://local.org/deplicateOf> ?dup.
}
}
```

Esta consulta retorna la fecha de todos los reviews que no estén duplicados y que además dicha fecha no tenga el formato YYYY-MM-DD

Luego observando los resultados manualmente, se prsta atención also primeros resultados y se trata de encontrar con qué patrón están formados para luego anotarlo y realizar la misma consulta esta vez filtrando los reusltados ocn dicho patrón. Iterando este proceso hasta que queden 0 resultados.

Resultados:

Se descubrieron los siguientes patrones

http://schema.org/datePublished

Expresión regular	Cantidad de reviews
`^[A-z]+ [0-9]{2}, [0-9]{4}$` & 14715	
`^[0-9]{4}-[0-9]{2}-[0-9]{2}.+` & 17700	
`^[0-9]{4}-[0-9]{2}-[0-9]{1} [0-9]{2}:[0-9]{2}:[0-9]{2}$` & 7179	
`^[0-9]{2}\\.[0-9]{2}\\.[0-9]{4}$` & 7172	
`^[A-z]+ [0-9]{2}, [0-9]{4}.+` & 5049	
`^[A-z]{3} [0-9]{1}, [0-9]{4}` & 1847	
`^[0-9]{2}\\.[0-9]{1}\\.[0-9]{4}$` & 410	
`^[0-9]{1}\\.[0-9]{1}\\.[0-9]{4}$` & 120	
`[0-9]{2}/[0-9]{2}/[0-9]{2}` & 67	
`^[0-9]{1}\\.[0-9]{2}\\.[0-9]{4}$` & 30	

http://purl.org/dc/terms/date

Expresión regular	Cantidad de reviews
`^[0-9]{4}-[0-9]{2}-[0-9]{2}.+`	33833
`^[A-z]+.[A-z]{2}.[0-9]{4}-[0-9]{2}-[0-9]{2}$`	33058
`^[A-z]{3}\\n.*[0-9]{1}\\n.*[0-9]{4}$`	25543
`^[A-z]{3}\\n.*[0-9]{2}\\n.*[0-9]{4}$`	17770
`^[A-z]+ [0-9]{2}, [0-9]{4}$`	17069
`[0-9]{2}((?!-).)[0-9]{2}((?!-).)[0-9]{4}`	11720
`^[0-9]{2} [A-zéû]+ [0-9]{4}`	4147
`^[0-9]{2}((?!-).)[0-9]{2}((?!-).)[0-9]{4}`	4093
`^[0-9]{2}((?!-).)[0-9]{1}((?!-).)[0-9]{4}`	2803
`^[A-z]{3} [0-9]{1}, [0-9]{4}`	2783
`^[A-z]+ [0-9]{2}[a-z]{2} [0-9]{4}\\.$`	2058
`^[0-9]{2} [A-z]{3} [0-9]{4}`	1730
`^[0-9]{1}((?!-).)[0-9]{1}((?!-).)[0-9]{4}$.`	1103
`^[0-9]{2}-[0-9]{2}-[0-9]{4}$`	1024
`^[A-z]+ [0-9]{1}[a-z]{2} [0-9]{4}\\.$`	746
`^[0-9]{1}((?!-).)[0-9]{2}((?!-).)[0-9]{4}$`	279

http://schema.org/publishDate

Expresión regular	Cantidad de reviews
`[0-9]{2}/[0-9]{2}/[0-9]{2}` & 2511	
`^[0-9]{2}\\.[0-9]{2}\\.[0-9]{4}$` & 2207	

http://schema.org/dtreviewed

Expresión regular	Cantidad de reviews
`^[0-9]{1}((?!-).)[0-9]{2}((?!-).)[0-9]{4}$`	14762
`^[0-9]{1}((?!-).)[0-9]{1}((?!-).)[0-9]{4}$`	6358
`^[0-9]{2}((?!-).)[0-9]{2}((?!-).)[0-9]{4}$`	4193
`^[0-9]{2}((?!-).)[0-9]{1}((?!-).)[0-9]{4}$`	1590

Una vez encontrados los patrones, sólo fue cuestion de correr un algoritmo en Java dónde se contemplan todo los patrones y se le aplica la correción apropiada al String y se vuelve a almacenar en el review. También en varios casos hubo que verificar dónde se encontraba el día y dónde el mes como por ejemplo para casos como éste `^[0-9]{2}\\.[0-9]{2}\\.[0-9]{4}\$` Manualmente se observan los resultados de ese patrón y se puede observar si los primeros 2 números son mes o día. También hubo casos no tratables como `^[A-z]+ [0-9]{2}` donde no estaba explicitado el año.

`http://purl.org/stuff/rev#rating is not in the expected range (1-5)`

RatingProperty is not in the expected range (1-5): Cuando nos encontramos con un review en esta situación, resulta casi imposible incluso manualmente corregir este error con la información provista por el review o el documento que contiene el review. Si el rating del review es 7, ¿Cuál es el máximo valor que el autor tenía en mente cuando generó la evaluación?. Uno podría pensar que lo más lógico sería 10, pero no se puede saber a ciencia cierta, por lo menos con la información del entorno al review. Para encontrar la respuesta, se necesita la información de todos los reviews del dominio que contiene el review problemático.

Estrategia: Para resolver el problema de rango en los ratings de los reviews entonces, lo primero que hay que hacer es una evaluación, en la cual se pueda observar qué valores tomaron los ratings de otros reviews correspondientes al mismo sitio web hosting del review. Esto sólo se aplicó par ala ontología purl, dado que apra schema los valores erróneos eran muy pocos y no se jsutificaba. Para implementarlo, lo primero que se realizó fue la siguiente consulta

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT distinct ?g ?rating ?id
WHERE{
GRAPH ?g{
?review <http://local.org/id> ?id .
?review <http://purl.org/stuff/rev#rating> ?rating .
FILTER NOT EXISTS{
?review <http://local.org/duplicateOf> ?dup .
}
FILTER NOT EXISTS{
```

```
?rat <http://purl.org/stuff/rev#maxRating> ?best .
}
FILTER NOT EXISTS{
?rat <http://purl.org/stuff/rev#minRating>  ?worst .
}
FILTER ((xsd:integer(?rating) > 5) || (xsd:integer(?rating) < 1)).
}
}
```

Esta consulta retorna todos las urls de los documentos que contienen el review problemático incluyendo el id del review y el rating correspondiente a dicho review.

A partir de esta información, puede obtenerse el dominio de la URL, y almacenarse toda la información en un mapa de Java, donde para cada dominio se guarde los valores de los ratings encontrados, y la cantidad de veces que se encontró cada valor de rating.

Resultados: El mapa resultante de esta evaluación fue el siguiente.

Dominio: www.yellowbot.com

Valor	Cantidad
0	13

Dominio: www.chip.de

Valor	Cantidad
6	1
8	2
9	1
19	1
25	1
27	1
29	1
31	1
30	1
34	1
33	2
38	1
36	3
37	1
42	1
43	3
40	1
46	1
47	4
44	1
45	4
51	9
50	10
49	8
48	6
55	7
54	15
53	5
52	12
59	8
58	18
57	11
56	19
63	25
62	23
61	12
60	12
68	62
69	34
70	50
71	62
64	24
65	39
66	35
67	32
76	82
77	82
78	81
79	86
72	81
73	70
74	72
75	73
85	120
84	92

Dominio: www.gamezebo.com

Valor	Cantidad
0	5

Dominio: www.gogobot.com

Valor	Cantidad
0	1

Dominio: www.ormigo.com

Valor	Cantidad
0	9
-1	24

Dominio: www.chroniclesofchaos.com

Valor	Cantidad
0	17
6	391
7	776
8	851
9	497
10	101

Dominio: www.kollermedia.at

Valor	Cantidad
6	19
7	26
8	26
9	7
10	7

En base a esta información se infirieron los siugientes rangos:

Dominio	Rango mínimo	Rango máximo
www.gamezebo.com	0	5
www.yellowbot.com	0	5
www.chroniclesofchaos.como	0	10
www.kollermedia.at	1	10
www.chip.de	1	100

Luego estos resultados se corroboraron manualmente revisando todos los valores que poseía el dominio tanto dentro del rango 1-5 como fuera . Desde ya que no se puede asegurar la veracidad de los rangos encontrados, dado que por ejemplo, podría suceder que para el dominio www.kollermedia.at por mas absurdo que parezca el rango real es 1-11 pero no existe ningún review con el valor 11 en su rating. Nota: Manualmente se verificó que ormigo poseía simplemente reviews mal generados.

Una vez con estos resultados se puede proceder a corregir los valores de los reviews del dataset.

Estrategia: Disponiendo de los rango correctos para cada dominio se recorrieron todos los reviews de cada dominio resultante, y se les adicionaron las propiedades rev:minRating y rev:maxRating con los valores correspondientes. Esto no sólo para los reviews con ratings erróneos encontrados en la etapa de evaluación (fuera del rango 1-5) dado que, para los reviews de cada dominio

con rango encontrado, por más que el rating se encuentre dentro del rango 1-5 y aparente ser correcto, ya se sabe que dicho dominio se maneja con rangos diferentes y el valor real del rating cambiaría (no es lo mismo el valor 3 en un rango 1-100 que en un rango 1-5)

Resultados:

Se le asignaron propiedades a:

www.gamezebo.com	3905
www.yellowbot.com	8955
www.chroniclesofchaos.como	4966
www.kollermedia.at	9968
www.chip.de	13751
Total	41545

8.5. Curado nº 5 - Análisis experto

Propiedades inexistentes o semánticamente incorrectas

Son los casos de itemreviewed, dtreviewed, publishDate y reviewBody, ya analizados en la evaluación. Estrategia: En este caso para todas el procedimiento es exactamente el mismo, y es bastante sensillo, crear una nueva propiedad con el mismo sujeto y objeto, pero la propiedad indicada para cada caso y luego borrar la tripleta incorrecta

itemreviewed	itemReviewed
dtreviewed, publishDate	datePublished
description	reviewBody

Comentarios en forma de reviews

Objetivo: Corregir la clase de los recursos que son de tipo schema:UserComments pero fueron clasificados como schema:Review

Estrategia: Lo que se intentará es cambiar la clase Review del recurso, y transferir todas las propiedades a una equivalente en la clase UserComments. Por lo que primero se debe inspeccionar qué propiedades continen dichos recursos.

```
SELECT distinct ?p
WHERE{
?s a <http://schema.org/Review> .
?s <http://schema.org/review> ?review .
?review ?p ?o .
}
```

Las propiedades resultantes fueron

Propiedad encontrada	Equivalente en Comment
http://schema.org/description	http://schema.org/commentText
http://schema.org/reviewBody	http://schema.org/commentText
http://schema.org/datePublished	http://schema.org/commentTime
http://schema.org/author	http://schema.org/creator

Entonces simplemente para solucionar el problema alcanza con la siguiente consulta SPARQL:

```
DELETE {
GRAPH ?g{
?s <http://schema.org/review> ?review .
?review <http://schema.org/description> ?description .
?review <http://schema.org/reviewBody> ?body .
?review a <http://schema.org/Review> .
?review <http://schema.org/author> ?author .
?review <http://schema.org/datePublished> ?date .
}
}
INSERT{
GRAPH ?g{
?s <http://schema.org/comment> ?review .
?review <http://schema.org/commentText> ?description .
?review <http://schema.org/commentText> ?body .
?review <http://schema.org/commentTime> ?date .
?review <http://schema.org/creator> ?author .
?review a <http://schema.org/UserComments> .
}
}
WHERE{
GRAPH ?g {
?s a <http://schema.org/Review> .
?s <http://schema.org/review> ?review .
OPTIONAL {  ?review <http://schema.org/description> ?description . }
OPTIONAL { ?review <http://schema.org/reviewBody> ?body . }
OPTIONAL { ?review <http://schema.org/datePublished> ?date . }
OPTIONAL { ?review <http://schema.org/author> ?author . }
}
}
```

Por último, el problema encontrado con los objetos de la propiedad author, que deberían ser recursos de tipo schema:Person o schema:Organization, necesitan un arreglo. Estrategia: Primero es necesario saber qué representan esos textos utilizados como objeto de author, una observación de esos valores mostró que en su gran mayoría representan usernames dentro de la página publicadora, y el resto nombre y apellido. Esto genera un problema a la hora de llevar a cabo la corrección, ya que de por sí, schema:Person utilizada una propiedad

para el nombre (givenName) y otra para el apellido (familyName), y no hay una propiedad específica para un username, aunque existe la propiedad alternativeName que se utiliza para los nombres que no encajen en las demás propiedades que podría ser utilizado para el username. Pero cómo saber automáticamente cuando es un username, y cuando es un nombre completo, y más aún, en los nombres completos, cómo saber cuál es el nombre y cuál es el apellido?. Esto no puede ser automáticamente resuelto, por lo que se optó por incluir a todos (tanto usernames como nombres completos) dentro de la propiedad name, heredada de schema:Thing. Entonces como procedimiento final, se cambio el objeto de la spropiedades author, por un recurso de tipo Person, con una propiedad name conteniendo como objeto el antiguo valor de author. Realizado con la siguiente consulta SPARQL:

```
DELETE {
GRAPH ?g{
?s <http://schema.org/author> ?author .
}
}
INSERT{
GRAPH ?g{
?s <http://schema.org/author> _:foo .
_:foo a <http://schema.org/Person> .
_:foo <http://schema.org/name> ?author .
}
}
WHERE{
GRAPH ?g {
?s a <http://schema.org/Review> .
?s <http://schema.org/author> ?author .
}
}
```

Capítulo 9

Integración de los Datos

Como quedó explicado en la sección estrategia, integración comprende los procedimientos que se realizan con el objetivo de obtener una visión más unificada del dataset. Esto puede significar, modificar los datos y ontologías y también agregar información faltante en los ítems/reviews. Este es el paso más importante de todo el proceso y la posibilidad de conseguir realizar una aplicación correcta que satisfaga los requerimientos depende del éxito obtenido en el mismo. En base a los requerimientos establecidos, se pueden pensar en distintos procedimientos que son necesarios para poder explotar los datos correctamente:

Unificación de vocabularios: En el primer paso del proceso se realizó una selección de los vocabularios con los que se trabajará, estos vocabularios modelan el mismo dominio de información pero muchas veces de forma distinta. Explotar información modelada de diferentes maneras puede resultar innecesariamente engorroso, y también para quienes en un futuro quieran hacer uso del dataset (ya que deberán consutlar la información previamente conociendo todas las ontologías de review existentes).
Unificación de autores: Es muy importante para lograr cumplir el objetivo de los requerimientos que si dos reviews distintos fueron generados por el mismo autor, esto quede explícitamente asentado, para poder realizar los algoritmos de recomendación.
Unificación de ítems: El paso más difícil y más importante, si un ítem está modelado dos veces de forma distinta debería saberse que se trata del mismo.
Unificación de formato de ratings: Si bien este paso sería válido también en la etapa de curación, se decidió que encajaba mejor acá. Se trata de llevar todos los ratings a un formato único numérico. Hay sitios que establecen los ratings de distintas formas como por ejemplo “4 out of 5”. En sí no es un problema sintáctico dado que no hay un rango establecido para las propiedades de rating, pero va a presentar problemas a la hora de utilizar la información en conjunto, ya que se requerirá tener en cuenta todos los casos distintos de formatos de ratings. Puede verse como un problema semántico, tomando en cuenta el ejemplo 4 out of 5, no es más que una sobrecarga de propiedades, ya que como valor de

la propiedad rating, se especifica no sólo el valor de dicha propìedad, sino que además el valor de la propiedad maxRating.
Unificación de tipos de ítems: Esto es muy útil para darle más posibilidades a la aplicación, y es agrupar los ítems según su tipo (Libro, película, Hotel, Auto, etc), de manera que por ejemplo si un usuario quiere que le recomienden un libro, la aplicación sólo le ofresca libros. Para la ontología schema, esto puede ser más sencillo dado que los ítems ya se encuentran con su tipo identificado con la clase, en el caso de libro, sería cuestión de buscar ítems con la clase schema:Book. El problema es que esto puede no ocurrir siempre, Un libro, puede estar clasificado como schema:Product, por lo que necesitaría que también lo clasifiquen como Book.

Por razones de que la tesis ya cubrió el desarrollo suficiente, sólo se intentarán implementar los puntos 1, 3 y 4 de la integración. Ya que los otros dos son demasiado abarcativos y pueden requerir demasiado trabajo.

9.1. Unificación de vocabularios

Objetivo: Conservar los reviews del dataset en una única ontología. Lo que simplificará las búsquedas, agregaciones y otras operaciones. Además de contener datos semánticamente más parecidos. Y también facilitará la comprensión del dataset a quienes quieran hacer uso del mismo en un futuro (ya que sólo deberán concentrarse en una sola ontología)
Estrategia: Deberá como primer paso realizarse una selección de vocabulario, en la cual se escoja uno para que quede como el único en el dataset. En este caso de estudio el vocabulario elejido es Review Ontology, ya que todas sus propiedades son suficientes para cumplir los requerimientos y este es mucho más simple, lo que facilitará su utilización. El vocabulario de schema poseé demasiadas propiedades que son irrelevantes para este caso de estudio. Además vale la pena recordar los problemas encontrados en el capítulo evaluación causados por la ontología. Como segundo paso, buscar para cada propiedad de schema:Review utilizada en el dataset su contraparte en la ontología purl.
La equivalencia entre las propiedades ya fueron realizadas en el capítulo de selección de vocabularios. Pero sólo fue realizado con las propiedades relevantes al caso de estudio. Las siguientes propiedades (que valga la redundancia son irrelevantes al caso de estudio) fueron utilizadas dentro del dataset, y no se estableció su equivalente en el otro vocabulario: http://schema.org/about y http://schema.org/keywords. Si bien, la falta de estas propiedades no impactará sobre la aplicación final, es conveniente no perder datos para la publicación del dataset, ya que a alguien podría resultarle útil. En la etapa de evaluación se observó que la propiedad about, se utilizó en el dataset sólo para poner contener el valor “This is required” en todos los casos. Por lo que no incluir esta

propiedad en el dataset no producirá una pérdida de información. Y la propiedad keywords puede ser reemplazada por http://purl.org/dc/terms/subject. Ya que en la documentación de dublin core, proponen la utilización de esta propiedad utilizando keywords. El caso de reviewRating, que tiene como objeto un schema:Rating, no puede ser explicitamente reemplazado, ya que esa propiedad sólo es utilizada para desacoplar el rating del review, situación que no se da en la otra ontología. Pero las propiedades dentro del rating se encuentran todas con su equivalente en purl, por lo tanto no causará ningún problema. Por último, schema:Person que es rango de la propiedad author, debería ser mapeado a foaf:Person que es rango de la propiedad reviewer. El problema se encuentra en lo analizado en la etapa de curado, que se observó que los nombres de los autores podía ser nomrbes completos o usernames. Y no existe una propiedad dentro del vocabulario de foaf:Person que sea tan genérico como para contemplar todo slos casos posibles, como sí existe en vCard. Además de que todos los previos valores de reviewer están conformados por un vCard, como también ya se analizó en la etapa de curado. Por estos motivos se determinó continuar utilizando vCards en lugar de recursos de tipo foaf, y dichos vCards utilizarán la propiedad fn (que contempla todos los casos mencionados). Finalmente el mapeo de propiedades quedó:

Propiedad schema	Propiedad de reemplazo
name	title
reviewBody	text
datePublished	dc:date
author	reviewer
reviewer	reviewer
ratingValue	rating
bestRating (reviewRating)	maxRating
worstRating (reviewRating)	minRating
url	vcard:url
keywords	dc:subject
name (author)	vcard:fn (reviewer)
comments	hasComment
author (comment)	commenter
text (comment)	text

No hubo problemas con el algoritmo que realizaba el procedimiento.

9.2. Unificación de formatos de ratings

Objetivo: Disponer de todos los ratings de los reviews, con el mismo formato. Un número sin decimales, para facilitar la implementación de operaciones como: "Listar los reviews más populares de determinada fecha" o "Hacer un promedio de puntaje para un ítem determinado"
Estrategia: Exactamente el mismo principio que para corregir los formatos de las fechas, sólo que un poco más simple porque sólo se realizará para una propiedad

en lugar de 4.
Primero se realizó una consulta SPARQL buscando los valores de ratings que no estén dentro de los correctos.

```
SELECT distinct ?value (count (distinct ?review) as ?cantidad)
WHERE{
?review <http://local.org/id> ?id .
?review <http://purl.org/stuff/rev#rating> ?value .
FILTER NOT EXISTS{
?review <http://local.org/duplicateOf> ?dup .
}
FILTER (!REGEX(str(?value), "^[0-9]{1}$", "i")) .
FILTER (!REGEX(str(?value), "^[0-9]{2}$", "i")) .
FILTER (!REGEX(str(?value), "^[0-9]{3}$", "i")) .
}GROUP BY ?value ORDER BY DESC(?cantidad)
```

Y luego analizando los resultados se fueron agregando los siguientes filtros a la consulta:

```
FILTER (!REGEX(str(?value), "^[0-9]{1} $ ", "i")) .
FILTER (!REGEX(str(?value), "^[0-9]{2}\\.[0-9]{1}$", "i")) .
FILTER (!REGEX(str(?value), "^[0-9]{1}\\.[0-9]{1}$", "i")) .
FILTER (!REGEX(str(?value), "^[0-9]{1},[0-9]{1}$", "i")) .
FILTER (!REGEX(str(?value), "^[0-9]{1} ", "i")) .
FILTER (!REGEX(str(?value), "^Rating [0-9]{1}", "i")) .
FILTER (!REGEX(str(?value), "^About.com Rating  [0-9]{1}", "i")) .
FILTER (!REGEX(str(?value), "^[0-9]{1}\\.[0-9]{1} ", "i")) .
FILTER (!REGEX(str(?value), "Rating [0-9]{1}", "i")) .
FILTER (!REGEX(str(?value), "\\([0-9]{1} ", "i")) .
```

Una vez que los resultados arrojados por la consulta fueron 0. Se realizó el algoritmo que lleve el rating al formato correspondiente según su expresión regular.

Resultados:

Expresión regular	Cantidad de ratings
`^[0-9]{1} $`	3722
`^[0-9]{2}\\.[0-9]{1}$`	250
`^[0-9]{1}\\.[0-9]{1}$`	63695
`^[0-9]{1},[0-9]{1}$`	45927
`^[0-9]{1}`	26314
`^Rating [0-9]{1}`	149
`^About.com Rating  [0-9]{1}`	2613
`^[0-9]{1}\\.[0-9]{1}`	5603
`Rating [0-9]{1}`	3300
`\\([0-9]{1}`	4176

Capítulo 10

Publicación del Dataset Curado

Para este paso, se escogió en un primer lugar el aprovicionamineto de un servicio web para ser accedido públicamente. Esto beneficia a los usuarios que requieran hacer un uso inmediato y simple de la información contenida. Ya que si se decidiera proporcionar un volcado de datos, el usuario deberá tener conocimiento de la manupulación del disco, así como recursos de hardware y tiempo de configuración.

Para proceder con la publicación del dataset de la forma mencionada, el motor que se optó por utilizar es Fuseki, ya que proporciona un medio de implementación extremadamente sencillo, debido a que funciona perfectamente en conjunto de un dataset TDB, que es con el que actualmente se cuenta.
Fuseki además, tiene la ventaja de ser fácilmente configurable, y no requiere instalación. En el caso de haber elegido otros motores, habría que haber hecho una conversión del dataset para obtenerlo en un formato adecuado, además del esfuerzo necesario para instalar y configurar toda la aplicación.

Por otro lado se consideró también que podría existir la necesidad por parte de algunos usuarios (la minoría), en hacer un uso más complejo del dataset (recordar que la utilización de endpoints sparql tiene sus restricciones), para lo cual requerirían manipular el dataset completo por su cuenta.
Resolver esta situación, no representó casi inconveniente, debido a la alta capacidad de compresión de datos que poseé una base de datos TDB, que pasó de 8,4Gb a 1,0Gb. Dicho dataset con esas características puede ser fácilmente publicado para una posible descarga muy fácilmente.

Capítulo 11

Explotación del Dataset

Capítulo 12

Conclusiones y trabajo futuro

Capítulo 13

Bibliografía

Bibliografía

Apéndice A

Publicaciones