UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

TÍTULO: Normalización e identificación de objetos duplicados sobre contenido extraído de la web

AUTORES: Gonzalo Mastronardi

DIRECTOR/A: Alejandro Fernandez

CODIRECTOR/A: Sergio Firmenich

CARRERA: Lic. en Sistemas

Resumen

La web es la principal fuente de información disponible, aunque los datos carecen de estructura y significado para las maquinas. La web semántica emerge para solucionar esto, pero su adopción es lenta, y la construcción de aplicaciones que utilicen los datos de la web implica extraerlos de manera manual. Dada la imprecisión de estas herramientas, se propuso, en primera instancia, la construcción un framework de normalización, y luego, un segundo framework de identificación de duplicados, con el objetivo de recolectar, estructurar y normalizar información dispersa, otorgándole sentido para las maquinas.

Palabras Clave

Web semántica – Información – Extracción de objetos - Calidad de datos – Framework - Normalización de objetos – Identificación de duplicados – Dbpedia -

Conclusiones

Procesar los datos recolectados a través de los frameworks de normalización e identificación de duplicados no solo nos permite obtener una fuente de datos limpia y confiable, sino que tambien nos posibilita consolidar objetos duplicados en una única entidad con la máxima información posible. Este proceso no solo otorga coherencia a los datos, sino que tambien le agrega significado a través de anotaciones proporcionadas por la web semántica, permitiendo una comprensión mas profunda y precisa para las maquinas.

Trabajos Realizados

- Presentación de la web como la mayor fuente de información disponible, destacando la desestructuración de los datos
- Se presentó el concepto de web semántica destacando su lenta adopción.
- Se analizó porque es importante extraer información de la web, las herramientas disponibles para ello, y las dificultades que ello conlleva.
- Construcción de un framework de normalización y un framework de identificación de duplicados.
- Realización de las pruebas correspondientes y evaluación de ambos frameworks a través de la utilización de los mismos por parte de dos colegas
- Análisis y evaluación de los resultados obtenidos

Trabajos Futuros

Desarrollo de una interfaz gráfica que permita seleccionar de manera intuitiva y sencilla aquellos objetos que efectivamente se correspondan al mismo en la vida real, consolidando un único objeto con la mayor cantidad de información posible.

Fecha de la presentación: Febrero 2024

Normalización e identificación de objetos duplicados sobre contenido extraído de la web.

Mastronardi, Gonzalo Agustín

27 de febrero de 2024

Resumen

En la actualidad, la web se presenta como la principal fuente de información disponible. Esta información, distribuida entre los miles de sitios web, se muestra de manera legible para los usuarios, pero carece de estructura y significado para las máquinas. La concepción de la web semántica surge como una solución para superar esta limitación, incorporando significado a los datos mediante anotaciones semánticas provenientes de DBpedia. DBpedia es un proyecto que transforma la información de Wikipedia en datos estructurados, otorgando accesibilidad y utilidad a la información de manera que sea útil para las máquinas. A pesar de estos avances, la adopción de la web semántica es lenta, y la construcción de aplicaciones que utilicen los datos web implica su extracción de manera manual a través de diversas herramientas desarrolladas para ello. Estas herramientas no son del todo precisas, por lo que emerge la necesidad de llevar a cabo un proceso de limpieza o normalización de datos para trabajar con ellos fiablemente. Para abordar este problema, se desarrolló un framework de normalización de objetos, con el propósito de corregir los datos extraídos de los objetos de información y asegurar una fuente de datos fiable. Posteriormente, se implementó un segundo framework de identificación de duplicados, es decir, aquellos que se correspondan al mismo objeto en la vida real. De este modo, logramos recolectar, estructurar y normalizar información que se encontraba dispersa en la web, identificando potenciales duplicados para finalmente consolidar un único objeto con la máxima cantidad de información posible, otorgándole sentido a través de la identificación de manera precisa a un objeto correspondiente a la DBpedia.

Índice general

1. Introducción **3**
- 1.1. Buscando objetos en la web . . . 3
- 1.2. ¿Por qué extraer objetos? . . . 5
- 1.3. El desafío al extraer objetos . . . 5
- 1.4. Objetivos y aportes de la tesina . . . 7
- 1.5. Organización de este documento . . . 8

2. Antecedentes y contexto **10**
- 2.1. Extracción de objetos de la web . . . 11
- 2.2. Calidad de los datos . . . 12
- 2.3. Integración de datos . . . 12
- 2.4. Limpieza de datos y detección de duplicados . . . 13
  - 2.4.1. Identificando duplicados . . . 13

3. Estrategia general **15**
- 3.1. Framework de normalización de objetos . . . 17
- 3.2. Framework de tratamiento de duplicados . . . 18

4. Framework de normalización de objetos **20**
- 4.1. Diseño general . . . 21
- 4.2. Utilización del framework . . . 22
  - 4.2.1. Creando un nuevo normalizador . . . 23
  - 4.2.2. Configurando la aplicación normalizadora . . . 26

5. Framework de tratamiento de duplicados **31**
- 5.1. Diseño general . . . 33
- 5.2. Utilización del framework . . . 36
  - 5.2.1. Extendiendo el framework . . . 37
  - 5.2.2. Configurando la aplicación normalizadora . . . 41

6. Evaluación **44**
6.1. Selección de los participantes 44
6.2. Entrenamiento de los participantes 44
6.3. Preparación de la prueba 45
6.3.1. Cuestionario 47
6.4. Resultados 48
6.4.1. Análisis del cuestionario 55

7. Conclusiones y trabajo futuro **58**
7.1. Conclusiones 58
7.2. Trabajo futuro 59

Apéndice **61**

A. Información y enlaces de utilidad **61**

Capítulo 1

Introducción

1.1. Buscando objetos en la web

Con el correr de los años la web se convirtió en la fuente de información más grande y de mayor crecimiento disponible. Los contenidos web crecen diariamente en tamaño y diversidad, y es probable que mucha de la información que se necesita para cumplir con las actividades diarias ya se encuentre allí disponible. Con frecuencia, los usuarios de la web buscan productos para la compra (Amazon o MercadoLibre), personas (Facebook) y/o lugares. Al realizar cualquiera de estas búsquedas, cada sitio web ofrece la información de acuerdo a su modo y diseño, sin hacer foco en destacar de manera precisa y estructurada la información respecto de los objetos de la búsqueda, como pueden ser sus propiedades y las relaciones entre ellas. Esto se debe a que la web se encuentra en constante crecimiento y el objetivo no es entender los datos como pertenecientes a objetos individuales y estructurados, sino mostrarlos al usuario final de manera atractiva [1].

La falta de estructura y uniformidad en la presentación de elementos similares en diferentes sitios web dificulta la capacidad de aprovechar la información disponible en la web para la toma de decisiones. La información tiende a encontrarse dispersa en diversos recursos en linea, lo que complica el proceso de búsqueda, ya que los usuarios se ven obligados a navegar entre múltiples sitios para encontrar la información necesaria. Esta dispersión provoca un desafió para los usuarios, quienes suelen sentirse abrumados y pueden incluso no encontrar aquello que están buscando, generando la sensación de frustración. Por ejemplo, un usuario interesado en comprar un teléfono celular, realiza una búsqueda y los motores actuales ofrecen el mismo producto en un conjunto similares de sitios web, en los que dicho producto se muestra al usuario según la estructura y la información que el proveedor del sitio dispon-

ga. Esto conlleva a que el usuario deba inspeccionar esta lista de sitios web comparando un mismo producto en diferentes fuentes, pudiendo encontrar información que le es relevante esparcida en diferentes sitios para finalmente tomar la mejor decisión en base a sus necesidades.

La web de hoy día está dirigida a la lectura humana y orientada exclusivamente a la visualización, es decir, está construida de manera tal que su objetivo no es facilitar el procesamiento automático de los datos de cualquier sitio web, sino que solo busca presentarlos de manera legible y atractiva al usuario. Esta web está construida de modo tal que sus documentos contienen la suficiente información para que una máquina pueda presentarla, pero no comprenderla [2].

Debido al gran crecimiento del volumen de datos -y a su vez la cantidad de ellos que se producen cada segundo en millones de dispositivos- existe la necesidad de administrarlos de manera eficiente. En base a esto surge la pregunta si es posible extender la web añadiendo algún tipo de información a los documentos almacenados, de manera que las máquinas utilicen esta información extra para lograr comprender con mayor profundidad el contenido los documentos. Es aquí donde nace la idea de cambiar la web actual o tradicional en lo que se conoce como la Web Semántica.

El termino 'Web Semántica' fue introducido por primera vez por Tim Berners-Lee, fundador de la Worl Wide Web -WWW-, Uniform Resource Locator -URL- y HyperText Markup Languaje -HTML-. Esta web se centra en el significado de los datos en lugar de hacer hincapié en la estructura sintáctica. La web semántica -considerada una extensión de la WWW- utiliza la web no solo de acuerdo a las necesidades del humano, sino que también permite a la maquina entenderla de manera eficiente e inteligente para mejorar los resultados de búsqueda deseados.

La Web Semántica provee las tecnologías y estándares necesarios para lograr dos objetivos fundamentales:

- Añadir significado a la web actual, de manera que sea comprensible para las máquinas
- Posibilitar la ejecución automática de ciertas tareas que de algún otro modo se hubieran realizado manualmente.

Su esencia radica en enriquecer la información web con un conocimiento semántico, permitiendo una mayor comprensión por parte de las maquinas y facilitando la automatización de diversas operaciones.

1.2. ¿Por qué extraer objetos?

Si bien la Web Semántica propone una estrategia para afrontar el desafío de conseguir que un sitio web sea al mismo tiempo atractivo para el usuario e interpretable para las computadoras, su adopción es lenta. La gran mayoría de los sitios web actuales no implementan las propuestas de la Web Semántica. Por consiguiente, construir aplicaciones que aprovechen la información disponible en los sitios web implica ser capaz de extraer objetos de información a partir de datos no estructurados.

Al navegar por la web con un navegador web normal, probablemente el usuario encuentre varios sitios en los que considere la posibilidad de recopilar, almacenar y analizar los datos presentados, por ejemplo para:

- Comparar características y precios al momento de tener que comprar un producto electrónico tal como un teléfono celular
- Recuperar información a partir de una tabla que resulte interesante para realizar un análisis estadístico
- Obtener una lista de reseñas de un sitio de películas para realizar minería de texto, crear un motor de recomendaciones, detectar criticas falsas, etc.
- Monitorear un sitio de noticias en busca de nuevas historias sobre un tema particular de interés

Estos ejemplos son algunos que empujan la necesidad de extraer información para poder estructurarla con el objetivo de usar dichos datos en sistemas de recomendación o de toma de decisiones.

1.3. El desafío al extraer objetos

La naturaleza no estructurada de la web actual no siempre facilita la recopilación o exportación de datos de una manera fácil. Los navegadores web son muy buenos para mostrar imágenes, animaciones y presentar sitios web a través de diseños visualmente atractivos para los humanos, pero en la mayoría de los casos no exponen una forma simple de exportar sus datos [3].

Muchos sitios web en la actualidad brindan una interfaz al usuario llamada 'Application Programming Interface' -API- que proporciona un medio para que el mundo exterior acceda a su repositorio de datos de una manera estructurada, destinada a ser consumida y accedida por programas informáticos. Twitter, Facebook, LinkedIn y Google, por ejemplo, proporcionan APIs

para buscar y publicar tweets, obtener una lista de amigos y de "me gusta", ver con quién uno está conectado, etc. Estas APIs son excelentes medios para acceder a las fuentes de datos, aunque muchos sitios web no las proporcionan. Ante esta situación surge el concepto de 'Scraping' [4], con el fin de extraer datos de la web mediante herramientas desarrolladas para obtener la información necesaria en un formato que sea de utilidad para el desarrollador, manteniendo al mismo tiempo la estructura de los datos.

Aunque la web está llena de información que podríamos usar para tomar decisiones racionales y efectuar una compra, raramente lo hacemos. Tomar una decisión basada en información algunas veces requiere procesos de decisión complejos, incluso basándose en múltiples criterios que la gente desconoce. Cuando la circunstancia lo amerita, algunas personas recurren a hojas de cálculo[5]. Una búsqueda de una hoja de cálculo para comprar, por ejemplo, teléfonos celulares, retornará un conjunto de plantillas predefinidas para ayudar a los compradores a comparar y seleccionar celulares. Estas hojas ayudan a las personas a identificar los atributos clave que deberían buscar y almacenar de manera estructurada. Algunas personas que dominan su funcionamiento las usan para crear plantillas similares, aunque en estos casos el uso de dichas hojas se limita a las mismas funcionalidades que nos proveen los sitios web, es decir: seleccionar candidatos, tener una vista general de las propiedades, ordenarlos y compararlos atributo por atributo [6].

En busca de simplificar el manejo de información y facilitar la interacción con los objetos de datos en entornos web, se desarrolló 'Web Objects Ambient (WOA)' [2], una plataforma que permite a los usuarios, sin conocimientos técnicos profundos, crear su propio espacio de información al reutilizar contenido web existente. Utilizando WOA, los usuarios pueden modelar sus objetos de interés al definir plantillas sobre páginas web específicas, utilizando recopiladores de datos que seleccionan elementos del DOM y le asignan nombres, valores y tipo de objeto [1]. Las plantillas resultantes permiten extraer información de diferentes sitios web, como por ejemplo Garbarino o Movistar, estructurando y almacenando datos previamente dispersos para su posterior análisis y toma de decisiones.

Si bien WOA ofrece una eficaz herramienta para estructurar y recopilar información, es importante señalar que esta metodología no incorpora mecanismos para garantizar la calidad de los datos extraídos. La plataforma

[1]Estos tipos de objeto están definidos en DBpedia. DBpedia es el repositorio de información estructurada más extenso en la web, siguiendo el patrón de un grafo de conocimiento abierto (OKG - Open Knowledge Graph). Este proyecto realiza una transformación de la información proveniente de Wikipedia en datos estructurados. Su objetivo es proporcionar accesibilidad y utilidad a la información de manera que resulte beneficiosa para las máquinas.

se centra en simplificar el proceso de extracción para permitir a los usuarios definir plantillas y recopiladores, pero la validación y aseguramiento de la precisión de los datos extraídos es responsabilidad del usuario. Como resultado, dentro de este marco de recolección de objetos, nos enfrentamos a problemas que afectan la calidad de los datos, como por ejemplo:

- Campos incompletos o vacíos: debido a que alguna fuente puede omitirlos por considerarlos irrelevantes, o bien hay campos que son requeridos pero se dejan en blanco.
- Errores de sintaxis: debido a la aparición de caracteres especiales o codificación.
- Errores de semántica: datos que no coinciden con los valores esperados, es decir, que no se adaptan al contexto. En un campo llamado 'Precio', el valor esperado debería ser un monto determinado y no, por ejemplo, 'Argentina'.
- Atributos duplicados: en los que la información puede llegar a repetirse.
- Campos con información incorrecta: generalmente debido a problemas que surgen al momento de cargar los datos.
- Información extra innecesaria: de manera que no resulta relevante.

Estos problemas de calidad de datos son comunes en diversas áreas de investigación, como en descubrimiento de conocimiento en bases de datos (Knowledge Discovery), en almacenamiento de datos y en la integración de sistemas y servicios electrónicos[7]. En respuesta a esto, surge la necesidad de desarrollar un sistema dedicado a llevar a cabo procesos de limpieza de datos sobre la información recolectada. La implementación de dicho sistema es sumamente necesaria para garantizar una mejor calidad de los datos almacenados.

1.4. Objetivos y aportes de la tesina

A partir de las dificultades y los problemas mencionados anteriormente basados en la recolección de objetos a través de las herramientas destinadas para ello, en este caso mediante WOA, se propuso, en primer término, la construcción de un framework de normalización. Este framework tiene como objetivo limpiar y/o corregir los datos asociados a los objetos de información recolectados. De esta manera buscamos contar con una fuente de

datos ordenada y confiable lista para la siguiente etapa: comparar los objetos para identificar potenciales duplicados. En este contexto se ha desarrollado un segundo framework dedicado a la tarea de comparación. Este framework nos posibilita la identificación de objetos candidatos a duplicados, es decir aquellos que pertenezcan al mismo objeto en la vida real. El problema de identificar pares de registros que representen la misma entidad (registros duplicados) es uno de los pasos esenciales en el proceso de limpieza de datos. De este modo buscamos cumplir con los principales objetivos planteados, la limpieza de los datos en primer término y luego la identificación de posibles candidatos a duplicados.

1.5. Organización de este documento

En el siguiente capítulo abordaremos el contexto y los antecedentes relevantes, centrándonos en aquellas investigaciones previas que han evaluado el tema de estudio. Veremos aquellas soluciones propuestas que si bien comparten similitudes con la problemática abordada, no demostraron ser eficaces o han presentado ciertos tipos de limitaciones.

En el tercer capítulo, presentaremos la estrategia general que abarca la solución propuesta. Describiremos, en primer termino, la solución para la problemática de normalización de objetos y, posteriormente, se abordara la solución para la de identificación de duplicados, dando un marco general para comprender la solución propuesta.

En los dos capítulos siguientes se desarrollaran en profundidad y de manera técnica las soluciones para cada problemática en particular. En ellos describiremos la estructura y el funcionamiento de cada framework desarrollado. Se detallara el diseño de cada framework, así como la utilización cada uno, junto a algunos ejemplos que ayudaran a su comprensión.

En el capitulo de evaluación, analizaremos las soluciones propuestas por dos participantes a quienes se les presentaron los frameworks desarrollados. Estos participantes llevaron a cabo una evaluación detallada y extendieron los frameworks para poner a prueba su funcionamiento práctico. Con el objetivo de obtener una evaluación integral se les realizó un cuestionario específico con una serie de preguntas las cuales abordaron diversos aspectos, desde la usabilidad y la eficacia hasta la identificación de posibles mejoras o soluciones. Su participación nos otorgo valiosas contribuciones para mejorar y perfeccionar la solución propuesta.

Por ultimo, en el capitulo de conclusiones y trabajo futuro, realizaremos un breve análisis abarcativo del trabajo, llegando a conclusiones basadas en los resultados obtenidos. También exploraremos opciones para trabajos

futuros que, por limitaciones de tiempo u otros motivos, no se incluyeron en el presente.

Capítulo 2

Antecedentes y contexto

Los usuarios de la web con más frecuencia buscan objetos específicos de su interés como por ejemplo productos (Mercado Libre o Amazon), personas (Facebook), o bienes raíces (ArgenProp) en lugar de palabras clave simples que se pueden encontrar en páginas web. Los motores de búsqueda actuales muestran al usuario la información según el modo y el diseño que dispongan y no proveen los datos de manera estructurada y precisa. Esto sucede debido a la gran escala que hay en la web, por lo que dichos motores no tienen como objetivo principal estructurar la información [1].

Dado que la mayoría de los motores de búsqueda devuelven resultados basados en las actividades recientes de los usuarios y no indexan toda la base de datos de la web, se convierte en una necesidad tener un motor de búsqueda unificado y totalmente funcional que pueda explorar todo el mundo de la World Wide Web -WWW- antes de producir su resultado.

Hay dos tipos de problemas que enfrenta cualquier motor de búsqueda mientras trabaja con la información proporcionada en la web [8]:

- En primer lugar, el desafío de asignar la consulta a la información disponible en internet y producir los resultados deseados e inteligentes.
- En segundo lugar, el obstáculo para identificar eficientemente los datos preferidos de los documentos vinculados dados.

La web semántica puede abordar este problema con la ayuda de anotaciones semánticas. Una anotación se trata de asignar a un objeto dentro de un texto una referencia a un repositorio semántico en el que hay más conocimiento. Las tecnologías webs semánticas se están volviendo cada vez mas populares y penetrando en el mercado industrial. Organizaciones como Schema.org y Facebook están intentando guiar en la web. El motor de búsqueda más utilizado, Google está intentando estructurar su motor de una manera

más semántica mediante una herramienta denominada 'Knowledge Graph', la cual a partir de la búsqueda de una palabra clave muestra en una caja información genérica de la búsqueda, aunque los resultados en la mayoría de los casos siguen siendo insuficientes.

La web semántica se centra en el significado de los datos en lugar de hacer hincapié en la estructura sintáctica. A su vez, se están llevando a cabo varias investigaciones sobre la parte de ontología (relación entre las entidades) de la web semántica, ya que es su columna vertebral, un lenguaje de esquema y representación.

2.1. Extracción de objetos de la web

La enorme cantidad y el constante aumento de datos disponibles en el mundo hizo que nuestra generación viva en una zona borrosa que está inundada de datos. Las computadoras personales hacen que el asunto sea bastante eficiente y nos ayudan a guardar fácilmente nuestros datos. La minería de datos utilizando computadoras ayuda a automatizar o al menos simplificar la búsqueda de datos. Muchas bases de datos o medios de almacenamiento necesitan usar la minería de datos para lograr mejores estrategias de búsqueda, intentando encontrar patrones o relaciones en donde haya gran cantidad de información [8].

Debido al gran volumen y la diversidad de información disponible, se hace necesario gestionar de manera eficiente los datos de la web. El propósito principal es extraer estos datos con el fin de estructurarlos y analizarlos. La extracción automática de datos de la web es un hecho que ha sido investigado ampliamente [1] [9]; sin embargo, luego del proceso de extracción de datos, nos enfrentamos a errores semánticos, los cuales suelen darse debido a un problema en la fuente, o bien por parte de las herramientas utilizadas. La mayoría de los enfoques confían en el hecho de que los datos que se extraen contienen la información correcta, y que podemos utilizar dicha información para analizarla [1] y tomar decisiones.

El hecho de extraer objetos de la web automáticamente es un gran desafío, y se ha empleado mucho trabajo para llevar a cabo este proceso a través de sistemas diseñados con dicho propósito [9] [10]. Tales sistemas, dado un conjunto de páginas web como entrada, intentan automáticamente crear programas denominados 'wrappers', capaces de extraer objetos de interés sobre dichas páginas. Que estos wrappers extraigan correctamente los objetos y toda la información relevante es realmente complejo, e incluso si somos capaces de crear dichos wrappers, luego del proceso de extracción surgen nuevos problemas. Un ejemplo de esto es la posibilidad de que los objetos residan

en diferentes fuentes y, como resultado, la salida del proceso de extracción contenga múltiples objetos duplicados que, en última instancia, se refieren al mismo objeto en la realidad.

2.2. Calidad de los datos

Las bases de datos cumplen un rol muy importante en cualquier sistema de información. Debemos tener en cuenta que los datos que obtenemos son resultado de un proceso de extracción, y dicho proceso no resulta perfecto. Esto provoca la necesidad de hacer hincapié en mejorar la calidad de la información ya que resulta lo mas valioso. Muchas industrias y sistemas dependen de la precisión de sus bases de datos para poder llevar a cabo diferentes operaciones, incluso la calidad de la información almacenada en estas bases puede tener costos significantes en un sistema que confía en la información almacenada para poder funcionar y guiar el negocio. En un sistema libre de errores y con información perfectamente limpia, si se quiere generar una vista de los datos, la tarea consiste en relacionar dos o más tablas teniendo en cuenta sus campos clave.

Es fundamental destacar que la calidad de los datos no se supervisa de manera rigurosa y que estos tampoco se definen de manera consistente. Por lo tanto, la calidad de los mismos suele verse comprometida por muchos factores, entre ellos, errores de entrada de datos (por ejemplo, Samsng en lugar de Samsung), falta de restricciones de integridad (por ejemplo, permitir entradas como EdadEmpleado = 457) y múltiples convenciones para registrar información (por ejemplo, 'Calle 50 e/ 27 y 28' y 'Calle 50 1447'). Para empeorar las cosas, en bases de datos administradas independientemente, no solo los valores, sino también la estructura y la semántica sobre los datos también pueden diferir [11].

Esta consideración es de suma importancia, ya que la calidad de los datos comprometida desde un principio puede conducir a mayores imprecisiones a la hora de extraer y analizar los objetos.

2.3. Integración de datos

Al momento de extraer e integrar datos de diferentes fuentes para implementar un almacén de datos, es posible que surjan diferencias o conflictos sistemáticos. Estos problemas caen dentro de un término general denominado heterogeneidad de datos. Distinguimos entre dos tipos de heterogeneidad de datos: estructural y léxica [11].

La heterogeneidad estructural ocurre cuando los campos de las tuplas están estructurados de manera diferente en distintas bases de datos. Por ejemplo, en una base de datos, la dirección del cliente puede registrarse en un campo llamado dirección, mientras que, en otra base de datos, la misma información puede almacenarse en múltiples campos como calle, ciudad, estado y código postal.

Por otro lado, la heterogeneidad léxica se presenta cuando las tuplas tienen campos idénticamente estructurados en las bases de datos, pero los datos utilizan diferentes representaciones para referirse al mismo objeto del mundo real. Un ejemplo común es almacenar una dirección de diferentes maneras o utilizar diversos formatos para las fechas.

En línea con los objetivos propuestos en este trabajo, uno de los principales desafíos radica en identificar registros recolectados de diversas fuentes de datos que se refieran a la misma entidad en el mundo real, abordando así la cuestión de la heterogeneidad. Para intentar solventar esto y que la integración de datos pueda realizarse correctamente se vuelve esencial contar con datos de calidad. En este sentido, resulta indispensable llevar a cabo un proceso de limpieza de datos, conectando de manera directa la calidad de los datos con la resolución de los problemas de heterogeneidad recién mencionados.

2.4. Limpieza de datos y detección de duplicados

Una vez recolectada la información de interés, es necesario comenzar con una transformación y estandarización de los datos con el objetivo de mejorar la calidad de los mismos para que sean comparables y más legibles. La transformación de datos refiere a conversiones simples que se pueden aplicar a los mismos para que se ajusten a su tipo correspondiente, mientras que la estandarización refiere al proceso de transformar la información representada en ciertos campos a un formato específico. Este proceso es crucial cuando la información está representada de diversas maneras en diferentes fuentes de datos y, luego de su almacenamiento, se requiere su conversión a una representación uniforme. La estandarización de datos es un proceso que nos conduce de manera más rápida a la identificación de duplicados.

2.4.1. Identificando duplicados

Generalmente, el proceso de detección de duplicados está precedido por una etapa de preparación de datos, en la que los mismos son almacenados

de manera uniforme en una base de datos, resolviendo el problema de la heterogeneidad estructural [11].

Identificar objetos duplicados a través de registros provenientes de diferentes páginas web es uno de los pasos clave en la integración de datos [10]. Los objetos que refieren a la misma entidad en la vida real se encuentran en diferentes fuentes, y cada una de las fuentes publica los datos de acuerdo a su propio esquema. Dichos esquemas son desconocidos por el usuario y no nos garantizan que todos los objetos dentro del mismo esquema estén publicados acorde a este. Esto genera que los mismos atributos puedan tener diferentes nombres en diferentes fuentes, que algunos atributos sean omitidos en algunas fuentes debido a que son considerados irrelevantes, o bien que varíen los valores de ciertos de ellos.

Hay que reconocer que las páginas web no siguen las mismas restricciones que una base de datos relacional, por lo que podemos tener un mismo registro repetido dos o mas veces en el mismo sitio, o el valor de un atributo puede no corresponderse con su tipo. También es importante tener en cuenta que los registros que obtenemos son resultado de un proceso de extracción, y dicho proceso no resulta perfecto. Esto provoca que los registros contengan atributos erróneos o parcialmente válidos, por lo que no podemos confiar totalmente en los valores de los atributos para los siguientes procesos que se quisieran llevar a cabo.

La mayoría de los enfoques no tienen en cuenta estos aspectos [12] [13] y confían en el hecho de que las herramientas de extracción son perfectas y que los valores pueden ser considerados como verdaderos o confiables.

Capítulo 3

Estrategia general

Introducción

Como se mencionó en los capítulos previos, la información que necesitamos para cumplir con las actividades diarias se encuentra dispersa en la web y los contenidos dentro de ella aumentan a gran velocidad. Como consecuencia, surge la necesidad de recopilar datos de interés sobre diferentes sitios con el objetivo de estructurarlos para luego poder utilizarlos en sistemas de recomendación o toma de decisiones. Un ejemplo de esta dispersión de datos se da al momento de querer comparar productos electrónicos -tales como teléfonos celulares- para poder efectuar una compra, teniendo que recorrer una variedad de sitios para comparar características o precios.

A través de la herramienta WOA[2] el usuario tiene la posibilidad de definir plantillas en diferentes sitios web para recolectar objetos de información y que estos residan en un entorno común. De este modo la información que antes se encontraba dispersa en la web ahora se encuentra estructurada para poder analizarla y tomar decisiones. Sin embargo estas tareas no pueden llevarse a cabo sin antes realizar una limpieza y corrección de datos para asegurarnos una mejora en la calidad de los mismos. Esto es necesario debido a que dentro del marco de recolección de objetos ocasionalmente surgen problemas que afectan la calidad de los datos. Estos inconvenientes pueden derivar de errores tanto sintácticos como semánticos, campos con información redundante o incorrecta, así como la existencia de campos vacíos.

A partir de la recolección y estructuración de los datos, y con motivo de limpiar y corregir la información de dichos objetos para asegurar una mejora en la calidad de los mismos, se decidió desarrollar un 'Framework de normalización de objetos'. A través de este framework se busca generar tanto una transformación como estandarización de los datos con el fin de obtener una

fuente de datos prolija para cumplir con las tareas del siguiente proceso, cuyo objetivo es la identificación de duplicados. Para cumplir con esta última tarea se desarrollo un 'Framework de tratamiento de duplicados' con el propósito de encontrar aquellos elementos que se correspondan al mismo objeto en la vida real.

Como recién se mencionó, hay dos tareas que llevar a cabo: la normalización de datos en primer instancia y luego la identificación de duplicados. Cada una de estas tareas debe ser configurable, por lo que se decidió basarse en una aplicación de dos pasos encadenada, en la que se ofrecerá un framework para cada uno de estos pasos.

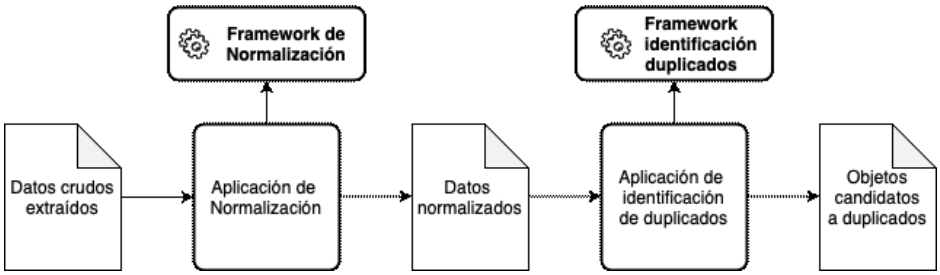


Figura 3.1: Flujo de datos

Como se observa en la figura, en primer término los datos crudos extraídos son procesados a través de una aplicación construida a partir del framework de normalización con el objetivo de obtener dichos datos normalizados. Luego de esta etapa los datos se procesan a través de otra aplicación construida a partir del framework de identificación de duplicados, con el objetivo de obtener aquellos objetos que se corresponden al mismo en la vida real.

Cuando hablamos de Framework nos referimos a una aplicación semi completa y reutilizable, que otorga grandes ventajas con el fin de cumplir con los objetivos de desarrollo propuestos. En primera instancia se destaca la reusabilidad a través de componentes genéricos, los cuales pueden ser utilizados una y otra vez para crear aplicaciones nuevas. En segunda instancia, y no menos importante, se destaca la capacidad de extensión del framework por parte del desarrollador, permitiéndole a este añadir nuevas funcionalidades a través de métodos gancho específicos. Estos métodos gancho o 'Hot Spots' son las partes del framework en donde ocurre la adaptación, y es donde el desarrollador usando el framework agrega su propio código con una funcionalidad específica. En contraste con los hotspots, los 'Frozen Spots' son aquellas partes de código que no cambian, es decir aquella parte que llama a los hotspots. Aquí entonces contaremos con dos frameworks los cuales serán reutilizables y nos proveerán extensibilidad cada uno a través de sus hotspots

o métodos gancho.

En cuanto a la clasificación de los frameworks podemos mencionar aquellos de 'caja blanca' y 'caja negra'. Un framework de caja blanca requiere que el desarrollador conozca los detalles de la estructura interna del mismo y cómo esta diseñado, proveyendo extensibilidad a través de la herencia de clases base y redefiniendo métodos gancho. Tal es el caso para el actual trabajo, en el cual necesitamos conocer la estructura y funcionamiento del mismo para poder adaptarlo a las necesidades propias. En cambio, un framework de caja negra no requiere conocimiento interno por parte del desarrollador, el cual podrá extender el framework a través de la definición e integración de componentes que cumplan con una interfaz determinada.

A continuación se presentan los frameworks desarrollados junto a sus características y propósitos generales.

3.1. Framework de normalización de objetos

Con el objetivo de lograr una mejora en la calidad de los datos almacenados es necesario procesar los objetos de información previamente recolectados realizando tareas de limpieza a través del "Framework de normalización de objetos".

Como se mencionó previamente el presente framework entra en la categorización de 'caja blanca', ya que el programador debe conocer su estructura interna para proveer extensibilidad y definir nuevas estrategias de normalización en caso de que las predefinidas no sean las adecuadas para los campos que se buscan normalizar.

Mediante el actual framework se otorga al desarrollador la posibilidad de normalizar tanto un único objeto como una colección de los mismos. Dicha tarea de normalización se realizará sobre cada uno de los objetos a través de estrategias específicas que se aplicaran sobre cada uno de sus campos. Para cumplir con este objetivo, previamente será necesario definir una configuración en la cual se indicará qué estrategia de normalización aplicar por cada campo de los objetos a normalizar. Abordaremos este tema con mayor profundidad en el próximo capítulo.

Luego de recolectar los objetos de información, el próximo paso conlleva la identificación de aquellos campos a normalizar, junto a la estrategia específica que se aplicará sobre cada uno de ellos. El framework proporciona una estructura que posibilita la normalización de cada uno de los objetos y, como se menciono anteriormente, la definición de nuevos normalizadores. De este modo, luego de identificar los campos a normalizar, se deben pensar y definir las estrategias específicas de normalización para cada campo. Poste-

riormente, se procede a definir una configuración en la que se especificará qué estrategia aplicar sobre cada uno. A modo de ejemplo, para el campo 'precio' se asignará la estrategia de normalización 'MonetaryAmountNormalizer'. Esta estrategia realiza una transformación en el formato del valor numérico del campo con el objetivo de lograr una estructura que abarque signo, puntuación y decimal, garantizando uniformidad para todos los objetos. De igual manera, la estrategia 'MemoryNormalizer' se asignará al campo 'memoria' con el objetivo de estructurar el campo de igual manera para todos los objetos, añadiendo la sigla 'GB' luego del valor correspondiente [1].

Este proceso de normalización busca que todos los objetos extraídos mantengan una estructura y un formato uniforme en todos sus campos con el objetivo aumentar la legibilidad y la calidad de los datos.

Luego de que los objetos son procesados a través del presente framework se obtiene una fuente de datos limpia y prolija lista para la segunda etapa de desarrollo correspondiente a la búsqueda de potenciales duplicados.

3.2. Framework de tratamiento de duplicados

Para identificar y resolver duplicados, se diseñó un 'Framework de tratamiento de duplicados' con el objetivo de encontrar dos o mas objetos que se correspondan al mismo objeto en la vida real.

A partir de este framework es posible, mediante la selección de un objeto y una fuente de datos, obtener candidatos a duplicados para el objeto seleccionado. Luego del proceso de normalización de objetos disponemos de diferentes alternativas a la hora de seleccionar el candidato a comparar. Una de ellas comienza con la recolección un objeto y su posterior procesamiento a través del framework de normalización, para luego identificar si posee candidatos a duplicados a través del presente framework. Otra alternativa para obtener el objeto referencia es mediante su selección sobre todos aquellos que fueron previamente recolectados y normalizados, es decir sobre la fuente de datos disponible. Este objeto sera comparado con todos sus pares bajo un plan estratégico en particular de manera de obtener un valor de similitud entre 0 y 1 para cada uno de los objetos con los que se compare.

[1] Para llevar a cabo la normalización de algunos campos se utilizaron expresiones regulares con el objetivo de encontrar información que coincida con determinados patrones. De esta manera es posible filtrar u obtener aquellos datos que aporten valor, logrando mejoras en la calidad de la información al quedarse con aquella que es de interés y dejando de lado la que no es relevante.

La comparación de objetos se realizará en primer instancia campo por campo, seleccionando para cada uno de ellos una estrategia de comparación, obteniendo un valor entre 0 y 1 para cada uno, para luego a partir de dichos valores emplear una nueva estrategia con el fin de obtener el resultado final de similitud del objeto entero. Al momento de seleccionar la estrategia a emplear, el desarrollador será responsable de definir una configuración en la que indicara qué estrategia de comparación utilizará sobre cada uno de los campos. Luego de realizada esta tarea resta definir aquella estrategia que se empleara para calcular el valor de similitud del objeto en base a los resultados de comparación de cada campo.

Al igual que el framework de normalización de objetos se trata de un framework de caja blanca, es decir, es necesario conocer su estructura interna con el objetivo de definir nuevas estrategias de comparación además de las predefinidas, proveyendo extensibilidad a través de métodos gancho. De esta manera es posible contar con diferentes técnicas de comparación a la hora de calcular la similitud entre dos campos u objetos, seleccionando la mas adecuada según corresponda.

Al finalizar el proceso de comparación se obtiene un archivo el cual contiene el objeto candidato que se seleccionó para comparar junto a todos aquellos objetos que hayan sido considerados candidatos a duplicados, cada uno con su valor de similitud.

En el próximo capitulo se discuten detalles del 'Framework de normalización de objetos'. Para ilustrar su uso en el presente trabajo se utiliza como caso de estudio el dominio de compra de teléfonos celulares, el cual se tratará de la siguiente manera:

Recorreremos diversas paginas web que ofrezcan teléfonos celulares y definiremos un template en cada dominio diferente para poder realizar la extracción de los datos a través de WOA. Buscaremos generar una base de datos con modelos de teléfonos que se repitan en diferentes paginas y a la vez otros que sean lo mas similar posible, para poder luego normalizar los datos, analizar los resultados y realizar comparaciones.

Capítulo 4

Framework de normalización de objetos

Introducción

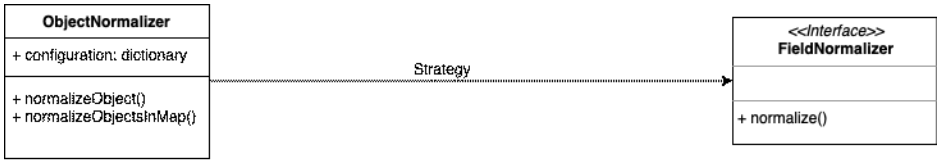
Con el objetivo de normalizar y limpiar los objetos previamente recolectados mediante la herramienta WOA, se propuso la construcción de un framework que cumpla con dichas tareas para poder contar con una fuente de datos prolija y confiable.

Para comenzar hemos de mencionar algunas de las principales características de este framework:

- Se trata de un framework de **caja blanca**, ya que es necesario conocer su estructura interna y el funcionamiento de las clases a extender.
- Es **reutilizable**, debido a que es posible construir diferentes aplicaciones a partir de él.
- Es **extensible**, ya que se puede ampliar su funcionalidad mediante nuevas funciones y componentes.

Mediante el presente framework el desarrollador será capaz de normalizar cada uno de los objetos recolectados a través un proceso de transformación de datos. A continuación se detalla la estructura y el funcionamiento del framework, y la manera en que cada uno de los objetos es normalizado para poder cumplir con la transformación de datos.

4.1. Diseño general



El presente framework actúa como una especie de plantilla a partir de la cual el desarrollador deberá completar los detalles específicos de su aplicación o problemática, es decir, provee las bases o la estructura sobre la cual sera posible construir nuevas aplicaciones.

El diseño del framework se encuentra basado en el patrón 'Strategy', el cual es un patrón de comportamiento que gestiona la relación y la responsabilidad entre objetos, delegando tareas en los colaboradores correspondientes. Esto permite la selección del algoritmo que ejecuta determinada acción en tiempo de ejecución, y a su vez brinda mayor flexibilidad en tiempo de programación y ejecución.

Como se observa en el diagrama contamos con la clase 'ObjectNormalizer' la cual será encargada de gestionar la normalización de objetos a través de la invocación del método correspondiente, otorgando la posibilidad de normalizar un único objeto o bien una colección de ellos. Normalizar un objeto implica normalizar cada uno de sus campos, por lo que será necesario contar con una estrategia de normalización especifica para cada uno de ellos. Seleccionar aquella estrategia de normalización que se aplicará sobre cada campo es tarea del desarrollador, por lo que en primer termino debe chequear si la estrategia que quiere utilizar está definida. En caso de que no lo esté, o aquellas estrategias disponibles no sean las adecuadas para el campo que se quiere normalizar, debe construir una nueva estrategia. Para poder realizar esta última tarea se presenta la interfaz 'FieldNormalizer', la cual cuenta con un método gancho o 'Hook method' que deberá implementarse en cada nueva estrategia. Esta interfaz actuará como colaboradora de la clase 'ObjectNormalizer', ya que a través de esta última se le delega la tarea de normalizar cada campo según la estrategia seleccionada. La implementación de cada estrategia debe realizarse mediante la reimplementación del método gancho 'Normalize()' en cada una de las nuevas clases.

Tanto la clase 'ObjectNormalizer' como la interfaz 'FieldNormalizer' forman la estructura principal del framework, siendo la base sobre la cual sera posible construir nuevas aplicaciones.

La clase 'ObjectNormalizer' representa la parte congelada del framework o el 'FrozenSpot', ya que forma parte de aquellos elementos que permanecen

inmutables en su estructura. Cuenta con el método 'NormalizeObjectsInMap()' el cual recibirá el archivo que contenga la colección de objetos a normalizar. Su principal tarea será recorrer el archivo recibido con el objetivo de normalizar cada uno de sus objetos. Para normalizar de cada uno de ellos se utilizará el método 'NormalizeObject()', el cual se ha de invocar en caso de querer normalizar un único objeto en lugar de una colección. Este último método es responsable de iterar sobre cada campo del objeto recibido, y por cada uno de ellos invocar la estrategia de normalización que corresponda. Para que a través del método 'NormalizeObject()' sea posible invocar la estrategia de normalización específica según el campo que se este normalizando es imprescindible definir una configuración en la que se deberá indicar, a través de un diccionario, el nombre del campo junto a la estrategia que se aplicará sobre cada uno de ellos. De esta manera, al momento de instanciar la clase 'ObjectNormalizer' para crear un nuevo normalizador, será necesario enviar la configuración predefinida con el fin de que esta clase pueda utilizarla para invocar a los normalizadores de campo.

La interfaz 'FieldNormalizer', en cambio, representa aquella parte variable del framework que puede extenderse por los usuarios del mismo, denominada 'HotSpot'. Este hotspot representa el punto en el que variaran las diferentes aplicaciones que se creen según el dominio del problema. A través de esta interfaz el usuario es capaz de extender el framework mediante la definición de nuevas estrategias de normalización. Según el dominio de objetos a normalizar será necesario aplicar una estrategia en particular sobre cada campo. En caso de que la estrategia a aplicar no se encuentre definida o las existentes no sean de utilizad, el usuario deberá crear la propia. Para ello, la interfaz cuenta con el método 'Normalize()', el cual deberá ser reimplementado en cada nueva estrategia, ya que es quien finalmente contendrá la lógica para normalizar el campo específico.

4.2. Utilización del framework

En la etapa actual y a través del presente framework el objetivo principal es, a partir de un archivo con objetos json y datos crudos, retornar tales objetos normalizados. Normalizar un objeto implica normalizar cada uno de sus campos, por lo tanto crear un normalizador que cumpla con el objetivo de normalizar cada objeto implica, por un lado, identificar los campos que se quieren normalizar y reconocer qué estrategia de normalización aplicar sobre cada uno; y luego construir un normalizador compuesto, es decir, que contenga cada uno de los nombres de los campos del objeto junto al normalizador a aplicar sobre cada uno.

Como se mencionó, el primer paso a la hora de utilizar el framework es identificar los campos a normalizar y elegir la estrategia de normalización para cada uno de ellos, por lo tanto, es necesario comprobar si existe alguna subclase concreta de las que ofrece el framework que cumpla con la tarea que se pretende. En caso de que ninguna de las subclases sea de utilidad ya que los campos no se corresponden, sera necesario construir una nueva. A continuación se detalla cómo es posible realizar esta tarea a través de la interfaz 'FieldNormalizer'.

4.2.1. Creando un nuevo normalizador

Como se manifestó anteriormente, para poder extender el framework y definir una nueva clase normalizadora contamos con la ayuda de la interfaz 'FieldNormalizer'.

```
const debugField = require("../../lib/debuggers").field;
const debugInfo = require("../../lib/debuggers").info;


module.exports = class FieldNormalizer {
  constructor() {
    //if setted, console.logs fields properties
    //export DEBUG=field,info
    this.debugField = debugField;
    this.debugInfo = debugInfo;
  }

  //it must be implemented by the subclasses
  normalize(anObject, attribute) {
    throw new TypeError("Must override method");
  }

  //it receives a text and an array with those words intented to delete in it.
  //it returns the text modified.
  deleteIfExists(aValue, wordsToDelete){
    if(!aValue){
      return "";
    }
    let result = aValue;
      for (var k in wordsToDelete) {
        if (result.toUpperCase().includes(wordsToDelete[k].toUpperCase())) {
          let regex = new RegExp(wordsToDelete[k], "gi");
          result = result.replace(regex, "");
        }
      }
        //remove extra whitespaces
      result = result.replace(/^\s+|\s+$/g, "").replace(/\s+/g, " ");
      return result;

  }

};
```

Figura 4.1: Interface 'FieldNormalizer'

Podemos ver que la interface cuenta con el método ***normalize****(anObject,*

attribute) el cual debe ser implementado por las clases concretas que definamos y que extenderán de dicha interface. Dentro de este método cada clase implementara su lógica en particular dependiendo la estrategia que se haya elegido.

A su vez, cuenta con el método ***deleteIfExists***, el cual cumple la función de eliminar, dentro de un texto, aquellas palabras que se reciban como parámetro. Dicho método se encuentra implementado dentro de esta interface ya que eliminar palabras de un texto puede ser una tarea recurrente que se realice en diferentes estrategias, de esta manera, cualquier nueva estrategia que se defina podría hacer uso de este método.

A continuación se presentan los pasos a seguir utilizando la interfaz con el fin de crear un nuevo normalizador de campo, tomando como ejemplo la creación de una clase concreta denominada 'CleanNormalizer', la cual formara parte de la estructura principal del framework y podrá ser reutilizada por las diferentes aplicaciones que se creen a través del mismo. La tarea de esta clase concreta consistirá, en linea con el método presentado recientemente *deleteIfExists*, en invocar al mismo con el objetivo de eliminar sobre el campo al cual se aplique esta estrategia aquellas palabras que se reciban como parámetro.

En primera instancia es necesario crear la clase concreta que extienda a la interfaz para poder implementar el 'método gancho' que contendrá la lógica correspondiente. En este caso la clase 'CleanNormalizer' sera quien extienda a la interfaz 'FieldNormalizer':

```
const FieldNormalizer = require("./FieldNormalizer");

module.exports = class CleanNormalizer extends FieldNormalizer {
  constructor(params) {
    super();
    this.params = params;

  }

  normalize(anObject, attribute) {
    let name = anObject[attribute];
    name = this.deleteIfExists(name, this.params);
    anObject[attribute] = name;
  }


};
```

Figura 4.2: Clase 'CleanNormalizer'

Esta clase cuenta con un método constructor, el cual debe recibir a través de un parámetro aquellas palabras que el usuario quiera eliminar indicadas

a través de un arreglo. La tarea, o la estrategia principal en esta clase, será comprobar si alguna de las palabras recibidas forman parte del campo sobre el que se aplique esta estrategia, con el objetivo de eliminarlas. La lógica de normalización de las clases que se creen deberá implementarse dentro del método 'normalize()', ya que es el 'hotspot' o la parte donde ocurrirá la adaptación de acuerdo a la estrategia a implementar. En este caso solo se invoca al método definido en la interface ya que utilizaremos la clase solo y únicamente con ese propósito.

Como se puede observar, a través del método 'Normalize()' se recibe tanto el objeto como el campo a normalizar. El objetivo entonces es, eliminar para el campo del objeto recibido, aquellas palabras que se indicaron por parámetro al momento de crear la clase normalizadora, para luego retornar el nuevo valor normalizado.

Luego de ejecutado el método y obtenido el valor normalizado, el siguiente paso es actualizar el campo del objeto con dicho valor.

De este modo construimos una nueva estrategia de normalización o nuevo normalizador de campo, el cual estará disponible para poder utilizar sobre el campo que el usuario seleccione. Para poder invocar esta estrategia se deberá crear un arreglo con aquellas palabras que se deseen eliminar sobre el campo en cuestión, y enviar dicho arreglo como parámetro al momento de instanciar la clase. Esta estrategia de normalización puede ser reutilizada sobre diferentes campos creando un nuevo 'CleanNormalizer' y seleccionando para cada uno de ellos que palabras se desean remover.

Así como se definió una nueva clase concreta de normalización, el usuario deberá definir las que crea necesarias. De este modo y luego de creado un nuevo normalizador en concreto la estructura principal del framework se vería de la siguiente manera:

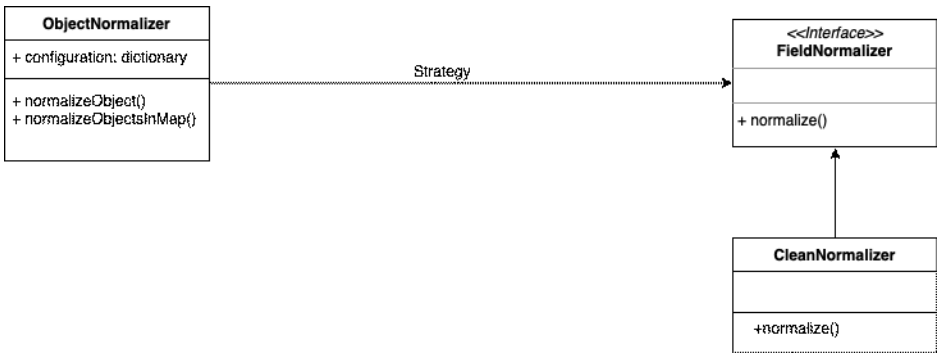


Figura 4.3: Estructura principal

Si bien el actual representa un framework de caja blanca, ya que es nece-

sario conocer su estructura interna para poder subclasificar la interfaz 'Field-Normalizer', muchos de los normalizadores de campo que el usuario defina podrán ser reutilizables. Por lo tanto, con el paso del tiempo, es posible disponer de un amplio catalogo de normalizadores de manera tal que para la mayoría de los dominios no será necesario crear nuevas subclases, transformándose el framework de caja blanca en un framework de caja negra.

Como se mencionó, normalizar un objeto implica normalizar cada uno de sus campos, por lo que primero se ha de identificar aquellos campos a normalizar y la estrategia a aplicar sobre cada uno de ellos. Recientemente vimos como crear un nuevo normalizador en caso de que los disponibles no se correspondan con la estrategia a emplear, o simplemente no existan. Por lo tanto, luego de identificar los campos a normalizar y contar con todas las estrategias implementadas, es necesario configurar la aplicación normalizadora de manera de seleccionar qué estrategia de normalización se aplicará sobre cada uno de los campos de los objetos recolectados.

4.2.2. Configurando la aplicación normalizadora

Para poder normalizar los objetos recolectados a través del presente framework es necesario configurar la aplicación que sera responsable de ejecutar dicha tarea. Luego de haber recolectado todos los objetos de interés es necesario identificar aquellos campos que se quieran normalizar, y para cada uno de ellos tener definida una estrategia de normalización. Anteriormente quedo demostrado de qué manera el desarrollador puede crear un nuevo normalizador de campo, por lo tanto, luego de tener definidos aquellos que se van a utilizar, solo resta configurar la aplicación normalizadora.

Siendo el dominio actual 'teléfonos celulares' se optó por definir diferentes normalizadores según la estrategia a aplicar sobre cada campo. En primer instancia se presenta la estructura de los objetos recolectados, es decir los campos en los que se almacenara la información, y luego se presentan las estrategias para normalizar cada uno de ellos.

Luego de recorrer diferentes sitios de venta de celulares y analizar aquella información que puede resultar de importancia a la hora de comparar y comprar un teléfono, se decidió formar una estructura de objeto que permita almacenar la siguiente información:

- Nombre
- Precio
- Marca

- Sistema Operativo
- Cámara principal
- Memoria de almacenamiento
- Batería
- Procesador
- Velocidad de procesador
- Tamaño de pantalla

Por lo tanto a la hora de normalizar cada uno de los objetos es necesario pensar en una estrategia de normalización para cada una de las propiedades recién mencionadas. Dichas estrategias se presentan a continuación:

Para el caso del **nombre** y con el objetivo de mantener la información lo mas completa posible, se optó por mantener el dato original eliminando solo aquellas palabras que el usuario considere innecesarias o redundantes, como por ejemplo la palabra 'Celular'. Estas serán definidas a través de un arreglo y enviadas al método de normalización correspondiente, el cuál se encargara de eliminarlas. A su vez es posible chequear por caracteres especiales extras no tengan significado.

El normalizador correspondiente al **precio** se encargara de transformar el valor a un formato uniforme para todos los objetos, teniendo en cuenta el signo $ y la puntuacion o decimales.

Para la **marca** se comprobara si alguno de los campos del objeto contiene alguna de las marcas que el usuario haya predefinido en un arreglo, de manera de mantener solo ese valor.

Para el **sistema operativo** se empleara una estrategia similar a la anterior, en donde el normalizador se encargara de comprobar si el campo contiene alguno de los sistemas operativos definidos en un arreglo.

En cuanto a la **cámara** y a la **memoria** los métodos correspondientes se encargaran de obtener el valor numérico, añadiendo la sigla MP o GB respectivamente.

En el caso de la **batería** su valor sera representado a través de la unidad 'mAh' (miliamperios hora), obteniendo el valor numérico y realizando las transformaciones necesarias.

Para el **procesador** la estrategia consiste en obtener la cantidad de núcleos o 'cores' que representen su valor. En cuanto a la **velocidad** del mismo se transformara el valor numérico con el objetivo de representarlo en 'GHz'.

Luego de haber identificado los campos a normalizar e implementado la estrategia para cada uno de ellos, el próximo paso sera configurar la aplicación normalizadora. Definiendo una configuración es posible que la aplicación conozca que estrategia de normalización debe aplicar sobre cada campo. Para ello se debe definir una variable en la que se indicaran, a través de pares de datos clave-valor, los nombres de los campos junto a la estrategia que se aplicara sobre cada uno de ellos.

Al momento de definir la configuración es necesario importar aquellas estrategias que fueron implementadas previamente de manera que estén disponibles para seleccionar la adecuada. Con el objetivo de facilitar esta tarea y hacer más legible el código se decidió definir una clase denominada 'FieldNormalizerFactory', de manera que funcione de repositorio con todas aquellas estrategias de normalización disponibles. Esto facilita la tarea al momento de definir la configuración y seleccionar las estrategias, ya que a través de esta clase es posible acceder a todas y cada una de ellas.

A continuación se presenta la clase 'FieldNormalizerFactory' con todos los normalizadores de campo disponibles que forman parte de la aplicación normalizadora creada a partir del actual framework. Su función principal es exportar o hacer visible cada clase normalizadora a través de una propiedad, de manera tal que al momento de definir la configuración y seleccionar alguna de ellas se cree una instancia de su clase.

```
const MonetaryAmountNormalizer = require("./fields/MonetaryAmountNormalizer");
const MemoryNormalizer = require("./fields/MemoryNormalizer");
const BrandNormalizer = require("./fields/BrandNormalizer");
const ScreenSizeNormalizer = require("./fields/ScreenSizeNormalizer");
const MegapixelNormalizer = require("./fields/MegapixelNormalizer");
const BatteryNormalizer = require("./fields/BatteryNormalizer");
const TitleNormalizer = require("./fields/TitleNormalizer");
const OSNormalizer = require("./fields/OSNormalizer");
const ProcessorNormalizer = require("./fields/ProcessorNormalizer");
const SpeedProcessorNormalizer = require('./fields/SpeedProcessorNormalizer')
const CleanNormalizer = require('./fields/CleanNormalizer')


module.exports = {
  monetaryAmountNormalizer: new MonetaryAmountNormalizer(),
  memoryNormalizer: new MemoryNormalizer(),
  brandNormalizer: new BrandNormalizer(),
  screenSizeNormalizer: new ScreenSizeNormalizer(),
  megapixelNormalizer: new MegapixelNormalizer(),
  batteryNormalizer: new BatteryNormalizer(),
  titleNormalizer: new TitleNormalizer(),
  osNormalizer: new OSNormalizer(),
  processorNormalizer: new ProcessorNormalizer(),
  speedProcessorNormalizer: new SpeedProcessorNormalizer(),
  cleanNormalizer: params => new CleanNormalizer(params)
};
```

Figura 4.4: 'FieldNormalizerFactory' como repositorio de normalizadores

A través de esta clase se almacenan todas las estrategias de normalización disponibles, exportando cada una de ellas a través de una propiedad con el fin de que puedan ser invocadas desde otros puntos de la aplicación de manera más sencilla y legible.

Continuando con la configuración finalmente es momento de declarar la variable sobre la que se definirá la misma. Al momento de declarar esta variable es necesario indicar, a través de pares de datos clave-valor, el nombre del campo a normalizar junto a la estrategia a aplicar, como se muestra a continuación:

```
let unusefulWordsForName = [
  "Celular",
  "Liberado",
  "Libre",
  "Cuotas",
  "Sin",
  "Interes",
  "Original",
  "Nuevo",
  "Modelo",
  "\\*1\\*",
];

//configuration for each field of the JSON object to normalize
var configuration = {
  name: fieldNormalizerFactory.cleanNormalizer(unusefulWordsForName),
  price: fieldNormalizerFactory.monetaryAmountNormalizer,
  ramMemory: fieldNormalizerFactory.memoryNormalizer,
  storageMemory: fieldNormalizerFactory.memoryNormalizer,
  mainCamera: fieldNormalizerFactory.megapixelNormalizer,
  processor: fieldNormalizerFactory.processorNormalizer,
  screenSize: fieldNormalizerFactory.screenSizeNormalizer,
  battery: fieldNormalizerFactory.batteryNormalizer,
  operatingSystem: fieldNormalizerFactory.osNormalizer,
  speedProcessor: fieldNormalizerFactory.speedProcessorNormalizer
};
```

Figura 4.5: Configuración de la aplicación normalizadora

En primer instancia se definió un arreglo con aquellas palabras que se creyeron innecesarias para un campo en particular y por lo tanto deben eliminarse. En este caso, se corresponden al campo nombre, y dicho arreglo sera enviado como parámetro al momento de seleccionar la estrategia 'CleanNormalizer', ya que, al momento de su instanciación, dicha clase requiere un parámetro con aquellas palabras a eliminar. Esta clase o estrategia se aplicara sobre el campo 'name' dentro de la variable de configuración, como se puede ver al inicio de su declaración. Continuando con esta configuración se modelan los campos a normalizar junto a la estrategia a aplicar sobre cada uno de ellos.

A modo de ejemplo, para el campo denominado ‘price’ se aplicará el algoritmo de normalización ‘MonetaryAmountNormalizer’. Del mismo modo se debe seleccionar qué método de normalización se aplicará sobre cada campo. De esta manera, luego de definir la configuración de normalización de la aplicación, el próximo paso será crear una instancia de la clase ’ObjectNormalizer’ enviando la configuración recién definida.

```
//creates an ObjectNormalizer with the previous configuration
const normalizer = new ObjectNormalizer(configuration);

//reads the input file and send it to the normalizer
jsonfile
  .readFile(inputFile)
  .then(result => normalizer.normalizeObjectsInMap(result))
  .then(result => jsonfile.writeFile(outputFile, result))
  .catch(err => console.error(err));
```

Figura 4.6: Creación de una instancia de la clase ’ObjectNormalizer’ enviando la configuración predefinida

Al crear una instancia de la clase ’ObjectNormalizer’ y enviar la configuración predefinida la aplicación normalizadora queda configurada y la tarea restante es enviar el archivo con los objetos a normalizar.

Con el fin de leer el archivo con los objetos a normalizar a través de la aplicación se utilizó el modulo ’jsonfile’, el cual facilita la tarea al momento de leer o escribir archivos JSON. De este modo, luego de obtener el archivo a través de este modulo es necesario invocar al método ’NormalizeObjectInMap()’, el cual será responsable de recibir la colección de objetos a normalizar. Su tarea será recorrer el archivo recibido e invocar al método ‘NormalizeObject()’ para cada uno de los objetos. La función de este último método es iterar sobre la configuración predefinida, invocando al método de normalización que se seleccionó para cada campo, enviando la información asociada al mismo. Luego de haber pasado por el proceso específico de normalización, cada campo del objeto será actualizado.

Finalmente, luego de que cada objeto haya pasado por este proceso de transformación de datos, la información de cada uno de ellos será más clara y consistente, generando datos de mayor calidad los cuáles serán utilizados en la etapa de identificación de duplicados.

Capítulo 5

Framework de tratamiento de duplicados

Introducción

Luego de recolectar los objetos de información y procesar los mismos a través del framework de normalización, estos quedan listos para la actual etapa de identificación de duplicados. El problema de detectar entidades o registros duplicados ha sido estudiado bajo diferentes nombres como 'Entity Matching' (Correspondencia de entidades) o 'Record Linkage' (Enlace de registros), entre otros. Todos ellos tienen el mismo objetivo, dado un conjunto de entidades, identificar el subconjunto que contenga aquellas que se correspondan al mismo objeto del mundo real.

Al momento de realizar la detección de duplicados la mayoría de los enfoques realizan comparaciones sobre objetos que provienen de diferentes fuentes, y por lo general su estructura de datos no es la misma. Por ejemplo, la base de datos de un sitio podría almacenar el nombre y el apellido de una persona en diferentes campos ('nombre': "Juan Carlos", 'apellido': "Medina"), mientras que otra base de datos de otro sitio podría almacenar solo las iniciales de la persona junto a su apellido en un solo campo ('nombre':"J. C. Medina"). Por lo tanto surgen diferencias al momento de recolectar la información.

En el presente trabajo, y al igual que en la mayoría de los enfoques,también se comparan objetos de diferentes fuentes, a diferencia que aquí todos ellos cuentan con una misma estructura de datos, ya que previamente utilizamos WOA como herramienta de recolección de objetos. Recordemos que WOA permite al usuario definir una estructura de datos antes de recolectar los objetos. Esto permite contar con la información estructurada desde un primer

momento, lo que luego nos facilitará las tareas al momento de realizar las operaciones o las comparaciones necesarias al utilizar el framework.

A través del presente framework es posible realizar comparaciones entre los objetos recolectados con el objetivo de identificar potenciales candidatos a duplicados, es decir, aquellos que se correspondan al mismo objeto en la vida real. Esto lo haremos seleccionando un objeto como candidato para que luego sea comparado con sus pares. Estas comparaciones se pueden realizar a través de diferentes algoritmos de similitud o comparación. Aquí en particular emplearemos algoritmos de similitud de cadenas de texto de manera de obtener un valor entre 0 y 1 para cada campo comparado.

Mencionamos que cada campo será comparado a través de una estrategia en particular. Será el usuario programador quien deba seleccionar aquella estrategia o método de comparación a aplicar sobre cada uno de los diferentes campos, a través de la definición de una configuración previa. Una vez realizada la comparación obtendremos un valor de similitud entre 0 y 1 para cada par de campos comparados. Luego de obtenidos estos valores de similitud para cada campo, se deberá obtener el valor de similitud final del objeto entero. Diferentes estrategias pueden aplicarse con el fin de obtener dicho valor de similitud. Al igual que en los casos anteriores, si la idea o la estrategia no se encuentra implementada, deberá llevarse a cabo. Finalizado este proceso se obtendrá un archivo con el objeto referencia -el cual fue seleccionado para comparar- y aquellos que hayan sido considerados candidatos a duplicados junto a su valor final de similitud.

Como se mencionó en los capítulos previos, el proceso general de la actual tesina consta de una aplicación de dos pasos encadenada, por lo que cada uno de los frameworks que se ofrecen deben brindar la posibilidad de construir aplicaciones que sean configurables, siendo esta una de las principales características de ambos frameworks. A través del actual framework el objetivo es otorgar una herramienta para facilitar al usuario la identificación de objetos duplicados mediante la gestión de diferentes estrategias de comparación, las cuales serán aplicadas sobre cada uno de los campos al momento de construir la aplicación.

Al igual que el framework de normalización de objetos, el actual cuenta con las siguientes características:

- Es un framework de **caja blanca** debido a que es imprescindible conocer su estructura interna y el funcionamiento de las clases para poder extender su funcionalidad.

- Es **reutilizable** ya que a través de él se pueden construir nuevas aplicaciones.

- Es **extensible**, ya que se puede extender su funcionalidad a través del desarrollo de nuevas funciones o componentes.

Esto brinda la posibilidad de adaptar el framework de acuerdo a las necesidades particulares del usuario en base a las comparaciones que deseen realizar, a través de la definición de diferentes estrategias.

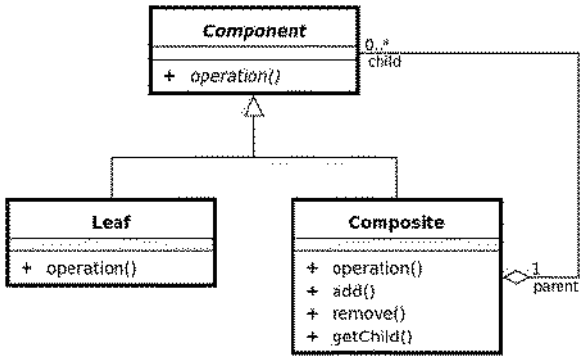
A continuación se presenta el diseño de la estructura del framework y el funcionamiento del mismo.

5.1. Diseño general

Con el objetivo de reconocer si dos objetos son considerados candidatos a duplicados es necesario comparar sus campos a través de diferentes estrategias, de manera de obtener un valor de similitud para cada uno de ellos y a partir de dichos valores emplear una nueva estrategia para obtener el valor final de similitud entre dos objetos.

Para ello se construyó un framework el cual se encuentra basado en uno de los patrones de diseño estructurales denominado 'Composite'. Este patrón compone objetos en una estructura de árbol para representar la jerarquía de las partes y el todo, de modo que los objetos que componen el árbol pueden tratarse de manera individual -a través de la clase Leaf- o como una composición de objetos -mediante la clase Composite-. La superclase de estas se la llama 'Component', y su interface se utiliza para interactuar con ambos objetos de la estructura del patrón Composite sin la necesidad de realizar una diferenciación entre objetos simples y compuestos. Si el receptor es una hoja el requerimiento es manejado directamente, y si es un objeto compuesto, el Composite delega el requerimiento a su componente hijo, el cual sera una hoja o un nuevo Composite.

La estructura del patrón es la siguiente:

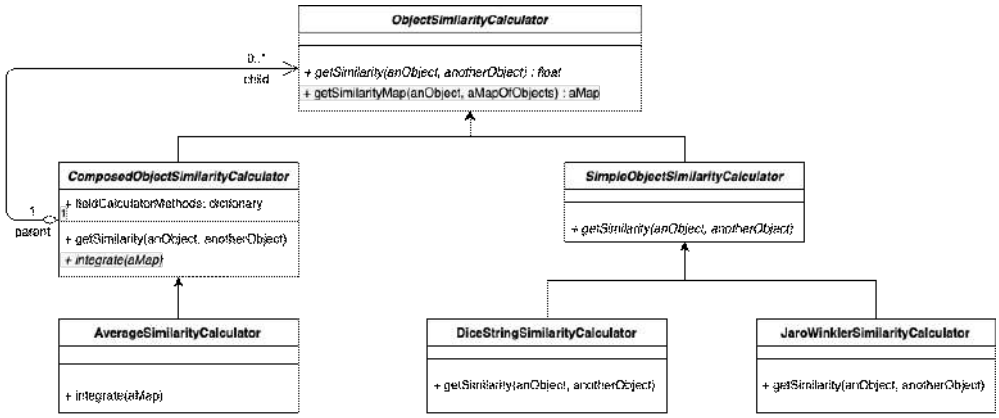


Donde:

- Component es la clase abstracta o interfaz que deben extender o implementar tanto los objetos simples como las agrupaciones de los mismos.
- Leaf representa un objeto simple u objeto "hoja" (no poseen hijos). Define comportamientos para objetos primitivos.
- Composite representa un componente compuesto o agrupación, es decir, que está a su vez compuesto de otros componentes ya sean objetos hoja o nuevos Composite.

Los clientes usan la interfaz de Component para interactuar con los objetos en la estructura del Composite. Si el receptor es una hoja la interacción es directa, mientras que si es un Composite se debe llegar a los objetos "hijos", lo que puede llevar a utilizar operaciones adicionales.

Conociendo el diseño de este patrón es posible realizar un paralelismo con el diseño del actual framework:



La interfaz Component del patrón esta representada por la clase 'ObjectSimilarityCalculator'. A través de ella se definen dos operaciones. Por un lado el método 'getSimilarityMap' el cual recibe un objeto junto a un mapa de ellos, y contendrá la lógica para poder comparar el objeto candidato contra cada uno del mapa. Y por otro lado la operación 'getSimilarity()' deberá ser implementada por las clases que hereden de ella, en este caso tanto por la clase hoja como por la clase compuesta.

Para representar las clases 'Leaf' u hojas sera necesaria la ayuda de una clase intermedia denominada 'SimpleObjectSimilarityCalculator', la cual funcionara como un 'HotSpot' a través del cual se podrá extender el framework por las nuevas clases que funcionaran como hojas. Cada una de estas

clase hoja representará una estrategia de comparación de campos diferente que se deberá implementar a través del método getSimilarity(). La clase 'DiceStringSimilarityCalculator', por ejemplo, se utilizará para realizar la comparación entre 2 cadenas de texto, mientras que la clase 'JaroWinkler' realizara la misma tarea utilizando un algoritmo diferente.

La representación de la clase Composite sera a través de la clase 'ComposedObjectSimilarityCalculator', la cual también deberá reimplementar el método 'getSimilarity'. A través de este último se reciben dos objetos, cada uno compuesto por sus campos correspondientes, y se itera sobre cada uno de ellos para poder compararlos. La clase cuenta con una variable de configuración en la que se representarán los campos de los objetos a comparar, y cada uno de ellos hará referencia a una instancia de la clase 'ObjectSimilarityCalculator'. Estas instancias pueden ser a objetos de la clase 'SimpleObjectSimilarityCalculator', es decir a estrategias concretas de comparación, o nuevamente a la clase 'ComposedObjectSimilarityCalculator', lo que permite poder comparar campos compuestos, es decir que contengan otros campos dentro. De esta manera es posible generar estructuras en forma de árbol.

Por lo tanto, al momento de realizar la comparación entre dos objetos, el método 'getSimilarity' dentro de la clase compuesta recorrerá la configuración predefinida, y por cada campo del objeto invocara nuevamente al método 'getSimilarity', el cual se corresponderá a una estrategia de comparación si es una hoja, o al nuevo objeto compuesto en caso que corresponda.

A la hora de comparar dos campos se obtendrá un valor entre 0 y 1, el cual indicará el grado de similitud entre ellos, debiendo luego implementarse una estrategia para integrar los valores de similitud de todos los campos y calcular la similitud final entre dos objetos. Para facilitar esta tarea se presenta el método *'integrate()'* que representa otro de los métodos gancho o 'HotSpots' del framework, a partir del cual el desarrollador podrá definir sus propias estrategias de integración. De este modo, al momento de utilizar el framework para realizar la comparación de objetos, se debe instanciar la clase correspondiente según la estrategia de integración que se quiera utilizar.

Como extensión de este framework, la clase 'AverageSimilarityCalculator' calculará el valor promedio de todos los valores obtenidos para cada campo y arrojará un valor entre 0 y 1 como valor de similitud final. Como parte de la estrategia seleccionada, dos objetos se consideraran candidatos a duplicados si superan un umbral el cual el usuario se encargara de predefinir.

Al finalizar el proceso se obtendrá un archivo el cual contendrá el objeto seleccionado como referencia para comparar, junto a sus pares que hayan superado el umbral predefinido, cada uno con su valor final de similitud.

5.2. Utilización del framework

Luego de recolectar los objetos de información y haberlos procesado a través del framework de normalización es momento de compararlos con la ayuda del actual framework. Al momento de realizar la comparación de objetos es necesario tomar uno de ellos como referencia para que sea comparado con todos sus pares, con el fin de identificar aquellos objetos que realmente se correspondan al mismo objeto seleccionado.

Para poder utilizar el framework debemos construir una aplicación que haga uso de él. Comparar dos o mas objetos implica comparar cada uno de sus campos, por lo que al momento de construir la aplicación de comparación se deben tener en cuenta diferentes consideraciones. En primer medida, identificar el objeto candidato a comparar con los demás, y luego la fuente o el conjunto de objetos sobre los que se realizaran las comparaciones.

Es imprescindible reconocer o seleccionar aquellos campos de los objetos sobre los que se realizaran las comparaciones y de qué manera se realizaran dichas comparaciones. Para ello, el usuario previamente deberá definir una configuración en la que indicara, a través de pares de datos clave-valor, sobre qué campos se realizaran las comparaciones y qué estrategia de comparación empleará sobre cada uno de ellos. En este momento será necesario comprobar si existe alguna estrategia o subclase concreta de las que ofrece el framework que cumpla con la tarea que el usuario pretende. En caso de que ninguna subclase sea de utilidad, o la estrategia deseada no exista, será necesario construir una nueva. Más adelante veremos como realizar esta tarea.

Luego de haber comparado campo a campo un objeto con otro y haber obtenido los valores de similitud para cada uno de ellos, el próximo paso es identificar aquella estrategia de integración a utilizar para calcular el valor de similitud final entre dos objetos. Al igual que al momento de seleccionar la estrategia de comparación para cada campo, en este caso es necesario seleccionar una estrategia de integración para calcular el valor de similitud final según todos los valores arrojados, por lo que se debe chequear si existe alguna subclase o estrategia de las que ofrece el framework que cumpla con el propósito deseado. La estrategia de integración seleccionada será la que se utilizará durante todo el proceso, debiendo ejecutar nuevamente la aplicación en caso de querer emplear una estrategia diferente. En tal caso, y considerando que la estrategia a emplear no se encuentre disponible, el usuario deberá extender el framework definiendo la propia.

Actualmente como estrategia de integración contamos con la clase 'AverageSimliarityCalculator', la cual, al momento de ser instanciada, recibirá a través de un parámetro la configuración definida previamente por el usuario, en donde se indicó para cada campo que estrategia de comparación emplear.

De esta manera, es posible invocar por cada campo a cada estrategia de comparación para calcular y almacenar el valor de similitud de cada uno de ellos, para finalmente poder calcular el valor final de similitud entre dos objetos a través del promedio general de cada uno de los valores.

Como se mencionó anteriormente, uno de los pasos a la hora de configurar la aplicación es seleccionar la estrategia de comparación para cada uno de los campos. Si la estrategia deseada no existe, o ninguna de las disponibles es de utilidad, es necesario construir una nueva. A continuación veremos cómo.

5.2.1. Extendiendo el framework

Mencionamos que para comparar campos simples entre dos objetos el programador debe definir una estrategia en particular. Con el objetivo de realizar dicho trabajo y llevar a cabo las implementaciones correspondientes, contamos con ayuda de la clase 'SimpleObjectSimilarityCalculator', la cual representa el HotSpot sobre el que el programador se podrá 'enganchar' para poder implementar las diferentes estrategias de comparación.

Diferentes alternativas fueron aquí propuestas e implementadas, proveyendo extensión al actual framework. Las mismas se presentan a continuación con el objeto de mostrar la funcionalidad que provee cada una de ellas, y a su vez brindar una guía con los pasos a seguir para poder extender el framework.

Las estrategias que se presentarán a continuación fueron pensadas principalmente para comparar dos grandes grupos de datos, como son números y cadenas de texto.

Para el caso de los números presentamos dos estrategias:

'SimpleNumberSimilarityCalculator': dados dos valores retornará un valor de 0 o 1 dependiendo si son iguales o no. Simplemente toma dos valores y comprueba igualdad. La utilizaremos cuando queramos saber con exactitud si dos valores son iguales o no, por ejemplo, al momento comparar el campo memoria, ya que nos interesa saber si cuentan exactamente con la misma capacidad.

La siguiente estrategia, **'NumberDistanceSimilarityCalculator'** calculará la distancia porcentual entre dos valores, debiendo indicar a través de un parámetro el porcentaje de distancia admitido entre ellos. De esta manera, si la distancia entre ellos se encuentra dentro del porcentaje indicado como parámetro, obtendremos un valor positivo; caso contrario, el valor obtenido será de 0. La utilizaremos cuando en lugar de querer obtener un valor exacto 0 o 1, como en el caso anterior, queramos obtener un valor intermedio que nos indique de cierta manera un grado de similitud. La utilizaremos, por ejemplo, al comparar el campo 'batería' o el campo 'precio', ya que en tales

casos nos interesa más saber si dos valores son cercanos en lugar de saber si son iguales.

Para comenzar y poder implementar cualquier estrategia, es necesario crear una clase que extienda de la interfaz 'SimpleObjectSimilarityCalculator' e implemente el método 'getSimilarity()'. En la siguiente figura se presenta el diseño de la clase 'NumberDistanceSimilarityCalculator' y la reimplementacion del método correspondiente.

```
const SimpleObjectSimilarityCalculator = require("../SimpleObjectSimilarityCalculator");

module.exports = class NumberDistanceSimilarityCalculator extends SimpleObjectSimilarityCalculator {
  constructor(porcentualDistance) {
    super();
    //maximum average distance (allows a percentage of difference between 2 numbers)
    this.allowedAverage = porcentualDistance;

  }

  /** it receives 2 numbers or prices and returns a value of similarity
   * working with the average distance between them
  */
  async getSimilarity(aNumber, anotherNumber) {
    if (!(aNumber || anotherNumber)) return 0;


    let firstNumber = parseFloat(aNumber.replace(/[^0-9]/g, ""));
    let secondNumber = parseFloat(anotherNumber.replace(/[^0-9]/g, ""));
    if (firstNumber === secondNumber) {
      return 1;
    }
    let average
    if (firstNumber > secondNumber) {
      average = firstNumber / secondNumber;

    } else {
      average = secondNumber / firstNumber;

    }

    average = average -1;

    if (average < this.allowedAverage) {
      return 1 - average;
    } else {
      return 0;
    }
  }
};
```

Figura 5.1: Clase 'NumberDistanceSimilatiryCalculator' como extension de 'SimpleObject' y su implementación del método getSimilarity()

Esta clase extiende a 'SimpleObjectSimilarityCalculator' y tiene su propia implementación del método 'getSimilarity()'. En la figura podemos observar que este método recibe como parámetros dos valores numéricos los cuales serán comparados luego. Como bien se mencionó anteriormente, la comparación entre los campos arrojara un valor entre 0 y 1 para aquellos pares de valores a comparar. En este caso, la estrategia a implementar consistió en tomar ambos valores y calcular la distancia porcentual entre ellos de la siguiente manera: se admitirá una diferencia porcentual del 7 % de distancia

entre ellos, es decir, que si el numero mas grande excede por mas del 7 % al otro, no sera considerado, retornando un valor final de 0, mientras que si la distancia entre ambos números es del 7 % o menos, el resultado final a retornar sera 1 - la distancia porcentual, siendo dicho valor el de similitud final entre ambos. Este valor de distancia porcentual admitida puede ser modificado por el programador a través de un parámetro. A continuación se presenta un ejemplo de cómo calcular dicha distancia entre dos números.

Por ejemplo, para calcular la distancia porcentual entre los valores $66.000 y $63.000, la tarea a realizar sera dividir el numero mas grande por el mas chico, obteniendo un valor, en este caso, de 1,047. A este valor le restaremos 1 obteniendo así la diferencia porcentual entre el mayor y el menor numero, en este caso 0,047 % . Si esta distancia o este valor es menor al valor de diferencia admitido por el programador, el resultado a devolver sera 1 - la diferencia porcentual, en este caso 1 - 0,047 = 0,953 como valor de similitud final entre ambos.

De esta manera, si los valores se encuentran dentro del rango de diferencia porcentual admitida, el valor de retorno del método sera cercano a 1. Teniendo en cuenta que el dominio elegido ha sido teléfonos celulares, es altamente probable que si dos teléfonos varían en su precio tan solo en un 7 % contengan características similares. Por lo tanto, este valor de retorno alto servirá para aumentar la precisión en el valor de similitud final entre ambos objetos.

Continuando con las estrategias de comparación de campo se presentaran aquellas elegidas para comparar cadenas de texto. Dentro de Javascript existe un gestor de paquetes llamado 'npm', el cual permite importar clases o código a través de librerías, algunas de las cuales realizan cálculos o tareas que resultarían muy complejas de programar. Una de las librerías se la conoce como *'Loadash'*, la cual permite comparar dos cadenas de texto a través de su método esEqual(), el cual retornará verdadero o falso si ambos textos coinciden o no. El hecho de que retorne verdadero o falso puede funcionar en el caso que queramos ver si dos cadenas o dos palabras son iguales o no, aunque en este caso en lugar de saber si son iguales o no, queremos obtener un porcentaje de similitud entre las palabras o el texto a comparar, por lo que necesitamos funciones que nos otorguen un valor que se encuentre entre un rango de '0 a 1' o '1 a 100' para poder realizar luego comparaciones con mayor precisión. Para ello se tuvieron en cuenta diferentes métodos de comparación, los cuales cumplen con esta ultima condición y retornan valores entre 0 y 1. Estos métodos son el 'Coeficiente de Dice' y 'Jaro Winkler'.

Para poder realizar comparaciones utilizando estos algoritmos hemos de extender el framework del mismo modo que con las estrategias presentadas anteriormente. Recordemos que para ello contamos con la ayuda de la inter-

faz 'SimpleObjectSimilarityCalculator', de la cual nos 'engancharemos' para poder implementar estas estrategias.

La primera que presentamos para comparar cadenas de texto utiliza el índice de **'Soresen-Dice'**, el cual es un estadístico utilizado para comparar la similitud entre dos muestras. Este fue utilizado por primera vez en el contexto de la botánica [Sørensen, 1948] para comparar la similitud entre dos especies. Dados dos conjuntos, X e Y, se define como:

$$s = \frac{2|X \cap Y|}{|X| + |Y|}$$

donde `|X|e|Y|`son el número de elementos de cada conjunto. El índice de Sørensen equivale al doble del número de elementos comunes a ambos conjuntos dividido por la suma del número de elementos de cada conjunto. La aplicación de este método nos retornará un valor entre 0 y 1 que nos servirá luego al momento de calcular el valor de similitud final.

La siguiente estrategia elegida utiliza la distancia de **'Jaro-Winnkler'**, la cual es una variante de la distancia de Jaro, propuesta por Winkler. Su propuesta fue modificar la similitud para dar mayor peso a las cadenas que contengan prefijos comunes, ya que en la mayoría de los casos los prefijos tienen mayor valor que los sufijos al identificar cadenas similares. Entonces, se introduce una escala que da calificación mas favorable a las cadenas que coincidan desde el comienzo. Al aplicar este método también obtendremos un grado de similitud entre 0 y 1 al igual que en el caso anterior.

Hasta aquí hemos presentado diferentes estrategias de comparación de campos extendiendo el framework a través de la clase 'SimpleObjectSimilarityCalculator', la cual funciona como hotspot o enganche. Esto nos permitirá asignar estas estrategias a los campos para obtener un valor de similitud entre ellos. Ahora es momento de seleccionar una estrategia de integración para tomar una decisión con los valores obtenidos y así llegar a un único y final valor de similitud entre dos objetos. Para ello, el framework cuenta con la clase llamada 'ComposedObjectSimilarityCalculator', la cual también funciona como hotspot y nos provee esa extensibilidad que buscamos. En esta clase contamos con un método llamado 'Integrate()', el cual deberá reimplementar cada subclase que extienda de ella, y será donde estará implementada la lógica de cada estrategia de integración.

En este caso, la estrategia elegida constó en sumar los valores arrojados para cada campo y dividirlo por la cantidad de campos que sólo hayan arrojado algún valor, y por lo tanto no sean nulos. A esta estrategia la denominamos 'AverageSimilarityCalculator', y será aquella que debamos instanciar al mo-

mento de construir la aplicación de similitud y será la que vaya a ser utilizada durante todo el proceso para calcular la similitud entre todos los objetos. A través de esta clase podremos realizar las invocaciones a los métodos correspondientes, ya sea para comparar un objeto con otro o para comparar un objeto con un mapa de objetos.

Teniendo entonces diferentes estrategias para realizar comparaciones entre campos, y a su vez una estrategia para calcular el valor de similitud final entre objetos, es momento de configurar la aplicación que hará uso del framework.

5.2.2. Configurando la aplicación normalizadora

Para poder definir la configuración el primer paso es identificar el objeto candidato a comparar contra todos sus pares. Luego debemos decidir qué campos pertenecientes al objeto queremos tener en cuenta para comparar, y para cada uno de ellos elegir la estrategia de comparación que creamos mas adecuada. Para poder realizar esto será necesario definir una variable de configuración en donde indicaremos qué estrategia utilizar para cada campo. Debemos chequear que la estrategia se encuentre definida o forme parte del framwework, caso contrario debemos extenderlo añadiendo la funcionalidad. Al igual que en el framework anterior contamos con una clase Factory la cual nos facilitara la importación de las estrategias de comparación.

```
var configuration = {
  name: FieldTypeCalculatorFactory.jaroWinklerSimilarityCalculator,
  processor: FieldTypeCalculatorFactory.jaroWinklerSimilarityCalculator,
  ramMemory: FieldTypeCalculatorFactory.simpleNumberSimilarityCalculator,
  storageMemory: FieldTypeCalculatorFactory.simpleNumberSimilarityCalculator,
  mainCamera: FieldTypeCalculatorFactory.simpleNumberSimilarityCalculator,
  price: FieldTypeCalculatorFactory.numberDistanceSimilarityCalculator(0.1),
  battery: FieldTypeCalculatorFactory.jaroWinklerSimilarityCalculator,
  processor: FieldTypeCalculatorFactory.jaroWinklerSimilarityCalculator,
  screenSize: FieldTypeCalculatorFactory.simpleNumberSimilarityCalculator,
  speedProcessor: FieldTypeCalculatorFactory.jaroWinklerSimilarityCalculator,
  operatingSystem: FieldTypeCalculatorFactory.jaroWinklerSimilarityCalculator
};
```

Figura 5.2: Declaracion de la variable de configuracion donde se establece la estrategia de comparacion a utilizar para cada campo

En este caso se tuvieron en cuenta la totalidad de los campos para comparar, y las estrategias elegidas fueron las siguientes:

- Para comparar cadenas de texto se opto por seleccionar la estrategia de Jaro-Winkler y utilizar dicha estrategia para todas las variables del tipo, para

analizar así su efectividad. Podemos también combinar y elegir para algunos campos la estrategia de Dice, o al contrario, utilizar esta ultima estrategia para todos los campos, pero en principio optamos por esta selección.

- Para el campo precio en particular se eligió la estrategia que calcula la distancia porcentual entre dos valores, admitiendo un porcentaje de distancia del 10 % entre ellos.

- Y para los demás valores numéricos optamos por la comparación simple entre ellos, la cual nos devuelve 0 o 1 si son idénticos o no.

De este modo tenemos lista la variable de configuración.

Posteriormente debemos seleccionar aquella estrategia de integración que sera utilizada en todo el proceso para calcular el valor de similitud final para cada objeto. Para ello debemos instanciar dicha clase a la cual le pasaremos la configuración antes definida, ya que es la superclase quien contiene el método con la lógica que recorre esta configuración e invoca al método de comparación correspondiente para cada campo. Esto lo haremos de la siguiente manera:

```
const averageSimilarityCalculator = new AverageSimilarityCalculator(
  configuration
);
```

Figura 5.3: Declaración de la variable 'averageSimilarityCalculator' como instancia de la clase ObjectSimilarityCalculator

Luego de haber instanciado esta clase podremos invocar al método 'getSimilarityMap()' al cual le enviaremos el objeto candidato y un archivo con todos sus pares a comparar. Lo haremos de la siguiente manera:

```
//compares a json object against a file with multiple objects
averageSimilarityCalculator
  .getSimilarityMap(inputObject, inputFile)
  .then((result) =>
    fs.writeFileSync(outputFile, JSON.stringify([...result]), "utf-8")
  )
  .catch((err) => console.error(err));
```

Figura 5.4: Llamada al método 'getSimilarityMap()' de la clase ObjectSimilarityCalculator

El método 'getSmilarityMap' recorrerá el mapa de archivos recibido y por cada objeto del mapa invocara al método 'getSimilarity()' enviando el objeto correspondiente de la iteración junto al objeto candidato. El método

'getSimilarity()' invocado que se ejecutará dependerá de qué clase lo este invocando, si se trata de una clase hoja, es decir un campo simple, o una clase compuesta, es decir, un campo compuesto.

Luego de comparados todos los campos para cada objeto y obtenido el valor de similitud final para cada uno, la aplicación retornara como salida un nuevo archivo con los objetos candidatos que hayan superado el umbral previamente definido, cada uno con su valor final de similitud.

Capítulo 6

Evaluación

En esta sección, detallaremos el proceso de evaluación, para el cual involucramos a dos colegas programadores con el objetivo de poner a prueba ambos frameworks, ampliando sus funcionalidades y obteniendo respuestas a un conjunto de preguntas específicas, para realizar un análisis de los resultados obtenidos.

6.1. Selección de los participantes

En primera instancia se invitó a participar a dos colegas compañeros de la universidad, con los que compartí cursada entre los años 2018 y 2019. Ambos han finalizado la carrera de licenciatura en sistemas y actualmente se encuentran trabajando en el ámbito, programando en diferentes lenguajes y con sólidos conocimientos en javascript. Sus conocimientos adquiridos en el área a lo largo de estos años, y a su vez la relación cercana con ellos, fue el motivo por el cual fueron seleccionados.

6.2. Entrenamiento de los participantes

Se realizó una reunión presencial con ambos en donde se les presentó la propuesta en general, los conceptos principales y la estructura de ambos frameworks junto con las tareas a realizar, las cuales describiremos a continuación. Se respondieron y aclararon las dudas de manera de asegurarnos que haya quedado clara la consigna. Todo lo charlado se les presentó luego escrito en un repositorio git, el cual contiene toda la información detallada con la idea principal del proyecto, la estructura de los frameworks y las tareas a realizar. Dicho repositorio se encuentra accesible para que cualquier per-

sona lo pueda clonar y realizar las pruebas correspondientes si así quisiera[1]. En él se encuentra detallado cómo funciona y cómo se debe extender cada framework. Su enlace se encuentra también disponible en el Apéndice A.

6.3. Preparación de la prueba

Además de dar las instrucciones de cómo funciona cada framework y cómo extenderlo para hacer uso de él, se les otorgó a los participantes un dataSet con los objetos sobre los que trabajarán. Los mismos fueron extraídos a través de la herramienta WOA y almacenados en el entorno que ésta provee. Estos objetos se almacenan en formato jSon y están caracterizados por el tipo de objeto 'mobilePhone' que se define en la ontología de DBpedia. DBpedia es un repositorio en la web, abierto y gratuito, con información estructurada proveniente de Wikipedia, y su objetivo es la definición de conocimiento semántico a través de grafos para dar sentido a la información de la web. Se recorrieron diversas paginas web y se extrajeron alrededor de 100 objetos de diferentes fuentes como Fravega, Garbarino, Compumundo, MercadoLibre, etc. Para cada objeto se seleccionó y extrajo la información más relevante, como por ejemplo el título, precio, memoria, cámara, batería, y otros. Es importante recordar que cada sitio presenta la información a su modo de acuerdo a la estructura y diseño de su pagina, por lo que buscamos transformar los datos de manera de que cada campo mantenga un formato homogéneo. A su vez, la información recolectada puede contener datos innecesarios o basura, de los cuales nos intentaremos deshacer para obtener una fuente de datos lo mas limpia y prolija posible.

Algunos ejemplos en la heterogeneidad de datos son:

- Para el campo batería:

  3000 mAh

  Li-Po 3000

  Li-Ion 5000

  5000 mA

- Cámara:

  12MP+64MP+12MP

  64 Mpx

  Cámara Principal 64 MP, f 1.8, 26 mm (ancho), 1 1.73 , 0.8, PDAF

[1] https://github.com/gonmastronardi/tesina-test

13.0 MP + 2.0 MP

- Precio:

  $87.732,67

  $18.379

- Velocidad Procesador:

  1300 MHz

  1.8 GHz

  Octa Core 2.3GHz

Tareas

Inicialmente se les pidió a los participantes que utilicen y extiendan el framework de normalización con el objetivo normalizar cada uno de los campos de los objetos recolectados para transformar el dataSet con el fin de obtener una fuente de datos lo mas limpia y depurada posible.

A cada participante se le asignó un par de campos específicos para normalizar, ya que son muchos y no buscamos normalizar todos ellos, sino evaluar y testear la funcionalidad del framework. Además a ambos se les pide por igual que normalicen los campos :

- 'operatingSystem'
- 'mainCamera'

para comparar sus soluciones, quedando entonces:

- Participante 1:
  - 'operatingSystem'
  - 'mainCamera',
  - speedProcessor
  - ramMemory o storageMemory
- Participante 2
  - 'operatingSystem'
  - 'mainCamera',
  - speedProcessor

- ramMemory o storageMemory

Luego de crear las estrategias de normalización para los campos recién mencionados, configurar y ejecutar el framework, deberán de procesar el archivo resultante a través del siguiente framework con el objetivo de identificar potenciales candidatos a duplicados, es decir aquellos objetos que se correspondan al mismo objeto en la vida real. Para ello tendrán que crear nuevas estrategias de comparación para aplicar sobre cada campo de manera de obtener un valor de similitud entre ellos. Luego, y a partir de los valores de similitud obtenidos para cada campo, deberán idear una nueva estrategia de integración de datos para obtener el valor de similitud final entre ambos objetos.

Como solución, se acordó que cada participante cree una nueva rama o branch sobre el repositorio que se les brindó para publicar sus resultados, por lo tanto allí se encuentran las soluciones propuestas. Esto luego nos permitió realizar un análisis comparativo. En el apéndice se adjuntan los enlaces a los repositorios con el código desarrollado por cada participante y las respuestas textuales de cada uno de ellos.

Para poder realizar la evaluación se analizó el código que cada participante publicó, comparandolo para poder obtener conclusiones.

6.3.1. Cuestionario

Luego de que hayan utilizado ambos FW y hayan completado las tareas correspondientes, se les realizó a los participantes un conjunto de preguntas para evaluar su experiencia de uso en general. Estas fueron:

- ¿Cuáles son las fortalezas que encuentra en cada framework?
- ¿Cuáles las debilidades?
- ¿Qué tan sencillo le pareció la extensión de cada framework?
- ¿Haría alguna modificación en la estructura de alguno de ellos?
- ¿Qué le falta a cada framework?
- ¿Cuál fue la experiencia general de uso?
- ¿Conoce algo similar?
- Finalmente, ¿en qué situación recomienda el uso de ellos?

6.4. Resultados

Comenzamos realizando una revisión y análisis del código publicado los participantes, donde podemos observar que, en cuánto al primer framework, ambos cumplieron con los patrones basados en herencia, extendiendo adecuadamente las clases solicitadas. Ambos participantes llevaron a cabo una correcta definición de la configuración, invocando las diferentes estrategias para cada campo.

A la hora de definir las estrategias de normalización para cada campo, el participante 1 parece comprender la consigna y realizar un trabajo correcto, mientras que el desarrollo del participante 2 parece ser algo confuso. Esto lleva a comunicarme con él y mediante una videollamada logramos aclarar las dudas y resolver los problemas, ya que, en lugar de normalizar los campos, estaba realizando las extensiones a modo de filtrado. Aclarado esto, se resuelve en el momento, y sube las actualizaciones de código al repositorio. Se vuelve a analizar el código y esta vez si la labor fue correcta.

Luego de analizar las estrategias propuestas de normalizacion asignada a cada participante, pasamos a detallar el desarrollo propuesto por cada uno para los campos asignados:

Para el campo ***'speedProcessor'*** asignado al participante numero 1, su estrategia fue, en primer medida, almacenar las coincidencias en Mhz y Ghz. Luego de ello transforma los valores de Ghz a Mhz y finalmente almacena la coincidencia de mayor valor en un nuevo campo.

Para el campo ***'ramMemory'*** busca las coincidencias de los valores en GB, y en caso de encontrar algún valor, lo retorna en una nueva variable recibida como parámetro a través del constructor.

El siguiente participante realiza lo propio para los campos asignados, en donde para el campo ***'processor'***, si encuentra el valor pasado como parámetro en cualquiera de los campos del objeto, lo almacena y lo retorna en una nueva variable.

Continuando con su desarrollo, al momento de normalizar el campo ***'battery'***, comprueba si el campo no contiene la cadena 'mah', y en caso afirmativo busca numero entero y le concatena dicha sigla

Para el campo ***'mainCamera'*** - uno de los campos en común a normalizar - ambos realizan el mismo trabajo y obtienen el mayor valor numérico dentro del campo, siendo ese el resultado final. El participante 1 devuelve solo el valor numérico, mientras que el segundo participante le concatena la cadena 'MP', lo cual me parece la decisión mas prolija y se corresponde a la solución que yo mismo propuse.

Con respecto al siguiente campo en comun, ***'operativeSystem'***, ambos resuelven de manera similar, con algunas diferencias en las lineas de código,

pero obteniendo el mismo resultado. Para dicho campo, corroboran si alguna de las palabras que se envían como parámetro se encuentran en el campo correspondiente. Si esto es así, retornan el valor encontrado en una nueva variable.

Luego de analizar el código realizamos las pruebas en la pc.

Al ejecutar el código del participante 2, observo que para el campo cámara no funcionaba el normalizador y encuentro que el nombre del campo en la configuración no se correspondía con el del objeto, por lo que se modifica y luego de eso funciona correctamente. Lo mismo sucedió con el campo operativeSystem, por lo que se corrigió y finalmente funciona. Para el campo batería la solución funciona perfectamente, mientras que para el restante y ultimo campo, processor, la solución no parece funcionar.

Luego nos movemos al repositorio del primer participante para hacer las pruebas y testear el funcionamiento del código, y todo funciona a la perfección al primer intento, normalizando cada campo correctamente y almacenando los valores buscados en nuevos campos.


```
    {
        "properties": {
            "battery": "2815 mAh",
            "mainCamera": "12 Mpx",
            "name": "Apple iPhone 12 Pro (128 GB) - Plata",
            "operatingSystem": "iOS",
            "price": "$274.990",
            "processor": "Apple A14 Bionic",
            "ramMemory": "6 GB",
            "screenSize": "6.1 \"",
            "speedProcessor": "2x3.1 GHz Firestorm, 4x1.8 GHz Icestorm",
            "storageMemory": "128 GB"
        },
        "type": "http://dbpedia.org/ontology/MobilePhone",
        "url": "https://www.mercadolibre.com.ar/apple-iphone-12-pro-128-gb
pdp_filters=category:MLA1055#searchVariation=MLA16163680&position=5&searcl
    },
```


Figura 6.1: Objeto extraído de la web sin normalizar

```
{
  "battery": "2815 mAh",
  "mainCamera": "12 Mpx",
  "name": "Apple iPhone 12 Pro (128 GB) - Plata",
  "operatingSystem": "iOS",
  "price": "$ 274.990,00",
  "processor": "Apple A14 Bionic",
  "ramMemory": "6 GB",
  "screenSize": "6.1 \"",
  "speedProcessor": "2x3.1 GHz Firestorm, 4x1.8 GHz Icestorm",
  "storageMemory": "128 GB",
  "url": "https://www.mercadolibre.com.ar/apple-iphone-12-pro-12
pdp_filters=category:MLA1055#searchVariation=MLA16163680&position=
  "dBPediaType": "http://dbpedia.org/ontology/MobilePhone",
  "operatingSystemName": "IOS",
  "maxMegapixels": 12,
  "speedProcessorMhz": 3100,
  "ramMemoryGB": 6
},
```


Figura 6.2: Objeto ya normalizado donde observamos que aquellos campos normalizados aparecen con sus valores en nuevos campos sobre el final del mismo

Podemos observar en la segunda imagen, que los campos normalizados aparecen al final, donde para cada uno de ellos sólo figura la información relevante.

En conclusión, y con respecto al primer framework, ambos participantes cumplen con la extensión del mismo de manera correcta y finalmente plantearon soluciones acordes a lo buscado. Con un participante hubo que limar o terminar de plantear correctamente el objetivo, mientras que el otro participante lo cumplió a la perfección.

Continuando con el segundo framework es momento de analizar las soluciones planteadas para la detección de objetos duplicados.

Como se menciono previamente, este framework podemos extenderlo de manera de, crear nuevas estrategias para obtener la similitud entre los campos, y nuevas estrategias para obtener la similitud entre dos objetos enteros.

En el caso del primer participante, la extensión del FW se realiza creando una nueva estrategia para poder calcular la similitud entre campos al momento de compararlos. Crea una nueva clase denominada 'LevenshteinComparator', que calcula la distancia de Levenshtein entre dos cadenas. La distancia de Levenshtein se calcula como el numero mínimo de ediciones de caracteres individuales (inserciones, eliminaciones o sustituciones) necesarias para transformar una cadena en la otra. En este caso particular, el participante calcula la distancia de Levenshtein y, en base al resultado obtenido, define el valor de similitud final de ambas cadenas. Si el numero de ediciones fue de 1 o 2, quiere decir que las cadenas son similares, por lo que retorna

un valor de 0,8. Caso contrario, el valor final de retorno es igual a 0.

Esta función, entonces, podremos invocarla al momento de definir la configuración y seleccionar qué estrategia de comparación aplicar sobre cada campo.

Luego, para obtener el valor de similitud final entre dos objetos, utiliza la función provista por el framework la cual calcula el promedio general de los valores obtenidos para cada campo.

Al momento de testear la funcionalidad encuentro que se demora algun/os minuto/s en finalizar y en algunos casos retorna undefined o null. Sin embargo, esta estrategia es una opción que encuentro sumamente valida para aplicar, mas allá que haya que realizar alguna modificación en el código para que termine de funcionar de manera correcta.

Continuando con el segundo framework, comenzamos con el análisis de la solución propuesta por el segundo participante. En un principio se observa que extiende el framework creando estrategias tanto para comparar campos individuales, como también creando una estrategia de integración para poder calcular el resultado final de similitud.

En cuanto a la estrategia de comparación de campos crea una clase denominada 'WordSimilarityCalculator', la cual realiza la comparación entre dos cadenas y calcula la similitud entre ellas en función de las palabras que tengan en común. Devuelve un valor entre 0 y 1, donde 0 indica que no tienen palabras en común y 1 indica que todas las palabras en la cadena A tienen al menos una coincidencia en la cadena B. Esta estrategia es una opción valida en algunos casos dependiendo lo que busque el desarrollador, ya que si la función evalúa dos palabras muy similares como por ejemplo 'banana' y 'anana', devolverá 0. Queda en manos del programador seleccionar esta estrategia o evaluar la posibilidad de utilizar otra, como por ejemplo, una que compare carácter por carácter, para obtener un resultado mas preciso dependiendo sus necesidades.

En cuanto a la estrategia de integración de datos y para calcular el valor final del objeto en base a los resultados obtenidos para cada campo, lo que realiza la estrategia implementada es calcular la suma de todos los valores, asignando 0 a aquellos valores que sean nulos o indefinidos, y calcular el promedio de acuerdo a la cantidad de campos. Esta estrategia, a diferencia de la que otorga el FW asigna valor de 0 a los indefinidos o nulos, mientras que la otra no los tiene en cuenta. Esto nos generaría una diferencia en los valores finales reduciendo el valor de similitud.

A la hora de testear el código, se observa que el participante utiliza la estrategia de comparación de campos implementada y la aplica a todos ellos al momento de definir la configuración:

```
var configuration = {
    name: new normalizers.CleanNormalizer(
   unusefulWordsForName),
    price:  new normalizers.MonetaryAmountNormalizer(),
    processor: new normalizers.ProcessorNormalizer("
   Snapdragon", "processor"),
    batery: new normalizers.BatteryNormalizer("battery"),
    os: new normalizers.OSNormalizer(["Android", "IOS"],"
   operativeSystem"),
    camera: new normalizers.MainCameraNormalizer("mainCamera
   "),

};
```

y observando los resultados arrojados nos encontramos con lo siguiente:

OBJETO TOMADO COMO REFERENCIA

```
  {
    "mainCamera": "12 Mpx",
    "name": "Apple iPhone 12 mini (64 GB) - (PRODUCT)RED",
    "operatingSystem": "iOS",
    "price": "$ 180.899,00",
    "processor": "Apple A14 Bionic",
    "ramMemory": "4 GB",
    "screenSize": "5.4 \"",
    "storageMemory": "64 GB",
    "url": "https://www.mercadolibre.com.ar/apple-iphone-12-
   mini-64-gb-productred/p/MLA16163667?pdp_filters=category:
   MLA1055#searchVariation=MLA16163667&position=3&
   search_layout=stack&type=product&tracking_id=8c05bfac-7d2a
   -41a3-890a-5f556b233654",
    "dBPediaType": "http://dbpedia.org/ontology/MobilePhone",
    "operatingSystemName": "IOS",
    "maxMegapixels": 12,
    "ramMemoryGB": 4
  },
```

RESULTADO FINAL COMPARADO CONTRA EL MISMO OBJETO

```
    {
     "mainCamera": "12 Mpx",
     "name": "Apple iPhone 12 mini (64 GB) - (PRODUCT)RED",
     "operatingSystem": "iOS",
     "price": "$ 180.899,00",
     "processor": "Apple A14 Bionic",
     "ramMemory": "4 GB",
     "screenSize": "5.4 \"",
     "storageMemory": "64 GB",
     "url": "https://www.mercadolibre.com.ar/apple-iphone
   -12-mini-64-gb-productred/p/MLA16163667?pdp_filters=
```

```
    category:MLA1055#searchVariation=MLA16163667&position=3&
    search_layout=stack&type=product&tracking_id=8c05bfac-7d2a
    -41a3-890a-5f556b233654",
11      "dBPediaType": "http://dbpedia.org/ontology/MobilePhone
    ",
12      "operatingSystemName": "IOS",
13      "maxMegapixels": 12,
14      "ramMemoryGB": 4
15    },
16    0.8
```

RESULTADOS DE COMPARACIÓN ENTRE CAMPOS

```
1   Map(0) {
2     name: 1,
3     processor: 1,
4     ramMemory: 1,
5     storageMemory: 1,
6     mainCamera: 1,
7     price: 1,
8     battery: null,
9     screenSize: 1,
10    speedProcessor: null,
11    operatingSystem: 1
12  }
13
```

En el ultimo JSON observamos los resultados de la comparación entre estos dos objetos para los campos que fueron indicados en la configuración, y que finalmente nos terminaran de dar el valor de similitud final entre ambos objetos. Observamos que ese valor es de 0,8 ya que dos de los diez campos tomados para comparar son nulos, lo que reduce el valor de similitud.

Para este objeto tomado como referencia los primeros 5 resultados arrojados fueron los siguientes:

- "Apple iPhone 12 mini (64 GB) - (PRODUCT)RED", con 0,8 de similitud
- "Apple iPhone 11 (64 GB) - Blanco", con 0.5791666666666666 de similitud
- "Apple iPhone 12 Pro (128 GB) - Plata", con 0.5125
- "iPhone 11 Pro 512 GB Gris espacial", con 0.47916666666666663
- "Apple iphone 11 pro max 64 gb Plateado 64 GB", con 0.425

y en sexto lugar

- "Nokia 7.1 64 GB acero brillante 4 GB RAM", con 0.35 de similitud.

Si bien el resultado es bastante acertado, esperaríamos un valor igual a 1 para el primer objeto, ya que se esta comparando contra si mismo. Con respecto a los demás objetos sucedería algo similar, en donde si tomáramos los valores nulos o indefinidos como 0, se reduciría la precisión de los resultados obtenidos.

Pasamos ahora a ejecutar la solución del otro participante, y tomamos el mismo objeto como referencia con la configuración que él mismo haya definido para comparar los resultados.

En este caso define la siguiente configuración:

```
var configuration = {
    name: fieldComparators.jaroWinklerSimilarityCalculator,
    processor: fieldComparators.
   jaroWinklerSimilarityCalculator,
    ramMemory: fieldComparators.
   jaroWinklerSimilarityCalculator,
    storageMemory: fieldComparators.
   jaroWinklerSimilarityCalculator,
    mainCamera: fieldComparators.
   jaroWinklerSimilarityCalculator,
    price: fieldComparators.jaroWinklerSimilarityCalculator,
    battery: fieldComparators.jaroWinklerSimilarityCalculator
   ,
    screenSize: fieldComparators.
   jaroWinklerSimilarityCalculator,
    speedProcessor: fieldComparators.
   jaroWinklerSimilarityCalculator,
    operatingSystem: fieldComparators.levensteinComparator,
    operatingSystemName: fieldComparators.
   levensteinComparator
  };
```

Utiliza gran parte de la configuración predefinida y para el campo sistema operativo utiliza el algoritmo de comparación de Levensthein. El resultado obtenido a partir de la comparación del objeto

- "Apple iPhone 12 mini (64 GB) - (PRODUCT)RED"

fue la siguiente:

En primer lugar obtenemos el mismo objeto con resultado final de similitud igual a 1.

Luego:

- “Apple iPhone 11 (64 GB) - Blanco”, con 0.9298422005786348 de similitud
- “iPhone 11 Pro 512 GB Gris espacial”, con 0.8806523636699796
- “Apple iPhone 12 Pro (128 GB) - Plata”, con 0.8795335586033262
- “Apple iphone 11 pro max 64 gb Plateado 64 GB”, con 0.8501331140866024

y en sexto lugar

- “Nokia 7.1 64 GB acero brillante 4 GB RAM”, con 0.8192404776160679

Al observar estos resultados y compararlos con los del anterior participante, resulta interesante ver que los 6 primeros resultados obtenidos son los mismos, solo que con el tercer y cuarto objeto invertidos. A su vez, podemos remarcar una notable diferencia en cuanto al valor de los resultados obtenidos, siendo estos últimos bastante mayores en comparación a los anteriores, por lo que podemos deducir que para este último caso podríamos filtrar los objetos utilizando un umbral mas alto o mas estricto, mientras que para el caso anterior deberíamos utilizar un umbral menor o considerar una menor precisión.

6.4.1. Análisis del cuestionario

En términos generales, las propuestas de ambos participantes, junto con sus soluciones y resultados, han sido positivas. Han demostrado comprender la consigna y han aplicado de manera adecuada los frameworks, extendiéndolos de manera correcta.

Para finalizar su participación y extraer las conclusiones finales, se les realizó a los participantes un conjunto de preguntas, las cuales repasaremos a continuación.

Se les consultó, en primer término, cuales son las **fortalezas** y las **debilidades** que encuentran en ambos frameworks. Ambos coinciden en que son sencillos de utilizar y destacan la facilidad de extensión y configuración de los mismos. El participante 1, además, menciona que a su parecer el framework de comparación es el que mas valor puede aportarle a un proyecto, punto en el que estoy de acuerdo, aunque su utilización nos dará una mejor performance si antes utilizamos el primer framework. Por ello es que se motiva la utilización en cadena de ambos.

En cuanto a las debilidades uno de los participantes menciona:

- *"no tienen comunicación directa los frameworks, es decir que la ejecución de uno no implica la ejecución de otro, si bien son frameworks separados, tendrían que tener un túnel de comunicación entre ellos para hacer mas ameno al usuario el uso de ambos"*

Si bien son frameworks independientes y la ejecución de uno no implica la ejecución del otro, el usuario tiene la posibilidad de llevar a cabo ambas tareas ejecutándolos en cadena a través de un script. En relación a esto, al ejecutar el primer framework, es posible generar automáticamente una copia del archivo normalizado resultante y depositarlo en la carpeta de datos de entrada del segundo framework, evitando la necesidad de buscarlo en otro directorio.

Ambos participantes destacan falencias en cuanto a la documentación, solicitando que sea mas accesible al lector general y menos técnica, y que para ello exista un apartado para cada tarea especifica.

Uno de ellos menciona que no se denota con exactitud los limites de caja blanca en relación con caja negra a nivel de código. En mi opinión, los hotspots están definidos con claridad en cuanto a su extensión y puntos de conexión, aunque podría ser beneficioso aclararlo de manera mas detallada en la documentación.

Otro punto mencionado por uno de los participantes, es su preferencia por no declarar los normalizadores o comparadores en un archivo separado al .config que los ejecuta, y sugiere que sería conveniente incluirlos directamente en el archivo de configuración. Esta es una consideración valida que queda a criterio de quien utilice el framework.

Cuando se les consultó sobre la simplicidad de la extensión de ambos frameworks, ambos coincidieron que fue sencillo luego de haber leído la documentación y comprender la función de cada archivo. Uno de los participantes destacó, además, que le pareció beneficioso que ambos FW tengan la misma estructura.

Luego se les preguntó que le falta a cada framework. Ambos hicieron hincapié en mejorar la documentación agregando más ejemplos para que los desarrolladores tengan una comprensión clara del abanico de tareas que se pueden realizar con ambos FW. El participante 1 menciona la carencia de un ejemplo sólido de integración para ejecutar el mismo dataset con ambos frameworks. Por otro lado, el participante 2 expresó que elaboraría una documentación especifica para cada FW, detallando su arquitectura, puntos de extensión, modos de ejecución y casos de prueba.

Cuando se les consultó si harían alguna modificación en la estructura de alguno de los frameworks, el primer participante mencionó que modificaría la estructura de carpetas y los nombres. Por ejemplo, para el normalizador de campos, cambiaría "field" por "attribute". En cuanto a la estructura de

carpetas no especifica como lo haría. El siguiente participante propuso migrar el proyecto a Typescript en lugar de Javascript, ya que considera que puede ofrecer una estructura mas robusta para manejar posibles errores que puedan ocasionarse debido a modificaciones del usuario. Además, sugiere la modificación en la estructura de directorios, proponiendo la creación de una carpeta denominada 'public', donde se encuentren los puntos de extensión del framework, de manera que la parte congelada del framework quede separada.

Cuando se les consultó si conocían algo similar a estos frameworks, ambos coincidieron en que no. En el ultimo tiempo, Google comenzó a mostrar en sus resultados de búsqueda, para algunos elementos en particular, una tarjeta a la derecha de los resultados de búsqueda, en donde se muestra el nombre del objeto buscado con sus características, los diferentes sitios en los que se ofrece y su precio por cada uno. En ultima instancia, nuestro objetivo es similar, en donde para cada articulo que queramos identificar sus duplicados, podamos fusionar los resultados obtenidos en un único objeto con la mayor cantidad de información posible, incluyendo los diferentes sitios en los que se ofrece junto a sus respectivos precios.

Finalmente, se les pregunto a los participantes en que situaciones recomendarían la utilización de estos frameworks. El primer participante mencionó que se lo recomendaría a cualquier persona que esté involucrada en un proyecto en Javascript, como por ejemplo, cualquier web en node.js, y que constantemente necesite limpiar ciertas entradas de datos que no necesariamente están en una base de datos. El otro participante considera que es beneficioso para combinarlo con bases de datos como mongoDB, que almacenan atributos en archivos JSON.

Capítulo 7

Conclusiones y trabajo futuro

7.1. Conclusiones

Comenzamos este trabajo destacando la gran cantidad de información que se encuentra disponible en la web, su crecimiento exponencial, y la complejidad asociada a la búsqueda de información y la toma de decisiones. Hicimos énfasis en la desestructuración de la web y la manera en que los sitios presentan la información sin preocuparse por otorgarle estructura y significado a los datos.

A raíz de ello exploramos el concepto de web semántica, exponiendo su objetivo y señalando su lenta adopción. Reconocimos que la construcción de aplicaciones capaces de aprovechar plenamente la información de la web a menudo implica la extracción manual de datos, lo cual constituye un desafio significativo.

Introdujimos la herramienta WOA, resaltando su funcionalidad y los desafíos a los que nos enfrentamos en el marco de recolección de objetos. Fue a raíz de estas dificultades que delineamos los objetivos y aportes de este trabajo. En primer lugar, nos propusimos la construcción de un Framework (FW) de normalización con el fin de limpiar y corregir los datos recolectados, obteniendo así una fuente de datos limpia y confiable para la siguiente fase: la detección de objetos duplicados mediante un segundo FW.

Seleccionamos un dominio de objetos específico, en este caso, teléfonos celulares, y recopilamos información de diversas fuentes a través de WOA para llevar a cabo las pruebas. Definimos la estructura, diseño y uso de ambos frameworks, identificando hotspots o puntos de extensión para su personalización.

Desarrollamos estrategias de normalización para el primer FW, adaptadas al dominio de objetos, y realizamos pruebas correspondientes. De manera

similar, definimos estrategias de comparación para el segundo FW, evaluando la similitud entre distintos objetos.

Posteriormente, para evaluar los FW desarrollados, convocamos a dos colegas programadores, presentándoles el proyecto y solicitándoles que los evaluaran mediante extensiones y la definición de estrategias propias para realizar pruebas. Luego de su implementación, se realizaron entrevistas para evaluar su experiencia de uso y discutir los resultados obtenidos.

A partir de lo desarrollado en base a los objetivos propuestos en este trabajo y luego del análisis de la evaluación realizada a ambos colegas, podemos concluir que la solución es aplicable para los programadores, y que es posible, a través de la utilización de ambos frameworks, definir estrategias para normalizar los objetos e identificar potenciales candidatos a duplicados.

Los participantes destacaron fortalezas en ambos frameworks, resaltando su simplicidad de uso y la facilidad de extensión y configuración. Sin embargo, reconocieron la necesidad de mejorar la comunicación entre los frameworks y pidieron por una documentación más sencilla, específica y con ejemplos detallados. Para ello se sugiere una mayor integración de ejemplos y una documentación específica para cada framework, abordando arquitectura, modos de ejecución y casos de prueba.

En cuanto a modificaciones estructurales, se propusieron cambios en la nomenclatura y estructura de carpetas, así como la migración a Typescript para mejorar la robustez.

A pesar de las áreas de mejora señaladas, ambos participantes elogiaron la facilidad de extensión de los frameworks.

En términos de posibles áreas de utilización, sugieren su uso en proyectos JavaScript para realizar limpieza de datos, y su combinación en proyectos con bases de datos como MongoDB, ya que almacenan atributos en archivos JSON.

7.2. Trabajo futuro

Para concluir este trabajo y abordar una tarea pendiente importante, nos centraremos en la integración de los objetos duplicados. Esta es una fase que nos asegurara la calidad y la integridad de la información almacenada.

Una vez que hemos obtenido el archivo que contiene los potenciales candidatos a duplicados, la tarea principal consiste en habilitar al usuario para que determine qué elementos verdaderamente representan el mismo objeto en la realidad.

El objetivo principal es obtener un único objeto que encapsule la máxima cantidad de información posible.

En este proceso de integración, proponemos la incorporación de una interfaz gráfica intuitiva que esté diseñada para facilitar la revisión y selección de objetos duplicados.

Se sugiere una propuesta que implique presentar todos los objetos resultantes en una pantalla, cada uno con sus campos acompañados de casillas de verificación. De esta manera, el usuario puede inicialmente seleccionar los objetos idénticos y, posteriormente, para cada campo, elegir el valor que considere más preciso o legible.

A pesar de ser una solución práctica, esta tarea fue relegada debido a la extensión que implicaría y a la ausencia de la implementación de una interfaz gráfica.

Apéndice A

Información y enlaces de utilidad

Aquí se destaca información importante para terminar de comprender el proyecto.

En primer término se proveen los enlaces a los repositorios GitHub con el código fuente de ambos frameworks. Estos son los repositorios utilizados personalmente donde se encuentra la última versión de código con el que se realizaron las pruebas de manera local, y donde se encuentran detalladas las diferentes estrategias de normalización y comparación que se aplicaron sobre todos los campos. Luego se presenta otro enlace a GitHub con el código fuente presentado a los colegas programadores para que testeen y extiendan ambos frameworks. Dicho código se encuentra actualizado y mejorado pero no contiene todas las estrategias de normalización implementadas, sino solo algunas a modo de ejemplo y guía. En este ultimo enlace se encuentran las instrucciones de como utilizar ambos frameworks.

- Framework de normalización.
- Framework de identificación de duplicados

A continuación provee el enlace al repositorio GitHub donde se encuentra el código fuente presentado a los colegas programadores para que puedan clonar y realizar las pruebas correspondientes. Dicho repositorio puede ser clonado para realizar las pruebas:

Repositorio Git.

El siguiente enlace contiene las respuestas pertenecientes a la evaluación realizadas a uno de los participantes. Las mismas se encuentran tal cual el participante las respondió:

Respuestas participante 1.

Las respuestas del participante 2 se presentaran a continuación tal cual fueron respondidas, ya que sus respuestas fueron recibidas a través de correo

electrónico:

- ¿Cuáles son las fortalezas que encuentra en cada framework?

  La primera fortaleza es la capacidad de extension del mismo, dado a que la configuracion de una estrategia no solo depende del core del framework, sino que se pueden agregar configuraciones alternativas y personalizadas que pueden influir en un desarrollo mucho mas especifico segun el usuario y segun el campo que se quiera evaluar. Otra fortaleza que se puede ver es la estructuracion de los archivos, dado a que se puede identificar y organizar bien las nuevas caracteristicas agregadas por el usuario.

- ¿Cuáles las debilidades?

  Una debilidad que puedo encontrar es tratar que no tienen comunicacion directa los frameworks, es decir que la ejecucion de uno no implica la ejecucion de otro, si bien son frameworks separados, tendrian que tener un tunel de comunicacion entre ellos para hacer mas ameno al usuario el uso de ambos, y la lectura de la documentacion mas prescisa en cuanto a esta caracteristica. Tambien no se denota con exactitud los limites de la caja blanca en relacion de la caja negra de cada uno a nivel de codigo. No tener un limite del buffer de entrada, dado a que si envio un archivo masivo de datos por el codigo visualizado no seria viable el uso, dado que el procesamiento es secuencial.

- ¿Qué tan sencillo le pareció la extensión de cada framework?

  Al principio me parecio medio rebuscado, todo el analisis quese tenia que hacer, pero luego de leer la documentacion y entender de que trataba no tuve problema alguno.

- ¿Haría alguna modificación en la estructura de alguno de ellos?

  Si, particularmente yo migraria el proyecto para que se trabaje con TypeScript en vez de JavaScript, pero por comodidad de desarrollo, aparte de ofrecerme una estructura mas robusta a errores que puedan a llegar a ocasionarse por modificaciones del usuario. Tambien haria en apartados la documentacion de ambos framework, con casos de ejemplo, testing, explicacion de la arquitectura. A nivel de codigo haria una estructuracion para que los campos de caja blanca sean solo de una carpeta que no se pueda tocar el contenido directo del framework. Es decir crear una carpeta denominada ‘public‘ donde dentro de la misma, tenemos varias carpetas identificadas con la accion que se puede implementar, extendiendo asi el contenido del index para que pueda

ejecutarse correctamente, por lo que no habria conflicto de modificacion del core inicial del framework ante la modificacion de un usuario. Tambien podria hacerce un paquete externo de node, para que se pueda usar sin tener el codigo directo en el proyecto. En el caso de que se pudiera incorporar una estrategia de paralelismo para evitar el problema de ingreso de datos masivos

- ¿Qué le falta a cada framework?

  Yo haria particularmente una documentación destinada para cada uno, donde se especifica toda la arquitectura del mismo junto con sus partes de caja blanca, destacando los modos de ejecucion y varios casos de prueba con alguritmos ya escritos dentro del mismo framework

- ¿Conoce algo similar?

  Al principio asilime una similitud hacia aplicaciones de viajes y hosting (tipo Trivago o Booking) pero, a este nivel de desarollo, no conozco ninguna tecnologia ni algo similar que se compare a estos framework

- Finalmente, ¿en qué situación recomienda el uso de ellos?

  El normalizer esta bueno para todo lo que es gestion adquisitiva de cada persona, dado a que cuando una persona quiera comprar algo, ya tiene todas las opciones precargadas. Otra situacion que si estaria buena es para la gestion y administracion de aerolineas, donde cada vuelo va a tener su conjunto de información y el administrador puede hacer un seguimiento mas directo dado a una categoria del mismo.

Bibliografía

[1] Ortona S. An analysis of duplicate on web extracted objects. In: Proceedings of the 23rd International Conference on World Wide Web - WWW '14 Companion; 2014. .

[2] Bosetti G, Firmenich S, Rossi G, Winckler M, Barbieri T. Web objects ambient: An integrated platform supporting new kinds of personal web experiences. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics); 2016. .

[3] vanden Broucke S, Baesens B. Practical Web Scraping for Data Science; 2018.

[4] Nylen EL, Wallisch P. Web Scraping. In: Neural Data Science; 2017. .

[5] Quinn AJ, Bederson BB. AskSheet: efficient human computation for decision making with spreadsheets. In: Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing; 2014. p. 1456–1466.

[6] Fernández A, Bosetti G, Firmenich S, Zaraté P. Logikós: Augmenting the Web with Multi-criteria Decision Support. In: International Conference on Decision Support System Technology. Springer; 2019. p. 123–135.

[7] Elfeky MG, Verykios VS, Elmagarmid AK. TAILOR: A record linkage toolbox. In: Proceedings - International Conference on Data Engineering; 2002. .

[8] Singh A, Sinha U, Sharma DK. Semantic Web and Data Visualization. In: Data Visualization and Knowledge Engineering. Springer; 2020. p. 133–150.

[9] Dalvi N, Kumar R, Soliman M. Automatic wrappers for large scale web extraction. Proceedings of the VLDB Endowment. 2011;4(4):219–230.

[10] Lenzerini M. Data integration: A theoretical perspective. In: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems; 2002. p. 233–246.

[11] Elmagarmid AK, Ipeirotis PG, Verykios VS. Duplicate record detection: A survey. IEEE Transactions on knowledge and data engineering. 2006;19(1):1–16.

[12] Ravikanth M, Vasumathi D. Record matching over query results from multiple web databases with duplicate detection. Journal of Advanced Research in Dynamical and Control Systems. 2018;.

[13] Kannan A, Givoni IE, Agrawal R, Fuxman A. Matching unstructured product offers to structured product specifications. In: Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining; 2011. p. 404–412.