Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

Análisis y comparacion de paquetes para el desarrollo de web scraping

Autor: David Márquez Mínguez

Tutor: Juan José Cuadrado Gallego

2022

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

Trabajo Fin de Grado

Análisis y comparacion de paquetes para el desarrollo de web scraping

Autor: David Márquez Mínguez

Tutor: Juan José Cuadrado Gallego

Tribunal:

Presidente: . . . . . .

Vocal 1º: . . . . . .

Vocal 2º: . . . . . .

Fecha de depósito: . . . . de . . . . . de . . . . .

Resumen

Resumen..... correo de contacto: David Márquez Mínguez <david.marquez@edu.uah.es>.

Palabras clave: Trabajo fin de /grado, LaTeX, soporte de español e inglés, hasta cinco....

Abstract

Abstract..... contact email: David Márquez Mínguez <david.marquez@edu.uah.es>.

Keywords: Bachelor final project , LaTeX, English/Spanish support, maximum of five....

Resumen extendido

Con un máximo de cuatro o cinco páginas. Se supone que sólo está definido como obligatorio para los TFGs y PFCs de UAH.

Índice general

Índice de figuras

Índice de tablas

Índice de listados de código fuente

Capítulo 1

Introducción y objetivos

1.1 Contexto

La *World Wide Web* o lo que comúnmente se conoce como la web, es la estructura de datos más grande en la actualidad, y continúa creciendo de forma exponencial. Este gran crecimiento se debe a que el proceso de publicación de dicha información se ha ido facilitando con el tiempo.

Tradicionalmente el proceso de inserción y extracción de la información se realizaba a través del copy-paste. Aunque este método en ocasiones pueda ser la única opción, es una técnica muy ineficiente y poco productiva, pues provoca que el conjunto final de datos no esté bien estructurado. El web scraping o minado web trata precisamente de eso, de automatizar la extracción y almacenamiento de información extraída de un sitio web [2].

La forma en la que se extraen datos de internet puede ser muy diversa, aunque comúnmente se emplea el protocolo HTTP, existen otras formas de extraer datos de una web de forma automática [3]. Este proyecto, se centra en la metodología existente de obtención de información, de como se tratan los datos y la forma en la que se almacenan. Durante los siguientes apartados se realizará una especificación mas concreta del objetivo del proyecto, así como de la estructura y limitaciones del mismo.

1.2 Motivación

El proceso de extracción y recopilación de datos no estructurados en la web es un área interesante en muchos contextos, ya sea para uso científico o personal. En ciencia por ejemplo, los conjuntos de datos se comparten y utilizan por múltiples investigadores, y a menudo también son compartidos públicamente. Dichos conjuntos de datos se proporcionan a través de una API [1] estructurada, pero puede suceder que solo sea posible acceder a ellos a través de formularios de búsqueda y documentos HTML. En el uso personal también ha crecido a medida que han comenzado a surgir servicios que proporcionan a los usuarios herramientas para combinar información de diferentes sitios web en su propias colección de páginas.

Además de ser un ambito interesante, el minado web también es un area muy requerida, algunos de las campos de mayor demanda tienen relacion con la venta minorista, mercado de valores, análisis de las redes sociales, investigaciónes biomédicas, psicológia...

[1] Conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizada por otro software como una capa de abstracción. [4]

1.3 Objetivo y limitaciones

Existen muchos tipos de técnicas y herramientas para realizar web scraping, desde programas con interfaz gráfica, hasta bibliotecas software de desarrollo. El objetivo de esta tesis es realizar un análisis cuantitativo de las diferentes técnicas y paquetes software para el desarrollo de web scraping.

¿Cuál es la solución más rentable para el minado web? ¿Cuál de las soluciones tiene un mejor rendimiento? Para responder a esta pregunta, se realizará un estudio comparando las diferentes características de los paquetes software, con el objetivo finalmente de poder determinar cuál es el más óptimo en términos de memoria, rendimiento...

En cuanto a las restricciones para el funcionamiento de un scraper, estas pueden ser varias, ya sean legales o por la incapacidad de acceder a una gran parte del contenido no indexado en internet. Aunque el uso de los scrapers está generalmente permitido, en algunos países como en Estados Unidos, las cortes en múltiples ocasiones han reconocido que ciertos usos no deberían estar autorizados [2]. El desarrollo de este proyecto no se verá perjudicado por este tipo de cuestiones, pues solo se limitará al estudio y análisis de los mismos.

1.4 Estructura del documento

Para poder facilitar la composición de la memoria se detalla a continuación la estructura de la misma:

1. **Bloque I: Introducción.**
   - **Capítulo 1: Introducción.**

     En la introducción se especifica tanto el contexto como la motivación a realizar el proyecto, así como las limitaciones esperadas durante la realización del mismo.
2. **Bloque II: Marco teórico.**
   - **Capítulo 2: Web scraping, extracción de datos en la web.**

     Durante este capítulo se explica en que consiste el web scraping, sus posibilidades prácticas y aspectos más generales.
   - **Capítulo 3: Introducción a los paquetes seleccionados y proceso de búsqueda.**

     Se realiza la selección de paquetes y se dictamina cuál ha sido la razón por la que los paquetes han sido seleccionados. Además, se especifican las características principales de cada uno, así como una visión general de sus funcionalidades.
3. **Bloque III: Marco práctico.**
   - **Capítulo 4: Selección de variables de análisis y proceso de estudio.**

     Durante este capítulo se especifica el proceso de análisis a realizar, cuáles son los test a los que los paquetes serán sometidos y que variables se tomarán a estudio para los mismos.
   - **Capítulo 5: Análisis y comparativa de paquetes.**

     Una vez introducidos todos los paquetes y el estudio al que van a ser sometidos, se realizá la comparativa de los mismos. Inicialmente, los paquetes serán analizados uno por uno y finalmente se hará una comparativa con los datos obtenidos.
4. **Bloque IV: Conclusiones y futuras líneas de trabajo.**

Capítulo 2

Web scraping, extracción de datos en la web

2.1 En que consiste realmente el web scraping

En la actualidad el web scraping se puede definir como una *"solución tecnológica para extraer información de sitios web, de forma rápida, eficiente y automática, ofreciendo datos en un formato más estructurado y más fácil de usar [5]"*. Sin embargo, esta definición no ha sido siempre así, los métodos de minado web han evolucionado desde procedimientos más pequeños con ayuda humana, hasta sistemas automatizados capaces de convertir sitios web completos en conjuntos de datos bien organizados.

Existen diferentes herramientas de minado, no solo capaces de analizar los lenguajes de marcado o archivos JSON, también capaces de realizar un procesamiento del lenguaje natural para simular cómo los usuarios navegan por el contenido web.

2.1.1 Extracción de los datos

En realidad el minado web es una práctica muy sencilla, se extraen datos de la web y se almacenan en una estructura de datos para su posterior análisis o recuperación. En este proceso, un agente de software, también conocido como robot, imita la navegación humana convencional y paso a paso accede a tantos sitios web como sea necesario [6]. Las fases por las que pasa el agente software en cuestión se determinan a continuación:

1. **Fase de búsqueda:** Esta primera fase consiste en el acceso al sitio web del que se quiere obtener la información. El acceso se proporciona realizando una solicitud de comunicación HTTP. Una vez establecida la comunicación, la información se gestiona a partir de los métodos GET y POST usuales.

2. **Fase de extracción:** Una vez que el acceso ha sido permitido, es posible obtener la información de interés. Se suelen emplear para este propósito expresiones regulares o librerías de análisis HTML. Los diferentes softwares empleados para este propósito se especifican en la sección 2.1.2.

3. **Fase de transformación:** El objetivo de esta fase es transformar toda la información extraída en un conjunto estructurado, para una posible extracción o análisis posterior. Los tipos de estructuras más comunes en este caso son soluciones basadas en cadenas de texto o archivos CSV y XML.

Una vez que el contenido ha sido extraído y transformado en un conjunto ordenado, es posible realizar un análisis de la información de una forma más eficaz y sencilla que aplicando el método tradicional. El proceso descrito se resume en la ilustración 2.1.

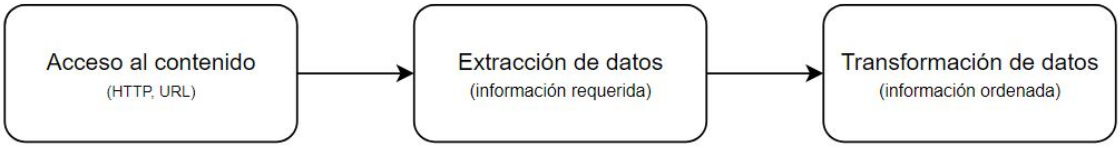

Figura 2.1: Fases del web scraping

Por otro lado, a lo largo del apéndice A, se ilustra el funcionamiento del agente software en cada una de las fases descritas anteriormente, así como de su comportamiento con el servidor web al que se desea acceder.

2.1.2 Herramientas software disponibles

El software disponible que emplean los web scrapers puede dividirse en varios enfoques, ya sean bibliotecas de programación de propósito general, frameworks, o entornos de escritorio.

Por lo general tanto los frameworks como los entornos de escritorio presentan una solución más sencilla e integradora con respecto a bibliotecas de programación. Esto es debido a que ambos, no se ven afectados a los posibles cambios HTML de los recursos a los que accede. Además, estas necesitan la integración de otras múltiples bibliotecas adicionales para el acceso, análisis y extracción del contenido.

Este trabajo se desarrolla sobre bibliotecas de programación, las cuales se implementan como un programa software convencional utilizando estructuras de control y de datos del propio lenguaje. Por lo general, bibliotecas como *curl [7]* conceden acceso al sitio web deseado haciendo uso del protocolo HTTP, mientras que los contenidos extraídos se analizan a través de funciones como la coincidencia de expresiones regulares y la tokenización.

Comprender como las bibliotecas obtienen los datos de los sitios web, pasa por conocer las diferentes formas en las que los documentos HTML se organizan. Existen dos técnicas, dependiendo si se realiza un renderizado previo o no [8]. La primera técnica consiste simplemente en parsear, es decir, realizar un análisis léxico-sintáctico sobre estructuras XML o HTML. Se suelen emplear tanto heurísticas determinadas, como expresiones XPath o selectores CSS para su realización. Por otro, lado si es necesario que parte de la lógica del sitio web pase al lado del cliente, este deberá pasar por un proceso de renderizado previo

Con el paso del tiempo, cada vez se extiende más el uso de bibliotecas de desarrollo como JQuery, encargadas de pasar parte de la lógica del lado del servidor al lado del cliente con el objetivo de favorecer la interactividad. Estas páginas no podrán ser analizadas si no se renderizan antes.

2.1.2.1 XPath

XPath es un lenguaje que permite construir expresiones que recorren y procesan un documento XML [9]. Puede utilizarse para navegar por documentos HTML, ya que este es un lenguaje similar en cuanto a estructura a XML. *XPath* es comúnmente utilizado en el web scraping para extraer datos de documentos HTML, además utiliza la misma notación empleada en las URL para navegar por la estructura del sitio web en cuestión.

2.1.2.2 Selectores CSS

El segundo método de extracción de datos en documento HTML se realiza a través de lo que se conoce como selectores CSS [10]. CSS es el lenguaje utilizado para dar estilo a los documentos HTML, por otro lado, los selectores son patrones que se utilizan para hacer coincidir y extraer elementos HTML basados en sus propiedades CSS.

Hay múltiples sintaxis de selector diferentes, estas se corresponden con la forma en la que el documento CSS está estructurado. En el fragmento de código A.3 se hace uso de los selectores *'.text-primary'* y *'.lister-item-header a'* para acceder al contenido web deseado.

2.1.3 Tipologias

Dependiendo de como se acceda y extraiga la información, existen dos técnicas de web scraping. Se mencionan los siguientes supuestos a continuación:

- Si la información que se almacena no procede de sitios web concretos, sino que durante el análisis de páginas web se encuentran enlaces que retroalimentan el análisis de otras nuevas, el método se conoce como web crawling [11].
- Si la información se extrae de sitios web concretos, donde ya se conoce como extraer y generar un valor por la información, la técnica se conoce como web scraping genérico. Mientras que en el web crawling el resultado de ejecución es la obtención de nuevas páginas, en el web scraping el resultado es la propia información.

Es decir, la principal diferencia entre ambas, es que mientras los web scrapers extraen información de páginas webs concretas, los web crawlers almacenan y acceden a las páginas a través de los enlaces contenidos en las mismas. En la figura 2.1 se mostraba la arquitectura en fases de un web scraper, veamos a continuación como es la arquitectura de un web crawler.

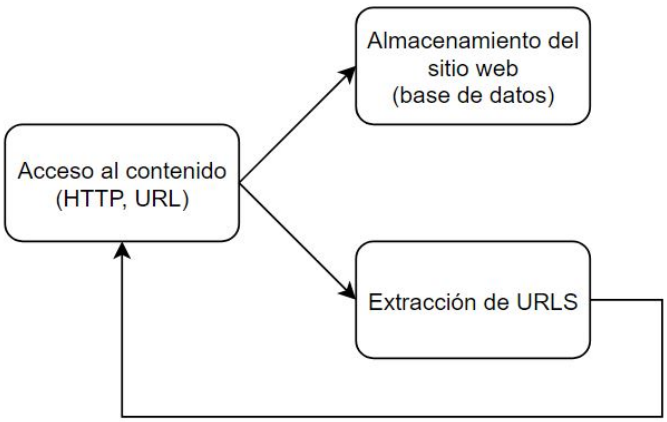

Figura 2.2: Fases del web crawling

Ya sea empleando cualquiera de estas dos tipologías, existen páginas que no pueden ser analizadas o rastreadas. Esto es debido a que algunos sitios web solo están disponibles con una autorización previa, o necesitan información especial para su acceso.

2.2 Posibilidades prácticas del web scraping

Son muchas las aplicaciones prácticas de la minería web, la mayoría de estas entran en el ámbito de la ciencia de los datos. En la siguiente lista se exponen algunos casos de su uso en la vida real [12]:

- Bancos y otras instituciones financieras utilizan minería web para analizar a la competencia. Inversores también utilizan web scraping, para hacer un seguimiento de artículos de prensa relacionados con los activos de su cartera.
- En las redes sociales se emplea minería de datos para conocer la opinión de la gente a cerca de un determinado tema.
- Existen aplicaciones capaces de analizar diferentes sitios web y encontrar los mismos productos a precio reducido. Incluso capaces de detectar ofertas de artículos a tiempo récord.
- etcétera

El minado web contiene infinidad de aplicaciones, muy diversas e interesante capaces de automatizar el trabajo y conseguir la información de forma ordenada.

No obstante, muchos sitios web ofrecen alternativas como el uso de APIs o ficheros estructuras para el acceso a dichos datos. En general el web scraping es una técnica que consume bastantes recursos, por lo que el desarrollador debe limitar su uso si existen otras alternativas, como las APIs, que proporcionan los mismos resultados.

2.3 Retos del web scraping

La forma en la que se crean los sistemas de extracción web se ha discutido desde diferentes perspectivas a lo largo del tiempo. Se emplean métodos científicos como el aprendizaje automático, la lógica o el procesamiento del lenguaje natural para lograr su implementación.

Uno de los principales retos a afrontar tiene relación con las fuentes cambiantes de información. A menudo, una herramienta de extracción tiene que obtener datos de forma rutinaria de una determinada página o sitio web que puede evolucionar con el tiempo. Estos cambios se producen sin previo aviso, son imprevisibles, por lo que es bastante probable que los raspadores web se vean sometidos a cambios. Por ello, surge la necesidad crear herramientas flexibles capaces de detectar y afrontar modificaciones estructurales de las fuentes web relacionadas.

Otros problemas recaen sobre la información extraída. En primer lugar, uno de los aspectos que se deben tener en cuenta al obtener información trata sobre la fiabilidad de la misma. Aunque la información exista y se pueda ser analizada, esta puede que no sea correcta. La gramática y la ortografía pueden ser un problema en la fase de análisis, ya que información puede perderse o ser falsamente recogida. Por otro lado, tanto aplicaciones que tratan con datos personales, como software de minado deben ofrecer garantías de privacidad. Por lo tanto, los posibles intentos de violar la privacidad del usuario deben ser identificados y contrarrestados a tiempo y de forma adecuada.

Puesto que multitud de técnicas de minería web requieren ayuda humana, un primer reto consiste en proporcionar un alto grado de automatización, reduciendo así al máximo el esfuerzo humano. Sin embargo, la ayuda humana puede desempeñar un papel importante a la hora de elevar el nivel de precisión alcanzado por un sistema de extracción de datos web, por ello la clave está en encontrar un equilibrio entre automatización e intervención humana.

Por último, a pesar de que las herramientas de web scraping han evolucionado con el tiempo, los aspectos legales están algo inexplorados pues dependen de los términos y condiciones de cada sitio web en cuestión.

2.4 Aspectos ético-legales del web scraping

Para comprender los aspectos legales del web scraping, debemos recordar el robot o agente software definido en el apartado 2.1.1. Este agente software, previamente examinado por el servidor, es el que se encarga de acceder y realizar un recorrido por el contenido web.

Durante el acceso al contenido, se espera que este agente se ajuste a los términos de uso del sitio en cuestión, así como el cumplimiento del archivo *'robots.txt'* [1], con el objetivo de evitar accesos no deseados y sobrecargas en el servidor.

Puesto que el documento *'robots.txt'* no es de obligado cumplimiento, a lo largo de los años la reputación del web scraping ha decrecido de forma significativa. Muchos agentes software no siguen las indicaciones determinadas, por lo que definir la cantidad de accesos y archivos a los que se accede dependerá de la ética de cada desarrollador.

Con el objetivo de tener una cierta garantía de que nuestro agente software cumple con los aspectos ético-legales, se deben tener en consideración las siguientes cuestiones [14]:

- Leer los términos de uso de la página web en al que se vaya a realizar el minado.
- Inspeccionar y cumplir con el documento robots.txt, para ser capaces de identificar los accesos del servidor.
- Realizar peticiones al servidor de forma controlada. Puede que el índice de solicitudes al servidor no este especificado en el documento, si esto sucede debemos determinar un número de solicitudes razonable, por ejemplo, una solicitud por segundo.

[1] Archivo alojado en el servidor web, que gestiona el tráfico del mismo e indica los documentos a los que no se debe acceder de forma automática [13].

Capítulo 3

Paquetes de programación orientados al web scraping

3.1 Búsqueda de paquetes destinados al web scraping

Como se especificó en la sección 1.3 este trabajo se limitará a realizar una comparativa de los programas software de minado web más frecuentes. Esta comparativa se efectúa con el fin de conocer cuál o cuáles de estos programas o paquetes software son los más rentables para este propósito.

¿Como saber que paquetes software destinados al minado web son los más comunes? Durante todo este capítulo se procederá a la búsqueda, selección e introducción de programas software empleados para el web scraping. Cabe destacar que Python y R serán los lenguajes de programación con los que se trabajará tanto para el desarrollo de la herramienta de comparación, como para el proceso de extracción de datos.

El primer paso consiste en buscar todos los elementos posibles que conforman la población de paquetes software del mercado, ya sean de Python o R. La búsqueda de estos paquetes se ha realizado a través de las distintas fuentes de información mostradas a continuación:

1. GitHub [15]. Gran parte de los desarrolladores de estos programas, publican su trabajo en estos repositorios de código abierto.
2. CRAN [16]. The Comprehensive R Archive Network es una red de servidores ftp y web que almacena versiones idénticas de código y documentación para R.
3. PyPi [17]. Python Package Index es un repositorio de software para Python, útil para la búsqueda de paquetes de un determinado propósito.

3.2 Bibliotecas de Python encontradas durante el proceso de búsqueda

A continuación, se hará una breve sinopsis de los paquetes encontrados. Esta introducción tiene como objetivo conocer el funcionamiento de los paquetes, cuáles son sus funciones y como es el proceso de extracción. Es posible que algunos de los paquetes hayan sido desarrollados tanto en R como en Python. En ese caso, por un lado, la introducción se realzará de forma conjunta, sin embargo, será interesante ver el código del mismo en ambos casos y como se comportan estos ante los distintos test de evaluación.

3.2.1 Beautiful Soup

Beautiful Soup [18] es una de las librerías de Python más comunes en el ámbito del web scraping, está diseñada para extraer datos de documentos XML y HTML.

La técnica que emplea este algoritmo consiste en la creación de un árbol de análisis, donde se disponen todos los elementos del documento como nodos del propio árbol. Una vez disponible, distintos analizadores pueden navegar, buscar y modificar este con el fin de obtener la mayor cantidad posible de información.

Beautiful Soup tiene a html.parse como analizador estándar de documentos HTML, pero también admite varios analizadores de terceros. En la tabla 3.1 se muestran los distintos analizadores disponibles, así como un pequeño resumen de las ventajas y desventajas de estos.

Tipo de analizador	Forma de uso	Ventajas	Desventajas
html	bs(markup, "html.parser")	Notablemente rapido	Mas lento que lxml
lxml html	bs(markup, "lxml")	Muy rapido	Dependencia de C
lxml xml	bs(markup, "xml")	Muy rapido y soporta xml	Dependencia de C
html5lib	bs(markup, "html5lib")	Analiza igual que un buscador	Muy lento

Tabla 3.1: Analizadores disponibles en Beautiful Soup

El empleo de distintos analizadores supondrá una importancia menor si se aplica sobre documentos bien formados, pues la solución aportada presentará la misma estructura que el propio documento original. En caso contrario, el uso de diferentes analizadores creará diferentes soluciones para un mismo documento. Veamos algún ejemplo.

Se emplea el analizador lxml sobre un documento HTML sencillo pero con erratas. Vemos como la solución aportada propone la inclusión de nuevas etiquetas *<head>* y *<body>*, sin embargo, ¿qué ha ocurrido con la etiqueta * *?.

```
> BeautifulSoup("<a></p>", "lxml")
# <html><body><a></a></body></html>
```

En lugar de ignorar la etiqueta * *, el analizador html5lib la empareja con una etiqueta * * de apertura. También añade una etiqueta <head>que el analizador lxml habia obviado.

```
> BeautifulSoup("<a></p>", "html5lib")
# <html><head></head><body><a><p></p></a></body></html>
```

Al igual que lxml, html.parse ignora la etiqueta de cierre * *. Podemos observar que este analizador ni siquiera intenta crear un documento HTML bien formado añadiendo etiquetas *<html>* o *<body>*.

```
> BeautifulSoup("<a></p>", "html.parser")
# <a></a>
```

Como vemos diferentes analizadores crearan diferentes soluciones en caso de que el documento a analizar no este bien formado. Por ello, si deseamos analizar múltiples documentos de los que no conocemos su origen o estructura sería deseable especificar el tipo de analizador con el fin del obtener la solución deseada.

En cuanto al proceso de extracción que se emplea en este algoritmo es sencillo. En primer lugar, el documento ya sea HTML o XML se convierte al completo en caracteres Unicode. Tras ello se crea un

árbol de objetos donde cada uno de ellos representa una etiqueta o tag del documento HTML ('<body>, '<p>', ...). Finalmente, un analizador especificado por parámetro, recorre el árbol buscando las partes del mismo que se desean.

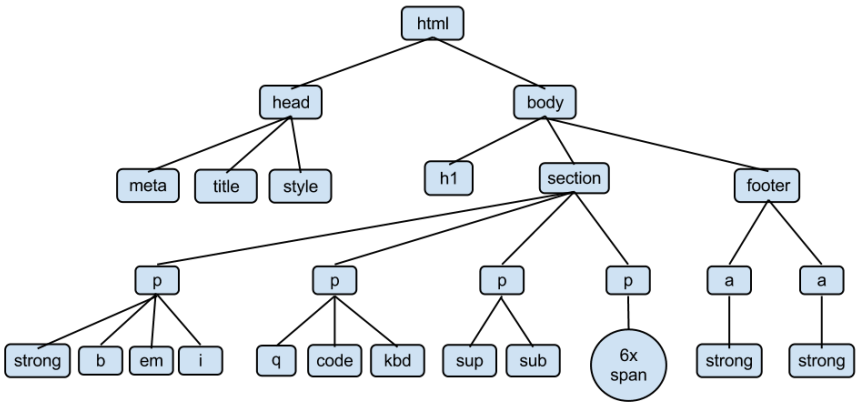

Figura 3.1: Arbol de objetos de Beautiful Soup[1]

Este algoritmo, no solo permite el recorrido automático del árbol en busca del texto del documento al completo, sino que permite además la posibilidad de recorrer el mismo de forma manual, por lo que es posible acceder a todos los objetos del árbol empleando métodos de navegación como *soup.head, soup.parent, soup.next_sibling, ...*

3.2.2 JusText

JusText [19] es una herramienta para eliminar el contenido repetitivo, como los enlaces de navegación, encabezados y pies de página de las páginas HTML. Está diseñado para preservar principalmente el texto que contiene frases completas.

La técnica que emplea este algoritmo consiste en lo que se conoce como segmentación. La idea es formar bloques de texto dividiendo la página HTML en etiquetas. Una secuencia de dos o más etiquetas como *
*, * , ...*, separaría los bloques.

Aunque no sea habitual, puede ocurrir que el contenido de estos bloques no sea homogéneo, es decir, que dentro de un mismo bloque haya una mezcla de información importante con contenido basura. La forma en la que JusText separa el contenido basura, denominado boilerplate, del contenido valioso se basa en el volumen de palabras con sentido gramatical. Aplicando esta heurística es posible hacer una distinción entre bloques como la que sigue:

- Los bloques cortos que contienen un enlace son casi siempre del tipo boilerplate.
- Los bloques que contienen muchos enlaces son casi siempre del tipo boilerplate.
- Los bloques largos que contienen texto gramatical forman parte del contenido valioso, mientras que todos los demás bloques largos son casi siempre del tipo boilerplate.
- Tanto los bloques buenos como los bloques de tipo boilerplate tienden a crear grupos, es decir, un bloque boilerplate suele estar rodeado de otros bloques de su mismo tipo y viceversa.

La dificultad real está en decidir cuando un texto es gramatical y cuando no. Aunque puede ser complicado, JusText emplea una simple heurística basada en el volumen de palabras con sentido gramatical.

Mientras que un texto gramatical suele contener un cierto porcentaje de este tipo de palabras, como las listas y enumeraciones, los contenidos de tipo boilerplate suelen carecer de ellas. En definitiva, los bloques largos y algunos bloques cortos pueden clasificarse con una confianza muy alta. El resto de bloques cortos pueden clasificarse observando los bloques circundantes.

Con el fin de facilitar el trabajo heurístico, JusText realiza un preprocesamiento del documento HTML. Durante esta fase, se elimina el contenido de las etiquetas *<header>*, *<style>* y *<script>*. Ademas, el contenido de etiquetas como *<select>* se clasifica inmediatamente como tipo boilerplate. Lo mismo ocurre con los bloques que contienen el símbolo de copyright ©.

Tras el proceso de segmentación y preprocesamiento se procede a la clasificación de bloques, donde a cada uno de estos bloques se le asigna una clase dependiendo de su naturaleza:

1. Malo: bloques de tipo boilerplate.
2. Bueno: bloques pertenecientes al contenido principal.
3. Corto: bloques demasiado cortos para tomar una decision sobre su naturaleza.
4. Casi bueno: bloques intermedios entre cortos y buenos.

¿Qué algoritmo sigue JusText para determinar esta clasificación? En el fragmento de código 3.1 se muestra cuál es el proceso de dicha clasificación para cada una de los cuatro tipos distintos de bloques.

Listado 3.1: JusText - Algoritmo de clasificación

```
if link_density > MAX_LINK_DENSITY:
    return 'bad'

# short blocks
if length < LENGTH_LOW:
    if link_density > 0:
        return 'bad'
    else :
        return 'short'

# medium and long blocks
if stopwords_density > STOPWORDS_HIGH:
    if length > LENGTH_HIGH:
        return 'good'
    else :
        return 'near-good'

if stopwords_density > STOPWORDS_LOW:
    return 'near-good'
else :
    return 'bad'
```

Podemos observar que se definen dos tipos de variables, la densidad y la longitud. Mientras que la longitud es el número de caracteres de bloque, la densidad se define como la proporción de caracteres o palabras dentro de una una etiqueta de tipo *<a>*, o una lista de parada.

El algoritmo toma como parámetros dos enteros definidos como LENGTH_LOW(70) y LENGTH_HIGH(200), ademas de tres números de coma flotante, MAX_LINK_DENSITY(0.2), STOPWORDS_LOW(0.3) y STOPWORDS_HIGH(0.32). Los dos primeros establecen los umbrales para dividir los bloques por su longitud en tres tipos, cortos, medianos y largos. Los dos últimos dividen los bloques por la densidad de palabras de parada en bajos, medianos y altos.

La distinción entre bloques es muy sencilla, hasta ahora JusTest ha sido capaz de clasificar los bloques buenos y malos, pero ¿qué ocurre con los bloques cortos y los bloques casi buenos? JusText reclasifica este tipo de bloques en función de los bloques circundantes. Los bloques ya clasificados como buenos o malos sirven como piedras base en esta etapa y su clasificación se considera fiable, por lo que nunca se modifica. Esta reclasificación se puede ver resumida en la imagen 3.2.

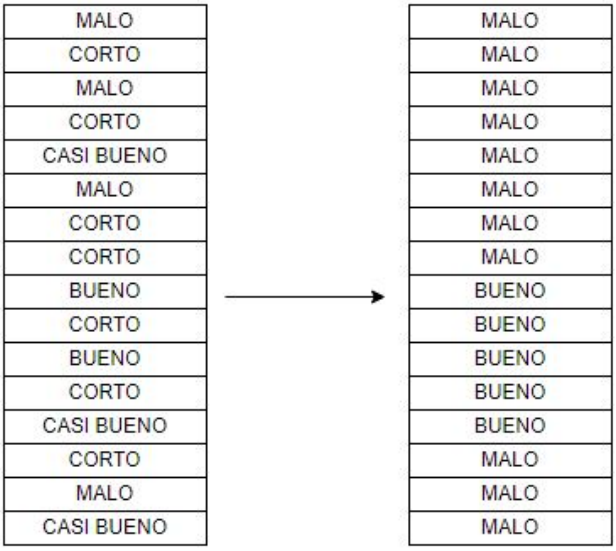

Figura 3.2: JusText - Reclasificación de bloques

La idea que subyace a la reclasificación es que los bloques 'boilerplate' suelen estar rodeados de otros bloques 'boilerplate' y viceversa. Los bloques casi buenos suelen contener datos útiles del corpus si se encuentran cerca de bloques buenos. Los bloques cortos suelen ser útiles sólo si están rodeados de bloques buenos por ambos lados.

3.2.3 Inscriptis

Inscriptis [20] es una biblioteca de conversión de HTML a texto basada en Python. Esta librería es especialmente adecuada para aplicaciones que requieran representaciones de texto con una alta precisión y calidad del contenido HTML.

A diferencia de otros algoritmos de extracción Inscriptis no solo tiene en cuenta el texto extraído, sino que además la estructura del mismo es muy importante, por lo que este texto extraído podría acercarse más al obtenido por cualquier navegador web.

Veamos una pequeña comparación entre la extracción de texto de Beautiful Soup 3.2.1, con la extracción de texto de Inscriptis, donde se tiene un fragmento HTML como el siguiente, como objeto de prueba.

```
<ul>
    <li>first</li>
    <li>second</li>
<ul>
```

Si aplicásemos Beautiful Soup sobre este fragmento HTML el texto extraído sería algo como *'firstsecond'*, donde no se tiene en cuenta el diseño ni la estructura del documento. Sin embargo, si aplicamos Inscriptis sobre el mismo fragmento HTML, la salida obtenida sería la siguiente:

```
* first
* second
```

Inscriptis, no solo admite construcciones tan simples como la anterior, permite analizar construcciones mucho más complejas, como las tablas anidadas, y subconjuntos de atributos HTML y CSS donde es esencial una conversión precisa de HTML a texto.

La técnica que emplea Inscriptis se conoce como reglas de anotación, es decir, mapeos que permiten anotar el texto extraído. Estas anotaciones se basan en la información estructural y semántica codificada en las etiquetas y atributos HTML utilizados para controlar la estructura y diseño del documento original.

Con el fin de asignar etiquetas y/o atributos HTML a las anotaciones, se emplea un algo parecido a un diccionario. En el diccionario mostrado a continuación, la etiqueta *<h1>* produce las anotaciones heading y h1, una etiqueta * * con una clase que contiene el valor toc da como resultado la anotación table-of-contents, y todas las etiquetas con un atributo cite se anotan con citation.

```
{
  "h1": ["heading", "h1"],
  "h2": ["heading", "h2"],
  "b": ["emphasis"],
  "div#class=toc": ["table-of-contents"],
  "#class=FactBox": ["fact-box"],
  "#cite": ["citation"]
}
```

Imaginemos que se dispone de un documento HTML como el que sigue y unas reglas de anotación determinadas.

```
<h1>Chur</h1>
<b>Chur</b> is the capital and largest town of the Swiss canton of the Grisons
and lies in the Grisonian Rhine Valley.
```

En este ejemplo, la salida habitual según el diccionario anterior debería ser una etiqueta de cabecera y otra de énfasis, veamos la salida que proporciona el proceso de asignación.

```
{"text": "Chur\n\nChur is the capital and largest town of the Swiss canton
      of the Grisons and lies in the Grisonian Rhine Valley.",
"label": [[0, 4, "heading"], [0, 4, "h1"], [6, 10, "emphasis"]]}
```

Como era de esperar la obtención del texto es precisa, pero no solo del texto sino de su estructura. Además, la asignación de etiquetas también se ha realizado de forma correcta.

3.2.4 Goose3

Goose [21] es una librería de programación cuyo objetivo es la extracción de texto en artículos. Inicialmente, el proyecto fue iniciado con Java, pero han surgido nuevas versiones en diferentes lenguajes de programación como Scala o Python.

El objetivo del software es tomar cualquier artículo de noticias o página web de tipo artículo y no solo extraer lo que es el cuerpo principal, sino también se extraen todos los metadatos e imágenes del mismo. Goose extraerá: (1) texto principal del artículo, (2) imagen principal del artículo, (3) videos introducidos en el artículo, (4) descripción y (5) meta tags.

La técnica que emplea Goose3 para la extracción de texto es muy similar a la que se emplea en Beautiful Soup 3.2.1, donde un árbol se recorre con el fin de obtener la mayor cantidad de información valiosa posible.

Listado 3.2: Goose3 - Algoritmo de extracción

```
def childNodesWithText(cls, node):
    root = node
    if root.text:
        elm = lxml.html.HtmlElement()
        elm.text = root.text
        elm.tag = 'text'
        root.text = None
        root.insert(0, elm)
    for _, elm in enumerate(list(root)):
        idx = root.index(elm)
        if elm.tag == 'text':
            continue
        if elm.tail:
            tmp = cls.createElement(tag='text', text=elm.tail, tail=None)
            root.insert(idx + 1, tmp)
    return list(root)
```

Si observamos el algoritmo 3.2, si se determina que en un cierto nodo del árbol contiene texto, se crea un nuevo nodo y este se inserta en una lista. Con este proceso todos los nodos que contienen información valiosa son agrupados y almacenados para su posterior uso.

Al igual que con Beautiful Soup, Goose puede emplear distintos analizadores, html de lxml o el conocido como soup parser de lxml. Pero no solo es posible configurar el analizador, también se puede determinar el agente de usuario que se empleará para la extracción.

```
> g = Goose({'browser_user_agent': 'Mozilla', 'parser_class':'soup'})
```

3.2.5 News-please

News-please [22] es un web scraper y web crawler de código abierto escrito en Python desarrollado para cumplir cinco requisitos: (1) extracción de noticias de cualquier sitio web, (2) extracción completa del sitio web, (3) alta calidad de la información extraída, (4) facilidad de uso y (5) mantenibilidad.

A diferencia de otros web scrapers ya mencionados anteriormente, news-please emplea herramientas ya existentes, las cuales se amplían con nuevas funcionalidades con el objetivo de cumplir con los requisitos señalados.

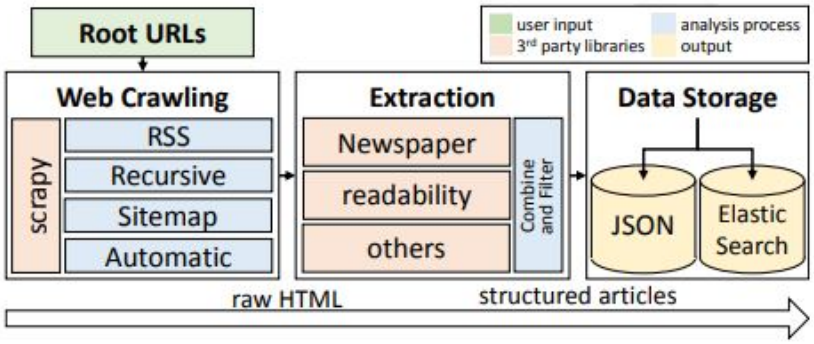

Figura 3.3: News-please - Proceso de scraping y crawling

En cuanto al proceso de web crawling, news-please lo divide en dos subtareas, en primer lugar se descarga el documento HTML empleando el framework *Scrapy*. En segundo lugar, con el objetivo de encontrar todos los artículos publicados en dicho documento se admiten cuatro técnicas:

1. RSS [1]: análisis de los canales RSS para encontrar los artículos recientes.
2. Recursiva: seguimiento de los enlaces enlaces internos en las páginas rastreadas.
3. Sitemap [2]: analiza los sitemaps en busca de enlaces a todos los artículos.
4. Automático: prueba los sitemaps, vuelve a ser recursivo en caso de error.

Estos cuatro enfoques también pueden combinarse, por ejemplo, iniciando dos instancias de news-please en paralelo, una en modo automático para obtener todos los artículos publicados hasta el momento y otra instancia en modo RSS para recuperar los artículos recientes.

Para el proceso de web scraping, se emplean herramientas ya existentes en el mercado con el objetivo de obtener la información deseada, es decir, título, párrafo principal, contenido principal, autor, fecha e imagen principal. Por un lado, news-please ha decidido utilizar herramientas como *Newspaper* 3.2.10 o *Readability* 3.2.8 para el proceso de extracción del contenido principal en artículos, mientras que por otro se emplea *Regex* para la extracción de fechas de publicación.

3.2.6 Trafilatura

Trafilatura [23] es un paquete de Python que descarga, analiza y extrae datos de páginas web. Combina dos bibliotecas de web scraping ya existentes, *Readability* 3.2.8 y *JusText* 3.2.2 como redes de seguridad y fallbacks.

La técnica que emplea el algoritmo de extracción de este paquete se fundamenta en una cascada de filtros basados en reglas y heurística de contenido. Veremos que al igual que muchos otros algoritmos de extracción, usa los árboles de objetos como estructura de datos.

[1] formato XML para distribuir contenido en la web
[2] Un sitemap es un archivo que enumera las URL visibles de un determinado sitio, el objetivo principal es revelar dónde pueden buscar contenido las máquinas.

En primer lugar, se realiza lo que se conoce como delimitaciones de contenido, donde mediante expresiones XPath dirigidas y atributos HTML comunes, se excluyen partes no deseadas del código HTML creando un árbol de objetos. Las mismas operaciones se realizan para los comentarios en caso de que formen parte de la extracción.

Listado 3.3: Trafilatura - Delimitación de contenido

```
# clean + use LXML cleaner
tree = tree_cleaning(tree, inc_tables, inc_images)

# convert tags, the rest does not work without conversion
tree = convert_tags(tree, inc_formatting, inc_tables, inc_images, inc_links)

# comments first, then remove
if inc_comments is True:
    tree = extract_comments(tree, deduplicate, config)
else:
    if favor_precision is True:
        tree = prune_unwanted_nodes(tree, REMOVE_COMMENTS_XPATH)
```

Si se detecta que la extracción realizada ha sido posiblemente defectuosa, se ejecuta lo que se conocen como algoritmos de respaldo. Estos aplican una heurística basada en la longitud de las líneas, la relación texto/marcado y la posición/profundidad de los elementos en el árbol HTML. Una vez aplicados estos algoritmos, su resultado se compara con la extracción 'casera' y se determina la extracción más eficaz, sobre todo en términos de longitud e impurezas.

Por último, es caso de que ambas soluciones provoquen una salida defectuosa, se ejecuta una extracción base con el fin de buscar elementos de texto 'salvajes' que probablemente se hayan pasado por alto. Esto implica la búsqueda de cualquier elemento con contenido textual útil.

Listado 3.4: Trafilatura - Algoritmo de respaldo y extraccion base

```
postbody, len_text, sure_thing = extract_content(tree, ...)
# compare if necessary
if no_fallback is False:
    postbody, temp_text, len_text = compare_extraction(tree, ...)
    # add baseline as additional fallback
    if len(postbody) == 0:
        postbody, temp_text, len_text = baseline(filecontent)
# rescue: try to use original/dirty tree
elif sure_thing is False and len_text < MIN_EXTRACTED_SIZE:
    postbody, temp_text, len_text = baseline(filecontent)
```

Como resultado de toda esta consecucion de algoritmos se obtienen los textos principales y los posibles comentarios de los documento HTML analizados, con la posibilidad de conservar elementos estructurales como párrafos, títulos, listas, comillas, código, saltos de línea, formato de texto en línea... En la extraccion, también se incluyen metadatos, es decir, título, nombre del sitio, autor, fecha, categorías y etiquetas.

3.2.7 Dragnet

Dragnet [24] es un web scraper escrito en Python y construido sobre el entorno de cálculo numérico 'numpy/scipy/Cython'. Como hemos visto hasta ahora, el objetivo de cualquier algoritmo de extracción es separar el contenido principal del contenido de navegación, los bloques de publicidad, los avisos de copyright y similares en las páginas web. La mayoría de estos algoritmos emplean heurísticas determinas para este fin. Dragnet se encarga de realizar esta separación de contenido empleando el aprendizaje automático.

El algoritmo comienza dividiendo el documento HTML en una secuencia de bloques utilizando el DOM [3] y un conjunto específico de etiquetas como * *, *<p>* o *<h1>*, que modifican la estructura del propio documento. Iterando por el DOM, cada vez que el algoritmo se encuentra alguna de las etiquetas mencionadas anteriormente, se crea un nuevo bloque. Aquellos bloques sin contenido de valor, son descartados.

Una vez separado el documento en bloques, se deben asignar un conjunto de características a cada uno. Estas serán útiles en un clasificador con el objetivo de predecir el contenido con valor, del contenido 'boilerplate' a nivel de bloque. Cualquier bloque con más del 10 % de los tokens extraídos se etiqueta como contenido valioso.

La primera característica consiste en la densidad de texto y enlaces. La intuición es que los bloques de contenido tienen una mayor densidad de texto y una menor densidad de enlaces que los bloques denominados boilerplate, ya que muchos de estos ultimos son bloques que consisten en breves fragmentos de palabras o son principalmente texto ancla.

El segundo tipo de característica está diseñada heurísticamente para capturar información semántica en el código HTML que dejan los programadores. Muchos atributos como *id* o *class*, y etiquetas HTML incluían tokens tipo *comment*, *header* y *nav*. Estos nombres descriptivos son utilizados por los programadores cuando escriben CSS y Javascript y, puesto que se eligen con el objetivo de que tengan sentido para el programador, incorporan cierta información semántica sobre el contenido del bloque.

Token	Content : No Content	Percent of blocks
menu	1 : 372.6	2.2 %
widget	1 : 314.1	4.6 %
nav	1 : 68.9	3.3 %
facebook	1 : 18.3	1.3 %
top	1 : 13.3	1.9 %
twitter	1 : 8.5	2.3 %
title	1 : 3.3	10.5 %
header	1 : 2.9	3.7 %
comment	3.2 : 1	21.3 %
author	4.9 : 1	7.9 %
thread	5.0 : 1	3.0 %
avatar	42.1 : 1	3.2 %

Tabla 3.2: Comparación de tokens en el atributo *class*

En la tabla 3.2 se enumeran algunos tokens seleccionados en el atributo de clase junto con sus ratios de probabilidad de contenido y no contenido. Los tokens de la parte superior de la tabla tienen más probabilidades de aparecer en bloques sin contenido, mientras que los de la parte inferior tienen más probabilidades de aparecer en bloques con contenido. Para una mejor visualización, se forman dos grupos, según estén asociados a bloques de no contenido o de contenido.

[3] El DOM define la manera en que objetos y elementos se relacionan entre sí en el navegador y en el documento. [25]

Finalmente, la tercera característica incluye varias ideas. Por un lado, la relación entre la longitud del texto y el número de etiquetas HTML tiende a ser mayor en los bloques de contenido. En segundo lugar, las secciones de la página que no son de contenido tienden a estar agrupadas, de modo que la diferencia en la proporción de contenido y etiquetas de un bloque a otro tiende a ser pequeña en las regiones que no son de contenido. La idea final es combinar la proporción de etiquetas de contenido y la diferencia en la proporción de etiquetas de contenido en un enfoque de agrupación de k-means no supervisado, de modo que los bloques sin contenido se agrupen naturalmente cerca del origen.

3.2.8 Readability

3.2.9 HTML-Text

HTML-Text [26] es una librería software destinada a la extracción de texto en documentos HTML. Las principales diferencias entre emplear expresiones XPath *.xpath('//text()')* o emplear otros algoritmo de extracción como Beautiful Soup *.get_text()* son: (1) Por un lado, el texto extraído es limpio, ya que no contiene estilos en línea, javascript o comentarios los normalmente no son visibles para los usuarios, (2) en segundo lugar, se normalizan los espacios en blanco de una forma más inteligente que empleando *.xpath('normalize-space())*, (3) por último, se añaden nuevas líneas para que el texto de salida se parezca más a cómo se representa en los navegadores.

3.2.10 Newspaper3k

3.3 Paquetes de R encontrados durante el proceso de búsqueda

3.4 Paquetes seleccionados para el proceso de análisis

Capítulo 4

Selección de variables de análisis y proceso de estudio

Capítulo 5

Presupuesto

Blah, blah, blah.

Bibliografía

[1] "Python simple crawling using beautifulsoup," https://medium.com/milooproject/python-simple-crawling-using-beautifulsoup-8247657c2de5. [Ultimo acceso 08/diciembre/2021].

[2] Wikipedia, "Web scraping," https://es.wikipedia.org/wiki/Web_scraping. [Ultimo acceso 18/octubre/2021].

[3] B. Zhao, *Web Scraping*, 05 2017, pp. 1–3.

[4] Wikipedia, "Interfaz de programación de aplicaciones," https://en.wikipedia.org/wiki/API. [Ultimo acceso 18/octubre/2021].

[5] O. Castrillo-Fernández, *Web Scraping: Applications and Tools*, ser. Report No. 2015 / 10. European Public Sector Information Platform, 2015.

[6] D. Glez-Peña, A. Lourenco, H. López-Fernández, M. Reboiro-Jato, and F. Fdez-Riverola, "Web scraping technologies in an API world," *Briefings in Bioinformatics*, vol. 15, no. 5, pp. 788–797, April 2013.

[7] CRAN, "curl: A modern and flexible web client for r," https://cran.r-project.org/web/packages/curl/index.html. [Ultimo acceso 26/octubre/2021].

[8] D. Francisco López, "Revisión de los paquetes para realizar web scraping en r: Análisis cualitativo y cuantitativo," Master's thesis, Universidad de Alcalá Escuela Politécnica Superior, 2018.

[9] Wikipedia, "Xpath," https://es.wikipedia.org/wiki/XPath. [Ultimo acceso 04/noviembre/2021].

[10] W3Schools, "Css selector reference," https://www.w3schools.com/cssref/css_selectors.asp. [Ultimo acceso 04/noviembre/2021].

[11] Wikipedia, "Web crawler," https://en.wikipedia.org/wiki/Web_crawler. [Ultimo acceso 02/noviembre/2021].

[12] S. Broucke Vande and B. Baesens, *Web Scraping for Data Science with Python*, 11 2017, pp. 14–16.

[13] Google, "Introducción a los archivos robots.txt," https://developers.google.com/search/docs/advanced/robots/intro?hl=es. [Ultimo acceso 27/octubre/2021].

[14] M. Beckman, S. Guerrier, J. Lee, R. Molinari, S. Orso, and I. Rudnytskyi, "An introduction to statistical programming methods with r, chapter 10," https://smac-group.github.io/ds/index.html. [Ultimo acceso 27/octubre/2021].

[15] "Github," https://github.com/. [Ultimo acceso 07/diciembre/2021].

[16] "Cran," https://cran.r-project.org/. [Ultimo acceso 07/diciembre/2021].

[17] "Pypi," https://pypi.org/. [Ultimo acceso 07/diciembre/2021].

[18] "Beautiful soup documentation," https://www.crummy.com/software/BeautifulSoup/bs4/doc/. [Ultimo acceso 08/diciembre/2021].

[19] M. Belica, "Justext," https://github.com/miso-belica/jusText. [Ultimo acceso 09/diciembre/2021].

[20] A. Weichselbraun, "Inscriptis - a python-based html to text conversion library optimized for knowledge extraction from the web," *Journal of Open Source Software*, vol. 6, no. 66, p. 3557, 2021. [Online]. Available: https://doi.org/10.21105/joss.03557

[21] M. Lababidi, "Goose3," https://goose3.readthedocs.io/en/latest/. [Ultimo acceso 11/diciembre/2021].

[22] F. Hamborg, N. Meuschke, C. Breitinger, and B. Gipp, "news-please: A generic news crawler and extractor," in *Proceedings of the 15th International Symposium of Information Science*, March 2017, pp. 218–223.

[23] A. Barbaresi, "Trafilatura: A Web Scraping Library and Command-Line Tool for Text Discovery and Extraction," in *Proceedings of the Joint Conference of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, 2021, pp. 122–131. [Online]. Available: https://aclanthology.org/2021.acl-demo.15

[24] M. E. Peters and D. Lecocq, "Content extraction using diverse feature sets," in *Proceedings of the 22nd International Conference on World Wide Web*, ser. WWW '13 Companion. New York, NY, USA: Association for Computing Machinery, 2013, pp. 89–90. [Online]. Available: https://doi.org/10.1145/2487788.2487828

[25] Wikipedia, "Document object model," https://es.wikipedia.org/wiki/Document_Object_Model. [Ultimo acceso 14/diciembre/2021].

[26] M. Korobov, "Html-text," https://github.com/TeamHG-Memex/html-text. [Ultimo acceso 15/diciembre/2021].

[27] CRAN, "robotstxt: A 'robots.txt' parser and 'webbot'/'spider'/'crawler' permissions checker," https://cran.r-project.org/web/packages/robotstxt/index.html. [Ultimo acceso 28/octubre/2021].

[28] ——, "xml2: Parse xml," https://cran.r-project.org/web/packages/xml2/index.html. [Ultimo acceso 29/octubre/2021].

[29] ——, "rvest: Easily harvest (scrape) web pages," https://cran.r-project.org/web/packages/rvest/index.html. [Ultimo acceso 29/octubre/2021].

[30] ——, "magrittr: A forward-pipe operator for r," https://cran.r-project.org/web/packages/magrittr/index.html. [Ultimo acceso 29/octubre/2021].

Apéndice A

Funcionamiento básico de un web scraper

Siguiendo las directrices determinadas en la sección 2.1.1, se muestra el funcionamiento de un web scraper durante las tres fases definidas. Para ello se realizará un pequeño ejemplo mostrando el comportamiento del mismo y de como interactúa con el servidor web al que se desea acceder.

Para el desarrollo de este ejemplo se han empleado bibliotecas software basadas en el lenguaje de programación R, capaces de extraer datos de cualquier web. En este caso la web sujeta al análisis será, *imdb* encargada de asignar un ranking entre películas y series.

A.1 Fase de búsqueda

Antes de comenzar con la primera de las tres etapas, debemos asegurarnos que nuestro agente software cumple con todos los aspectos ético-legales descritos en la sección 2.4. Los términos y servicios de la página deberán ser leídos, al igual que el documento *'robots.txt'* con el objetivo de conocer cuáles son los accesos disponibles y el índice de solicitudes a realizar.

En el fragmento de código A.1 se muestra la solicitud al servidor realizada. A través de la biblioteca *robotstxt* [27] y haciendo uso de la función *get_robotstxt* se obtiene el documento deseado.

Es posible que la solicitud del documento no se realice correctamente, pues o bien la página no dispone del documento en ese instante, o la función ha fallado durante su solicitud. En cualquiera de estos dos escenarios, es posible realizar una doble comprobación accediendo al mismo a través de la propia URL, *https://www.imdb.com/robots.txt*.

Listado A.1: Solicitud del documento *robots.txt*

```
install.packages("robotstxt")
library("robotstxt")

robots <- get_robotstxt(domain = "https://www.imdb.com/")

head(robots)
```

```
# robots.txt for https://www.imdb.com properties
User-agent: *
Disallow: /OnThisDay
Disallow: /ads/
Disallow: /ap/
Disallow: /mymovies/
Disallow: /r/
Disallow: /register
Disallow: /registration/
...
```

Una vez se ha accedido al documento *'robots.txt'*, conociendo los posibles accesos al servidor, y determinando el número de solicitudes máximas por segundo, es posible acceder y descargar el fichero HTML de forma segura. Para este propósito se empleará la función *read_html* de la biblioteca *xml2* [28] la cual se detalla a continuación.

Listado A.2: Acceso y descarga del archivo HTML

```
install.packages("xml2")

library("xml2")

url <- "imdb.com/search/title/?count=100&release_date=2016,2016&title_type=feature"
html_doc <- read_html(url)
```

El valor que retorna la función *read_html*, se trata del archivo HTML integro, donde se incluyen cabecera, cuerpo y demás etiquetas del mismo. Una vez que se dispone de la página, es posible comenzar con la extracción de datos de interés de la misma.

A.2 Fase de extracción

Para el minado se empleará *rvest* [29], una de las bibliotecas software más comunes en este aspecto, diseñada para trabajar con *magrittr* [30] y facilitar tareas de la extracción. Además, será necesario el uso de funciones como *html_nodes()* y *html_text()*.

Durante esta fase de extracción se obtendrán tanto los títulos como el ranking asignado a cada película o serie. Para realizar la extracción de forma correcta, se deberá conocer la etiqueta HTML que envuelve dicha información.

Listado A.3: Extracción de datos de interés del documento

```
install.packages("rvest")
install.packages("magrittr")

library("rvest")
library("magrittr")

rank_data <- html_nodes(html_doc, '.text-primary') %>%
             html_text()

title_data <- html_nodes(html_doc, '.lister-item-header a') %>%
              html_text()

head(rank_data)
head(title_data)
```

```
[1] "1." "2." "3." "4." "5." "6."

[1] "Animales nocturnos"   "Train to Busan"       "La llegada (Arrival)"
[4] "Escuadrón suicida"    "Deadpool"             "Hush (Silencio)"
```

A.3 Fase de transformación

Una vez los datos han sido extraídos, la última fase consiste en la transformación de los mismos. Los datos desordenados deberán ser convertidos en estructuras de datos ordenadas y coherentes con el fin su posible almacenamiento en una base de datos.

Listado A.4: Transformación de datos en un data frame

```
movies_df <- data.frame(Rank = rank_data, Title = title_data)

head(movies_df)
```

```
  Rank                Title
1   1.   Animales nocturnos
2   2.       Train to Busan
3   3. La llegada (Arrival)
4   4.    Escuadrón suicida
5   5.             Deadpool
6   6.      Hush (Silencio)
```

Una vez los datos han sido ordenados en una estructura de datos propia, en este caso un data frame, es posible trabajar con ellos de forma más cómoda y sencilla.

Universidad de Alcalá
Escuela Politécnica Superior

Universidad
de Alcalá