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 [1].

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 [2]. 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. [3]

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 [1]. 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 [4]"*. 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 [5]. 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.

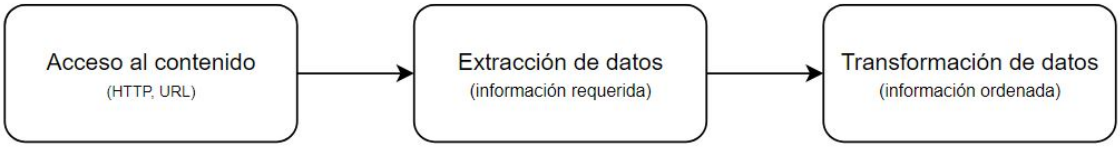

Figura 2.1: Fases del web scraping

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 [6]* 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 [7]. 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 [8]. 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 [9]. 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 [10].
- 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.

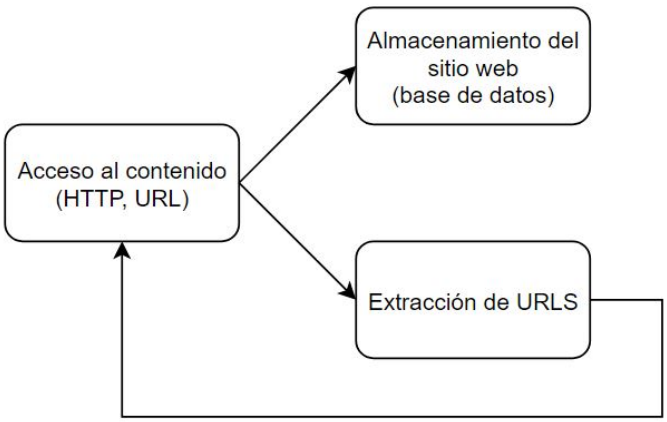

Figura 2.2: Fases del web crawling

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 [11]:

- 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 [13]:

- 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 [12].

Capítulo 3

Librerías de programación orientadas al web scraping

3.1 Búsqueda de librerías destinadas 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 [14]. Gran parte de los desarrolladores de estos programas, publican su trabajo en estos repositorios de código abierto.

2. CRAN [15]. 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 [16]. Python Package Index es un repositorio de software para Python, útil para la búsqueda de paquetes de un determinado propósito.

3.2 Librerías de Python encontradas durante el proceso de búsqueda

A continuación, se hará una breve sinopsis de las librerías encontradas en Python. 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.

Antes de comenzar con la sinopsis de paquetes, es conveniente revisar el apéndice B, donde se realiza una breve introducción de aquellos analizadores empleados en los mismos. Los 'emparejamientos' entre biblioteca-analizador se recogen en la tabla 3.1 a modo de resumen.

htmltext	**inscriptis**	**dragnet**	**boilerpy**	**newspaper**	**newsplease**	**justext**	**goose3**
lxml etree	lxml html	lxml etree	html parser	lxml etree	lxml etree	lxml html	lxml html

readability	**trafilatura**	**beautifulsoup**	**libextract**	**html2text**
lxml html & etree	lxml html & etree	lxml etree & html5lib & html parser	lxml html	html parser

Tabla 3.1: Emparejamientos biblioteca-analizador

Ya sea porque determinadas bibliotecas permiten al desarrollador seleccionar entre varios tipos de analizadores, o bien porque múltiples de ellos son necesarios para el correcto funcionamiento de la extracción, muchas de estas bibliotecas hacen uso de más de un analizador en su código.

3.2.1 Inscriptis

Inscriptis [17] además de ser una biblioteca de conversión de HTML a texto basada en Python, también tiene soporte como línea de comandos o como servicio web para tablas anidadas. A pesar de sus múltiples funcionalidades, a lo largo de esta sección nos centraremos en inscriptis como biblioteca de programación.

3.2.1.1 Estructura de la solución

A diferencia de otros algoritmos de extracción, Inscriptis no solo tiene en cuenta la calidad del texto extraído, la estructura del mismo también es muy importante. Esto provoca que el resultado obtenido se acerque más a un posible resultado aplicando el método tradicional a través de cualquier navegador web.

Veamos una pequeña comparación entre la extracción de texto de Beautiful Soup 3.2.2, con la extracción de texto de Inscriptis, donde se tiene un fragmento HTML como el siguiente como objeto de prueba.

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

Empleamos en primer lugar el metodo *get_text()* propio de Beautiful Soup sobre el fragmento HTML anterior. Veamos cuál es el resultado.

```
# firstsecond
```

Se puede observar que el formato obtenido no es el adecuado, puesto que no se ha respetado la estructura del documento base. Sin embargo, si aplicamos Inscriptis sobre el mismo fragmento HTML, la salida obtenida sería la siguiente:

```
# first
# second
```

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

3.2.1.2 Reglas de anotación

La técnica que emplea Inscriptis se conoce como reglas de anotación, es decir, mapeos que permiten realizar anotaciones sobre 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 lo que se conoce como perfil de anotaciones, algo parecido a un diccionario.

h1	['heading', 'h1']
h2	['heading', 'h2']
b	['emphasis']
div#class=toc	['table-of-contents']
#class=FactBox	['fact-box']
#cite	['citation']

Tabla 3.2: Inscriptis - Perfil de anotaciones

Si observamos el diccionario mostrado en la tabla 3.2, las etiquetas de tipo cabecera producen anotaciones de tipo *hn*, 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 como *citation*.

A modo de ejemplo, y con el fin de mostrar el correcto etiquetado del algoritmo, imaginemos que se dispone el fragmento de un documento HTML como el mostrado a continuación y unas reglas de anotación como las mostradas en la tabla 3.2.

```
<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.
```

A partir de este ejemplo, y basándonos en el diccionario anterior, la salida esperada debería ser una etiqueta de cabecera y otra de énfasis, veamos el resultado 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. Se muestra en el fragmento de código 3.1 como se pueden emplear las reglas de anotación dentro de un programa.

Listado 3.1: Inscriptis - Uso de reglas de anotación

```
import urllib.request
from inscriptis import get_annotated_text, ParserConfig

html = urllib.request.urlopen(url).read().decode('utf-8')

rules = {'h1': ['heading', 'h1'],
         'h2': ['heading', 'h2'],
         'b': ['emphasis'],
         'table': ['table']}

output = get_annotated_text(html, ParserConfig(annotation_rules=rules)
```

3.2.1.3 Postprocesamiento

Además, Inscriptis da la posibilidad al usuario de realizar una fase de postprocesamiento donde las anotaciones se realizan en un formato determinado. Un primer tipo de postprocesamiento es el que se conoce como surface, donde se retorna una lista de mapeos entre la superficie de anotación y su etiqueta.

```
[['heading', 'Chur'],
  ['emphasis': 'Chur']]
```

En segundo lugar, el postprocesamiento tipo xml retorna una etiqueta de versión adicional, propia de un documento XML convencional.

```
<?xml version="1.0" encoding="UTF-8" ?>
<heading>Chur</heading>

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

Por último, el postprocesamiento en tipo html crea un documento HTML que contiene el texto convertido y resalta todas las anotaciones. Se muestra en la figura 3.1 un ejemplo de la salida obtenida.

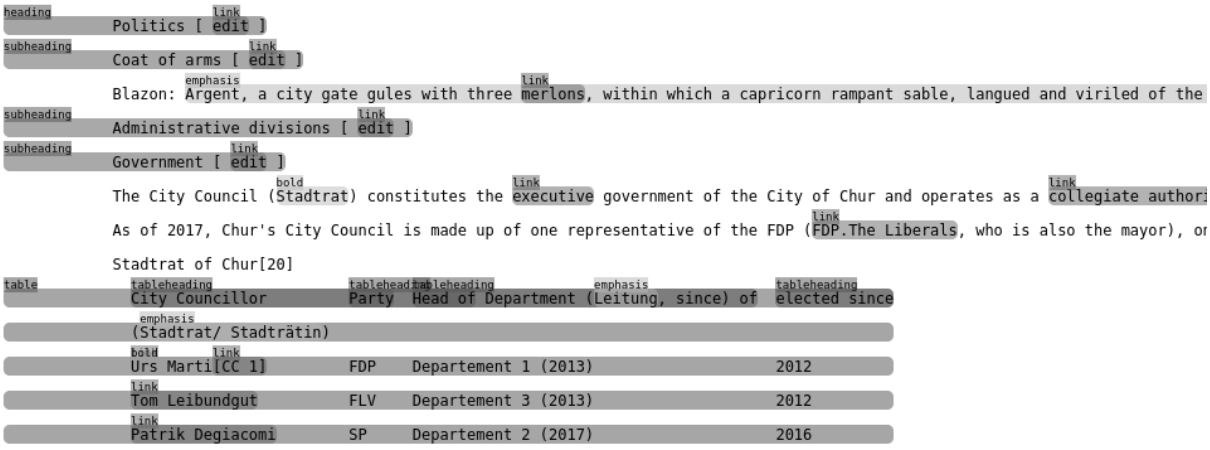

Figura 3.1: Inscriptis - Postprocesamiento html

3.2.2 Beautiful Soup

Beautiful Soup [18] es una de las librerías de Python más comunes en el ámbito del web scraping, diseñada para extraer datos de documentos XML y HTML. Como se muestra en la tabla 3.1 y a diferencia del resto de librerías, Beautiful Soup permite determinar el tipo de analizador que se empleará en la extracción, lo que flexibiliza el proceso de navegación, búsqueda y modificación de documentos.

3.2.2.1 Multiplicidad de analizadores

Beautiful Soup tiene a html.parse como analizador estándar de documentos HTML, pero ya hemos dicho que admite varios analizadores de terceros. En la tabla 3.3 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 3.3: Beautiful Soup - Analizadores disponibles

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.

Se emplea el analizador lxml sobre un documento HTML sencillo pero con erratas. Vemos como la solución aportada propone la inclusión de nuevas etiquetas *<html>* 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 * * como lo hace lxml, el analizador html5lib la empareja con una etiqueta * * de apertura. También añade una etiqueta <head>que el analizador lxml había 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.

3.2.2.2 Proceso de extracción

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 propio documento. Finalmente, un analizador especificado por parámetro, recorre el árbol buscando las partes del mismo que se desean.

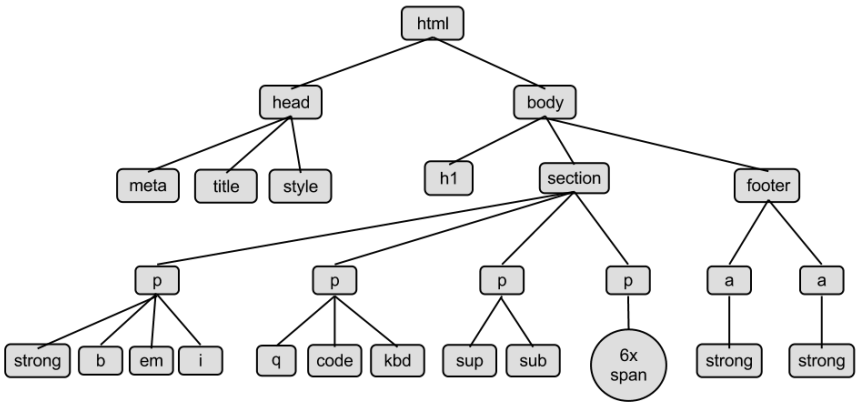

Figura 3.2: Arbol de objetos

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, ...*

3.2.3 JusText

JusText [19] es una herramienta para eliminar el contenido repetitivo, como los enlaces de navegación, encabezados y pies de página de los documentos HTML. Este algoritmo, está diseñado para preservar el texto que contiene frases completas.

3.2.3.1 Preprocesamiento

Previamente a cualquier fase y con el fin de facilitar el trabajo heurístico, JusText realiza un preprocesamiento del documento HTML. Durante este proceso, se elimina el contenido de ciertas etiquetas como *<header>*, *<style>* y *<script>*. Además, el contenido de etiquetas como *<select>* se clasifica inmediatamente como contenido basura. Lo mismo ocurre con los bloques que contienen ciertos símbolos especiales como el de copyright ©.

3.2.3.2 Segmentación

Tras una previa fase de preprocesamiento, se procede a lo que se conoce como segmentación. La idea es formar bloques de texto dividiendo la página HTML por etiquetas. Una secuencia de dos o más etiquetas como *
*, * *, ..., separaría los bloques.

Para la segmentación de bloques, la clave es que 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.

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, denominado 'boilerplate'. Se resumen a continuación algunos aspectos en relación.

- 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.
- 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.
- 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.

Con respecto al ultimo punto, decidir si un texto es gramatical o no 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 estas palabras, los contenidos de tipo boilerplate suelen carecer de ellas.

3.2.3.3 Clasificación de bloques

Tras la fase de preprocesamiento y segmentación se procede a la clasificación de bloques, donde a cada uno de estos bloques se le asigna una clase dependiendo de su naturaleza. En el fragmento de código 3.2 se muestra paso a paso como se determina el tipo de clase para cada bloque.

Listado 3.2: 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'
```

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

Además de los valores de densidad y longitud, el algoritmo toma como parámetros dos enteros definidos como LENGTH_LOW y LENGTH_HIGH, y también tres números de coma flotante, MAX_LINK_DENSITY, STOPWORDS_LOW y STOPWORDS_HIGH. Los dos primeros establecen los umbrales para dividir los bloques por su longitud. Los dos últimos dividen los bloques por la densidad de palabras de parada en bajos, medianos y altos.

Si volvemos a observar el algoritmo 3.2, nos damos cuenta de que solo se ha realizado una clasificación real sobre los bloques de tamaño medio y largo. En la tabla 3.4 se muestra a modo de resumen como JusText actúa sobre este tipo de bloques.

Block size	Stopwords density	Class
medium size	low	bad
long	low	bad
medium size	medium	near-good
long	medium	near-good
medium size	high	near-good
long	high	good

Tabla 3.4: JusText - Clasificación de bloques long & medium size

3.2.3.4 Reclasificación de bloques

¿Qué ocurre entonces con los bloques cortos y los bloques casi buenos? La reclasificación en este caso se realiza 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 figura 3.3.

Figura 3.3: 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.

3.2.3.5 Bloques de cabecera

En cuanto a los bloques de cabecera, aquellos encerrados en etiquetas del tipo *<h1>*, *<h2>*, ..., son tratados de una manera especial. El objetivo es conservar estos bloques en los textos determinados como buenos.

Para el tratamiento especial de bloques de cabecera se añaden dos etapas. La primera etapa, conocida como preprocesamiento, se ejecuta justamente después de la clasificación y justo antes de la reclasificación de bloques. La segunda etapa, conocida como postprocesamiento, se ejecuta después de la reclasificación.

1. Clasificación de bloques.
2. **Preprocesamiento de bloques de cabecera**.
3. Reclasificación de bloques.
4. **Postprocesamiento de bloques de cabecera**.

Durante esta fase de preprocesamiento, se buscan bloques de cabecera cortos que precedan a bloques buenos, y que al mismo tiempo no haya más caracteres entre el bloque de cabecera y el bloque bueno. El propósito de esto es preservar los bloques cortos entre el encabezado y el texto bueno que, de otro modo, podrían ser seleccionados como malos durante el proceso de reclasificación.

Por otro lado, en el postprocesamiento, se buscan de nuevo bloques de cabecera que precedan a bloques buenos, y que al mismo tiempo no haya más caracteres entre el bloque de cabecera y el bloque bueno. El propósito es que algunas cabeceras cortas y casi buenas puedan clasificarse como buenas si preceden a bloques buenos que, de otro modo, habrían sido clasificadas como malas tras de la reclasificación.

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

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

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

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

[5] 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.

[6] 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].

[7] 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.

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

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

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

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

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

[13] 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].

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

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

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

[17] 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

[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] 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].

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

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

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

[24] Wikipedia, "Analizador sintáctico," https://en.wikipedia.org/wiki/Parsing. [Ultimo acceso 22/diciembre/2021].

[25] I. Bicking, "lxml - xml and html with python," https://lxml.de/index.html. [Ultimo acceso 20/diciembre/2021].

[26] ——, "lxml.html," https://lxml.de/lxmlhtml.html. [Ultimo acceso 20/diciembre/2021].

[27] ——, "Beautifulsoup parser," https://lxml.de/elementsoup.html. [Ultimo acceso 20/diciembre/2021].

[28] ——, "html5lib parser," https://lxml.de/html5parser.html. [Ultimo acceso 20/diciembre/2021].

[29] Python, "html parser - simple html and xhtml parser," https://docs.python.org/3/library/html.parser.html. [Ultimo acceso 20/diciembre/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* [20] 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* [21] 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* [22], una de las bibliotecas software más comunes en este aspecto, diseñada para trabajar con *magrittr* [23] 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.

Apéndice B

Analizadores empleados en el web scraping

Este apéndice tiene como objetivo introducir aquellos aspectos más relevantes relacionados con los analizadores, también conocidos como *parsers*, empleados en el web scraping. A lo largo de esta sección se realizará una pequeña sinopsis de aquellas herramientas empleadas en los algoritmos de extracción de los paquetes definidos en la sección **??**.

A modo de introducción cabe destacar que un analizador sintáctico, es un programa informático que analiza una cadena de símbolos según las reglas de una gramática formal [24]. Generalmente, los analizadores se componen de dos partes, por un lado, un analizador sintáctico, y por otro un analizador léxico. Mientras que el analizador léxico crea tokens a partir de una secuencia de caracteres de entrada, el analizador sintáctico convierte los tokens en otras estructuras.

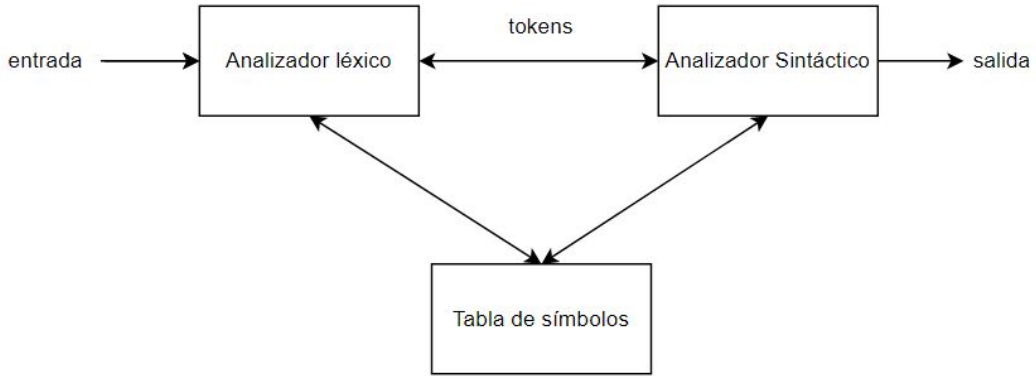

Figura B.1: Estructura basica de un analizador

B.1 lxml

Uno de los analizadores más comunes en todos los algoritmos de minado web es *lxml* [25]. Esta biblioteca de Python permite procesar tanto documentos XML como HTML.

Como la gran mayoría de analizadores, lxml convierte el texto de entrada en una estructura tipo árbol. Esto permite navegar por la propia estructura en busca de la información que se desea de forma sencilla.

En el caso del web scraping, la manera en la que lxml es capaz de encontrar información valiosa, es decir texto, es a través de expresiones XPath. Se muestra un pequeño ejemplo a continuación.

```
> html.xpath("string()")
# TEXTTAIL

> html.xpath("//text()")
# ['TEXT', 'TAIL']
```

Cabe destacar que el resultado dado por una expresión XPath es un objeto especial que conoce parte de su estructura. Esto permite ejecutar operaciones sobre este elemento y saber de dónde proviene a través de diferentes métodos.

B.1.1 lxml.html

¿Qué ocurre con los documentos HTML mal formados? Para este propósito, se creó lo que se conoce como *lxml.html* [26]. Un paquete de Python especial para tratar con documentos HTML, el cual a diferencia del analizador base, proporciona una API de elementos propia de HTML, así como una serie de utilidades para tareas comunes de procesamiento de los mismos.

```
> broken_html = "<html><head><body><h1>page title</h3>"
> parser = etree.HTMLParser()
> tree   = etree.parse(StringIO(broken_html), parser)
> result = etree.tostring(tree.getroot(), method="html")
# <html><head></head><body><h1>page title</h1></body></html>
```

Como se puede observar, incluso analizando documentos realmente mal formados, el analizador es capaz de crear una estructura propia de un documento HTML. A partir de ahí, es posible aplicar expresiones XPath con el fin de recorrer el árbol generado y recuperar información de valor.

B.1.2 soupparser

Otro de los analizadores propios de lxml es *soupparser* [27] propio del paquete BeautifulSoup. La forma en la que lxml interactúa con BeautifulSoup, es a través del módulo *lxml.html.soupparser* el cual proporciona una seria de funciones principales.

Por un lado, tanto *fromstring()* como *parse()* se emplean para analizar ya sea una cadena o un archivo HTML, por otro lado *convert_tree()* se emplea para convertir un árbol BeautifulSoup existente en una lista de elementos de nivel superior.

```
> tag_soup = '''<meta/><head></head><body>Hi all<p>'''
> root = fromstring(tag_soup)
# <html><meta/><head></head><body>Hi all<p/></body></html>
```

Imaginemos que se dispone de una cadena HTML mal formada como la mostrada en el ejemplo. Al ser una cadena, se emplea *fromstring()* para realizar un análisis de la misma. Como vemos la salida también puede provocar algún error, puesto que no es exactamente una estructura propia de HTML.

B.2 html5lib

html5lib [28] es un paquete de Python que implementa el algoritmo de análisis sintáctico de HTML5. Proporciona una interfaz similar a la del módulo lxml.html B.1.1 con metodos como *fromstring()* o *parse()* que operan de la misma manera que las funciones de análisis de HTML normales.

Al igual que lxml.html, este paquete también trabaja con árboles de objetos, pero en este caso normaliza algunos elementos y estructuras a un formato común. Por ejemplo, incluso si una tabla no tiene una etiqueta del tipo *<tbody>*, se inyectará una automáticamente.

```
> tostring(html5parser.fromstring("<table><td>foo"))
# '<table><tbody><tr><td>foo</td></tr></tbody></table>'
```

B.3 html.parser

html.parser [29] es un módulo que define una clase HTMLParse como base para analizar archivos HTML y XHTML. Este módulo trabaja alrededor de etiquetas, pues llama a métodos de manejo cuando se encuentran etiquetas de inicio, etiquetas finales, texto, comentarios y otros elementos de marcado.

```
> parser.feed('<p><a class=link href=#main>tag soup</p ></a>')
# Start tag: p
# Start tag: a
# Data     : tag soup
# End tag  : p
# End tag  : a
```

Como se puede observar, el algoritmo es capaz de detectar etiquetas tanto de apertura, como de cierre incluso en documentos HTML mal formados como este. Esto hace que la búsqueda de información valiosa, en este caso texto, sea sencilla.

Universidad de Alcalá
Escuela Politécnica Superior

Universidad
de Alcalá