

Natural Language Processing

Tema 7: Web Scraping

Alejandro Medrano

CONQUER BLOCKS

Índice

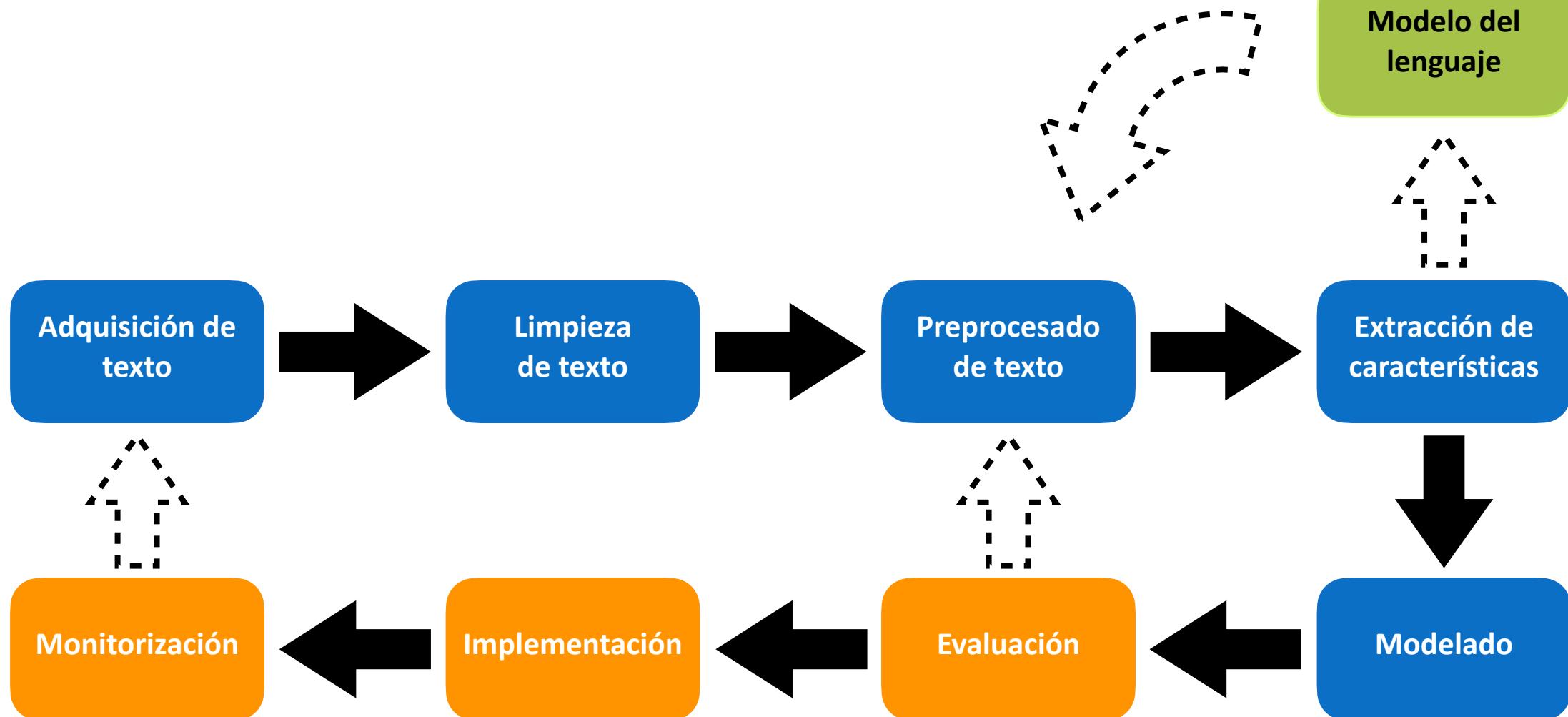
1. Introducción y objetivos
2. Web scraping estático
3. Web scraping dinámico
4. Consideraciones éticas y legales
5. Ejercicio práctico

Introducción y objetivos

Objetivos

1. Comprender la importancia y aplicabilidad del web scraping en el procesamiento de lenguaje natural.
2. Aprender a identificar y extraer datos textuales de sitios webs.
3. Revisión de las técnicas y tipos más comunes de web scraping. Así como su aplicabilidad y selección según las necesidades del proyecto y las características del sitio web.
4. Dominar el uso de herramientas y bibliotecas de Python para el web scraping.
5. Evaluar las consideraciones éticas y legales asociadas al web scraping. Así como comprender las mejores prácticas.
6. Aplicar conocimientos de web scraping en un contexto de NLP para resolver un problema real.

Flujo de trabajo en NLP



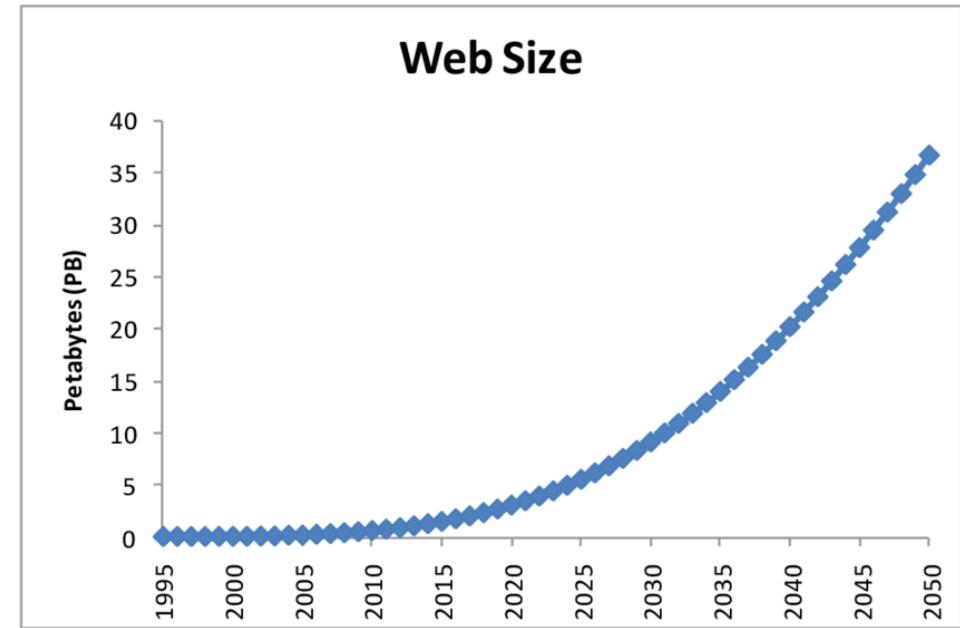
Web scraping

- **Web scraping** es una técnica empleada para **extraer** grandes cantidades de **información** desde **sitios web** mediante programas automatizados.
- Generalmente utilizaremos **análisis HTML**, interacción mediante **APIs** o manejo de páginas dinámicas con **JavaScript**.
- En el campo del NLP, los datos son tan importantes como el algoritmo. El web scraping proporciona acceso a vastos recursos de texto que pueden ser utilizados para entrenar modelos de lenguaje, análisis de sentimientos, y más.



Internet: un océano de datos

- Internet alberga una cantidad **exponencialmente creciente** de datos, con millones de GB de información generados cada día.
- Estos datos son una **mina de oro** para los data scientists, existe multitud de datos abiertos que podemos utilizar.
- La habilidad de transformar este océano de datos crudos en información útil y accionable es lo que hacen a los expertos en NLP tan valiosos.



[Trotman, Andrew & Zhang, Jinglan. \(2013\). Future Web Growth and its Consequences for Web Search Architectures.](#)

Se calcula que entre el 80% y el 95% del contenido de Internet es inaccesible a través de búsquedas convencionales y requieren de métodos de extracción especializados

Tipos de web scraping

- Existen multitud de tipos y técnicas de web scraping. Su elección dependerá de nuestras necesidades, la estructura del sitio web y el tipo de datos necesarios.
- Tipos más importantes de web scraping:
 - **Web scraping estático**: No depende de interacciones complejas del usuario para mostrar su contenido.
 - **Web scraping dinámico**: Usa herramientas automáticas para interactuar con sitios webs que requieren acciones del usuario, como clicks o ingreso de información, para revelar la información.
 - **API scraping**: Extracción de datos a través de APIs públicas o privadas que los sitios web proporcionan para acceder programáticamente a su información.

Web scraping estático

Web scraping estático

BeautifulSoup



- El **web scraping estático** se refiere a la extracción de datos de páginas web que **no requieren interacción** del usuario para mostrar su contenido completo. Estos datos son accesibles directamente en el código HTML inicial de la página.
- Proceso básico:
 1. **Cargar el HTML** de la página web mediante una petición HTTP.
 2. **Parsear el HTML** para extraer la información deseada utilizando herramientas específicas.
- En Python las principales herramientas son:
 - **Requests**: Simplifica envío de solicitudes HTTP.
 - **BeautifulSoup**: Facilita el web scraping permitiendo el parseo de documentos HTML y XML.

Requests: GET

```
import requests

# Enviar solicitud GET
url = ...
try:
    response = requests.get(url)
except:
    print("Error al abrir la página")
else:
    if response.status_code == 200:
        print("Contenido de la solicitud GET:")
        print(response.content)
    else:
        print("Error en la solicitud:", response.status_code)
```

BS4: Objeto sopa

```
import requests
from bs4 import BeautifulSoup

# Enviar solicitud GET
url = ...
response = requests.get(url) # Faltaría manejo de excepciones

# Parsea el contenido HTML usando html.parser
soup = BeautifulSoup(response.content, 'html.parser')
```

BS4: Búsqueda de elementos

```
# Buscar el primer elemento <title>
title = soup.find('title')
print(title)

# Buscar todos los elementos <a> (anclas/links)
links = soup.find_all('a')
for link in links:
    print(link.get('href'))
```

BS4: Acceder a los atributos de un elemento

```
# Supongamos que queremos obtener el href del primer link
first_link = soup.find('a')
href = first_link['href']
print(href)

# Tambien existe el método .get que es seguro si el atributo no existe
href = first_link.get('href')
print(href)
```

BS4: Navegación por el árbol de elementos

- Podemos navegar por la estructura del documento usándosela relaciones como *parent*, *children*, *next_sibling* y *previous_sibling*.

```
# Acceder al padre de un elemento
parent = first_link.parent
print(parent.name)

# Iterar sobre todos los hijos de un elemento
for child in parent.children:
    print(child)

# Acceder al siguiente y anterior elemento hermano
next_sibling = first_link.next_sibling
previous_sibling = first_link.previous_sibling
```

BS4: Extraer texto de los elementos

```
# Extraer todo el texto de un elemento  
print(first_link.text)
```

Web scraping dinámico

Web scraping dinámico

- El **web scraping dinámico** se refiere al conjunto de técnicas de extracción de datos de sitios web que cargan su contenido de manera asincrónica, generalmente mediante JavaScript.
- A diferencia del scraping estático, donde el contenido ya está presente en el HTML inicial, el scraping dinámico puede **requerir interacción con la página** (como hacer clic o desplazarse) para cargar los datos.
- La idea es **simular la interacción de un usuario real** con la página para que se ejecuten todos los scripts necesarios y se revele el contenido oculto.
- En Python la principal herramienta es **Selenium**.

Selenium



- **Selenium** es una poderosa librería de Python para automatizar navegadores web, permitiendo hacer web scraping dinámico.
- Selenium permite **simular** una amplia gama de **interacciones humanas** como clicks, ingreso de texto, y desplazamientos, lo que facilita la extracción de datos que de otro modo estarían inaccesibles.
- Únicamente debe ser usado cuando sea necesario, generalmente se trata de una herramienta lenta en comparación con técnicas de scraping estáticas, ademas de requerir de un buen manejo de tiempos de espera y excepciones para ser efectivo.

Consideraciones éticas y legales

Consideraciones éticas y legales

- Al realizar web scraping, es crucial no solo considerar qué se puede hacer técnicamente, sino también qué se debería hacer desde una perspectiva ética y legal.
- **Consideraciones éticas:**
 - **Respeto por la privacidad:** Evitar recolectar datos personales sin consentimiento, especialmente en jurisdicciones con regulaciones estrictas como el GDPR en Europa.
 - **Impacto en los sitios web:** Considerar el efecto del scraping en los recursos del servidor del sitio web, evitando causar una carga excesiva que podría degradar el servicio para otros usuarios.
 - **Transparencia y uso de datos:** Ser transparente sobre cómo se utilizan los datos extraídos y asegurarse de que su uso no infrinja los derechos de los individuos o entidades afectadas.
- **Consideraciones legales:**
 - **Cumplimiento de la ley:** Familiarizarse con y adherirse a las leyes locales que regulan la extracción de datos, incluyendo leyes de derechos de autor y de privacidad.
 - **Términos de servicio:** Respetar los términos de servicio de los sitios web, que a menudo incluyen cláusulas específicas sobre la prohibición o limitación del scraping.

Buenas prácticas

- **Respeto a los términos de servicio:** Antes de comenzar a hacer scraping, verifica y sigue los términos de servicio del sitio web, que pueden prohibir o limitar esta práctica.
- **Adherencia al archivo *robots.txt*:** Consulta el archivo *robots.txt* del sitio para respetar las restricciones impuestas sobre el scraping de ciertas áreas del sitio web.
- **Gestión de la carga del servidor:** Evita sobrecargar los servidores del sitio implementando retrasos entre solicitudes y evitando el scraping durante los picos de tráfico para minimizar el impacto en el rendimiento del servidor.
- **Privacidad y protección de datos:** Asegúrate de proteger la privacidad y cumplir con las regulaciones de protección de datos como el GDPR, evitando recolectar y almacenar datos personales sin consentimiento.
- **Uso ético de los datos:** Utiliza los datos recopilados de manera ética, para los fines permitidos, y mantén transparencia sobre cómo se utilizan estos datos.
- **Estrategias de extracción eficientes:** Optimiza tus scripts de scraping usando técnicas de cacheo y seleccionando precisamente los datos a extraer para mejorar la eficiencia y reducir la carga en los recursos del sitio web.

Ejercicio práctico



¡GRACIAS!

CONQUER **BLOCKS**