Procesamiento del lenguaje natural

Web scraping con "BeatifulSoup"

1. Introducción

En Data Science, sólo se puede hacer cosas de interés si se dispone del conjunto de datos adecuado. Una vez que tengamos buenos datos, se puede recurrir a *Pandas* o *Matplotlib* para analizar o visualizar tendencias. Pero, ¿cómo obtenemos esos datos en primer lugar?

Si se nos proporciona en un archivo *csv* o *json* bien organizado no habría problema, pero la mayoría de las veces necesitamos buscar los datos por nosotros mismos. Si los datos residen, por ejemplo, en una página web, se requerirá alguna forma de descargarlos. Aquí es donde BeautifulSoup resulta útil para trabajar con HTML y convertir su código en una estructura que podamos entender.

BeautifulSoup toma su nombre de una canción de la obra literaria "Alicia en el país de las maravillas", permitiendo tomar información de un sitio web de manera fácil y rápida y volcarla a un *DataFrame*.

2. Normas para hacer "scraping"

Cuando hacemos "scraping" en sitios web debemos asegurarnos de seguir algunas pautas para tratar a los sitios web y a sus propietarios con respeto. Por ello, cabe consultar siempre los términos y condiciones de un sitio web antes de realizar la extracción o "scraping". Por lo general, los datos recopilados no deben utilizarse con fines comerciales.

Es importante asimismo no generar más tráfico del debido a un servidor web haciendo más solicitudes de las necesarias. Como regla general de buenas prácticas, se puede realizar una solicitud a una página web por segundo. Si el diseño del sitio web cambiara, habría que adaptar el código a la nueva estructura de la página web a la que estés accediendo.

3. Solicitudes

Para obtener el HTML del sitio web, debemos en primer lugar realizar una *solicitud* que nos permita extraer el contenido de la página web. Python dispone de una librería para solicitudes que hace que obtener contenido sea realmente fácil. Todo lo que tenemos que hacer es importar la librería y luego introducir la URL con la que queremos trabajar:

```
import requests

res = requests.get("https://www.rtve.es/")
print(res.text)
```

Este código, por ejemplo, imprimiría el HTML de la página pasada como argumento en el URL.

4. El objeto BeautifulSoup

La impresión del código HTML "en bruto" no parece una buena opción. ¿Cómo podríamos extraer información relevante de una página? *BeautifulSoup* nos facilita recorrer una página HTML y extraer las partes que nos interesan.

Podemos importarlo usando la línea:

```
from bs4 import BeautifulSoup
```

Si, por ejemplo, imaginemos que tenemos el siguiente archivo HTML llamado "rainbow.html" ...

```
<body>
  <div> rojo </div>
  <div> naranja </div>
  <div> amarillo </div>
  <div> verde </div>
  <div> azul </div>
  <div> indigo </div>
  <div> violeta </div>
</body>
```

... podemos creamos un objeto *BeautifulSoup* para acceder a su contenido (en este caso previamente descargado):

```
soup = BeautifulSoup ("rainbow.html", "html.parser")
```

"html.parser" es un tipo de analizador ("parser"). Hay otras opciones como "lxml" y "html5lib" que tienen diferentes ventajas y desventajas, pero para "html.parser" es de entrada una buena opción.

Con las habilidades de solicitudes que acabamos de aprender, podemos usar un sitio web alojado en línea como HTML:

```
webpage = requests.get ("http://rainbow.com/rainbow.html", "html.parser")
soup = BeautifulSoup(webpage.content)
```

Cuando usamos BeautifulSoup en combinación con Pandas, podemos convertir sitios web en *DataFrames* que son fáciles de manipular y de los que es posible extraer información útil.

5. Tipos de objetos

BeautifulSoup divide la página HTML en varios tipos de objetos.

5.1. Etiquetas

Una etiqueta corresponde a una etiqueta HTML en el documento original. Estas líneas de código ...

```
soup = BeautifulSoup('<div id="example">An example div</div><p>An example p tag</p>')
print(soup.div)
```

... producirían una salida que se parece a :

```
<div id="example">An example div</div>
```

Accediendo a una etiqueta desde el objeto *BeautifulSoup* de esta manera se obtendrá la *primera* etiqueta de ese tipo en la página.

Se puede obtener el nombre de la etiqueta usando *.name* y un diccionario que represente los atributos de la etiqueta usando *.attrs*:

```
print (soup.div.name)
print (soup.div.attrs)
```

```
div
{'id': 'example'}
```

5.2. NavigableStrings

NavigableStrings son fragmentos de texto que se encuentran en las etiquetas HTML de la página. Se puede obtener la cadena dentro de la etiqueta llamando a *.string:*

```
print (soup.div.string)
```

Un ejemplo de div

5.3. Navegación por etiquetas

Para navegar a través de un árbol, podemos llamar a los nombres de las etiquetas. Imagina que tenemos una página HTML con este aspecto:

```
<h1> Las mejores galletas con chispas de chocolate del mundo </h1>
<div class = "banner">
  <h1> Ingredientes </h1>
</div>
<ul>
  <li> 1 taza de harina </li>
  <li> 1/2 taza de azúcar </li>
  <li> 2 cucharadas de aceite </li>
  <li> 1/2 cucharadita de bicarbonato de sodio </li>
  <li> ½ taza de chispas de chocolate </ li>
  <li> 1/2 cucharadita de vainilla <li>
  <li> 2 cucharadas de leche </li>
</ul>
```

Si quisiéramos obtener el primer elemento *h1*, lo haríamos de esta forma:

```
print(soup.h1)
```

<h1> Las mejores galletas con chispas de chocolate del mundo </h1>

Podemos obtener los hijos de una etiqueta accediendo al atributo .children:

```
for child in soup.ul.children:
    print(child)
```

 1 taza de harina
 1/2 taza de azúcar
 2 cucharadas de aceite
 1/2 cucharadita de bicarbonato de sodio
 ½ taza de chocolate chips
 1/2 cucharadita de vainilla
 2 cucharadas de leche

También podemos navegar *hacia arriba en* el árbol de una etiqueta accediendo al atributo .parents:

```
for parent in soup.li.parents:
  print(parent)
```

Este bucle imprimirá primero:

 1 taza de harina
 1/2 taza azúcar
 2 cucharadas de aceite
 1/2 cucharadita de bicarbonato de sodio
 ½ taza de chispas de chocolate
 1/2 cucharadita de vainilla </ li >
 2 cucharadas de leche

Luego, imprimirá la etiqueta que contiene el ul (es decir, la etiqueta del cuerpo del documento). Luego, imprimirá la etiqueta que contiene la etiqueta del cuerpo (es decir, la etiqueta html del documento).

5.4. "Buscar todo"

Si queremos encontrar todas las ocurrencias de una etiqueta, en lugar de solo la primera, podemos usar *.find_all()*.

Esta función puede tomar solo el nombre de una etiqueta y devuelve una lista de todas las apariciones de esa etiqueta.

```
print (soup.find_all ("h1"))
```

['<h1> Las mejores galletas con chispas de chocolate del mundo </h1>', '<h1> Ingredientes </h1>']

.find_all() es mucho más flexible que simplemente acceder a elementos directamente a través del objeto de *soup*. Con *.find_all()*, podemos usar expresiones regulares, atributos o incluso funciones para seleccionar elementos HTML de manera más inteligente.

5.5. Usando Regex

¿Y si quisiéramos cada *<ol>* y cada *<ul>* que contiene la página? Podemos seleccionar ambos tipos de elementos con una expresión regular en nuestro *.find_all()*:

```
import re
soup.find_all(re.compile ("[ou]l"))
```

¿Y si quisiéramos todas las etiquetas *h1 - h9* que contiene la página?

```
import re
soup.find_all(re.compile ("h[1-9]"))
```

5.6. Usando listas

También podemos especificar todos los elementos que queremos encontrar proporcionando a la función una lista de los nombres de etiquetas que estamos buscando :

```
soup.find_all (['h1', 'a', 'p'])
```

5.7. Usando Atributos

También podemos intentar hacer coincidir los elementos con atributos relevantes. Podemos pasar un diccionario al parámetro *attrs* de *find_all* con los atributos deseados de los elementos que estemos buscando. Si queremos encontrar todos los elementos con la clase "banner", por ejemplo, podríamos usar el comando:

```
soup.find_all (attrs = {'class': 'banner'}) ¡
```

O podemos especificar múltiples atributos diferentes! ¿Qué pasaría si quisiéramos una etiqueta con una clase de "banner" y la identificación "jumbotron"?

```
soup.find_all (attrs = {'class': 'banner', 'id': 'jumbotron'})
```

5.8. Usando una función

Si nuestra selección comenzara a complicarse mucho, se puede separar toda la lógica que estemos usando para elegir una etiqueta en su propia función.

```
def has_banner_class_and_hello_world (tag):
   return tag.attr ('class') == "banner" and tag.string == "Hola mundo"

soup.find_all (has_banner_class_and_hello_world)
```

Este comando encontraría una etiqueta de este tipo:

 Hola mundo

pero no un elemento que se vea así:

 Hola mundo

O esto:

 ¡Qué pasa, mundo!

5.9. Selecciones para selectores CSS

Otra forma de capturar los elementos deseados con el objeto *soup* es usar selectores CSS. El método *.select()* tomará todos los selectores CSS que normalmente se usa en un archivo *.css*

<h1 class = 'results'> Resultados de búsqueda para: Funfetti </h1>
  Funfetti Spaghetti < / a> </ div>
  lasaña de Funfetti </ div>
 < a href = "cupcakes.html"> Cupcakes Funfetti
  Pastel Funfetti de nueces

Si quisiéramos seleccionar todos los elementos que tienen la clase 'RecetaLink', podríamos usar el comando:

```
soup.select(".recipeLink")
```

Si quisiéramos seleccionar el elemento que tiene el id 'selected', podríamos usar el comando :

```
soup.select("#selected")
```

Supongamos que queremos recorrer todos los enlaces a estas recetas que encontramos en nuestra búsqueda.

```
for link in soup.select(".recipeLink > a"):
  webpage = requests.get(link)
  new_soup = BeautifulSoup(webpage)
```

Este bucle pasará por cada enlace en cada *div .recipeLink* y creará un objeto fuera de la página web. Primero tomaría * Funfetti Spaghetti *, luego * Lasagna de Funfetti *, y así sucesivamente.

6. Lectura de texto

Cuando usamos *BeautifulSoup* para seleccionar elementos HTML a menudo queremos tomar el texto dentro del elemento para poder analizarlo. Podemos usar *.get_text()* para recuperar el texto dentro de cualquier etiqueta en la que queramos llamarlo.

<h1 class = "results"> Resultados de búsqueda para: Funfetti </h1>

Si este es el HTML que se ha utilizado para crear el objeto *soup*, podemos realizar la llamada:

```
soup.get_text ()
```

Que devolverá:

'Resultados de la búsqueda para: Funfetti'

Como se puede observar lo anterior combinó el texto dentro de la etiqueta *h1* externa con el texto contenido en la etiqueta *span* dentro de ella. Si quisiéramos separar los textos de diferentes etiquetas podríamos especificar un carácter separador:

```
soup.get_text ('|')
```

Ahora el comando devuelve:

'Resultados de la búsqueda para: | Funfetti'

7. Ejemplo final

En el siguiente ejemplo se muestra el funcionamiento de *BeatifulSoup.*

```
import requests
from bs4 import BeautifulSoup

prefix = "https://content.codecademy.com/courses/beautifulsoup/"
webpage_response = requests.get('https://content.codecademy.com/courses/beautifulsoup/shellter.html')

webpage = webpage_response.content
soup = BeautifulSoup(webpage, "html.parser")

# Se guardan las referencias a las diferentes tortugas de la página capturada
turtle_links = soup.find_all("a")

# Se guardan generan nuevos URL a partir de las referencias anteriores
links = []
for a in turtle_links:
 links.append(prefix + a["href"])

# Los datos de las tortugas se guardarán en un diccionario
turtle_data = {}

#follow each link:
for link in links:
   webpage = requests.get(link)
   turtle = BeautifulSoup(webpage.content, "html.parser")
   turtle_name = turtle.select(".name")[0].get_text() // se obtiene el nombre de la tortuga
   turtle_data[turtle_name] = turtle.find("ul").get_text("|").split("|") // se obtiene cada
característica, que se guarda por separado
   print(turtle_data[turtle_name])
```