Actividad 3 - James Bo de Interness

Data, data, data. I cannot make bricks without data - Sherlock Holmes

ExactasPrograma - Datos

Invierno 2020

En esta actividad nos vamos a centrar en la descarga sistemática de información desde sitios Web. Esta actividad se conoce coloquialmente como *crawlear datos* y la idea básica es traer un gran volumen de información de un conjunto de sitios sin tener que hacerlo manualmente. Los sitios que suelen ser objeto de esta práctica, que en algunos casos está cercana a no ser del todo legal, suelen poner algunas trabas dentro de sus sistemas para evitar que la información pueda ser extraída de una manera sencilla.

Para mostrar un escenario posible de uso, empecemos con un ejemplo hipotético del mundo del comercio electrónico: para tener idea de los comportamientos comerciales de algunas empresas, sería interesante tener los datos de los precios de los productos ofrecidos antes y durante el último *hot sale*. Si uno fuera capaz de tener esa información, sería bastante sencillo saber si las ofertas son reales o los productos fueron aumentados el día antes para que el descuento parezca mucho y en realidad siguen manteniendo el precio. Claramente, los sitios de comercio electrónico no quieren que se sepa si ellos realizan este tipo de prácticas y, por medio de distintas técnicas, buscan detener los accesos a sus sitios cuando no son realizados por usuarios humanos.

La actividad que presentamos se basa en utilizar una herramienta hecha en Python que permite controlar la recuperación de información a partir de sitios. Por ejemplo, es posible que se le indique que visite todas páginas o *links* mencionados en una página determinada. De esta forma, uno puede comenzar a navegar por una primer página y luego hacer que el programa siga los distintos links que encuentra, avanzando en la visita a otros sitios (o partes del mismo sitio) hasta que lo interrumpamos o hasta que se cumpla algún criterio objetivo pre-establecido (por ejemplo, obtener información de 1000 productos).

Para que nos mantengamos dentro del escenario académico, en lugar de obtener información de productos y precios, vamos centrar esta actividad en la recuperación y análisis de textos científicos.

El paquete que vamos a utilizar es uno de los más populares para hacer *crawling* y se llama `scrapy` (https://scrapy.org/). Este paquete no viene instalado *por default* así que deberán agregarlo siguiendo las instrucciones disponibles en su sistema. A continuación damos una idea de cómo sería la instalación usando `pip` (además, agregamos otro paquete que sirve para pasar páginas web a texto plano que se pueda leer):

```
pip install Scrapy --user
pip install html2text --user
```

Si no les funciona o necesitan más detalles, pueden ir directamente a la página de instalación del paquete: https://docs.scrapy.org/en/latest/intro/install.html.

Calentando motores

Luego de instalados los paquetes, vamos a hacer un poco de exploración de datos usando los siguientes tres artículos (por ahora no tienen que hacer nada, solo mirar los títulos):

- Fanelli D (2009) *"How Many Scientists Fabricate and Falsify Research? A Systematic Review and Meta-Analysis of Survey Data"*. PLoS ONE 4(5): e5738. https://doi.org/10.1371/journal.pone.0005738.

- A. Amini et al., *"Learning Robust Control Policies for End-to-End Autonomous Driving From Data-Driven Simulation"*. IEEE Robotics and Automation Letters, vol. 5, no. 2, pp. 1143-1150, April 2020, https://doi.org/10.1109/LRA.2020.2966414.
- Mohand Ousaid A, Millet G, Haliyo S, Régnier S, Hayward V (2014) *"Feeling What an Insect Feels"*. PLoS ONE 9(10): e108895. https://doi.org/10.1371/journal.pone.0108895.

A continuación, empezamos las actividades para ver cómo se pueden bajar datos y hacerles un procesamiento *divertido*:

1. No está de más, asegurarse que lo que instalamos y que funciona. Para esto, la manera más sencilla es abrir una terminal y ejecutar el comando:

```
scrapy -h
```

   Si todo fue bien, debería mostrar una salida similar a esta:

```
Scrapy 2.2.1 - no active project

Usage:
  scrapy <command> [options] [args]

Available commands:
  bench         Run quick benchmark test
  commands
  fetch         Fetch a URL using the Scrapy downloader
  genspider     Generate new spider using pre-defined templates
  runspider     Run a self-contained spider (without creating a project)
  settings      Get settings values
  shell         Interactive scraping console
  startproject  Create new project
  version       Print Scrapy version
  view          Open URL in browser, as seen by Scrapy

  [ more ]      More commands available when run from project directory

Use "scrapy <command> -h" to see more info about a command
```

2. `Scrapy` tiene un modo que permite hacer pruebas bastante poderosas sin tener que configurar demasiado, es lo que vamos a utilizar en esta primera etapa. Desde una terminal, hacer:

```
scrapy shell
```

   Si fue todo bien, debería haber entrado en la consola de `Scrapy` en donde podemos poner comandos en `Python`. Vamos a traer el primero de los artículos. Este comando se conecta al sitio de PLoS ONE y trae la página que contiene al artículo. Los mensajes que aparecen deberían ser similares a esto:

```
In [1]: fetch("https://doi.org/10.1371/journal.pone.0005738")
2020-08-24 11:00:10 [scrapy.core.engine] INFO: Spider opened
2020-08-24 11:00:11 [scrapy.downloadermiddlewares.redirect] DEBUG: Redirecting (302) to <
    GET https://dx.plos.org/10.1371/journal.pone.0005738> from <GET https://doi.org
    /10.1371/journal.pone.0005738>
2020-08-24 11:00:12 [scrapy.downloadermiddlewares.redirect] DEBUG: Redirecting (301) to <
    GET https://journals.plos.org/plosone/doi?id=10.1371/journal.pone.0005738> from <GET
    https://dx.plos.org/10.1371/journal.pone.0005738>
2020-08-24 11:00:14 [scrapy.downloadermiddlewares.redirect] DEBUG: Redirecting (301) to <
    GET https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0005738> from <
    GET https://journals.plos.org/plosone/doi?id=10.1371/journal.pone.0005738>
2020-08-24 11:00:14 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://journals.plos.
    org/plosone/article?id=10.1371/journal.pone.0005738> (referer: None)
```

Dentro de las cosas que muestran, están los distintos pasos que tuvo que hacer para poder conseguir la página pedida. El último mensaje nos da una pista de que la operación fue un éxito debido a que dice "`Crawled`".

Para verificar que anduvo bien, podemos ejecutar el siguiente comando:

```
In [2]: view(response)
Out[2]: True
```

La función `view` debería mostrar en nuestro navegador la página conteniendo el artículo (es posible que se vea rara, ya que los archivos de estilo están en el servidor de la revista y no se bajan con el artículo). Toda la información recolectada en el *crawling* queda asociada a la variable `response`, ya vamos a ver que tiene mucha información interesante y que se puede utilizar como cualquier variable en `Python`.

3. Si bien la página con el artículo contiene la información que nos interesa (el texto del artículo propiamente dicho), también incluye mucha data adicional que ensuciará el análisis que nos interesa hacer.

   Vamos a usar un paquete de `Python` desde la consola de `Scrapy` para quedarnos solo con el texto del artículo:

```
In [3]: import html2text
   ...: converter=html2text.HTML2Text()
   ...: converter.ignore_links=True
   ...: converter.ignore_images=True
   ...: converter.ignore_tables=True
   ...: texto=converter.handle(response.css('*').get())
   ...: #texto=texto.encode('ascii', 'ignore')
   ...: #Descomentar la linea anterior si hay problemas de codificacion al intentar
    guardar el archivo
```

   Con estos comandos, vamos a tener todo el texto del artículo asociado a la variable `texto`. Con el comando `response.css('*').get()` se puede acceder a toda la información de la página (posiblemente haya más cosas que no interesan, pero es la manera segura de tener todo). El último comando (`texto.encode('ascii', 'ignore')`) es para asegurarnos que no quede ningún carácter raro que no sea texto común, a veces quedan algunos y después molestan, aunque dependiendo de la instalación de cada máquina, puede ser necesario ejecutar esa línea o no. Podemos mirar cómo queda haciendo un `print`, pero como vamos a utilizar el texto más adelante, vamos a guardarlo en un archivo:

```
In [4]: f = open("paper1.txt", 'w')

In [5]: f.write(texto)
Out[5]: 62587

In [6]: f.close()
```

   Ahora, desde un editor de texto de nuestro gusto, podemos ver el contenido del archivo[1]. ¡Manos a la obra!

   Podrán notar que el archivo incluye un montón de cosas adicionales, como alguna propaganda, los *links* para bajar las imágenes y todas las referencias. Se puede afinar un poco la conversión para que ciertas cosas nos las incluya, pero como la estructura cambia de revista a revista, es más seguro quedarse con todo el texto del artículo.

4. Con el primer artículo bajado y guardado en un archivo, vamos a hacer un poco de exploración de su contenido: Bienvenidos al análisis de texto basado en herramientas computacionales.

[1] Recuerden que el archivo lo va a crear en el directorio desde donde llamaron al `Scrapy`.

Primero necesitamos transformar el texto para que sea una lista de palabras. Definan la función `separar_texto(pals)` que recibe una cadena de caracteres (un `string`, sería el texto del artículo que guardamos antes en un archivo) y lo separa en palabras, devolviendo una lista de palabras (debe incluir, repeticiones, la idea es que estén todas las palabras). Aquí pueden usar su sentido común para determinar qué es una palabra y aprovechar para limpiar el texto obtenido de cadenas que no lo sean (por ejemplo, números o código de producto o similares).

Veamos un ejemplo basado en un texto sencillo:

```
In [7]: texto = """Sherlock Holmes is a fictional private detective created by British
   ...: author Sir Arthur Conan Doyle. Referring to himself as a consulting detective
   ...: in the stories, Holmes is known for his proficiency with observation, deduction,
   ...:  forensic science, and logical reasoning that borders on the fantastic, which he
   ...:  employs when investigating cases for a wide variety of clients, including
   ...: Scotland Yard."""

In [8]: mis_palabras = separar_texto(texto)

In [9]: print(mis_palabras)
['Sherlock', 'Holmes', 'is', 'a', 'fictional', 'private', 'detective', 'created', 'by', '
    British', 'author', 'Sir', 'Arthur', 'Conan', 'Doyle.', 'Referring', 'to', 'himself',
    'as', 'a', 'consulting', 'detective', 'in', 'the', 'stories,', 'Holmes', 'is', 'known'
    , 'for', 'his', 'proficiency', 'with', 'observation,', 'deduction,', 'forensic', '
    science,', 'and', 'logical', 'reasoning', 'that', 'borders', 'on', 'the', 'fantastic,'
    , 'which', 'he', 'employs', 'when', 'investigating', 'cases', 'for', 'a', 'wide', '
    variety', 'of', 'clients,', 'including', 'Scotland', 'Yard.']
```

5. Ahora, el siguiente paso es generar un diccionario con las apariciones de todas las palabras. Vamos a definir la función `generar_dic(pals_sep)` que recibe una lista de palabras (la generada por la función anterior) y crea un diccionario que contiene como clave a las palabras y como dato la cantidad de veces que aparece en la lista. Por ejemplo, siguiendo con el código anterior, sería:

```
In [10]: freq_pals = generar_dic(mis_palabras)

In [11]: print(freq_pals)
{'stories': 1, 'detective': 2, 'fantastic': 1, 'fictional': 1, 'Sir': 1, 'a': 3, 'in': 1,
    'author': 1, 'forensic': 1, 'reasoning': 1, 'when': 1, 'by': 1, 'Yard': 1, 'as': 1, '
    clients': 1, 'logical': 1, 'deduction': 1, 'observation': 1, 'for': 2, 'Referring': 1,
     'he': 1, 'of': 1, 'borders': 1, 'created': 1, 'with': 1, 'on': 1, 'to': 1, 'Holmes':
    2, 'that': 1, 'science': 1, 'consulting': 1, 'investigating': 1, 'cases': 1, 'Conan':
    1, 'known': 1, 'himself': 1, 'proficiency': 1, 'is': 2, 'Doyle': 1, 'the': 2, 'which':
     1, 'employs': 1, 'including': 1, 'Arthur': 1, 'his': 1, 'Sherlock': 1, 'variety': 1,
    'and': 1, 'British': 1, 'private': 1, 'Scotland': 1, 'wide': 1}
```

6. Lo siguiente será hacer un histograma que muestre la frecuencia de aparición de las palabras (cuántas palabras se repiten una sola vez, cuántas dos y así). Esperamos que, a esta altura, esto lo puedan resolver sin ayuda adicional.

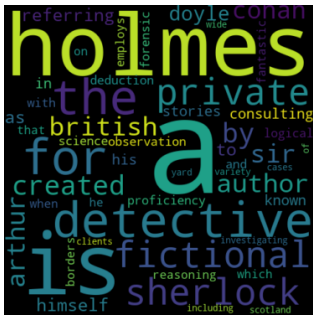
7. Ahora, vamos a mostrar una manera de ver las palabras más frecuentes usando lo que se conoce como *nube de palabras*. Para hacerlo en `Python` ya tenemos un paquete que lo resuelve, se llama `WordCloud`. Si no lo tienen instalado, deberán ejecutar:

```
pip install wordcloud --user
```

Para usarlo, tenemos el siguiente código de ejemplo:

```
In [13]: import matplotlib.pyplot as plt
    ...: from wordcloud import WordCloud
    ...: wordcloud = WordCloud(width=480, height=480, margin=0)
    ...: wordcloud.generate_from_frequencies(freq_pals)
    ...: plt.imshow(wordcloud, interpolation='bilinear')
    ...: plt.axis("off")
    ...: plt.margins(x=0, y=0)
    ...: plt.show()
```

Este código debería generar una figura similar a:



8. Si bien el texto es corto, se puede entender la idea de esta figura, cuanto mayor frecuencia tiene la palabra, más grande aparece en relación al resto. En esta nube de palabras se ve que aparecen varias palabras que no son específicas de novelas de misterio, descripción de un personaje o similar, sino que son palabras que aparecerían en cualquier texto.

   Cuando se intenta visualizar la importancia de las palabras en los textos, conviene filtrar previamente aquellas que no sean específicas. En la función `generar_dic` estamos incluyendo todas las palabras que aparecen en el texto recibido como parámetro. Lo que tenemos que hacer es escribir una nueva función, basada en la anterior, que reciba además del texto una lista de palabras que se desean filtrar: `generar_dict_filt(palabras_separadas, palabras_filtro, n)`. Como resultado también deberá devolver un diccionario, pero en este caso tendrá todas las palabras que estén el texto pero que no estén incluidas en la lista `palabras_filtro`. Adicionalmente, esta función recibe el parámetro `n` que indica la cantidad de palabras que debe incluir el diccionario. Si este parámetro es 0 o negativo, deben incluirse todas las palabras, si es un número positivo, solo deben incluirse las `n` palabras más frecuentes. Para probarla, pueden definir ustedes mismo la lista con las palabras que no quieran que se incluya en la nube de palabras resultante. En el ejemplo del texto de Sherlock Holmes, algunas palabras a filtrar serían: 'is' y 'a'.

9. Ahora vamos a ir un poco más allá y vamos a analizar qué pasa con las nubes de palabras en distintas novelas de misterio. Vamos a analizar los textos que están libremente disponibles en los siguientes *links*:

   a) A Study in Scarlet (A. Conan Doyle) http://www.gutenberg.org/files/244/244-h/244-h.htm
   b) The Murder on the Links (Agatha Christie) http://www.gutenberg.org/files/58866/58866-h/58866-h.htm
   c) The Man who was Thursday a Nightmare (G. K. Chesterton) http://www.gutenberg.org/files/1695/1695-h/1695-h.htm
   d) Whose Body? (Dorothy L. Sayers) http://www.gutenberg.org/files/58820/58820-h/58820-h.htm

   Deberán construir los histogramas de frecuencia y las nubes de la palabras (para las 20 palabras más frecuentes) para los libros incluidos en estos *links*. Recuerden definir el filtro de palabras para limitar el análisis a las palabras de interés.

10. Para terminar con el día. Ahora deberán construir los histogramas de frecuencia y las nubes de la palabras (para las 20 palabras más frecuentes) de los tres artículos que mencionamos al principio. Recuerden definir el filtro de palabras para limitar el análisis a las palabras de interés.

Terminando (por hoy) Les dejamos planteadas las siguientes preguntas para guiar los últimos dos puntos y sacar conclusiones:

I) ¿Puede observar diferencias entre las nubes de palabras individuales de cada uno de los géneros?

II) ¿Qué conclusiones saca si agrupan las palabras por género?

III) ¿Son todas las palabras distintas o hay alguna en común?

IV) ¿Se meten *cosas* que no sean palabras en algunas?

V) ¿Dónde se mencionan más *nombres propios*?