Web scraping and social media scraping – crawling

Jacek Lewkowicz, Dorota Celińska-Kopczyńska

University of Warsaw

March 24, 2019

What will we be working on today?

- We should already know how to gather data from a single website
- We will learn how to move from site to site
- and how to write simple crawlers

Note: Crawling is a broad topic: here we cover the popular cases.

Convention

- In snippets, we will highlight in violet the areas where you may put your own content
- In commands, the areas in [] are optional
- UNIX-like systems use "/" as the path separator and DOS uses "\". In this presentation the paths will be written in UNIX-like convention if not stated otherwise

Examples of crawlers covered today

- Generating lists of links with python or using stdin to provide rules
- Getting links from the given webpage and moving towards
- Moving to internal or external links

Scraping similar items from many sites

- In many cases we need the same information from a bunch of pages
- Typical example: getting departments for all the employers of an enterprise (each person has a standardized page)
- In such case we need to write a scraper for a single page and provide the rules how to move from one to another

Generating rules

- Inspect how the links in the original website are generated
- If there are sequences of numbers used
  - use list comprehension, ranges, etc – standard Python
- If there are some easily determined strings:
  - provide them as an internal list
  - write a rule to generate them
  - provide them via standard streams

Generating starting lists – examples

- While knowing the exact (usually low) number of entries:

```
pages = ['http://website1.com', 'http://website2.com']
```

- Providing, e.g., sequence of numbers:

```
pages = ('http://website1.com/{}'.format(l) for l in xrange(1,246,1))
pages = ('http://website1.com/' + ``1'')
```

- Using standard stream in (later you will pipe through it):

```
import fileinput
from sys import stdin
pages =  ('http://website1.com/{}'.format(l.strip()) for l in stdin)
```

Customized starting lists in Scrapy

- You have to override the `__init__()` function:

```
class YourSpider(scrapy.Spider):
    name = 'spidername'

    def __init__(self, category=None, *args, **kwargs):
        super(YourSpider, self).__init__(*args, **kwargs)
        # here you specify the rules for generation of starting urls
        self.start_urls = ('http://website1.com/'.format(l) for l in xrange(1,246,1) )
```

Subpages dependent on the input

- Let us assume that you have to extract data from a website that has multiple subpages
- You want to extract data both from the starting page and from the dependent ones
- And you want to do it in one spider, instead of running two of them
- Example: Each person in the enterprise has their standardized main website but the projects they were involved in are a subpage of their main website. You need both the information from the main website and the projects one was involved in

Extracting data from the known subpages

- General principles of generating rules apply in such situation (you can use them in BS)
- As you remember by default Scrapy collects data for an item from one website from the starting list
- In Scrapy you will have to yield a new `Request` and specify new `parse()` function, which will in the end yield either new `Request` or an item

Multiple pages for one item in Scrapy

```
from scrapy import Request
# specify new URL dependent on the already scraped part of the output from the website
URLnew = 'http://website1.com' +  "".join(item['name'])
yield Request(URLnew, meta='item':item, callback=self.parse_new)

    def parse_new(self, response):
        item = response.meta['item']
        # specify here new rules for objects belonging to item, as usual
        item['something'] = response.xpath('your/xpath/here/text()').getall()
        yield item
```

Unknown links

- A separate group of crawlers are the crawlers in which you cannot precisely determine the syntax of the subpages
- However you are able to determine the rules to find them within the website's source code

Unknown links – general idea

- You have to provide a rule to find the links within the source code – all methods are allowed (XPaths, finding parts of the source code, searching for a string...)
- Then get those links and, e.g., insert them into a set or a list
- In the end you will iterate over the chosen structure, adding new links if they have not been already gathered until manual stop or running out of links

Example – Extracting links

- Here we will present an example of a function which extracts links from a website
- We will prepare a function which starts with a random article
- Then it chooses a random article from the returned list and crawls towards
- And the procedure is repeated until the program is stopped or there are no article links on the new page

Example – Extracting links in Beautiful Soup

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import datetime
import random
import re

random.seed(datetime.datetime.now())
def getLinks(articleUrl):
    html = urlopen("http://en.wikipedia.org"+articleUrl)
    bsObj = BeautifulSoup(html)
    return bsObj.find("div", "id":"bodyContent").findAll("a", href=re.compile("^(/wiki/)((?!:).)*$"))

links = getLinks("/wiki/Edvard_Munch")
while len(links) > 0:
    newArticle = links[random.randint(0, len(links)-1)].attrs["href"]
    print(newArticle)
    links = getLinks(newArticle)
```

Example – crawling in Scrapy

```
import scrapy
from scrapy.spiders import CrawlSpider, Rule
from scrapy.linkextractors import LinkExtractor

class MySpider(CrawlSpider):
    name = 'example.com'
    allowed_domains = ['example.com']
    start_urls = ['http://www.example.com']

    rules = (
        # Extract links matching 'category.php' (but not matching 'subsection.php')
        # and follow links from them (since no callback means follow=True by default).
        Rule(LinkExtractor(allow=('category.php', ), deny=('subsection.php', ))),

        # Extract links matching 'item.php' and parse them with the spider's method parse_item
        Rule(LinkExtractor(allow=('item.php', )), callback='parse_item'),
    )

    def parse_item(self, response):
        self.logger.info('Hi, this is an item page! %s', response.url)
        item = scrapy.Item()
        item['id'] = response.xpath('//td[@id="item_id"]/text()').re(r'ID: (+)')
        item['name'] = response.xpath('//td[@id="item_name"]/text()').getall()
        item['description'] = response.xpath('//td[@id="item_description"]/text()').getall()
        return item
```

Example – following external links in BS

```
#list of external links found on a page
def getExternalLinks(bsObj, excludeUrl):
    externalLinks = []
    #links that start with "http" or "www" that do not contain #the current URL
    for link in bsObj.findAll("a",href=re.compile("^(http|www)((?!"+excludeUrl+").)*$")):
        if link.attrs['href'] is not None:
            if link.attrs['href'] not in externalLinks:
                externalLinks.append(link.attrs['href'])
                return externalLinks

def splitAddress(address):
    addressParts = address.replace("http://", "").split("/")
    return addressParts

def getRandomExternalLink(startingPage):
    html = urlopen(startingPage)
    bsObj = BeautifulSoup(html)
    externalLinks = getExternalLinks(bsObj, splitAddress(startingPage)[0])
    if len(externalLinks) == 0:
        internalLinks = getInternalLinks(startingPage)
        return getNextExternalLink(internalLinks[random.randint(0,len(internalLinks)-1)])
    else:
        return externalLinks[random.randint(0, len(externalLinks)-1)]

def followExternalOnly(startingSite):
    externalLink = getRandomExternalLink("http://www.website1.com/en/")
    print("Random external link is: "+externalLink)
    followExternalOnly(externalLink)
```