

Screen Scraping

aka harvesting aka spidering

By Imri Goldberg

What is it, and what's it good for?

- ▶ Screen Scraping is the process of parsing a web page for the information stored within.
- ▶ Not every web page 'wants' to be scraped, sometimes you have to force it.
- ▶ The rewards are the information, which may be used:
 - ▶ as input for a search engine
 - ▶ as source data for some aggregating service
 - ▶ as source data for research
 - ▶ etc...



Example

- ▶ Let's find the most influential artist in the world, using an algorithm similar to pagerank.
- ▶ What we need:
 - ▶ an implementation of the algorithm
 - ▶ input data.
- ▶ Since we don't have input data, we'll have to get it. How about allmusic.com?



Perliminaries

- ▶ IPython is very recommended!
- ▶ We need a local copy of the webpage
- ▶ To get it from a Python script, just use urllib2.
 - ▶ `d = urllib2.urlopen(url).read()`
- ▶ We'll discuss fetching issues later.
- ▶ What information do we need?
 - ▶ Let's start from a given artist, and progress along his/her connections, limiting ourselves to some depth.
 - ▶ For each artist, we'll extract:
 - ▶ the name
 - ▶ the genres
 - ▶ "influenced by"
 - ▶ other artists to follow



Technique no. 1 - regexps

- ▶ We'll open the html in some editor,
- ▶ locate the information we want,
- ▶ and construct a regexp to match it.
 - ▶ Rinse and repeat until satisfied



Example

- ▶ First, we'll fetch the page, and open it in an editor

- ▶ links look like:

```
<span class="libg"><a  
href="/cg/amg.dll?p=amg&sql=11:0xfqxqegldde">Alanis  
Morissette</a>
```

- ▶ We'll construct a regexp to catch it:

```
r = '''<span class="libg"><a  
href="( ?P<url>.+? )" >( ?P<name>.+? )</a>'''
```

- ▶ Now, let's find all matches using finditer:

```
l = [m.groupdict() for m in re.finditer(r, d)]
```

- ▶ But, we got more than just the arists! Let's filter:

```
l2 = [x for x in l if 'sql=11' in x['url']]
```



Technique no. 1 - regexps

- ▶ **Pros:**

- ▶ doesn't require any external libraries, very straightforward
- ▶ doesn't depend on the page being correct in any way

- ▶ **Cons:**

- ▶ Slow to implement, hard to read
- ▶ high maintenance



Generic Technique - firebug + any parser

- ▶ Firebug is the ultimate web-development firefox plugin.
- ▶ Method when harvesting:
 - ▶ Use 'inspect' to locate the information we want in the DOM
 - ▶ Use element attributes to describe the information we want
 - ▶ Catch it using the parsed data



Example

Pop/ Rock
' Alternative Pop/
Rock
' Contemporary
Singer/
Songwriter

Moods
' Cathartic
' Intense
' Playful
' Theatrical
' Literate
' Cerebral
' Provocative
' Passionate

Instruments
' Vocals
' Piano
' Keyboards

Other Entries
▪ [Movie Entry](#)
▪ [Classical Music Entry](#)

Member Of
▪ [Y Kant Tori Read](#)

Similar Artists
▪ [Jann Arden](#)
▪ [Fiona Apple](#)
▪ [Suzanne Vega](#)
▪ [10,000 Maniacs](#)
▪ [Rufus Wainwright](#)
▪ [Jeff Buckley](#)
▪ [Rachel Taylor Brown](#)
▪ [Karl Newhouse](#)
▪ [Artemic](#)

Influenced By
▪ [Joni Mitchell](#)
▪ [Kate Bush](#)
▪ [Robert Plant](#)
▪ [Rickie Lee Jones](#)

Followers
▪ [Kristeen Young](#)
▪ [Shannon Wright](#)
▪ [Yvonne Doll & The Locals](#)
▪ [Trophy Wife](#)
▪ [Anny](#)
▪ [Sarah Rabdau](#)
▪ [Jeanne Cherhal](#)
▪ [Christia Mantzke](#)
▪ [Hanne Hukkelberg](#)

HTML CSS Script DOM Net

```
Edit | a < span.libg < li < ul < td < tr < tbody < table#large-list < td < tr < tbody < table < td < tr < tbody < table < td < tr < tbody < tabl
```

```
<li>  
  <span class="libg">  
    <a href="/cg/amg.dll?p=amg&sql=11:3bfoxqrkldae">  
      Trophy Wife </a>  
    </span>  
  </li>  
</li>  
</li>  
</li>  
</li>  
</li>
```

Style Layout DOM

```
#review a.more, #large-list,  
#large-list a, #left-  
sidebar-list a: hover,  
.formed-sub a: hover, .large-  
list-title {  
  color: #6C6823;  
}  
#large-list a {  
  font-family: Trebuchet MS;  
  font-size: 11px;  
  line-height: 15px;  
  text-decoration: underline;  
}
```

blueberry.css (line 97)
new_global.css (line 675)
new_global.css (line 11)

http://allmusic.com/cg/amg.dll?p=amg&sql=11:3bfoxqrkldae

Technique no. 2 - lxml

- ▶ xml & html parser
- ▶ Pros:
 - ▶ very fast
 - ▶ useful searches (using css selectors, etc..)
- ▶ Cons:
 - ▶ Clunky interface
(who wants to learn xpath anyway?)
 - ▶ May choke on slightly malformed input



lxml - cssselect example

```
d = lxml.html.document_fromstring(d)
r = d.cssselect('span.libg')
for x in r:
    a = x.cssselect('a')
    print [(y.attrib['href'], y.text_content()) for y in
           a if 'sql=11' in y.attrib['href']]
```

► Even better:

```
r = d.cssselect('span.libg a')
print [(a.attrib['href'], a.text_content()) for a in links
       if 'sql=11' in a.attrib['href']]
```



Technique no. 3 - beautifulsoup

- ▶ quick & dirty html parser
- ▶ Pros:
 - ▶ Very easy interface
 - ▶ may survive strange tag misplacements
- ▶ Cons:
 - ▶ Slow
 - ▶ Based on HTMLParser, and may choke on input malformed below the tag level
 - ▶ for example, javascript code, with an html tag in quotes
- ▶ We'll skip giving a beautifulsoup example.



My technique - beautifulsoup interface over lxml

- ▶ Thin wrapper around lxml
- ▶ Pros:
 - ▶ Easy interface
 - ▶ As fast as lxml
- ▶ Cons:
 - ▶ Will choke on anything lxml chokes on
 - ▶ Not well maintained
 - ▶ “too easy” interface
- ▶ Example:

```
all_spans = soup.find_all('span', 'libg')
all_links = [s.a for s in all_spans if 'sql=11' in
             s.a.href]
print [(link.all_text(), link.href) for link in
       all_links]
```



Solving the original example

Code run-through

Fetching

- ▶ Some websites require you to stagger fetches
 - ▶ or they'll block you
- ▶ Most of the time, you'll be waiting on IO.
 - ▶ If you don't need to stagger fetches, you can speed things up by using some kind of threaded-map
- ▶ When harvesting from multiple websites, interleave fetches from different websites
 - ▶ generators are your friends
 - ▶ interleave recipe
- ▶ Smart exception handling



Issues not covered here

- ▶ Authentication, forms, cookies, etc:
 - ▶ consider using mechanize. Haven't used it myself though.
- ▶ Legal issues
 - ▶ IANAL
- ▶ Mixing & matching information from different sources
 - ▶ see blog post
- ▶ Unicode



Bonus: most influential artists

- ▶ Starting from Tori Amos as seed, and going 3 levels deep yields about 3000 artists
- ▶ Two most influential ones have significant lead over the rest
- ▶ Can you guess who they are?



Solution:

- ▶ 0.00213240829565 R.E.M.
 - ▶ 0.00213402389235 James Brown
 - ▶ 0.00220035153679 Muddy Waters
 - ▶ 0.0022104694257 Hank Williams
 - ▶ 0.00223807714946 The Who
 - ▶ 0.0022698455533 The Sex Pistols
 - ▶ 0.00242841080591 Sam Cooke
 - ▶ 0.00246575870548 The Beach Boys
 - ▶ 0.00253188657715 Jimi Hendrix
 - ▶ 0.00281672295461 T-Bone Walker
 - ▶ **0.00284912109211 David Bowie**
 - ▶ 0.00286253568126 The Kinks
 - ▶ 0.00305587166572 Joni Mitchell
 - ▶ 0.0033827693424 Little Richard
 - ▶ 0.00349458399127 Elvis Presley
 - ▶ 0.00368460716328 Louis Jordan
 - ▶ 0.00402967429683 The Velvet Underground
 - ▶ 0.00480785924763 The Rolling Stones
 - ▶ 0.00589614847126 Chuck Berry
 - ▶ **0.0111767252846 Bob Dylan**
 - ▶ **0.0130311417336 The Beatles**
-



Bonus: pagerank

- ▶ **Intuitive point of view:**
 - ▶ Each link to a different page is a vote of confidence.
 - ▶ How much is that vote worth?
 - ▶ As much confidence others put in you
- ▶ **Mathematical point of view:**
 - ▶ Starting at a random page, what are the chances a visitor choosing pages at random will land on a specific page?
- ▶ **Overview of the implementation...**

