Information extraction

3. Design considerations, crawling and scraping

Simon Razniewski
Winter semester 2019/20
Announcements

• Assignments
  • Do not plagiarize
  • Submit outputs where asked
• No lecture nor tutorial next week
• Automating extraction?
  • Stay tuned...
• Visualizing KGs
  • https://www.wikidata.org/wiki/Wikidata:Tools/Visualize_data
  • https://angryloki.github.io/wikidata-graph-builder/?property=P40&item=Q3044&iterations=100&limit=100
  • https://angryloki.github.io/wikidata-graph-builder/?property=P737&item=Q937&iterations=100&limit=100
  • https://gate.d5.mpi-inf.mpg.de/webyago3spotlxComp/SvgBrowser/
  • https://developers.google.com/knowledge-graph
Lucius Pinarius - Wikipedia
https://en.wikipedia.org/wiki/Lucius_Pinarius

Lucius Pinarius Scarpus (flourished 1st century BC) was a Roman who lived during the late Republic and the early Empire. He served as the Roman governor of Cyrene, Libya during the Final War of the Roman Republic.

Life • In fiction

Lucius Pinarius Scarpus – Wikipedia

• https://www.reddit.com/r/wikipedia/comments/dg6pnl/thedeadhatedate_of_lucius_pinarius_wasnt_added_so/
• https://www.wikidata.org/wiki/Wikidata:Project_chat#unknown_values_for_people_who_have_long-since_died

Lucius Pinarius Scarpus was a Roman who lived during the late Republic and the early Empire. He served as the Roman governor of Cyrene, Libya during the Final War of the Roman Republic. Wikipedia

Born: 67 BC (age 2,085 years)
Parents: Atia Balba Tertia, Julia Major
Outline

1. Design considerations
2. Crawling
3. Scraping
IE design considerations

1. What should be the output?
   - Type of information
   - Quality requirements

2. What is the best suited input?

3. Which method to get from input to output?
Outline

1. Design considerations
2. Crawling
3. Scraping
Acknowledgment

• Material adapted from Fabian Suchanek and Antoine Amarilli
Web Crawler

A Web crawler is a system that follows hyperlinks, collecting all pages on the way.

Donald John Trump (born June 14, 1946) is the 45th President of the United States.

The United States unites lots of states:
Some of the cooler ones are California and New York.
A crawler does BFS on URLs

1. Start with queue of important URLs

http://...  http://...  http://...
A crawler does BFS on URLs

http://...
http://...
http://...

2. Download

flag.png
A crawler does BFS on URLs

http://...

3. If page is “good”, add it to corpus
A crawler does BFS on URLs

http://...

4. Find URLs in page, enqueue them
A crawler does BFS on URLs

http://...
http://...
http://...

5. repeat the process until you covered all pages
   • within a certain depth
   • in a certain domain
   • with certain topics
   •
Finding new URLs

• In an HTML page
  • Hyperlinks `<a href="...">`
  • Media `<img src="...">, `<audio src="...">, `<video src="...">, `<source src="...">`
  • Frames `<iframe src="...">`
• JavaScript `window.open("...")` — undecidable in general
• Page text by regular expressions.
• In other kinds of files (PDFs...).
• In sitemaps provided specifically to crawlers.
Freshness Problem

• Content on the Web changes
• Different change rates:
  online newspaper main page: every hour or so
  published article: virtually no change
• Continuous crawling, and identification of change rates
  for adaptive crawling:
  If-Last-Modified HTTP feature (not reliable)
  Identification of duplicates in successive request
Freshness problem (2)

• Prediction problem: Estimate page change frequency
  • From previous change behavior
  • Or from page content

• Optimization problem: Decide crawl frequency
  • Fixed budget → How to distribute them
  • Flexible budget → Cost-benefit framework needed
Estimating change frequencies

- Cho and Molina, TOIT 2003
  - Model changes as Poisson processes (i.e., memoryless/statistically independent)
  - Extrapolate change frequency from previous visits
    - Daily visit for 10 days, 6 changes detected
    - Change frequency: 0.6 changes/day?
  - Extrapolation underestimates change frequency due to multiple change possibility

- Liang et al., IJCAI 2017
  - Monitor news websites
  - Build supervised prediction models based on page features

- Wijaya et al., EMNLP 2015
  - Wikipedia-specific
  - Learn state-change-indicating terms
  - E.g., engage, divorce
<table>
<thead>
<tr>
<th>Label</th>
<th>Verb</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>begin-deathdate</code></td>
<td>+(arg1) die on (arg2), +(arg1) die (arg2), + (arg1) pass on (arg2)</td>
</tr>
<tr>
<td><code>begin-birthplace</code></td>
<td>+(arg1) be born in (arg2), +(arg1) bear in (arg2), + (arg1) be born at (arg2)</td>
</tr>
<tr>
<td><code>begin-predecessor</code></td>
<td>+(arg1) succeed (arg2), +(arg1) replace (arg2), + (arg1) join cabinet as (arg2), +(arg1) join as (arg2)</td>
</tr>
<tr>
<td><code>begin-successor</code></td>
<td>+(arg1) lose seat to (arg2), +(arg1) resign on (arg2), + (arg1) resign from post on (arg2)</td>
</tr>
<tr>
<td><code>begin-termstart</code></td>
<td>+(arg1) be appointed on (arg2), +(arg1) serve from (arg2), + (arg1) be elected on (arg2)</td>
</tr>
<tr>
<td><code>begin-spouse</code></td>
<td>+(arg1) marry on (arg2), +(arg1) marry (arg2), + (arg1) be married on (arg2), -(arg1) be engaged to (arg2)</td>
</tr>
<tr>
<td><code>end-spouse</code></td>
<td>+(arg1) file for divorce in (arg2), +(arg1) die on (arg2), + (arg1) divorce in (arg2)</td>
</tr>
<tr>
<td><code>begin-youthclubs</code></td>
<td>+(arg1) start career with (arg2), + (arg1) begin career with (arg2), + (arg1) start with (arg2)</td>
</tr>
</tbody>
</table>
Distributing crawl resources
[Razniewski, CIKM 2016]

• Ingredients:
  • Benefit of an up-to-date website
    • Synonymous: cost of outdated website
  • Cost of a crawl action
  • Decay behavior

→ Page-specific recrawl frequency that maximizes benefit minus cost
Decay behaviour

\[ z_{\text{exp}}(t) = e^{-\lambda_{\text{exp}} t}. \]

\[ z_{\text{lin}}(t) = \max(1 - \lambda_{\text{lin}} t, 0). \]
Observed decay behaviour

Figure 7: Decay behaviour of soccer players at Manchester United (blue) and Bayern München (red), observed (solid lines), and approximated by exponential decay curves with $\lambda = 0.26$ and 0.36, respectively (dashed lines).
Average freshness $F$

\[
F(\lambda, u) = \frac{\int_0^u z(t) \, dt}{u}.
\]

\[
F_{\text{lin}}(\lambda, u) = 1 - \frac{\lambda \cdot u}{2}.
\]

\[
F_{\text{exp}}(\lambda, u) = \frac{1 - e^{-\lambda u}}{\lambda \cdot u}.
\]
Net income $NI$

- $B$... Benefit/time unit
- $F$... Average freshness
- $\Lambda$... Decay coefficient
- $u$... Update interval length
- $C$... Cost of an update

**Formulae**

- \[
    NI(u) = B \cdot F(\lambda, u) - \frac{C}{u}.
\]

- \[
    NI_{\text{lin}}(u) = B - \frac{B \cdot \lambda \cdot u}{2} - \frac{C}{u}.
\]

- \[
    NI_{\text{exp}}(u) = B \frac{1 - e^{-\lambda u}}{\lambda \cdot u} - \frac{C}{u}.
\]

**Optimum via common algebra**
Examples for address updates

Assumption: benefit over one year = 100 \times \text{cost of single crawl}

Actual ratio magnitudes lower, e.g., 0.003 Cents/crawl


(and for 580 \$ on Amazon EC2)
Duplicate pages

- Prevent multiple indexing and penalize content farms.
- Prevent duplicate URLs by canonicalization.
  
  http://example.com:80/foo

  = http://example.com/bar/../foo

  = http://www.example.com/foo

- Detect duplicate pages by using a hash function.
- Detect near-duplicates (dates, etc.) by using a similarity function.
  (e.g., Broder's MinHash from 1997, used in AltaVista and later Google)
Crawl scheduling

- Wait a minimal delay between requests to the same server.
  - Depends on the server (wikipedia.org vs your laptop).
  - Depends on the resource (large files...).
  - Generally, waiting at least one second is preferable.
- Requests to different servers can be parallelized.
- Requests should be run asynchronously.
- The HTTP connection should remain open.
- Requests can be distributed across multiple machines.
- Crawlers represent about 20% of Web traffic.
Crawler traffic

Traffic on a3nn.net as of September 2013 (out of 36593 requests).
Robot control (honor-based)

- **Robot Exclusion Standard**: http://example.com/robots.txt
  - Only at root level (not available for subfolders).
  - Filtering by **User-agent**.
  - **Disallow** directive to forbid certain pages.
  - Also: **Allow**, **Crawl-delay**, **Host**, **Sitemap**.

- **HTTP header**: **X-Robots-Tag** (less support):
  - **X-Robots-Tag**: noindex

- **Meta tag**: `<meta name="robots" content="noindex">`
  - Also **nofollow**, **nosnippet**, **noarchive**...

- **Links**: `<a href="secret/" rel="nofollow">`

- **Engine-specific** interfaces (e.g., Google Webmaster Tools).

  - **No guarantees**!

  https://www.mpi-inf.mpg.de/robots.txt
  https://www.google.de/robots.txt
Robot control with CAPTCHAs

How can we discriminate against robots?

• Completely Automated Public Turing test to tell Computers and Humans Apart (trademarked by CMU, but patented by AltaVista).
• Making a computer able to recognize humans.
• Can be any AI problem: add two numbers, listen to a word, recognize an animal in an image, etc.
ReCAPTCHAs

CAPTCHAs can be used to
• digitize books
  Show one word that we know (to validate the user),
  and one word that we want to digitize (to digitize the book)

- following
- finding

• Show ads
  Ask the user to type a slogan

• Do recognition of street numbers in
  Google street view images
Breaking CAPTCHAs

• Employ humans to remotely solve CAPTCHAs ("sweatshops", hundreds per hour)

• Sometimes there may be no ground truth → Try often enough

• Optical character recognition has improved and can solve some CAPTCHAs
“Robot Control” by Spider Traps

A spider trap (also: crawler trap, robot trap) is a set of web pages that cause a web crawler to make an infinite number of requests or cause a poorly constructed crawler to crash.

Example:

January 1st
• no meetings

Spider traps can be intentional or unintentional. Can be used to trap spiders that do not follow robots.txt :-)

http://foo.com/bar/foo/bar/foo/bar/foo/bar/.....
Deep web / dark web

• Pages that have no links to them.
• For instance, result pages from a search.
• 2001 estimate: the deep Web is hundreds of times larger than the reachable Web.
• Web form probing:
  ⇒ Need to figure out form constraints.
  ⇒ Need to come up with keywords.
  ⇒ Idea: feed back words from the website into the form.

We can use an existing Web crawl

<table>
<thead>
<tr>
<th>Pages</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClueWeb</td>
<td>1b</td>
</tr>
<tr>
<td>CommonCrawl</td>
<td>6b</td>
</tr>
<tr>
<td>Internet Archive</td>
<td>2b</td>
</tr>
<tr>
<td>enWikipedia</td>
<td>5m</td>
</tr>
<tr>
<td>Dresden web table corpus</td>
<td>125m</td>
</tr>
<tr>
<td>Twitter dumps 2016 US election</td>
<td>280m</td>
</tr>
<tr>
<td>Reddit dumps</td>
<td>...</td>
</tr>
<tr>
<td>Wikia dumps</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Insights from crawling mpi-inf.mpg.de

- URL ending *inclusion/exclusion* criteria need thought
- Long (machine-generated URLs) need exclusion
- Beyond that no issues
- 35 lines in Python
- Sequential runtime for 2000 pages: ~10 minutes
- Completeness?
Outline

1. Design considerations
2. Crawling
3. Scraping
Inputs

- Semi-Structured Data (Infoboxes, Tables, Lists ...)
- Text Documents & Web Pages
- Conversations & Behavior

Premium Sources (Wikipedia, IMDB, ...)
Web collections (Web crawls)

Methods

- Rules & Patterns
- Logical Inference
- Statistical Inference
- NLP Tools
- Deep Learning

Outputs

- Entity names, aliases & classes
- Entities in Taxonomy
- Rules & Constraint
- Relational Statements
- Canonicalized Statements

Scraping
Generated Web pages

Web page generation is the process of producing several similar Web pages from a KB.
Example: Generated Web pages

price 10 USD
author William Steig

"Shrek"
by W. Steig
only 10 USD!
Buy it!
Scraping aims to reconstruct the KB
Def: Wrapper

A wrapper for a set of pages generated from the same KB is a function that extracts strings from such a page.

(Technically, it is the inverse function of the function that generated the page. The strings still have to be disambiguated and put in relation to yield facts. Different applications have different more specific definitions of the "strings".)

Kushmerick: Wrapper Induction
Information is always in same place

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Genre</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Shrek - Der tollkühne Held</em></td>
<td>2001</td>
<td>Animation, Adventure, Comedy</td>
</tr>
<tr>
<td><em>Elvis: Aloha from Hawaii</em></td>
<td>TV 1973</td>
<td>Documentary, Music</td>
</tr>
</tbody>
</table>

*Shrek* - Der tollkühne Held (2001)

- **Your rating:** 7.9/10
- **Ratings:** 7,000 from 294,955 users
- **Metascore:** 84/100
- **Reviewed:** 94 user, 209 critic

An ogre, in order to regain his swamp, travels along with an annoying donkey in order to bring a princess to a scheming lord, wishing himself king.

- **Directors:** Andrew Adamson, Vicky Jenson
- **Writers:** William Steig (book), Ted Elliott (screenplay)
- **Stars:** Mike Myers, Eddie Murphy, Cameron Diaz

*Elvis: Aloha from Hawaii* (TV 1973)

- **Your rating:** 7.0/10
- **Ratings:** 1,928 from 582 users
- **Reviews:** 29 user, 3 critic

A 1973 concert by Elvis Presley taped at the Convention Center in Honolulu, Hawaii. This was the first program to ever be beamed around the world by satellite.

- **Directors:** Marly Pessetta, Gary Hevey
- **Stars:** Elvis Presley, James Burton, Jerry Schloff
Def: XPath

XPath is a formal language for selecting nodes in an XML document.

/ identifies the root node
K/T[i] identifies the i-th child with tag T of the node identified by K
K/T is K/T[1] if K has one T child

<html>
  <body>
    <h1>Aloha from Hawaii</h1>
    <p>This is a really great movie</p>
    <p>Stars:<i>Elvis Presley</i></p>
  </body>
</html>

[https://www.w3schools.com/xml/xml_xpath.asp]
[https://devhints.io/xpath]
Task: XPath

Write XPath expressions that identify nodes whose text is "Shrek", "W. Steig", and "84 min".

<html>
<body>
<b>Shrek</b>
<ul>
<li>Creator: <b>W. Steig</b></li>
<li>Duration: <i>84m</i></li>
</ul>
</body>
</html>
Scraping: Browser

• “Try XPath” Firefox addin

• `//h3[@class='pi-data-label pi-secondary-font']`

• Firefox console
  • `$x('//h3[@class='pi-data-label pi-secondary-font']')`

• `//h3[@class='pi-data-label pi-secondary-font'] //div[@class='pi-data-value pi-font']`
Scraping in Python - XPath

```python
# from https://lxml.de/parsing.html#parsing-html

import requests
import lxml
from lxml import etree

url='https://lotr.fandom.com/wiki/Frodo_Baggins'

req = requests.get(url)

html = etree.HTML(req.text)

output = html.xpath('//h3[@class="pi-data-label pi-secondary-font"]')

for e in output:
    print(e.text)
```

Other names
Titles
Birth
Death
Weapon
Race
Hair
Eyes
Culture
Actor
Def: Wrapper induction

Wrapper induction is the process of generating a wrapper from a set of Web pages with strings to be extracted.

```
<html/body/h1
/html/body/p[2]/i
```

Web page

+ "Shrek", "7.9"

= Strings to be extracted

= Wrapper
Wrapper induction

Wrapper Induction requires as input Web pages with strings to be extracted. These can come, e.g.,
• from a KB

    hasTitle(ShrekMovie, "Shrek")

• from manual extraction

• from manual annotation in a GUI
Detail Pages & List Pages

Wrappers can be learned across several detail pages:

Wrappers can also be learned across items in a list:
Data may exhibit structure

Dronkeys:
<ul>
<li>Eclair: female</li>
<li>Bananas: flexible</li>
</ul>

Shrek's kids:
<ul>
<li>Farkle: male</li>
<li>Fergus: male</li>
</ul>

```
family: tuple (  
    name: string  
    children: set (  
        child: tuple (name: string,  
                     gender: string)))```
ROADRUNNER: Learn types

ROADRUNNER is a system that can learn the Web page structure. Finds least upper bounds in regex lattice

Page 1:
<ul>
<li>Peanut</li>
</ul>

Page 2:
<ul>
<li>Charles</li>
</ul>

Wrapper:
<ul>
<li>[FIELD]</li>
</ul>

Crescenzi et al., VDLB 2001
ROADRUNNER: Learn types

ROADRUNNER is a system that can learn the Web page structure.

Page 1:
<ul>
<li>Peanut
</ul>

Page 2:
<ul>
<li>Charles
<li>Anne
</ul>

Wrapper:
<ul>
<li>(<li>[FIELD])+</ul>

Crescenzi et al., VDLB 2001
Def: Wrapper Application

Wrapper application is the process of extracting its strings from a Web page.

Web page

+ 
/html/body/h1
/html/body/p[2]/i

= 
"Elvis", "11"

Wrapper

= Strings
Def: Wrapper Application

Wrapper application is the process of extracting its strings from a Web page.

Web page

```
<html/body/h1
/html/body/p[2]/i

= "Elvis", "11"

Disambiguation, + relation

hasActor(e42, ElvisPresley)
hasRating(e42, "11.0")

Strings

Facts
```
Scrapping in Python — BeautifulSoup (1)

- Python library for pulling data out of HTML and XML files.

```html
<html>
<head>
<title>
The Dormouse's story
</title>
</head>
<body>
  Once upon a time there were three little sisters; and their names were <a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>, <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a> and ...

  soup.title
  # <title>The Dormouse's story</title>

  soup.title.string
  # u'The Dormouse's story'

  soup.title.parent.name
  # u'head'

  soup.a
  # <a class="sister" href="http://ex.com/elsie" id="link1">Elsie</a>

  soup.find_all('a')
  # [<a class="sister" href="http://ex.com/elsie" id="link1">Elsie</a>,
  #  <a class="sister" href="http://ex.com/lacie" id="link2">Lacie</a>,
  #  <a class="sister" href="http://ex.com/tillie" id="link3">Tillie</a>]
```
from bs4 import BeautifulSoup
import urllib3
import requests
from urllib.request import urlopen

site = "http://en.wikipedia.org/wiki/Max_Planck_Institute_for_Informatics"

page = requests.get(site, verify=False)
soup = BeautifulSoup(page.text, 'html.parser')
table = soup.find('table', class_='infobox vcard')

for tr in table.find_all('tr'):
    if tr.find('th'):
        print(tr.find('th').text + " : " + tr.find('td').text)

Abbreviation: MPI-INF
Formation: 1993; 26 years ago (1993)
Type: research institute
Headquarters: Saarbrücken, Saarland, Germany
Website: www.mpi-inf.mpg.de
XPath vs. BeautifulSoup vs ...

• **XPath**: Generic query language to select nodes in XML (HTML) documents
  • Queries can be issued from Python, Java, C, ...

• **BeautifulSoup**
  • Python library to manipulate websites as Python objects

• **Scrapy**
  • Python library to crawl websites

• **Selenium**
  • Actual scripted browser interaction
  → To get around Javascript etc.

Assignment 3

• No crawling (ethics...)
• 1x Extraction from dump — infobox treasure
  • Remember design considerations
  • XML format, but essential content not structured by XML tags
    → pattern matching/regex
• 2x Scraping
  • BeautifulSoup recommended, but XPaths are also fine as well

• Reading on large-scale WP extraction:
  DBpedia extraction framework
Take home

1. Think about goal, sources, methods
2. Crawling
   • BFS to achieve coverage
   • Challenges with traps and deep web
3. Scraping
   • Reverse-engineering of template-based websites