Informationsextraktion aus Websites

Michael Haas <haas@computerlinguist.org>

Service-Center Forschungsdaten, Universität Mannheim

22.01.2013

Lessons Learned - Kontext

- Mein Hintergrund: B.A. Computerlinguistik, Universität Heidelberg
- Projekte am Service-Center:
  - Manuel Trenz: Beobachtung der Preisveränderungen einer gegebenen Menge an Produkten auf Online-Shops und Preisvergleichern
  - Dominic Nyhuis: Durchsuchen der Online-Archive von 10 Zeitungen per Screen Scraping
  - Georg Wernicke: NER und Sentiment-Analyse auf Zeitungsartikeln
- Ziele für heute
  - Tutorial Screen Scraping
  - Folien als Referenz
  - Lessons Learned als Hinweise/best practices

Aufgabe

- Kunde benötigt Daten von Website
- Manuelle Extraktion mit HiWis und Copy&Paste zu aufwendig
- Automatisieren!

Python

Konkret: Kunde möchte Produktpreise über längeren Zeitraum überwachen

```
1 >> import urllib2
2 >> content = urllib2.urlopen("http://host/produkt/id")
3 HTTPError: HTTP Error 403: denied - contact webmaster@host
```

Python - Ninja Level 1

Website mag unseren User-Agent nicht!

```
1 >> request = urllib2.Request("http://host/produkt/123")
2 >> request.add_header('User-Agent', 'Mozilla/5.0')
3 >> opener = urllib2.build_opener()
4 >> content = opener.open(request).read()
5 >> content[0:30]
6 '<!DOCTYPE HTML><html lang="de"'
```

Python - Iteration

Kunde will mehrere Produkte überwachen

```
>> for p in products*100:
    request = urllib2.Request("http://host/produkt/" + p)
    request.add_header('User-Agent', 'Mozilla/5.0')
    opener = urllib2.build_opener()
    content = opener.open(request).read()
HTTPError: HTTP Error 421: too fast - contact webmaster@host
```

Python - Ninja Level 2

```
1 >> import time
2 >> for p in products*100:
3        request = urllib2.Request("http://host/product/" + p)
4        request.add_header('User-Agent', 'Mozilla/5.0')
5        opener = urllib2.build_opener()
6        content = opener.open(request).read()
7        time.sleep(5)
```

Python - Paranoid Ninja

Admin könnte Access Logs überwachen - Abstände der Zugriffe zufällig halten!

```
>> import random,time
>> for p in products*100:
     request = urllib2.Request("http://host/product/" + p)
     request.add_header('User-Agent', 'Mozilla/5.0')
     opener = urllib2.build_opener()
     content = opener.open(request).read()
     time.sleep(random.uniform(1,5))
```

Python - there is a lib for that

Alles zu kompliziert! Besser:

```
1 $ sudo easy_install-2.7 leechi
2 $ ipython2
3 >> import leechi
4 >> l = leechi.Leechi()
5 >> for p in product:
6      content = l.fetchDelayed("http://host/product/" + p)
```

Python - Leechi

```
>> l = leechi.Leechi(cookies=True, retry=3)
>> l.chooseRandomUA()
>> l.setCustomUA("Wget/1.9.1")
>> handle = l.obtainHandle("http://host/")
>> content = handle.read()
>> handle = l.obtainHandleDelayed("http://host/")
```

Python - Leechi - Source

- http://github.com/mhaas/leechi/
- http://pypi.python.org/pypi/Leechi/0.2
- Send Patches:
  - Periodisches Wechseln von UA
  - Unterstützung (anonymer) Proxy-Server
  - Tests

Python - HTML Parsing

Und nun?

```
>> content = """<html><body>
                <p class="price">preis ist: 5euro</p>
                </body></html>"""
>> import re
>> re.search(ur'preis ist: (\d{0,4})euro', content).group(1)
'5'
```

Python - HTML Parsing - Nie RegEx

- HTML/XML sind kontextfreie Sprachen
- Reguläre Ausdrücke beschreiben reguläre Sprachen
- Kontextfreie Sprachen sind mächtiger als reguläre Sprachen[1]

[1] "Parsing HTML with regex summons tainted souls into the realm of the living." http://stackoverflow.com/a/1732454

Python - HTML Parsing - BeautifulSoup

- Besser: BeautifulSoup 4
- Kann alles, auch Tagsuppe:
  - fehlende schließende Tags
  - mangelhaft kodierte Sonderzeichen
- http://www.crummy.com/software/BeautifulSoup/

Python - HTML Parsing - BeautifulSoup - Navigation

HTML-Attribut *class* sehr nützlich als Ziel.

```
1  $ sudo easy_install-2.7 beautifulsoup4
2  $ python2
3  >> content = """<html>
4                    <body>
5                     <p class="price">preis ist: 5e</p>
6                    </body>
7                   </html>"""
8  >> from bs4 import BeautifulSoup
9  >> soup = BeautifulSoup(content)
10 >> soup.find(class_="price")
11 <p class="price">preis ist: 5e</p>
```

Python - HTML Parsing - BeautifulSoup - Navigation

Container für Listen

```
>> content = """<ul id="priceList">
                <li>preis: 5e</li>
                <li>preis: 10e</li>
                <li>preis: 15e</li>
                </ul>"""
>> priceList = soup.find('ul', id='priceList')
>> for node in priceList.children: # priceList.contents
    re.search(ur"preis: (\d{1,4})e", node.string).group(1)
5
10
15
```

Python - HTML Parsing - BeautifulSoup - Navigation

Durch den Baum hangeln

```
>> content = """<div>
                    <h1 class="section-header">Preise</h1>
                    <h3>Zubehoer</h3>
                    <ul>
                     <li>preis: 5e</li>
                    </ul>
                   </div>"""
>> soup.div.h1.nextSibling.nextSibling
<ul><li>preis: 5e</li></ul>
>> soup.div.contents[2]
<ul><li>preis: 5e</li></ul>
>> soup.find(class_="section-header").nextSibling.nextSibling
<ul><li>preis: 5e</li></ul>
```

Python - HTML Parsing - BeautifulSoup - Navigation

```
>> for string in soup.strings:
    print string
Preise
Zubehoer
preis: 5e
```

- *soup.stripped_strings*: ohne Leerzeichen
- *soup.descendants*: depth-first search

BeautifulSoup - Suchen

- Nach Tag-Namen
- Nach Attributen
- Nach Text
- Kombinationen

Python - HTML Parsing - BeautifulSoup - Suchen - Tag

```
1 >> soup.h1
2 <h1 class="section-header">Preise</h1>
3 >> soup.find('h1')
4 <h1 class="section-header">Preise</h1>
5 >> soup.find_all('h1')[0]
6 <h1 class="section-header">Preise</h1>
```

Python - HTML Parsing - BeautifulSoup - Suchen - Attribut

```
>> content = """<div>
                    <h1>Preise</h1>
                    <h3 id="header">Zubehoer</h3>
                    <ul>
                     <li>preis: 5e</li>
                    </ul>
                   </div>'
>> soup.find(id="header")
<h3 id="header">Zubehoer</h3>
>> soup.find("h3", id="header")
<h3 id="header">Zubehoer</h3>
>> soup.find(id=re.compile('head'))
<h3 id="header">Zubehoer</h3>
>> soup.find(id=re.compile('head'))["id"]
 'header'
```

Python - HTML Parsing - BeautifulSoup - Suchen - Text

```
>> soup.find(text="Zubehoer")
u'Zubehoer'
>> soup.find(text="Zubehoer").parent
<h3 id="header">Zubehoer</h3>
>> soup.find_all(text=True)
[u'Preise', u'Zubehoer', u'preis: 5e']
>> soup.find_all(text=re.compile("[pP]reis"))
[u'Preise', u'preis: 5e']
```

Zwischenstand

Wir können:

- unerkannt Content herunterladen
- Content parsen und Information extrahieren

Spezialfall: Suchanfragen auf Websites automatisieren!

Suchmasken - Automatisierung von Formularen

- Suchmasken sind *<form>*-Objekte mit *<input>*-Feldern
- Übermittlung per HTTP GET oder POST
- Achtung: benötigt oft Cookies für Session Management!

```
1 Leechi(cookies=True)
```

Suchmasken - Automatisierung von Formularen

http://de.wikipedia.org/wiki/Spezial:Suche

```
1 <form id="search" method="get" action="/w/index.php">
2     <input value="Spezial:Suche" name="title" type="hidden"
          />
3     <input value="default" name="profile" type="hidden" />
4     <input id="searchText" name="search" />
5     <input value="Search" name="fulltext" type="hidden" />
6     <input type="submit" value="Volltext" />
7 </form>
```

Suchmasken - GET Request

http://de.wikipedia.org/w/index.php?title=Spezial%3ASuche&profile=def

```
>> import urllib
>> params = {   "title" : "Spezial:Suche",
                "fulltext" : "Search",
                "search" : "katze",
                "profile": "Default" }
>> urllib.urlencode(params)
'profile=Default&fulltext=Search&search=katze&title=Spezial
    %3ASuche'
>> content = l.fetchDelayed("http://de.wikipedia.org/w/index
    .php?" + urllib.urlencode(params))
```

Suchmasken - POST Request

- POST als Request Method
- Parameter als separaten Wert übergeben

```
1 >> l.obtainHandleDelayed(baseURL, urllib.urlencode(params))
```

Suchmasken - Wikipedia - Pagination

Suchergebnisse über mehrere Seiten verteilt

- Zusätzliche Parameter
  - *offset*
  - *limit*

```
>> params = {    offset : "20",
                 limit : "20",
                 "title" : "Spezial:Suche",
                 "fulltext" : "Search",
                 "search" : "katze",
                 "profile": "Default" }
```

http://de.wikipedia.org/w/index.php?title=Spezial:Suche&limit=20&offse

Suchmasken - Wikipedia - Alle Links extrahieren

Beispiel: extrahiere alle Links aus Suchergebnis

```
>> while has_more_results():
    params["offset"] = offset
    content = l.fetchDelayed("http://..?" + urllib.urlencode
        (params))
    soup = BeautifulSoup(content)
    for node in soup.find_all("div", class\_="mw-search-
        result-heading"):
        results.append(node.a["href"])
    offset += 20
```

Suchmasken - has_more_results()?

Wann haben wir alle Ergebnisse gesehen?

- Server liefert auf letzter Seite weniger Ergebnisse als *limit*
- Server liefert Fehler 404, 500 bei zu großem *limit*
- Server liefert doppelte Ergebnisse bei zu großem *limit*
- Am Besten: Vergleich mit Anzahl Ergebnisse - nicht immer korrekt
- Keine allgemeingültige Formel!

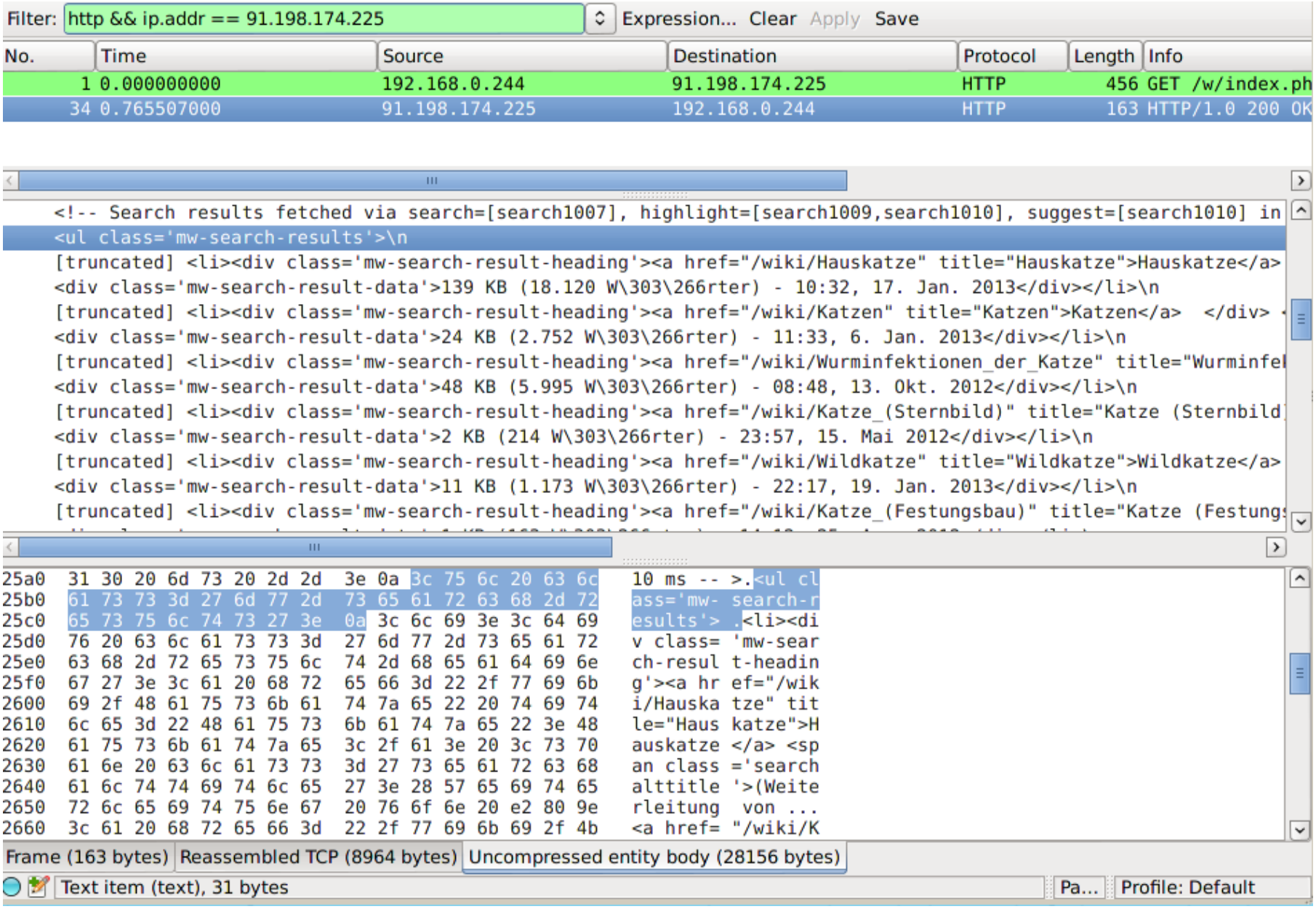
Zusammenfassung

- Kommunikation mit Server über Leechi
- Extraktion von Informationen aus DOM-Baum über BeautifulSoup
- Suchmasken: GET/POST requests mit allen Parametern, Cookies!
- Ende der Ergebnisliste: here be dragons!

Werkzeugkasten

- Python, BeautifulSoup, Leechi
- Browser: “View Source”, DOM Inspector
- Firefox: Web Console, Live HTTP Headers
- Wireshark

Wireshark


Filter: http && ip.addr == 91.198.174.225
Expression... Clear Apply Save
No. Time Source Destination Protocol Length Info
1 0.000000000 192.168.0.244 91.198.174.225 HTTP 456 GET /w/index.ph
34 0.765507000 91.198.174.225 192.168.0.244 HTTP 163 HTTP/1.0 200 OK
<!-- Search results fetched via search=[search1007], highlight=[search1009,search1010], suggest=[search1010] in
<ul class='mw-search-results'>\n
[truncated] Hauskatze
139 KB (18.120 W\303\266rter) - 10:32, 17. Jan. 2013 \n
[truncated] Katzen
24 KB (2.752 W\303\266rter) - 11:33, 6. Jan. 2013 \n
[truncated] <a href="/wiki/Wurminfektionen_der_Katze" title="Wurminfe
48 KB (5.995 W\303\266rter) - 08:48, 13. Okt. 2012 \n
[truncated] <a href="/wiki/Katze_(Sternbild)" title="Katze (Sternbild
2 KB (214 W\303\266rter) - 23:57, 15. Mai 2012 \n
[truncated] Wildkatze
11 KB (1.173 W\303\266rter) - 22:17, 19. Jan. 2013 \n
[truncated] <a href="/wiki/Katze_(Festungsbau)" title="Katze (Festung
25a0 31 30 20 6d 73 20 2d 2d 3e 0a 3c 75 6c 20 63 6c 10 ms -- >.<ul cl
25b0 61 73 73 3d 27 6d 77 2d 73 65 61 72 63 68 2d 72 ass='mw- search-r
25c0 65 73 75 6c 74 73 27 3e 0a 3c 6c 69 3e 3c 64 69 esults'> .
<di
25d0 76 20 63 6c 61 73 73 3d 27 6d 77 2d 73 65 61 72 v class= 'mw-sear
25e0 63 68 2d 72 65 73 75 6c 74 2d 68 65 61 64 69 6e ch-resul t-headin
25f0 67 27 3e 3c 61 20 68 72 65 66 3d 22 2f 77 69 6b g'><a hr ef="/wik
2600 69 2f 48 61 75 73 6b 61 74 7a 65 22 20 74 69 74 i/Hauska tze" tit
2610 6c 65 3d 22 48 61 75 73 6b 61 74 7a 65 22 3e 48 le="Haus katze">H
2620 61 75 73 6b 61 74 7a 65 3c 2f 61 3e 20 3c 73 70 auskatze <sp
2630 61 6e 20 63 6c 61 73 73 3d 27 73 65 61 72 63 68 an class ='search
2640 61 6c 74 74 69 74 6c 65 27 3e 28 57 65 69 74 65 alttitle '>(Weite
2650 72 6c 65 69 74 75 6e 67 20 76 6f 6e 20 e2 80 9e rleitung von ...
2660 3c 61 20 68 72 65 66 3d 22 2f 77 69 6b 69 2f 4b <a href= "/wiki/K
Frame (163 bytes)
Reassembled TCP (8964 bytes)
Uncompressed entity body (28156 bytes)
Text item (text), 31 bytes
Pa...
Profile: Default

Lessons Learned - Encodings

- Encoding matters: Parameter passend kodieren
- Encoding aus HTTP-Headern: *charset*-Parameter in *Content-Type*
- Alternative: *<meta http-equiv="Content-Type" ...>*
- Achtung: "*latin*-1" bedeutet oft "*cp1252*"

```
1 >> params = {"query" : u"Müller".decode("utf-8")}
2 >> l.fetchDelayed("http://host/", urllib.urlencode(params))
```

Lessons Learned - Tests

- Umfangreichere Projekte: Unit-Test-Frameworks!

```
1 >> items = crawl("Katze", from="01.01.", to="01.03.")
2 >> assert(len(items) == 84)
```

- Alternative: unglücklicher Kunde, unglücklicher Entwickler

Lessons Learned - Parsers

- Nicht jeder Parser in BeautifulSoup4 kommt mit jeder Tagsuppe klar [2]
- Gut, aber langsam: *html5lib* - *BeautifulSoup(content, "html5lib")*
- Auch *html5lib* funktioniert nicht immer; *lxml* auch brauchbar
- Parsing-Fehler subtil - DOM-Tree laden und wieder serialisieren
- Tests schreiben!

[2] http://www.crummy.com/software/BeautifulSoup/bs4/doc/#installing-a-parser

Lessons Learned - Session Management

- Session Management für einige Websites notwendig
- Session Management per Cookie
- Oder: Per Parameter in GET-Request - muss dann extrahiert & übergeben werden!
- Selten: verpflichtende, wechselnde Parameter (form input) - muss bei jedem Seitenwechsel extrahiert werden

Bonus: AJAX

- Keine Daten im HTML-Source, aber im DOM-Baum
- Dynamische Website mit AJAX
- Content wird dynamisch nachgeladen als JSON oder XML und DOM-Baum modifiziert
- URL finden:
  - Javascript-Code lesen
  - Wireshark
  - Browser-Extension

Bonus: AJAX

```
1 >> import json
2 >> content = urllib.urlopen('https://ajax.googleapis.com/
       ajax/services/feed/load?v=1.0&q=http://www.digg.com/rss/
       index.xml').read()
3 >> data = json.loads(content)
4 >> data["responseData"]["feed"]["author"]
5 u'Digg'
```