Part I: Acquiring Data from the Web

Solomon Messing

Department of Communication, Statistics
Stanford Social Science Data and Software (SSDS)

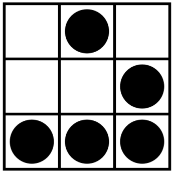
September 24, 2012

Overview

- How to get data from the web (cURL, APIs, JSON, XML)
- Extracting useful stuff from the results (Regex)
- Representing text as data
- Getting meaningful features (Regex, R)
- Document-term matrices
- Supervised and unsupervised approaches to analysis

What to expect

- Goal: Quantitative insight from haystack of messy data
- Labs: programming, lots of R!
- Working with your neighbors
- Adapting starter code + code found on Google, StackOverflow, etc.
- You learn better, I talk less, you have code to work with.



Motivation

- Massive growth in availability of text and other data
- 100K Tweets per min, 700K FB shares per min (DOMO), 200M emails—10 min = 1 LOC (Huggins)
- Proliferation of structured/semi-structured data at your fingertips: open-data, APIs, and scrape-able data sources
- Growth of open-source tools to acquire and analyze this data

Motivation: what's out there?

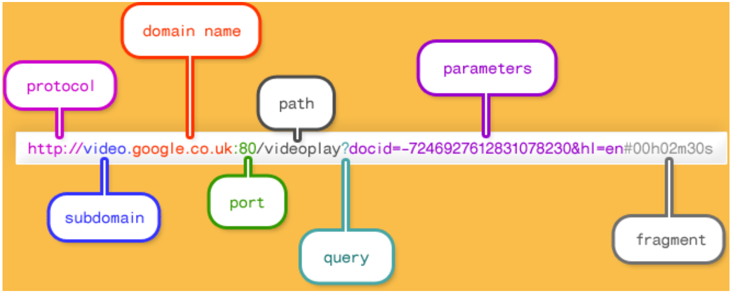
- Raw text, html tables, semi-structured HTML/XML.
- Data marketplaces, e.g.:
  http://www.infochimps.com/datasets
- APIs:
  - https://dev.twitter.com/
  - http://developer.nytimes.com/ articles, campaign finance, congress, entities, geography/population
  - http://developer.washingtonpost.com/ political speeches, campaign finance, White House visitors log
  - http://developer.yahoo.com/everything.html - search, finance, geo-coding, on-the-fly entity extraction, content analysis, term extraction.
  - https://api.facebook.com - access (a little) Facebook data.[1]

[1] e.g., get aggregate likes for NYTimes.com article

Data from the Web

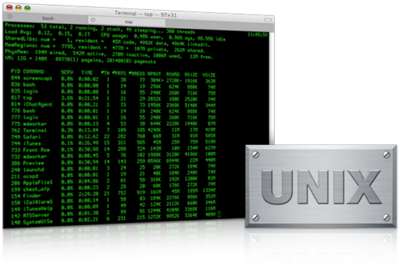
1. Hit a server
2. Parse it's response
3. Clean and transform into something useful
4. Often by merging it with something else.

Hit a server


domain name
protocol
path
parameters
http://video.google.co.uk:80/videoplay?docid=-7246927612831078230&hl=en#00h02m30s
subdomain
port
query
fragment

from: http://doepud.co.uk/blog/anatomy-of-a-url.php

Hit a server, in *.NIX



```
Processes:  53 total, 2 running, 2 stuck, 49 sleeping... 300 threads                11:05:59
Load Avg:  0.12,  0.15,  0.17    CPU usage:  0.48% user,  0.96% sys, 98.55% idle
SharedLibs: num =    1, resident =   45M code, 4992K data, 4964K linkedit.
MemRegions: num =  7795, resident =  477M +  107M private,  262M shared.
PhysMem:  194M wired,  542M active,  270M inactive, 1006M used,   12M free.
VM: 12G + 140M   89770(1) pageins, 201400(0) pageouts

  PID COMMAND      %CPU   TIME   #TH #PRTS #MREGS RPRVT  RSHRD  RSIZE  VSIZE
  849 screencapt   0.0%  0:00.02   1    38     77  304K+ 2720K- 1916K   363M
  836 bash         0.0%  0:00.00   1    14     19  256K   624K   888K    74M
  835 login        0.0%  0:00.00   1    16     55  240K   260K   716K    75M
  817 top          3.6%  0:21.54   1    21     29 2032K   188K  2520K    74M
  814 iChatAgent   0.0%  0:00.21   2    73     73 1956K  2368K  5140K   344M
  778 bash         0.0%  0:00.01   1    14     19  240K   624K   888K    74M
  777 login        0.0%  0:00.01   1    16     55  240K   260K   716K    75M
  775 mdworker     0.0%  0:00.13   4    53     38  644K  2220K  1940K    87M
  762 Terminal     0.3%  0:13.04   7   109    185 4240K    11M    17M   419M
  749 Safari       0.0%  0:12.62  22   282    760   66M    31M    91M   585M
  744 iTunes       0.1%  0:26.49  15   311    585   45M    25M    75M   519M
  733 Front Row    0.1%  0:30.98  14   288    724  143M    10M   154M   627M
  732 mdworker     0.0%  0:00.45   5    76    182 1988K  3120K  4196K   100M
  306 Preview      0.0%  0:30.94  14   143    259 8596K  6944K    22M   440M
  240 launchd      0.0%  0:00.01   3    24     25   20K   272K   184K    74M
  211 ocspd        0.0%  0:00.01   1    19     21   40K   184K   240K    74M
  206 AppleFileS   0.0%  0:04.06   2    61     58  316K   192K  1200K    83M
  199 check_afp    0.0%  0:00.23   2    25     20   68K   176K   272K    74M
  154 Finder       0.0%  2:29.20  21   752    919  161M    45M   185M  1334M
  150 iCalAlarmS   0.0%  0:00.14   1    58     83  184K  2276K   996K   352M
  149 iTunesHelp   0.0%  0:00.09   1    49     42  124K  2112K   660K   346M
  142 ATSServer    0.0%  0:01.30   2    99     91 1244K  4104K  3368K   116M
  140 SystemUISe   0.0%  0:02.21   6   231    215 1272K  9952K  3364K   409M
```

Unix/Linux was made for this. Windows was not.

Hit a server with cURL

- Use cURL (RCurl) to send GET or POST request to server for a URL/URI
- `curl http://thecaucus.blogs.nytimes.com`
- `curl -L http://t.co/KtxsapBV`
- `curl http://search.twitter.com/search.json?q=%40obama`
- The latter is a query string, can used to return custom results from MANY websites (Twitter, nytimes.com, Library of Congress, etc.).

Query string trivia: the following string has brought down many a web server: `system%28%27rm+-rf+%2F%27%29`

Hit a server (or an entire site!)

- Use wget or a crawler
- `wget http://thecaucus.blogs.nytimes.com`
- See also module 3 http://www.stanford.edu/~seanjw/module3/#8
- Heritrix http://crawler.archive.org/

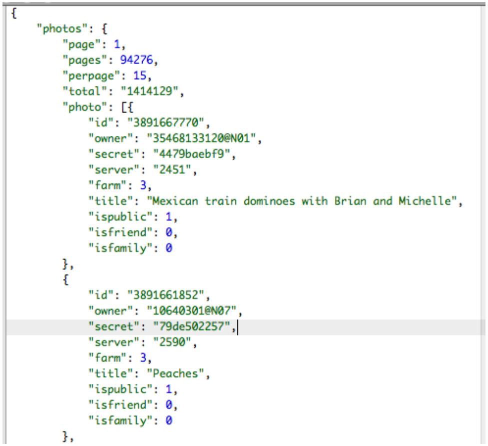
Hit a server (on schedule!)

- Use `cron`
- `crontab -e`
- in VIM type `* * * * * /4 /path/to/R CMD myfile.R` to run every Wed
- or perhaps `* * * * * /4 /path/wget http://thecaucus.blogs.nytimes.com`
- Save and you'll see: `crontab: installing new crontab`
- Type `crontab -l` to see your crontabls
- See `http://benr75.com/pages/using_crontab_mac_os_x_unix_linux` for more.

Parse the server's reply: JSON

- What the **** is JSON?
- JSON: Java script object notation
- for serializing objects, for the purposes of data interchange
- semi-structured, tree-like, and pretty simple
- Nice JSON viewer for Chrome and Firefox—Use this in the R code.
- Sean's overview from module 3 `http://www.stanford.edu/~seanjw/module3/#42`
- Widom's overview of JSON here: `http://www.db-class.org/course/video/preview_list`

Record and parse it: JSON



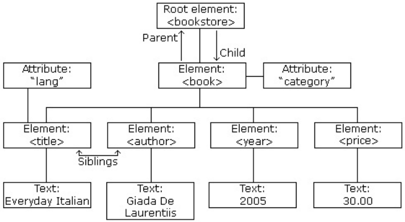
```
{
    "photos": {
        "page": 1,
        "pages": 94276,
        "perpage": 15,
        "total": "1414129",
        "photo": [{
            "id": "3891667770",
            "owner": "35468133120@N01",
            "secret": "4479baebf9",
            "server": "2451",
            "farm": 3,
            "title": "Mexican train dominoes with Brian and Michelle",
            "ispublic": 1,
            "isfriend": 0,
            "isfamily": 0
        },
        {
            "id": "3891661852",
            "owner": "10640301@N07",
            "secret": "79de502257",
            "server": "2590",
            "farm": 3,
            "title": "Peaches",
            "ispublic": 1,
            "isfriend": 0,
            "isfamily": 0
        },
```

- Base values
- Objects {} - label-value pairs
- Arrays [] - list of values
- nested sets of arrays - not a table
- NO SCHEMA
- NO SQL (hard to query)

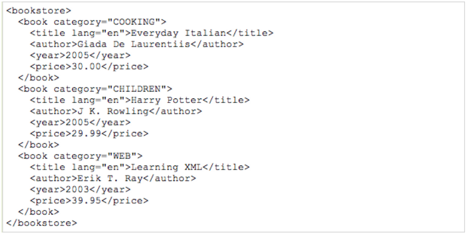
Record and parse it: XML

- What the **** is XML?
- XML: eXtensible Markup Language
- Like HTML, but tags describe data, not formatting
- Semi-structured, tree-like, much richer, more complicated than JSON
- Nice XML viewer for Chrome.
- Sean's overview in module 3 `http://www.stanford.edu/~seanjw/module3/#21`
- Widom's course: `http://www.db-class.org/course/video/preview_list`

Record and parse it: XML



Looks like:



```
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

Example from http://www.w3schools.com/xml/xml_tree.asp.

- Tagged elements
- Attributes
- Text
- Nested structure - not a table
- XML SCHEMA/DTD
- NO SQL (use XPATH/jQuery)

Lab 1

Lab 1: Getting useful data with Curl and JSON in R.

Clean things up: Regular Expressions

WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

from: https://xkcd.com/208/

Regex in action..

- To clean up data after scraping `http://www.r-bloggers.com/scrape-web-data-using-r/`.
- To extract useful information (state, latitude, longitude), when scraping a web page `http://solomonmessing.wordpress.com/2011/09/18/map-of-participants/`
- grep to create custom indicator variable features for analysis, agrep for approximate version of texttgrep. More on this later.
- For nice reference materials, look to `http://www.regular-expressions.info/reference.html`.

RegEx bare essentials

RegEx	Description	Example
bil	Match string ‘bil’	this is a 2 dollar bill.
dollar|bill	Match dollar or bill	this is a 2 dollar bill.
\d	Match any digit	this is a 2 dollar bill.
\w	Match any word (single letter)	this is a 2 dollar bill.
\w+	Match at least 2 letters	this is a 2 dollar bill.
\s	Match any whitespace	this_is_a_2_dollar_bill.
\S	Match NOT whitespace	this is a 2 dollar bill.
colo?ur	Optionally match character preceding ‘?’	Yanks color, Brits colour.
col.*r	match any character between l and r 0 or more times	Yanks color, Brits colour.

Regex in R for Cleaning and Feature Extraction

Command	What it does
`grep("dollar\\|bill", moneyStuff)`	return index of `moneyStuff` with item.
`gregexpr("\\d", "4a53f45e")`	return index of string in each match (why might this be a bad idea?)
`str_extract(moneyStuff, "\\d+")`	extracts groups of digits
`str_replace(moneyStuff, "(\\d+)", "\$\\1)")`	inserts dollar sign in front of numbers
`agrep("dollar\\|bill", moneyStuff, max.distance = .2)`	return index of anything with edit distance $\leq$ .2 from dollar or bill in `moneyStuff` with item. VERY useful to handle misspellings, plagiarism, etc.

Read up on edit distance here: http://www.stanford.edu/class/cs124/lec/med.pdf

Regex == AWK

Need to quickly extract/transform a 2 TB text file?

- `awk '/pattern/'` return every line in a file matching pattern
- `cat bigfile.csv | awk '(2 > 5&&3<2) {print 1,3}' > smallerfile.csv`
  Reads in all lines from bigfile.csv where 2nd column value > 5 and third column < 2 and prints to smallerfile.csv

Lab 2

Lab 2: Scraping and Regex