

Text Technologies

Crawls and Feeds

Victor Lavrenko

Copyright © Victor Lavrenko, 2010

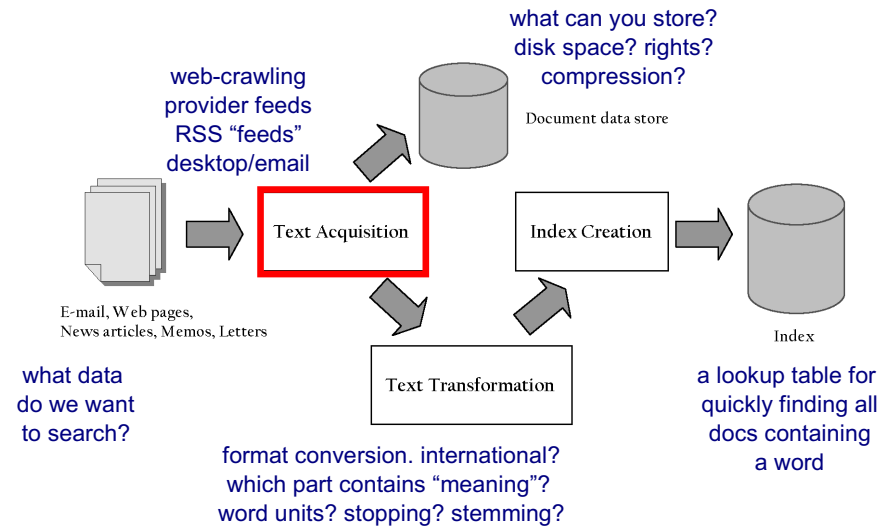
Overview

- Acquiring documents
 - desktop / enterprise search
 - document feeds
 - crawling the web
 - mechanics of crawling
- Extracting the content
 - XML markup and DOM
 - tag-plateau method

Copyright © Victor Lavrenko, 2010

1

Indexing Process (review)



Copyright © Victor Lavrenko, 2010

© Addison Wesley, 2008

Sources of data

- In general, three different sources of data
- Desktop / enterprise search
 - all local files, removable media, networked drives
- Web / site search
 - all published pages on a sub-network of interest
- Document feeds
 - RSS/Atom feeds (e.g. www.google.com/reader)
 - commercial feeds: AP, Reuters, AcquireMedia

Copyright © Victor Lavrenko, 2010

3

4

What do we want to search?

- Acquire everything that is feasible
 - do not filter by subject / quality / trustworthiness
 - no such thing as bad data or too much data
- Filtering is much easier once it's indexed
 - better models of subject / authority / spam
 - fast and space-efficient, if designed right
- Respect the rules
 - visible ≠ accessible ≠ storable ≠ presentable

Copyright © Victor Lavrenko, 2010

5

Desktop / enterprise search

- Finding documents easy:
 - desktop: traverse filesystem
 - enterprise: intercept file- / mail- / IM-server traffic
- Main challenge: update speed
 - users expect all recent changes to be searchable
 - register OS hooks for all file-change operations
 - network-mounted drives have to be re-indexed
- Desktop indices must be small
- Store and respect privacy / access permissions

Copyright © Victor Lavrenko, 2010

6

Document feeds

- Used for “published” documents
 - news, press releases, blog posts, forums, usenet
 - posts released at a fixed time, rarely updated
- Engine gets a list of everything available
 - advantageous to both publisher and the engine
 - timestamps determine what needs to be indexed
- Two types of feeds:
 - *push* feeds: commercial newswires, aggregators
 - *pull* feeds: free news, blogs via RSS/Atom

Copyright © Victor Lavrenko, 2010

7

Push feeds

- Usually commercial news / PR providers
 - vendor provides an API (usually closed-source)
 - (1) login, (2) subscribe, (3) register call-back, (4) feed
 - subscribe to everything if possible, filter yourself
 - call-back function called with each new post
 - usually passed as XML-annotated string
 - very little processing time (100 posts / second)
 - cache and move on, index asynchronously
- Vendors focus on latency (ms)
 - may provide classification, entity extraction tags

Copyright © Victor Lavrenko, 2010

8

Commercial feed example (XML)

```
<document>
<nil?>
<head>
<title>InvestSource Terminates Program With Physicians Adult
Daycare</title>
</head>
<body>
<body.head>
<headline>
<h1>InvestSource Terminates Program With Physicians Adult
Daycare</h1>
</headline>
<istributor>Market Wire</istributor>
</body.head>
<body.content>
<p>HUNTINGTON BEACH, CA -- (MARKET WIRE) --
<chron>02/22/07</chron> --
<org>InvestSource, Inc.</org>, a boutique media
relations/investor relations firm specializing in providing exposure points
for small- and micro-cap companies, announced today that it is severing
all ties with <org>Physicians Adult Daycare, Inc.</org>, a provider of
adult daycare services. Despite repeated attempts to work in conjunction
with Physicians Adult Daycare to determine the source of a campaign of
illegal &quot;spam&quot; emails promoting Physicians Adult Daycare,
InvestSource has been unable to resolve the issue. Consequently,
InvestSource finds no alternative other than to cease its own investor
relations program undertaken on behalf of
</body.content>
</body>
</document>
<xn:providerName>Market Wire</xn:providerName>
<xn:providerCode>17</xn:providerCode>
<xn:serviceName>US Press Releases</xn:serviceName>
<xn:serviceCode>1</xn:serviceCode>
<xn:publicationTime>2007-02-22T00:00:00-05:00</xn:publicationTime>
<xn:receivedTime>2007-02-22T00:00:12-05:00</xn:receivedTime>
<xn:vendorData>AMX:Publish Reason= ORIGINAL</xn:vendorData>
<xn:vendorData>AMX:Alert=FALSE</xn:vendorData>
<xn:vendorData>AMX:Headline Only=FALSE</xn:vendorData>
<xn:vendorData>AMX:Temporary=FALSE</xn:vendorData>
<xn:description>02/22/07 -- InvestSource, Inc., a boutique media
relations/investor relations firm specializing in providing exposure points for
small- and micro-cap companies, announced today that it is severing all ties
with Physicians Adult Daycare, Inc., a provider of adult daycare
services.</xn:description>
<xn:vendorData>MRKTWIRE:IWCategory=Financial Services:Investment
Services and Trading; Professional Services:Investor Relations;
</xn:vendorData>
<xn:vendorData>MRKTWIRE:Source=InvestSource, Inc.</xn:vendorData>
<xn:vendorData>MRKTWIRE:Geographic
Code=RE/HUNTINGTON_BEACH</xn:vendorData>
<xn:vendorData>MRKTWIRE:Geographic Code=RE/CA</xn:vendorData>
<!-- Classifier -->
<xn:industryCode>I/C/fini</xn:industryCode>
<xn:industryCode>NI/NEC</xn:industryCode>
<xn:subjectCode>NT/NEC</xn:subjectCode>
<xn:subjectCode>XC/any.company</xn:subjectCode>
<xn:subjectCode>XC/any</xn:subjectCode>
<xn:vendorData>AMX:Special Code=MC/HOT</xn:vendorData>
```

Pull feeds

- Usually free news/blogs/tweets via RSS/Atom
 - engine periodically checks a known URL
 - gets a list of all currently-available posts (XML)
 - list contains time-stamp, title, URL for each post
 - easy to determine what's new, and fetch only that
 - list contains a time-to-live (TTL) field
 - tells you how long list won't be updated (minutes)
- Main drawback: coverage
 - some sources provide a small subset via RSS

RSS example (blog)

```
<?xml version="1.0">
<rss version="2.0">
<channel>
<title> Information Retrieval Blog </title>
<link> http://www.hypothetical-ir-blog.com </link>
<description> New developments in Information Retrieval </description>
<language> en-us </language>
<pubDate> Mon Sep 29 00:41:50 BST 2008 </pubDate>
<ttl> 60 </ttl>
...
<item>
<title> Quantum IR </title>
<link> http://www.some-other-site.com/quantum-ir.html </link>
<description> A new model based on quantum mechanics </description>
<pubDate> Mon Sep 29 00:41:50 BST 2008 </pubDate>
<guid> http://www.hypothetical-ir-blog.com/#499 </guid>
</item>
...
```

Atom example (Twitter)

```
<entry>
<id>tag:search.twitter.com,2005:14648316600</id>
<published>2010-05-24T21:27:52Z</published>
<link type="text/html" href="http://twitter.com/villiedatruth/statuses/14648316600" rel="alternate"/>
<title> RT @MZTISH Too many people live in a fairytale mindframe...get real...our generation is a mess..I
feel sorry for my own kids//shame isn't it </title>
<content type="html">RT &lt;a href=&quot;http://twitter.com/MZTISH&quot;&gt;&gt;@MZTISH&lt;/a&gt; Too
many people live in a fairytale mindframe...get real...our generation is a mess..I feel sorry for my own
&lt;b&gt;kids&lt;/b&gt;//shame isn&apos;t it</content>
<updated>2010-05-24T21:27:52Z</updated>
<link type="image/png" href="http://a1.twimg.com/profile_images/917602038/0514101525_normal.jpg"
rel="image"/>
<twitter:geo> </twitter:geo>
<twitter:metadata> <twitter:result_type>recent</twitter:result_type> </twitter:metadata>
<twitter:source>&lt;a href=&quot;http://twitter.com/&quot;&gt;&gt;web&lt;/a&gt;</twitter:source>
<twitter:lang>en</twitter:lang>
<author>
<name>villiedatruth (Villie VL-Tunekz)</name>
<uri>http://twitter.com/villiedatruth</uri>
</author>
</entry>
```

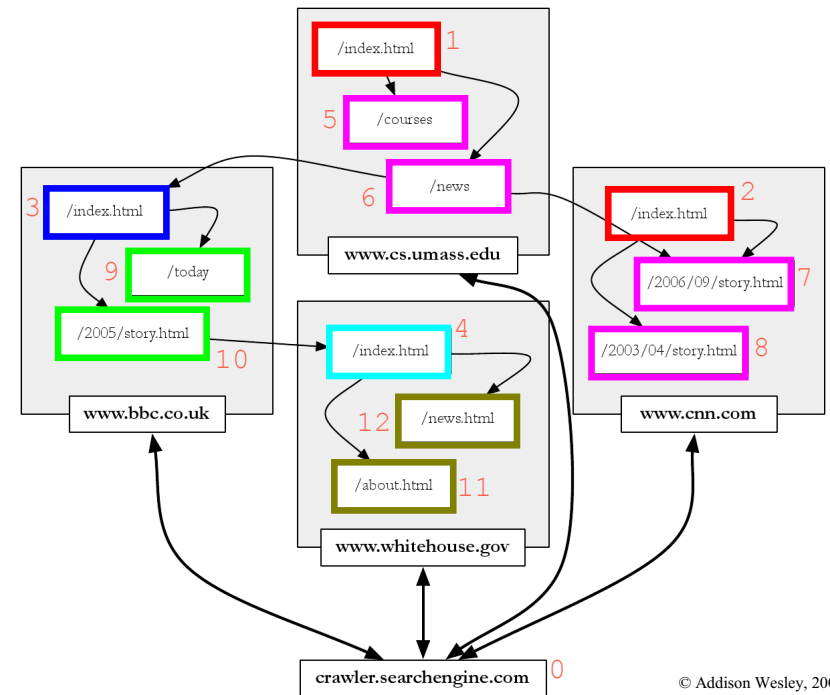
Web crawling

- Web is a graph (nodes=pages, edges=links)
- A crawler (spider) performs graph traversal:
 - frontier: priority queue of all pages to be crawled
 - initialized with a set of “seed” sites
 - priority reflects freshness, rotates hosts
 - can encode subject priorities for focused crawling
 - can be threaded if frontier is global / lockable

```
url = frontier.pop() # priority queue
html = get(url.host,url.page)
store (url,html) # save a copy
for each url in html: # add new links
    frontier.push(url,time)
```

Copyright © Victor Lavrenko, 2010

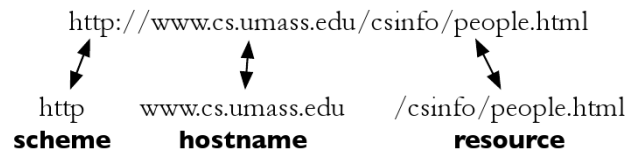
13



© Addison Wesley, 2008

© Addison Wesley, 2008

Fetching a page



- Links are defined by URLs
 - hostname: convert to IP address (123.456.789.10)
 - requires DNS query, possible performance bottleneck
 - connect to IP on port 80 and send
 - GET /csinfo/people.html HTTP/1.1
 - POST – for filling forms, typically not used
 - HEAD – to get status (exists, size, charset, timestamp)
 - low-bandwidth way to check freshness

Copyright © Victor Lavrenko, 2010

15

Respecting the rules

- <http://www.host.com/robots.txt>
 - defines who can access what
 - always check before crawling the site
 - ignoring is a quick way to get your IP blacklisted
- Remember: visible ≠ accessible ≠ storable

```
User-agent: *
Disallow: /private/
Disallow: /other/
Allow: /other/public/

User-agent: EdinburghCrawler
Disallow:
```

Copyright © Victor Lavrenko, 2010

16

On the web ≠ Public domain



“On the web” does not mean you can:

- download it using an automated program
- permanently store any portion of it
- publish any portion of it or even link to it (deep-linking)

Copyright © Victor Lavrenko, 2010

Keeping up with changes (2)

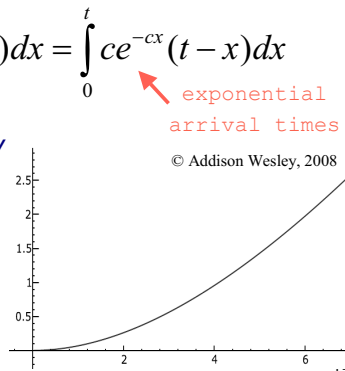
- Problem: don't know when page has changed

- page updates ~ Poisson (c)
 - A1: 1-at-a-time
 - A2: independent

- Expected age of a page:

$$\int_0^t P(\text{changed_at_time_x})(t-x)dx = \int_0^t ce^{-cx}(t-x)dx$$

- t ... days since we last crawled
- c ... expected changes per day
- c=1/7, t=7 → 2.6 days old



Copyright © Victor Lavrenko, 2010

- Priority = expected age

- “learn” c for popular urls
- be sure to rotate hosts

Keeping up with changes

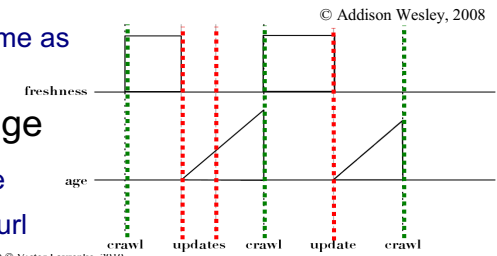
- When should we re-crawl a page?
 - stale page ... has changed since we crawled it
 - freshness ... 0 if page is stale, 1 if not

- Maximize number of fresh pages: $\sum_p Fresh(p)$
 - problematic: no “degree” of staleness

- stale by 1 minute same as stale by 1 month

- Minimize “age” of page

- # days it's been stale
- set as priority of the url



Copyright © Victor Lavrenko, 2010

Keeping up with changes (3)

- How do we learn the rate of updates c?

- c ... expected number of page changes per day
- we crawl page with intervals $\Delta t_1, \Delta t_2, \Delta t_3 \dots$
- let c_i be the number of changes during interval Δt_i
- then $c = c_1 + c_2 + c_3 \dots / \Delta t_1 + \Delta t_2 + \Delta t_3 \dots$

- Problem: only know if $c_i = 0$ or $c_i > 0$

- could make Δt_i very small ... lots of wasted effort
- one solution: $\left\{ \begin{array}{l} \text{if } c_i > 0: \text{ decrease } \Delta t \text{ by } \epsilon \\ \text{if } c_i = 0: \text{ increase } \Delta t \text{ by } \epsilon \end{array} \right\}$ converges to Δt s.t. $P(c_i=0) = 0.5$

Copyright © Victor Lavrenko, 2010

Focused crawling

- Suppose you're interested in one specific topic
 - and no infrastructure to crawl the entire web
- Possible to focus a crawl on a specific topic
 - priority = P(url will discuss topic of interest)
 - P ... classifier based on the following features
 - content of the pointing pages
 - anchor text of the pointing links
 - topic-weighted PageRank
- Crawler becomes an “agent”

Extracting the Content

- Most information online is some form of XML

```

tag      content
<title> Information Retrieval Blog </title>
<pubDate> Mon Sep 29 00:41:50 BST 2008 </pubDate>
<language> en-us </language>
<description> New developments in Information Retrieval </description>
    
```

- Tag = “variable name”, Content = “its value”
- Tag = “annotation” over a span of text
 - clarify what the content means (semantics)
 - different from HTML (tag = how text looks)
- XML parsers widely available

Tags Across Sources

- Tags and formats will vary across sources

```
RSS: <pubDate> Mon Sep 29 00:41:50 BST 2008 </pubDate>
```

```
Twitter: <published>2010-05-24T21:27:52Z</published>
<updated>2010-05-24T21:27:52Z</updated>
```

```
AM: <xn:publicationTime>2007-02-22T00:00:00-05:00</xn:publicationTime>
<xn:receivedTime>2007-02-22T00:00:12-05:00</xn:receivedTime>
```

- Will need a separate parser for every source
- Attempt to solve: NewsML
 - enforce consistent tags across news vendors
 - violated in creative ways:

```

<xn:vendorData>MRKTWIRE:Source=InvestSource, Inc.</xn:vendorData>
<xn:vendorData>MRKTWIRE:Geographic Code=RE/HUNTINGTON_BEACH</xn:vendorData>
    
```

dummy tag real tag masquerades as content

Extracting Content from Web-pages



```

<style type="text/css">
.cnnFRecBtm
{width:386px;float:right;margin:10px 0;clear:both;}

.cnnFRecBtmBot
{width:420px;margin:30px 0 15px 186px;}

.cnn_strycntnltft
{clear:both;}
</style>

</div>

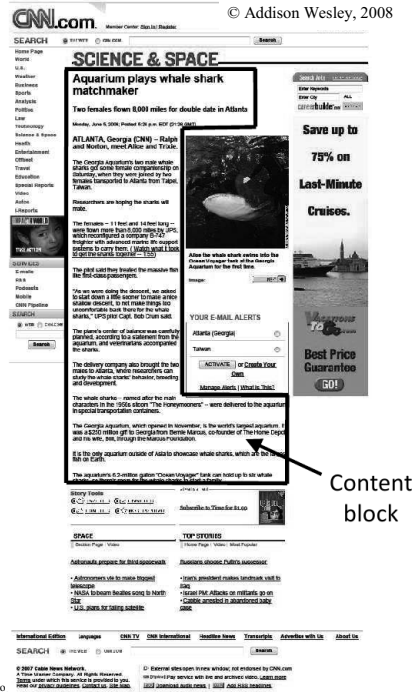
<div class="cnnFRecBtm" id="cnnStryRcmndBtm"></div>

<!-- google_ad_section_start --><!--
<script>
</script>
</div>
</div>
<!-- google_ad_section_end --></div>
<!-- CONTENT --><!--<!-- startclickprintinclude-->
<script language="JavaScript" type="text/javascript">var
clickExpire = ".1"/</script> <div class="cnn_stryltcont">
<div class="cnn_stryltcont"><div class="cnn_bulletbin
cnnStryHghLght"><!-- google_ad_section_start --><!--</li>The
    
```

- No semantic tags
 - Tags = how the page looks
 - Highly varied across sources
- What if we remove all tags?

Getting the text

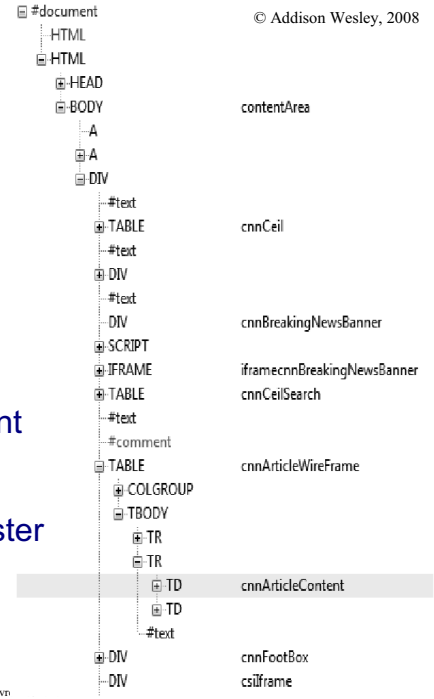
- Web pages contain junk
 - navigation controls
 - interactive / multimedia
 - advertisements
 - copyright notices
- Want to extract content
 - block of text containing description of the page
 - what you want to search



Copyright © Victor Lavrenko

Getting the text (2)

- Use DOM structure
 - document object model
 - logical layout of page
- Explicit markup
 - some sites will tag content
 - rare and site-specific
 - breaks with new webmaster
- Or learn statistically
 - brittle, needs re-training

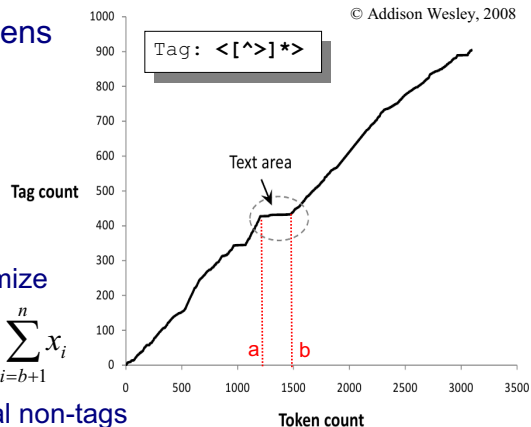


Copyright © Victor Lavr

Getting the text: tag plateau

- Idea: text portion contains fewer HTML tags
 - plot #tags vs. #tokens
 - look for a plateau
- Formally

- $x_i = 1 \dots a$ tag
- $0 \dots$ non-tag
- find a, b that maximize
- $c = \text{total tags} / \text{total non-tags}$



Copyright © Victor Lavrenko, 2010

Tag plateau: algorithm

- Naïve implementation: $O(n^3)$... hours per page

- Incremental: $O(n^2)$

$$\sum_{i=1}^{a-1} x_i + c \sum_{i=a}^b (1-x_i) + \sum_{i=b+1}^n x_i$$

L M R

```

for a = 1..n:
  L += X[a]
  ...
  for i = a+1..n:
    R -= X[i]
    M += 1-X[i]
  ...
    
```

- Dynamic programming: $O(n)$
 - under 10ms per page if implemented correctly
- Limitations:
 - single "content" section surrounded by "junk"
 - alternative: look at slopes in k-token slices of HTML

Copyright © Victor Lavrenko, 2010

Summary

- Acquiring documents:
 - *get everything you can*
 - desktop search: *refresh often, conserve disk*
 - document feeds: *push-and-pull saves bandwidth*
 - crawling the web: *priority queue by age/topic*
 - mechanics of crawling: *respect the rules!*
- Extracting the content:
 - XML tags will vary from one source to another
 - use tag plateau for webpages