Text Technologies

Crawls and Feeds

Victor Lavenko

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

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

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

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-omnified 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

Overview

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

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-changes operations
  - network-mounted drives have to be re-indexed
- Desktop indices must be small
- Store and respect privacy / access permissions

Commercial feed example (XML)
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 is 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

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, relative host
  - can encode subject priorities for focused crawling
  - can bethreaded if frontier is global/localizable

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

Respecting the rules

- [http://www.host.com/robots.txt](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

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)
```

RSS example (blog)

```
<rdf:RDF

<title>Information Retrieval Blog</title>
<link>
</link>
<description>New developments in information IR-blank</description>
<language>en</language>
<pubDate>Mon Sep 20 06:15:50 BST 2005</pubDate>
<guid>
</guid>
```

Atom example (Twitter)

```
<entry>
<title>Quantum Physics</title>
<link>http://www.somewhere.com/quantum.html</link>
<description>A new model based on quantum mechanics</description>
<published>Mon Sep 20 00:41:50 BST 2005</published>
<updated>Mon Sep 20 00:44:05</updated>
</entry>
```

Keeping up with changes

- When should we re-crawl a page?
  - stale page: has changed since we crawled it
    - freshness ... if page is stale, 1 if not
  - maximum number of fresh pages:
    - problematic: no "degree" of staleness
    - stale by 1 minute same as stale by 1 month
  - "age" of page:
    - stale if it's been stale
    - set as priority of the URL

Fetched a page

```
http://www.somewhere.com/robots.txt
```

Getting from the URL

```
```

```
http://www.somewhere.com:8080/robots.txt
```
**Keeping up with changes (2)**

- Problem: don't know when page has changed
  - page updates ~ Poisson (c)
  - All 1-at-a-time
  - Expected age of a page:
    \[
    \int_{0}^{\infty} P(\text{changed at time } x) \, dx = \int_{0}^{\infty} e^{-\lambda x} \, dx = \frac{1}{\lambda}
    \]
    - \( t \) days since we last crawled
    - \( c \) expected changes per day
    - \( c=1/7, t=7 \rightarrow 2.6 \text{ days old} \)
- Priority = expected age
  - "learn" c for popular urls
  - be sure to rotate hosts

**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 \( \Delta t_i \), \( \Delta t_2 \), \( \Delta t_3 \) ...
  - let \( c_i \) be the number of changes during interval \( \Delta t_i \)
  - then \( c = c_1 + c_2 + c_3 + ... \)
  - Problem: only know if \( c_i = 0 \) or \( c_i > 0 \)
  - could make \( \Delta t \) very small ... lots of wasted effort
  - one solution: decrease \( \Delta t \) by \( \epsilon \) each crawl if \( c_i = 0 \)

**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

**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"