Class Logistics

Midterm Exam (releasing tonight, due 11/07 via NYU Classes)
• Material from all six lectures on fundamentals of search engine
  • Full slides for today’s lecture will be posted tonight
  • Sign and scan the pledge page, turn it in with the answers
  • Print! Mark clearly which question each answer is for
  • Grading will be done by 11/11
  • Solutions will be briefly discussed in class on 12/13, before the final
• Individual take-home exam, do not discuss with other people
  • This is serious, do it on your own!

Class Logistics

Homework and Project
• HW1 graded, TAs will announce an office hour on Piazza
• HW2 grading will be done by 11/18
• HW3 will be released 11/07 (next Monday)
• Demo slots, sign up as soon as possible (8 groups have)
  • 5 more to go, first come first serve, not many options left
  • Do it now: G01, G06, G10, G11, G13
More on Project
Proposal (optional)
• You should have a good idea of what you want to do soon, before HW3 is completed
• If you want advice on your project, email us a proposal and/or stop by the office hour

What we look for
• Report describing motivation, design and experimental analysis
• Demo showing that things work. We will play with the system
• Code needs to be submitted and we will examine it
  • Due after demo

Important note on grading
• Everyone on the same team may not get the same grade in certain circumstances

Advanced Topics
Topics for Fernando’s lectures have yet to be finalized
• Likely topics: Personalization, Real-time Search, Ethics

No class or office hour 12/05: both Fernando and I are traveling.

11/14 Internet Advertising (Nitish Korula)
• How internet companies providing free services make their money
• Nitish is an engineering manager at Google NYC on Display Advertising, a former researcher at Google Research NYC

12/12 Knowledge & Structured Data (Andrei Dinu-Ionita)
• How search engines answers questions and be your assistant
• Andrei is an engineer at Google NYC on Search, a former student of this class!

12/13 Big Data
• A loose set of topics such as MapReduce

Last Lecture
Document Processing
• Encoding: Unicode and UTF-8
• Segmentation: statistical approach
•Normalization
  • Spelling check: edit distance
  • Related terms: general statistical approach & word2vec
Web Crawl

Core tasks of crawl can be summarized in one sentence:

Starting with a set of seed sites, find and prioritize contents, download and collect necessary information, and detect and remove low quality contents.

- Contents = textual, multimedia, etc.
- Necessary information = content, link structure, anchors, etc.
- Low quality contents = spam
Outline

Web Crawl Basics
- Size of the Web
- Important Protocols
- Downloading the Content

Crawl Scheduling
- Crawl Prioritization
- Distributed Crawl

Handling Quality Issues
- Duplicate Detection
- Spam Detection

Size of the Web

How large is the Web?
- Hidden Web backed by databases
- Videos, Images, etc.

Overall, over one hundred billion of pages in indexable Web
- Cuil, a start up search engine from 2008 to 2010, claimed to have indexed 120 billion pages

Billions of high quality pages
- Pages that are likely to show up in result pages for at least some user queries
- Not a static set: new contents are generated every day, e.g., News, Social Streams, and old contents may go away

Estimating Size of the Web

Search companies won’t disclose how many pages they crawl, for obvious reasons.

But, the companies do want to know the size of the Web, i.e., whether they’ve crawled enough.

For example, Bing wants to know how many pages Google is crawling, and vice versa.

How?
By checking the other search engine’s search results.
Estimating Size of the Web

Generate a set of test URLs

- Issue a set of random queries to various search engines and randomly pick pages from the query results → test set of URLs
- Check each URL within the test set against both search engines, record which one(s) contain the page

One can now estimate the probability of a URL appearing in the intersection given it appears in one of the search engines.

\[ |G \cap B| = |G| \cdot \Pr[G \cap B | G] = |B| \cdot \Pr[G \cap B | B] \]

At Bing:

- Size of the intersection \( |G \cap B| = |B| \cdot \Pr[G \cap B | B] \)
- Size of G \( |G| = |B| \cdot \frac{\Pr[G \cap B | B]}{\Pr[G \cap B | G]} \)
- Size of the Web \( |G \cup B| = |B| + |G| - |G \cap B| \) / \( \text{url}_{\text{query}} \% \)

In-Class Exercise

A random sample of 2000 URLs, 800 appear in Google alone, 400 appear in Bing alone, 600 appear in both.

If Bing crawls 10 billion pages, what’s the size of the Web?

Size of the intersection = 10 * (600 / 1000) = 6B
Size of Google = 10 * ((600 / 1000) / (600 / 1400)) = 14B
Size of the Web = (10 + 14 - 6) / (1800 / 2000) = 20B
Seeding the Crawl

Start with the well-known sites
• Open Directory Project (www.dmoz.org)
• Wikipedia (www.wikipedia.org)
• Official government websites
• Fortune 500 company websites
• US News & World Report top university websites
• Major news papers and magazine sites

Subsequent crawls will have better knowledge about the Web to begin with.

The Protocols

Steps involved in downloading the page at http://www.wikipedia.org/wiki/Nyu

DNS: www.wikipedia.org ➔ 208.80.154.225
HTTP protocol: GET /wiki/Nyu HTTP/1.0
TCP/UDP protocols: multi-packets data transfer
IP protocol: transfer individual packets machine to machine

DNS: Domain Name System

October 2016 Dyn cyberattack, largest DDoS attack
Translates meaningful domain names to IP addresses
• End users don’t have to memorize IP addresses
• Provides a necessary layer of abstraction:
  • The underlying IP addresses (i.e., machines) being mapped to a particular domain can change without the end user noticing.

Each domain has an authoritative DNS server
At the top are root servers, 12 logical servers world wide
• Their IP addresses are all public
• USC, Maryland, NASA, AMES, DoD, ARL, etc.
Hierarchical Address Resolution

What’s the implication for Crawlers?

It is critical to cache the DNS results!

HTTP: Hypertext Transfer Protocol

Similar to DNS, an application-layer protocol

- Does not deal with how data is transferred

Most important request methods

- **GET/HEAD:** for retrieving data and metadata
  - Ideally not changing the state of the server
  - User content visible in the URL → easy to crawl
- **POST:** for posting data, mostly used by hidden Web
  - Server state may be changed
  - User content not visible in the URL → difficult to crawl

HTTP is Stateless

The server is not required to remember anything from previous requests

Sessions, however, are very important in many Web applications, e.g., shopping cart

- **HTTP cookies:** a piece of data communicated between a site and the browser; each time a user visits the site, the site can retrieve the data stored and set new data back to the browser.
- **CGI parameters:** http://www.amazon.com/s?keywords=shoes&h=9778894a0cb2d03c754d87b60fcf1185a5f

Cookies for advertising (a.k.a., How does Site A know about the product I browsed on Site B two days ago?)

- Most likely, it's not Site A or Site B
- Online advertisement exchanges know a lot about you
- Each request for an ad is an HTTP request and therefore a cookie can be set and retrieved, often by the same company across different sites.
Transport Protocols: TCP

Transmission Control Protocol

• Dealing with packet loss
• Dealing with duplicate packets
• Dealing with out of order delivery
• Dealing with congestion control

Three way handshake connection establishment (refer to your networking class)

Transport Protocols: UDP

User Datagram Protocol

• No guarantee of delivery
• No guarantee of order or duplicate removal
• Provides checksums for data integrity
• No need for congestion control or connection establishment

UDP is ideal for live streaming or VoIP applications

• Extremely time sensitive, cannot afford to buffer the data for later delivery
• Many clients often connect at the same time, cannot afford to keep states at the server for each client

Internet Protocol, IPv4, IPv6

Internet layer protocol for sending a data packet from one IP address to the to another IP address

• Best effort delivery and no connection establishment
• IPv4 (32-bit address) not interoperable with IPv6 (64-bit address)

Routing algorithms are at the core of the IP protocol

• Each step moves the packet a bit closer to the final destination
**Crawl Protocol: Robot Exclusion Standard**

A very simple description of access permissions, available as the file `robots.txt` at the root of a site:

- E.g., http://en.wikipedia.org/robots.txt

Enforcement is up to the site operator

- For example, wget is disallowed by wikipedia.org, but often ignored
- Site operators usually have to block the IPs of serial offenders
- Established companies always respect robots.txt

```bash
# robots.txt: Use this to block things.

User-agent: * # Block all!
Disallow: /wiki
Disallow: /new-user
```

**Crawl Protocol: Sitemap**

Stored as the file `sitemap.xml` at the root directory and advises the crawler how to crawl the content on the site

- Inclusion protocol, complementary to `robots.txt`, which is exclusion
- Usages?
  - fresh content (not many links yet), archival content (links gone), rich content, less frequent changes, very frequent changes, etc.

Whether to follow the advice is up to the crawler operator.

**Crawl Protocol: `nofollow` & `redirect`**

```html
<html>
<head>...
<title>
<meta name="robots" content="noindex, nofollow">
</title>
</head>
<body>
  <a href="http://example.com/article" rel="nofollow">Article Example</a>
</body>
</html>
```
Downloading the Content
Lots of challenging issues that are not solved
• User-specific dynamic redirects
• User-specific dynamic contents

Challenging issues that are partially solved
• JavaScripts
  • Use a rendering engine
• Hidden Web
  • Construct reasonable queries to uncover the hidden content
• Boilerplates

Example Boilerplates
Advertisements
Navigational menus
• Top bar and left side menu
Informational boxes
• Stock quotes tables in financial articles
Common characteristics
• Not related to the main page content
• Often providing misleading signals for ranking
Detecting Boilerplates: a Simple ML Approach


Uses shallow text features to train a machine learning model and classify text blocks as either boilerplates or main contents.

A text block is defined as a contiguous set of text nodes uninterrupted by structural nodes (except pseudo-text nodes such as `<a>`, `<b>`, etc.)

---

DOM Structure of HTML Pages

---

Boilerplate versus Main Content: Intuition

Page level
- **Text length**: main contents tend to be longer
- **Presence of anchor links**: usually much denser in boilerplates
- **Positions in the document**: main contents tend to be in the middle, and surrounded by boilerplates
- **Word features**: e.g., boilerplates have more words starting with uppercase letter

Site level
- Boilerplates are similar across different pages on the same site
- Main contents are unique per page
Boilerplate versus Main Content: Useful Features

Simple features that work well

- **Number of words**
- **Text density**: number of words divided by the number of lines the text block occupies based on standard width
- **Link density**: number of links divided by the number of lines

- Positions are captured by adding above features from the previous and following text blocks

Machine Learning using News Collection

Off-the-shelf machine learning models: Decision Tree

Number of words and text density are highly correlated.

```
Algorithm 2 Classifier based on Number of Words:
curr_linkDensity <= 0.333333
| prev_linkDensity <= 0.555556
| | curr_numWords <= 15
| | | prev_numWords <= 4: BOILERPLATE
| | | prev_numWords > 4: CONTENT
| | | curr_numWords > 15: CONTENT
| | prev_linkDensity > 0.555556
| | curr_numWords <= 40
| | | prev_numWords <= 17: BOILERPLATE
| | | prev_numWords > 17: CONTENT
| | | curr_numWords > 40: CONTENT
| | curr_linkDensity > 0.333333: BOILERPLATE
```

Classification Results

(b) GoogleNews

- [Graph showing classification results for GoogleNews]
Outline

Web Crawl Basics

Crawl Scheduling
• Crawl Prioritization
• Distributed Crawl

Handling Quality Issues
• Duplicate Detection
• Spam Detection

Crawl Prioritization
The Web is evolving and the crawler needs to keep up
• A 2004 study (Ntoulas et al), 8% new pages per week, 25% new links per week, 80% pages gone after a year

What factors to consider in crawl prioritization?

An incomplete list:
Frequency of changes to the page
• Frequently changing pages need to be crawled more often

Nature of changes to the page
• Pages whose changes are in important parts need to be crawled as soon as possible

Impact to search
• Pages that are likely to impact search result quality need to be crawled more often, if they are changing

Freshness and Age
A crawled page is fresh at time t if the crawled content is the same as the live page at time t.

The age of a crawled page at time t is (t - t_c), where t_c is the time the page was last changed but the crawler has yet to crawl the changed content.
• Expected age can be computed by modeling the distributions of page updates

Maximizing freshness or minimizing age?
• Maximizing freshness: if a page changes very frequently, it will be better not to crawl it since it will never be fresh
• Minimizing age avoids that problem.
Search Impact


Estimating potential search impact for not yet crawled pages and prioritize crawl for those high potential impact pages.

---

Search Related Signals for Crawl Prioritization

Query independent
- PageRank

Query dependent
- Relevance score of the page for queries in the query log

Note that the page has not been crawled, so exact computation of those signals are not available.

Query independent signals are much easier to estimate than query dependent signals
- Thus most prior works have ignored the latter.

---

Query Independent Signals are not Good Enough

<table>
<thead>
<tr>
<th>Query: “NCAA Football”</th>
<th>“Judy Inklesnurff”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page</td>
<td>Query-independent score</td>
</tr>
<tr>
<td>1</td>
<td>$P_1$</td>
</tr>
<tr>
<td>2</td>
<td>$P_2$</td>
</tr>
<tr>
<td>3</td>
<td>$P_3$</td>
</tr>
<tr>
<td>100</td>
<td>$P_{201}$</td>
</tr>
</tbody>
</table>

---
Estimating Impact

Impact of a page on an individual query, \( I(p,q) \), is partially based on estimation of relevance score, \( R(p,q) \):

Overall search impact is thus a weighted summation over impact on individual queries. (\( V \) is the visibility function.)

\[
I(p,Q) = \sum_{q \in Q} f(q) \cdot I(p,q) = \sum_{q \in Q} f(q) \cdot V(R(p,q))
\]

Estimating Impact

A page is considered as having impact on a query if its estimated score will land the page in the top-K results

- Estimated score can be computed from matching of queries with meta data (URL and anchors)

Needy queries

- Frequency: common queries are more important
- Low current score: the search engine is not yet doing well
- It is very important to handle long tail queries well

Estimating \( V(R(p,q)) \): Query Sketch
Crawling the Web

Crawl 8.5 billion \(2^{33}\) pages in one week, assuming on average 64KB \(2^{16}\) per page

- \(2^{49}\) bytes per week = 888MB per second = 7+ Gbit/s
- Not possible even with Google Fiber everywhere!

Furthermore
- Must introduce delays to avoid overwhelming the sites and prevent them from serving real traffic
- Document processing to find new URLs takes additional time.
- Web crawl needs to be distributed to maximize throughput

Distributed Crawling

Multiple crawlers, each handles a subset of the discovered URLs. How to split the URLs?

- Hash over the hostname, not the full URL
  - Crawling delay is much easier to enforce if all pages on a single site is crawled by the same server
  - New URLs are more likely to stay on the same site since most links are between pages on the same site
  - Easier to remember the pages that have already been crawled

What happens if you need more machines?

- hash(host) mod 100 \(\rightarrow\) hash(host) mod 101
- In this conventional hashing, changing the number of machines will lead to reshuffling nearly everything
**Consistent Hashing**

Only $K/n$ items need to be reshuffled if a machine is added or removed, where $K$ is the total number of items and $n$ is the number of machines.

**Construction:**
- Hash machines randomly to many points on a circle.

**Search:**
- For a given hostname, hash it to a point on the same circle.
- Moving in clockwise direction until a machine point is found.
- Assign the given hostname to that machine.

**Lookup complexity**
- $O(\log n)$, where $n$ is the number of machines.

Why map each machine to many random points on the circle?
- To achieve load balancing as machines are added and removed.

Adopted by Amazon’s Dynamo, a distributed key-value store.
In-Class Exercise
Given the site to machine mapping on the right, identify where on the circle M2 and M3 are mapped.

Outline
Web Crawl Basics
Crawl Scheduling
Handling Quality Issues
• Duplicate Detection
• Spam Detection

Quality Issues in Web Crawl
Financial payoff is very big in being ranked high in major search engines
• Thus, lots of incentives for gaming the system
Robots-only Content
• Showing crawler a page that is entirely different from what the users see
(Near-)Duplicate content
• Benign mirror sites:
  E.g., www.dblp.org and http://www.informatik.uni-trier.de/~ley/db/
• Sites that are just scraping other site’s content
Link Spam
• Gaming the ranking signals based on link analysis, e.g., PageRank, Anchor
• Search Engine Optimization: white hat versus black hat
Robots-only Content (www.bmw.de circa 2006)

(Near)-Duplicate Detection

Easy to handle exact duplicates → hashing

But most intentionally created “duplicate” contents are designed to be slightly different from the original

• Different navigational menus and advertisements
• The stealer may even randomly modify the main content slightly
  • To defeat boilerplate detection based approaches

Shingling

A technique for detecting near duplicates on the Web scale

First proposed by Broder et al. 1997 in WWW Conference.

A w-shingle of a web page is a sub-sequence of w tokens from the page content

Pages are compared based on the sets of shingles we generate by sliding a window of size w over the content:

• Intuition: the more similar the two pages are, the more common shingles they will likely share
Shingling Example ($w = 2$)

The documents

\(d_1\): To be, or not to be, that is the question.
\(d_2\): To sleep perchance to dream, ay, there's the rub.
\(d_3\): To be or not to be, ay there's the point.

Step 1: generate 2-shingles & remove duplicates & sort

\(d_1\): be or, be that, is the, not to, or not, that is, the question, to be.
\(d_2\): ay there, dream ay, perchance to, a the, sleep perchance, the rub, there a, to dream, to sleep.
\(d_3\): ay there, be ay, be or, or not, or not, a the, the point, there a, to be.

Step 2: hash into integers

\(d_1\): 43, 14, 100, 204, 26, 106, 134, 172.
\(d_3\): 132, 110, 138, 204, 26, 14, 107, 93, 172.

Shingling Example ($w = 2$)

(absolute) Step 3: pick a subset, \(h \mod m = 0\), say \(m = 2\)

\(d_1\): 14, 26, 28, 106, 154, 172.
\(d_2\): 132, 134, 138.
\(d_3\): 132, 110, 204, 26, 14, 172.

Step 4: compute the intersection

\[
\frac{|A \cap B|}{|A \cup B|}
\]

Duplicate \((1, 3) = \frac{3}{6 + 6 - 3} = \frac{1}{3}\)

Duplicate \((1, 2) = 0\)

In-Class Exercise

D1: New York University is in New York City
D2: New York City has Columbia and NYU
D3: Columbia and NYU are both in New York City

Using the shingle approach, with \(w = 3\) and using all the shingles, which two documents are more similar?

\(D1\): new york, is in new, york city, new york university, university is in, york university is.
\(D2\): city has, columbia, and nyu, has, columbia and, new york city, city has.
\(D3\): and nyu are, both in, both in, new, columbia and nyu, in new york, new york city, nyu are both.

\(D1, D2\): new york, 1 / (6+5-1) = 0.1
\(D1, D3\): new york city, 1 / (6+7-1) = 0.08
\(D2, D3\): columbia and nyu, new york city, 2 / (5+7-1) = 0.18
Scaling to the Web

Computing the duplicate-ness for all pairs of pages on the Web is infeasible

- \((10 \text{ billion pages})^2 \times 0.01\text{s/pair} = 30 \text{ billion years!}\)

Leverage the same inverted index concept as we used in find "similar" documents to the query.

Scaling Up Shingling (w = 2)

Step 4: invert-indexing the shingles into pages

- \(\{14, 106, 108, 114, 118, 122, 124\}\)
- \(\{132, 134, 204, 206, 208, 210\}\)

Step 5: sort the shingles and put together pages with the same shingle, \(<\text{page}_1, \text{page}_2>\)

- \(\{3, 3\}, \{3, 3\}, \{3, 3\}, \{3, 3\}, \{1, 3\}\)

Step 6: sort the pairs and count, \(<\text{page}_1, \text{page}_2, \text{count}>\)

- \(\{1, 3, 3\}, \{2, 3, 2\}\)

Spam

Before PageRank and other link analysis techniques were adopted by almost all search engines, spammers concentrated on term spam.
Link Spam

Techniques to fool search engines into overvaluing the PageRank (or other similar quality measures) of a specific page, instead of improving its intrinsic quality.

How would you do it?
• Creating your own pages
• Modifying other pages
  • Spammers have to modify other pages, why?
  • Otherwise, the target page may not even be crawled!

What are those other pages?
• YouTube, Amazon
• blogs, social updates, etc.

Why Link Spam Works

PageRank revisited (simple version)

\[ Pr(p) = \beta \sum_{p_i \in M(p)} \frac{Pr(p_i)}{L(p_i)} + \frac{(1 - \beta)}{n} \]

• \( M(p) \): pages that link to \( p \)
• \( L(p) \): number of outgoing links on \( p \)
• \( n \): number of all pages on the Web
• \( \beta \): probability for a user to follow the link (to reach \( p \))

Support pages
• Created by the spammers just pointing to the target page, so their PageRank are not "leaked"
Why Link Spam Works

Let \( y \) be the unknown PageRank for \( t \), the target page, which links to support pages.

Let there be \( m \) support pages and \( n \) total pages on the Web.

So the PageRank of each supporting page is (ignoring random surfing):

\[
\beta y/m + (1 - \beta)/n
\]

Let there be a total of \( x \) PageRank coming from the accessible pages.

Two main sources for \( t \)'s PageRank (again, ignoring random surfing):

1. \( x \), as we assumed
2. Coming directly from the \( m \) support pages

\[
y = x + \beta m \left( \frac{\beta y}{m} + \frac{1 - \beta}{n} \right) = x + \beta^2 y + \beta(1 - \beta) \frac{m}{n}
\]

Solving for \( y \), we have:

\[
y = \frac{x}{1 - \beta^2} + \frac{\beta}{(1 + \beta)} \frac{m}{n}
\]

Let \( \beta \) be 0.85:

- Increased the external contribution (\( x \)) by 360%!
- The spammer links on accessible pages become more valuable
- Captured 46% of the spam portion (\( m/n \)) of the Web
- The more support pages, the higher this component

The basic link farm structure can be changed in many different ways, yet still be able to influence the PageRank.

- Detecting such link farm structures are difficult.
Combating Link Spam: TrustRank

There are pages the spammers can not get access to
- Human evaluated set of top pages
- Pages on exclusive domains: .edu, .mil, .gov, etc.
- Not news article pages or social stream pages
  - Those pages are high quality themselves, but their outgoing links can not be trusted
  - Some sites have started to forbid outgoing links in comments

Only count PageRank contributed by trustworthy pages.
Eliminates the two main sources of PageRank for $t$, leaving only the negligible random surging portion.

Combating Link Spam: Spam Mass

TrustRank is very aggressive, many of the links in those spammer accessible pages are probably legitimate

Instead, we can compute both scores
- The original PageRank
- The TrustRank

If a page’s PageRank is unusually higher than its TrustRank, then it is a strong indicator of the page being a spam page.

Combat Link Spam: Additional Techniques

For spammers to create those support pages, they have to register a lot of new domain names for those pages
- A single misbehaving domain can easily be detected

Those DNS records are accessible and can be analyzed for features that can indicate spam activities
Review of Lecture

Basics of crawling:
• Estimating size of the Web

Crawling scheduling:
• Impact-based prioritization
• Distributed crawl with consistent hashing

Quality
• (Near-)Duplicate detection
• Link spam prevention