Announcements

- Remaining classwork:
  - Assignment 3, due Wednesday November 13
  - Assignment 4, due Friday November 22
  - Final project, due Sunday December 15
  - Final exam, Tuesday December 17

Ranking and Search Engine Optimization

Info 427

Link analysis

- Which pages look more “important”?

PageRank

- Basic idea: assign a score to every web page
  - The score represents importance or quality, independent of the query
- The structure of the web can help
  - High-quality pages are linked to by many other pages
  - High quality pages

The PageRank Model

- Say you’re a user on the web, visiting a page
  - with probability 1-p, follow a random link on the page;
  - otherwise (probability p), visit a random webpage
  - continue doing this forever
- A page’s score is equal to the probability that the user is visiting that page at any moment in time
The PageRank Model

- Say you’re a user on the web, visiting a page
  - with probability $1-p$, follow a random link on the page;
  - otherwise (probability $p$), visit a random webpage
  - continue doing this forever

- A page’s score is equal to the probability that the user is visiting that page at any moment in time

$$r_j = \frac{p}{n} + \left(1 - p\right) \sum_{i:j \to j} \frac{r_i}{d_i}$$

<table>
<thead>
<tr>
<th>Probability of visiting page j randomly</th>
<th>Probability of visiting page j by following a link</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05 + 0.8(0.25/2) = 0.21</td>
<td>0.21 + 0.05 + 0.8(0.25/2) = 0.36</td>
</tr>
<tr>
<td>0.05 + 0.8(0.55/2) = 0.49</td>
<td>0.49 + 0.05 + 0.8(0.55/1) = 0.64</td>
</tr>
<tr>
<td>0.05 + 0.8(0.35/2) = 0.30</td>
<td>0.30 + 0.05 + 0.8(0.35/1) = 0.47</td>
</tr>
<tr>
<td>0.05 + 0.8(0.41/2) = 0.39</td>
<td>0.39 + 0.05 + 0.8(0.41/1) = 0.56</td>
</tr>
<tr>
<td>0.05 + 0.8(0.42/2) = 0.39</td>
<td>0.39 + 0.05 + 0.8(0.42/1) = 0.56</td>
</tr>
</tbody>
</table>

Computing PageRank

- The iterative algorithm from before can be used
  - Algorithm:
    - Assign an initial score to each node (e.g. 1/n, but doesn’t matter)
    - Then repeat:
      - Compute a new score for each node, using the scores computed during the last iteration for the other nodes, and the equation:
        $$r_j = \frac{p}{n} + \left(1 - p\right) \sum_{i:j \to j} \frac{r_i}{d_i}$$
      - Useful facts: This algorithm always converges, and
        - All of the scores are non-zero and in the range [0,1]
        - The scores sum to 1

An example

$$r_j = \frac{p}{n} + \left(1 - p\right) \sum_{i:j \to j} \frac{r_i}{d_i}$$

<table>
<thead>
<tr>
<th>Probability of visiting page j randomly</th>
<th>Probability of visiting page j by following a link</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25 + 0.05 + 0.8(0.25/2) = 0.46</td>
<td>0.46 + 0.05 + 0.8(0.25/2) = 0.56</td>
</tr>
<tr>
<td>0.25 + 0.05 + 0.8(0.55/1) = 0.72</td>
<td>0.72 + 0.05 + 0.8(0.55/2) = 0.87</td>
</tr>
<tr>
<td>0.25 + 0.05 + 0.8(0.35/1) = 0.67</td>
<td>0.67 + 0.05 + 0.8(0.35/2) = 0.76</td>
</tr>
<tr>
<td>0.25 + 0.05 + 0.8(0.41/1) = 0.68</td>
<td>0.68 + 0.05 + 0.8(0.41/2) = 0.77</td>
</tr>
<tr>
<td>0.25 + 0.05 + 0.8(0.36/1) = 0.69</td>
<td>0.69 + 0.05 + 0.8(0.36/2) = 0.78</td>
</tr>
</tbody>
</table>

Linear algebra interpretation

$$r_j = \frac{p}{n} + \left(1 - p\right) \sum_{i:j \to j} \frac{r_i}{d_i}$$

• Alternative formulation, using matrices
  - Define a matrix $A$, where $A_{i,j}$ is 1 if there is a link between page $i$ and $j$, and 0 otherwise
  - Define a matrix $B$, where $B_{i,j}$ is the probability that the user jumps to page $j$ given that he/she is at page $i$:
    $$B_{i,j} = \frac{p}{n} + \left(1 - p\right) \frac{1}{d_i} A_{i,j}$$

• Let $r$ be the (unknown) vector of all page ranks. Then:
  $$r = B^T \bar{r}$$

An eigenvector problem!

• Eigen decomposition finds solutions to the equation:
  $$Ax = \lambda x$$
  - where $A$ is a square matrix
  - $\lambda$ is a scalar (single real number), an eigenvalue
  - $x$ is a vector, an eigenvector

• In the PageRank problem, we are interested in the single eigenvector corresponding to eigenvalue 1
  $$\bar{r} = B^T \bar{r}$$
  - By the Perron-Frobenius theorem, there is always a solution
An enormous eigenvector problem!

\[ B_{ij} = \frac{p}{n} + (1 - p) \frac{1}{d_i} A_{ij} \]

\[ \vec{v} = B^T \vec{v} \]

- How to solve an eigen problem this huge?
  - There’s a simple way of finding the principal eigenvector
  - For any nondegenerate vector \( \vec{v} \), the principal eigenvector of \( B \) is,
    \[ B^T \vec{B} \vec{B}^T \vec{B}^T \ldots B^T \vec{v} = \lim_{n \to \infty} B^n \vec{v} \]
  - This computation is equivalent to the iterative algorithm we used earlier!

Google servers, 1998

- Google used to re-compute PageRank, by solving this eigen decomposition problem, about every month.

Google servers, 2008

- Now, updated constantly and dynamically (?)

FAQ on Pagerank

- Pages that link outside the corpus?
- Handling self-links?
- My PageRanks are very tiny?
- My PageRanks are greater than 1 or less than 0?
- My PageRanks don’t sum to 1?

Sample PageRanks

- Google is very protective of ranks, but makes coarse PageRanks available through the Google Toolbar

```
<table>
<thead>
<tr>
<th>Coarse PageRank</th>
<th>Site</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td><a href="http://www.cnn.com">www.cnn.com</a></td>
</tr>
<tr>
<td>9</td>
<td><a href="http://www.indiana.edu">www.indiana.edu</a></td>
</tr>
<tr>
<td>8</td>
<td><a href="http://www.in.gov">www.in.gov</a></td>
</tr>
<tr>
<td>7</td>
<td><a href="http://www.pr.com">www.pr.com</a></td>
</tr>
<tr>
<td>6</td>
<td><a href="http://www.indiana.edu/~djcran/">www.indiana.edu/~djcran/</a></td>
</tr>
<tr>
<td>5</td>
<td><a href="http://www.cs.indiana.edu/~djcran/">www.cs.indiana.edu/~djcran/</a></td>
</tr>
<tr>
<td>4</td>
<td><a href="http://www.inrd.com">www.inrd.com</a></td>
</tr>
<tr>
<td>3</td>
<td><a href="http://www.iurecsports.org/aquatics_fall_hours">http://www.iurecsports.org/aquatics_fall_hours</a></td>
</tr>
</tbody>
</table>
```

Alternative to PageRank

- There are two important types of pages on the web
  - *Authorities* contain important, trustworthy information
    - e.g. news articles, academic papers, corporate websites,
  - *Hubs* contain many links to other pages
    - e.g. web directories, libraries, blogs, Wikipedia entries...
HITS algorithm [Kleinberg97]

- Hypertext-Induced Topic Selection (HITS)
  - Like PageRank, but treats hubs and authorities differently
  - A good authority is linked to by many good hubs; a good hub links to many good authorities
  - Each page gets an authority score and a hub score,
    \[ a_j = \sum_{i=1}^{n} h_i, \quad h_i = \sum_{j=1}^{n} a_j, \]
    or in terms of the adjacency matrix \( A \),
    \[ \vec{a} = A^T \vec{h} , \quad \vec{h} = A \vec{a} . \]

PageRank vs. HITS

- Advantages of HITS
  - Explicitly models hub and authority structure of the web
  - Handles pages with low in-degree and/or out-degree better
  - Topic-specific

- Advantages of PageRank
  - Can compute PageRank of every page ahead of time; HITS requires more computation at query-time

- Disadvantages of HITS and PageRank
  - Unstable: minor link changes can cause large score changes
  - Possibility of nefarious users biasing the search results

Combining text and link analysis

- Nobody knows how Google does this
  - It’s their carefully protected “secret recipe”

- A reasonable strategy: Pages should have both high content similarity and high PR
  - if either is low, rank should be low, even if other is high
  - This suggests using a product of the scores,
    \[ \text{score}(\text{page}) = \text{sim}(\text{page, qry}) \times \text{pagerank}(\text{page}) \]

How Google did it (B&P 1998)

"...Google maintains much more information about web documents than typical search engines. Every hitlist includes position, font, and capitalization information. Additionally, we factor in hits from anchor text and the PageRank of the document. Combining all of this information into a rank is difficult. We designed our ranking function so that no particular factor can have too much influence. First, consider the simplest case – a single word query. In order to rank a document with a single word query, Google looks at that document’s hit list for that word. Google considers each hit to be one of several different types (title, anchor, URL, plain text large font, plain text small font, ...), each of which has its own type-weight. The type-weights make up a vector indexed by type. Google counts the number of hits of each type in the hit list. Then every count is converted into a count-weight. Count-weights increase linearly with counts at first but quickly taper off so that more than a certain count will not help. We take the dot product of the vector of count-weights with the vector of type-weights to compute an IR score for the document. Finally, the IR score is combined with PageRank to give a final rank to the document."
How Google did it (B&P 1998)

For a multi-word search, the situation is more complicated. Now multiple hit lists must be scanned through at once so that hits occurring close together in a document are weighted higher than hits occurring far apart. The hits from the multiple hit lists are matched up so that nearby hits are matched together. For every matched set of hits, a proximity is computed. The proximity is based on how far apart the hits are in the document (or anchor) but is classified into 10 different value "bins" ranging from a phrase match to "not even close". Counts are computed not only for every type of hit but for every type and proximity. Every type and proximity pair has a type-prox-weight. The counts are converted into count-weights and we take the dot product of the count-weights and the type-prox-weights to compute an IR score. All of these numbers and matrices can all be displayed with the search results using a special debug mode. These displays have been very helpful in developing the ranking system...

Google’s ranking factors

• Google uses >200 factors in deciding a page’s rank
  — Exact list not publicly known; likely factors include:
    — Position of keyword in page
      • Good: Keyword appears in domain name, URL, title, heading, large font, near top of page
    • Bad: invisible text, tiny text, hidden text (in HTML code)
  — Page content
    • Good: freshness of page (but not newness of site)
    • Bad: Pages with only images, many "poison words" (vulgaries and frequent spam words), copied text from other sites, repeated or unrelated words

Google’s ranking factors

— Links
  • Good: Modest number of outlinks (<100?), links to high-quality sites
  • Bad: Many invalid links, linking to many low-quality sites, content of link text changes frequently, single pixel links
— Incoming links
  • Good: lots of links from high-quality pages (high PageRank), keywords in anchor text of incoming links, older links, position of link on page (earlier is better), sudden spike in incoming links, links from certain "expert" sites
  • Bad: Suspiciously high spike in incoming links

Google’s ranking factors

— Past user behavior
  • Good: Many previous Google visitors have visited page (high click-through rate), long time spent on page, many users bookmark page
  • Bad: Many previous visitors quickly exited page
— Other factors
  • Good: .gov, .edu, or .gov hostname, freshness of page
  • Bad: Very long pages, very long URLs, very new site, immediate redirecting to another page, excessive Javascript, domain name registered for short period of time or about to expire, violating Google’s Terms of Service, being caught using deception

User-specific factors

• Google also uses customizes search results to a particular user — what it knows about you
  — Your location
User-specific factors

- Google also uses customizes search results to a particular user – what it knows about you
  - Your location
  - Your social connections
  - Your search history

Multiple algorithms

- Google is thought to use multiple ranking algorithms at any moment in time
  - i.e. your search results depend on which algorithm happens to answer your query

- Why do this?