Today’s topic

- Link-based ranking in web search engines

Web idiosyncrasies

- Distributed authorship
  - Millions of people creating pages with their own style, grammar, vocabulary, opinions, facts, falsehoods …
  - Not all have the purest motives in providing high-quality information - commercial motives drive “spamming”.
  - The open web is largely a marketing tool.
    - IBM’s home page does not contain *computer*.

More web idiosyncrasies

- Some pages have little or no text (gifs may embed text)
- Variety of languages, lots of distinct terms
  - Over 100M distinct “terms”!
- Long lists of links
- Size: >1B pages, each with ~1K terms.
  - Growing at a few million pages/day.
Link analysis

- Two basic approaches
  - Universal, query-independent ordering on all web pages (based on link analysis)
    - Of two pages meeting a (text) query, one will always win over the other, regardless of the query
  - Query-specific ordering on web pages
    - Of two pages meeting a query, the relative ordering may vary from query to query

Query-independent ordering

- First generation: using link counts as simple measures of popularity.
  - Two basic suggestions:
    - Undirected popularity:
      - Each page gets a score = the number of in-links plus the number of out-links (3+2=5).
    - Directed popularity:
      - Score of a page = number of its in-links (3).

Query processing

- First retrieve all pages meeting the text query (say venture capital).
- Order these by their link popularity (either variant on the previous page).

Spamming simple popularity

- Exercise: How do you spam each of the following heuristics so your page gets a high score?
  - Each page gets a score = the number of in-links plus the number of out-links.
  - Score of a page = number of its in-links.
Pagerank scoring

- Imagine a browser doing a random walk on web pages:
  - Start at a random page
  - At each step, go out of the current page along one of the links on that page, equiprobably
- “In the steady state” each page has a long-term visit rate - use this as the page’s score.

Not quite enough

- The web is full of dead-ends.
  - Random walk can get stuck in dead-ends.
  - Makes no sense to talk about long-term visit rates.

Teleporting

- At each step, with probability 10%, jump to a random web page.
- With remaining probability (90%), go out on a random link.
  - If no out-link, stay put in this case.

Result of teleporting

- Now cannot get stuck locally.
- There is a long-term rate at which any page is visited (not obvious, will show this).
- How do we compute this visit rate?
Markov chains

- A Markov chain consists of $n$ states, plus an $n \times n$ transition probability matrix $P$.
- At each step, we are in exactly one of the states.
- For $1 \leq i, j \leq n$, the matrix entry $P_{ij}$ tells us the probability of $j$ being the next state, given we are currently in state $i$.

\[
\sum_{j} P_{ij} = 1
\]

Ergodic Markov chains

- A Markov chain is ergodic if
  - you have a path from any state to any other
  - you can be in any state at every time step, with non-zero probability.

Markov chains are abstractions of random walks.

Exercise: represent the teleporting random walk from 3 slides ago as a Markov chain, for this case:

Ergodic Markov chains

- For any ergodic Markov chain, there is a unique long-term visit rate for each state.
  - Steady-state distribution.
- Over a long time-period, we visit each state in proportion to this rate.
- It doesn’t matter where we start.
Probability vectors

- A probability vector \( \mathbf{x} = (x_1, \ldots, x_n) \) tells us where the walk is at any point.
- E.g., \( (000\ldots1\ldots000) \) means we’re in state \( i \).

More generally, the vector \( \mathbf{x} = (x_1, \ldots, x_n) \) means the walk is in state \( i \) with probability \( x_i \).

\[ \sum_{j=1}^{n} x_j = 1. \]

Change in probability vector

- If the probability vector is \( \mathbf{x} = (x_1, \ldots, x_n) \) at this step, what is it at the next step?
- Recall that row \( i \) of the transition prob. Matrix \( \mathbf{P} \) tells us where we go next from state \( i \).
- So from \( \mathbf{x} \), our next state is distributed as \( \mathbf{xP} \).

Computing the visit rate

- The steady state looks like a vector of probabilities \( \mathbf{a} = (a_1, \ldots, a_n) \):
  - \( a_j \) is the probability that we are in state \( j \).

For this example, \( a_1 = \frac{1}{4} \) and \( a_2 = \frac{3}{4} \).

How do we compute this vector?

- Let \( \mathbf{a} = (a_1, \ldots, a_n) \) denote the row vector of steady-state probabilities.
- If we our current position is described by \( \mathbf{a} \), then the next step is distributed as \( \mathbf{aP} \).
- But \( \mathbf{a} \) is the steady state, so \( \mathbf{a} = \mathbf{aP} \).
- Solving this matrix equation gives us \( \mathbf{a} \).
  - (So \( \mathbf{a} \) is the (left) eigenvector for \( \mathbf{P} \).)
Another way of computing $a$

- Recall, regardless of where we start, we eventually reach the steady state $a$.
- Start with any distribution (say $x = (10...0)$).
- After one step, we’re at $xP$.
- After two steps at $xP^2$, then $xP^j$ and so on.
- “Eventually” means for “large” $k$, $xP^k = a$.
- Algorithm: multiply $x$ by increasing powers of $P$ until the product looks stable.

Pagerank summary

- Preprocessing:
  - Given graph of links, build matrix $P$.
  - From it compute $a$.
  - The entry $a_i$ is a number between 0 and 1: the pagerank of page $i$.
- Query processing:
  - Retrieve pages meeting query.
  - Rank them by their pagerank.
  - Order is query-independent.

The reality

- Pagerank is used in google, but so are many other clever heuristics
  - more on these heuristics later.

Query-dependent link analysis

- In response to a query, instead of an ordered list of pages each meeting the query, find two sets of inter-related pages:
  - Hub pages are good lists of links on a subject.
    - e.g., “Bob’s list of cancer-related links.”
  - Authority pages occur recurrently on good hubs for the subject.
Hubs and Authorities

- Thus, a good hub page for a topic *points to* many authoritative pages for that topic.
- A good authority page for a topic is *pointed to* by many good hubs for that topic.
- Circular definition - will turn this into an iterative computation.

High-level scheme

- Extract from the web a base set of pages that *could* be good hubs or authorities.
- From these, identify a small set of top hub and authority pages;
  - iterative algorithm.

Base set

- Given text query (say *browser*), use a text index to get all pages containing *browser*.
  - Call this the root set of pages.
- Add in any page that either
  - points to a page in the root set, or
  - is pointed to by a page in the root set.
- Call this the base set.
Assembling the base set

- Root set typically 200-1000 nodes.
- Base set may have up to 5000 nodes.
- How do you find the base set nodes?
  - Follow out-links by parsing root set pages.
  - Get in-links (and out-links) from a connectivity server.
  - (Actually, suffices to text-index strings of the form `href="URL"` to get in-links to `URL`.)

Distilling hubs and authorities

- Compute, for each page $x$ in the base set, a hub score $h(x)$ and an authority score $a(x)$.
- Initialize: for all $x$, $h(x) \leftarrow 1$; $a(x) \leftarrow 1$;
- Iteratively update all $h(x)$, $a(x)$; $\Leftarrow$ Key
- After iteration, output pages with highest $h()$ scores as top hubs; highest $a()$ scores as top authorities.

Iterative update

- Repeat the following updates, for all $x$:

  \[
  h(x) \leftarrow \sum_{\alpha \neq x} a(y) \\
  a(x) \leftarrow \sum_{\gamma \neq x} h(y)
  \]
Scaling

- To prevent the $h()$ and $a()$ values from getting too big, can scale down after each iteration.
- Scaling factor doesn’t really matter:
  - we only care about the relative values of the scores.

How many iterations?

- Claim: relative values of scores will converge after a few iterations:
  - in fact, suitably scaled, $h()$ and $a()$ scores settle into a steady state!
  - proof of this comes later.
- In practice, ~5 iterations get you close to stability.

Things to note

- Pulled together good pages regardless of language of page content.
- Use only link analysis after base set assembled
  - iterative scoring is query-independent.
- Iterative computation after text index retrieval - significant overhead.
Proof of convergence

- \( n \times n \) adjacency matrix \( A \):
  - each of the \( n \) pages in the base set has a row and column in the matrix.
  - Entry \( A_{ij} = 1 \) if page \( i \) links to page \( j \), else \( = 0 \).

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Hub/authority vectors

- View the hub scores \( h() \) and the authority scores \( a() \) as vectors with \( n \) components.
- Recall the iterative updates
  \[
  h(x) \leftarrow \sum_{y} a(y)
  \]
  \[
  a(x) \leftarrow \sum_{y} h(y)
  \]

Rewrite in matrix form

- \( h = Aa \).
- \( a = A'h \).

Substituting, \( h = A\A'h \) and \( a = A'Aa \).
Thus, \( h \) is an eigenvector of \( AA' \) and \( a \) is an eigenvector of \( A'A \).

Resources

- MIR 13
- The Anatomy of a Large-Scale Hypertextual Web Search Engine
  - http://citeseer.nj.nec.com/brin98anatomy.html
- Authoritative Sources in a Hyperlinked Environment
  - http://citeseer.nj.nec.com/kleinberg97authoritative.html
- Hypersearching the Web
- Dubhashi resource collection covering recent topics
  - http://www.cs.chalmers.se/~dubhashi/Courses/intense00.html