Introduction to Information Retrieval

CS276
Information Retrieval and Web Search
Pandu Nayak and Prabhakar Raghavan
Lecture 18: Link analysis

Today’s lecture – hypertext and links

- We look beyond the content of documents
  - We begin to look at the hyperlinks between them
- Address questions like
  - Do the links represent a conferral of authority to some pages? Is this useful for ranking?
  - How likely is it that a page pointed to by the CERN home page is about high energy physics
- Big application areas
  - The Web
  - Email
  - Social networks

Links are everywhere

- Powerful sources of authenticity and authority
  - Mail spam – which email accounts are spammers?
  - Host quality – which hosts are “bad”?
  - Phone call logs

- The Good, The Bad and The Unknown

Simple iterative logic

- The Good, The Bad and The Unknown
  - Good nodes won’t point to Bad nodes
  - If you point to a Bad node, you’re Bad
  - If a Good node points to you, you’re Good

Simple iterative logic

- Good nodes won’t point to Bad nodes
  - If you point to a Bad node, you’re Bad
  - If a Good node points to you, you’re Good
Simple iterative logic

- **Good** nodes won’t point to **Bad** nodes
  - If you point to a **Bad** node, you’re **Bad**
  - If a **Good** node points to you, you’re **Good**

Sometimes need probabilistic analogs – e.g., mail spam

Many other examples of link analysis

- Social networks are a rich source of grouping behavior
- E.g., Shoppers’ affinity – Goel+Goldstein 2010
  - Consumers whose friends spend a lot, spend a lot themselves

Our primary interest in this course

- Analogs of most IR functionality based purely on text
  - Scoring and ranking
  - Link-based clustering – topical structure from links
  - Links as features in classification – documents that link to one another are likely to be on the same subject
- Crawling
  - Based on the links seen, where do we crawl next?

The Web as a Directed Graph

Assumption 1: A hyperlink between pages denotes a conferment of authority (quality signal)

Assumption 2: The text in the anchor of the hyperlink describes the target page (textual context)
Anchor Text

WWW Worm - McBryan [Mcbr94]

- For *ibm* how to distinguish between:
  - IBM’s home page (mostly graphical)
  - IBM’s copyright page (high term freq. for ‘ibm’)
  - Rival’s spam page (arbitrarily high term freq.)

![Diagram of anchor text and page]

“A million pieces of anchor text with "ibm" send a strong signal

www.ibm.com

Indexing anchor text

- When indexing a document D, include (with some weight) anchor text from links pointing to D.

![Diagram of indexing anchor text]

Other applications

- Weighting/filtering links in the graph
- Generating page descriptions from anchor text

Citation Analysis

- Citation frequency
  - Bibliographic coupling frequency
    - Articles that co-cite the same articles are related
- Citation indexing
  - Who is this author cited by? (Garfield 1972)
  - Pagerank preview: Pinsker and Narin ’60s

The web isn’t scholarly citation

- Millions of participants, each with self interests
- Spamming is widespread
- Once search engines began to use links for ranking (roughly 1998), link spam grew
  - You can join a group of websites that heavily link to one another
In-links to pages – unusual patterns

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 a dead end, jump to a random web page.
- At any non-dead end, with probability 10%, jump to a random web page.
  - With remaining probability (90%), go out on a random link.
  - 10% - a parameter.

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 $i$. 

Markov chains

- Clearly, for all $i$, $\sum_j P_{ij} = 1$.
- Markov chains are abstractions of random walks.
- Exercise: represent the teleporting random walk from 3 slides ago as a Markov chain, for this case:

![Diagram of Markov chain]

Ergodic Markov chains

- For any (ergodic) Markov chain, there is a unique long-term visit rate for each state.
  - Steady-state probability 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 (row) vector $x = (x_1, \ldots, x_n)$ tells us where the walk is at any point.
  - E.g., $(0, 0, 1, 0, 0)$ means we’re in state $i$.
  - $\sum_i x_i = 1$.

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

Change in probability vector

- If the probability vector is $x = (x_1, \ldots, x_n)$ at this step, what is it at the next step?
- Recall that row $i$ of the transition probability matrix $P$ tells us where we go next from state $i$.
- So from $x$, our next state is distributed as $xP$.
  - The one after that is $xP^2$, then $xP^3$, etc.
  - (Where) Does the converge?

How do we compute this vector?

- Let $a = (a_1, \ldots, a_n)$ denote the row vector of steady-state probabilities.
- If our current position is described by $a$, then the next step is distributed as $aP$.
- But $a$ is the steady state, so $a = aP$.
- Solving this matrix equation gives us $a$.
  - So $a$ is the (left) eigenvector for $P$.
  - (Corresponds to the “principal” eigenvector of $P$ with the largest eigenvalue.)
  - Transition probability matrices always have largest eigenvalue 1.

Pagerank summary

- Preprocessing:
  - Given graph of links, build matrix $P$.
  - From it compute $a$ – left eigenvector of $P$.
  - 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.
  - But this rank order is query-independent
The reality

- Pagerank is used in google and other engines, but is hardly the full story of ranking
  - Many sophisticated features are used
  - Some address specific query classes
  - Machine learned ranking (Lecture 19) heavily used
- Pagerank still very useful for things like crawl policy

Hyperlink-Induced Topic Search (HITS)

- 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.
- Best suited for “broad topic” queries rather than for page-finding queries.
- Gets at a broader slice of common opinion.

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.

The hope

![Diagram of Hubs and Authorities]

Base set

- 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.

High-level scheme

- 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.
Introduction to Information Retrieval

**Visualization**

Get in-links (and out-links) from a *connectivity server*

---

**Iterative update**

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

  $$h(x) \leftarrow \sum_{y \rightarrow x} a(y)$$

  $$a(x) \leftarrow \sum_{y \leftarrow x} h(y)$$

---

**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 $h(x)$, $a(x)$;
- **Key**
- **After iterations**
  - output pages with highest $h()$ scores as top hubs
  - highest $a()$ scores as top authorities.

---

**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.

---

**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.

---

**Japan Elementary Schools**

### Hubs
- Kids' Space
- The Link Page
- The American School in Japan
- KAMISHIBUN Elementary School...
- KAMISHIBUN Elementary School...

### Authorities
- Kids' Space
- The Link Page
- The American School in Japan
- KAMISHIBUN Elementary School...
- KAMISHIBUN Elementary School...
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.

\[
\begin{pmatrix}
1 & 1 & 1 \\
1 & 2 & 0 \\
1 & 0 & 3
\end{pmatrix}
\]

Rewrite in matrix form

- \( h = Aa \).
- \( a = A^t h \).

Substituting, \( h = AA^t h \) and \( a = A^t A a \).

Thus, \( h \) is an eigenvector of \( AA^t \) and \( a \) is an eigenvector of \( A^t A \).

Further, our algorithm is a particular, known algorithm for computing eigenvectors: the power iteration method. Guaranteed to converge.

Issues

- Topic Drift
  - Off-topic pages can cause off-topic “authorities” to be returned
    - E.g., the neighborhood graph can be about a “super topic”
- Mutually Reinforcing Affiliates
  - Affiliated pages/sites can boost each others’ scores
    - Linkage between affiliated pages is not a useful signal

Connectivity servers
Connectivity Server

- Support for fast queries on the web graph
  - Which URLs point to a given URL?
  - Which URLs does a given URL point to?
- Stores mappings in memory from
  - URL to outlinks, URL to inlinks
- Applications
  - Crawl control
  - Web graph analysis
  - Connectivity, crawl optimization
  - Link analysis

Boldi and Vigna 2004

- Webgraph – set of algorithms and a java implementation
- Fundamental goal – maintain node adjacency lists in memory
  - For this, compressing the adjacency lists is the critical component

Adjacency lists

- The set of neighbors of a node
- Assume each URL represented by an integer
- E.g., for a 4 billion page web, need 32 bits per node
- Naively, this demands 64 bits to represent each hyperlink

Adjacency list compression

- Properties exploited in compression:
  - Similarity (between lists)
  - Locality (many links from a page go to “nearby” pages)
  - Use gap encodings in sorted lists
  - Distribution of gap values

Storage

- Boldi/Vigna get down to an average of ~3 bits/link
  - Why is this remarkable?
- (URL to URL edge)
- How?

Main ideas of Boldi/Vigna

- Consider lexicographically ordered list of all URLs, e.g.,
  - www.stanford.edu/alchemy
  - www.stanford.edu/biology
  - www.stanford.edu/biology/plant
  - www.stanford.edu/biology/plant/copyright
  - www.stanford.edu/biology/plant/people
  - www.stanford.edu/chemistry
**Boldi/Vigna**

- Each of these URLs has an adjacency list
- Main idea: due to templates, the adjacency list of a node is similar to one of the Z preceding URLs in the lexicographic ordering
- Express adjacency list in terms of one of these
  - E.g., consider these adjacency lists
    - 1, 2, 4, 8, 16, 32, 64
    - 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144
- Encode as (-2), remove 9, add 8

**Main advantages of BV**

- Depends only on locality in a canonical ordering
- Lexicographic ordering works well for the web
- Adjacency queries can be answered very efficiently
  - To fetch out-neighbors, trace back the chain of prototypes
  - This chain is typically short in practice (since similarity is mostly intra-host)
  - Can also explicitly limit the length of the chain during encoding
- Easy to implement one-pass algorithm

**Gap encodings**

- Given a sorted list of integers x, y, z, ..., represent by x, y-x, z-y, ...
- Compress each integer using a code
  - γ code - Number of bits = 1 + 2 \lfloor \log x \rfloor
  - δ code: ...
  - Information theoretic bound: 1 + \lfloor \log x \rfloor bits
  - ζ code: Works well for integers from a power law

---

**Resources**

- IIR Chap 21
- The WebGraph framework I: Compression techniques (Boldi et al. 2004)