15-853: Algorithms in the Real World

Indexing and Searching III (well actually II)
- Link Analysis
- Near duplicate removal

Indexing and Searching Outline

Introduction: model, query types
Inverted Indices: Compression, Lexicon, Merging
Vector Models: Weights, cosine distance
Latent Semantic Indexing:
  Link Analysis: PageRank (Google), HITS
  Near Duplicate Removal:

Link Analysis

The goal is to rank pages.
We want to take advantage of the link structure to do this.

Two main approaches:
- Static: we will use the links to calculate a ranking of the pages offline (Google)
- Dynamic: we will use the links in the results of a search to dynamically determine a ranking (Clever - hubs and authorities)

The Link Graph

View documents as graph nodes and the hyperlinks between documents as directed edges.

Can give weights on edges (links) based on
1. position in the document
2. weight of anchor term
3. # of occurrences of link
4. ...
The Link Matrix

An adjacency matrix of the link graph.
Each row corresponds to the Out-Links of a vertex
Each column corresponds to the In-links of a vertex
Often normalize columns to sum to 1 so the dot-product with self is 1.

Google: Page Rank (1)

**Goal:** to give a total order (rank) on the "importance" of all pages they index

**Possible Method:** count the number of in-links.

**Problems?**

Google: Page Rank (2)

**Refined Method:** weigh the source by its importance.

**Imagine the following:** Jane Surfer, has a lot of time to waste, and randomly selects an outgoing link each time she gets to a page. She counts how often she visits each page.

**Problem?**
Side Topic: Markov Chains

A **discrete time stochastic process** is a sequence of random variables \((X_0, X_1, ..., X_n, ...)\) where the 0, 1, ..., \(n\) ... are discrete points in time.

A **Markov chain** is a discrete-time stochastic process defined over a finite (or countably infinite) set of states \(S\) in terms of a matrix \(P\) of **transition probabilities**.

**Memorylessness property:** for a Markov chain
\[
\Pr[X_{n+1} = j \mid X_0 = i_0, X_1 = i_1, ..., X_n = i] = \Pr[X_{n+1} = j \mid X_n = i]
\]

Let \(\pi(t)\) be the probability of being in state \(i\) at time step \(t\).

Let \(\pi(t) = [\pi_0(t), \pi_1(t), ...]\) be the vector of probabilities at time \(t\).

For an initial probability distribution \(\pi(0)\), the probabilities at time \(n\) are
\[
\pi(n) = \pi(0) P^n
\]

A probability distribution \(\pi\) is **stationary** if \(\pi = \pi P\) (i.e. is an eigenvector of \(P\) with eigenvalue 1).

Side Topic: Markov Chains

A Markov chain is **irreducible** if the underlying graph (of \(P\)) consists of a single strong component (i.e. all states are reachable from all other states).

A **period** of a state \(i\) is the integer \(t\) such that \(P^n_{ii} = 0\) for all \(n \neq t, 2t, 3t, ...\) (i.e. only comes back to itself every \(t\) steps).

A state is **aperiodic** if it has period 1.

A Markov chain is **aperiodic** if all states are aperiodic.

A state is **recurrent** if upon entering this state the process will definitely return to the state.
Google: Page Rank (3)

Stationary probability of a Markov Chain.
Transition probability matrix is simply matrix with equal probability of leaving on each link
Remaining problem: Gives too high of a rank to the Yahoo's of the world, and too low to clusters with low connectivity into the cluster.
Solution: With some probability jump to a random page instead a random outgoing link. Transition probability matrix is:
\[ T = \epsilon U + (1 - \epsilon)A \]
U is the uniform matrix, and A is the normalized link matrix (each column sums to 1).

Hubs and Authorities (1)

Goal: To get a ranking for a particular query (instead of the whole web)
Assume: We have a search engine that can give a set of pages P that match a query
Find all pages that point to P (back set) and that P point to (forward set).
Include b, d and f in a document set D.

Google: Page Rank (4)

Want to find \( \pi \), such that \( T \pi = \pi \)
This corresponds to finding the principal eigenvector.
Methods:
- Power method: iterate \( p_{n+1} = Tp_n \), until it settles
pick \( p_0 \) randomly,
decomposing \( p_0 \) into eigenvectors \( \lambda_1 e_1 + \lambda_2 e_2 + \ldots \)
we have \( p_i = \lambda_1^i e_1 + \lambda_2^i e_2 + \ldots \)
- Multilevel method: solve on contracted graph, use as initial vector for power method on the expanded graph.
- Lanczos method: diagonalize, then iterate
All methods are quite expensive

Hubs and Authorities (2)

For a particular query we might want to find:
1. The "authorities" on the topic (where the actual information is kept)
2. The "hub" sites that point to the best information on the topic (typically some kind of link site)
Important authorities will have many pages that point to them
Important hubs will point to many pages
But: As with page-rank, just having a lot of pointers does not mean much
**Hubs and Authorities (3)**

- Vector $h$ is a weight for each document in $D$ for how good a hub it is.
- Vector $a$ is a weight for each document in $D$ for how good an authority it is.
- $T$ is the adjacency matrix.

Now repeat:
- $h = T^T a$
- $a = Th$
- until it settles.

**Hubs and Authorities (4)**

What is this process doing?

- $a_{u,t} = (T^T T) a_i$
- $h_{u,t} = (T^T T) h_i$

Power method for eigenvectors of $T^T T$ and $T^T T$

$SVD(T)$: first column of $U$ and first row of $V$

**Hubs and Authorities (5)**

**Problem (Topic Drift):** Hubs and Authorities will tend to drift to the general, instead of specific.

- E.g. you search for "Indexing Algorithms" which identifies all pages with those terms in it, and the back and forward pages.
- Now you use the SVD to find Hubs + Authorities.

These will most likely be pages on Algorithms in general rather than Indexing Algorithms.

**Solution:** weight the documents with "indexing" prominently in them, more heavily.

- $a = T W h$, $h = T^T W a$, where $W$ is a diagonal matrix with weights on the diagonal.
- Same as finding SVD of $T = TW$

**Extensions**

What about second eigenvectors?

Mix terms and links.
**Indexing and Searching Outline**

*Introduction:* model, query types  
*Inverted Indices:* Compression, Lexicon, Merging  
*Vector Models:* Weights, cosine distance  
*Latent Semantic Indexing:*  
*Link Analysis:* PageRank (Google), HITS  
*Near Duplicate Removal:*  

---

**Syntactic Clustering**

*Goal:* find pages for which either  
1. Most of the content is the same  
2. One is contained in the other (possibly with changes)  
*Constraints:*  
- Need to do it with only a small amount of information per document: a *sketch*  
- Comparison of documents should take time linear in the size of the sketch  
- Should be reasonably robust against adversary

---

**Syntacting Clustering of the Web**

*Problem:* Many pages on the web are almost the same but not exactly the same.  
- Mirror sites with local identifier and/or local links.  
- Spam sites that try to improve their "pagerank"  
- Plagiarized sites  
- Revisions of the same document  
- Program documentation converted to HTML  
- Legal documents  
These near duplicates cause web indexing sites a huge headache.
w-Shingling

View each shingle as a value.

S_w(A): the set of w-shingles for document A.

As shorthand, I will often drop the w subscript.
Hash

How about a random hash?
i.e. for each shingle $x$ take $h(x)$, then pick the minimum $k$ values.

$$S_k(A) = \min_k \{ h(x) : x \in S(A) \}$$

Some Theory: Pairwise independent

**Universal Hash functions:** (Pairwise independent)
- $H$ : a finite collection (family) of hash functions mapping $U \rightarrow \{0\ldots m-1\}$
- $H$ is **universal** if,
  - for $h \in H$ picked uniformly at random,
  - and for all $x_1, x_2 \in U$, $x_1 \neq x_2$
  $$\Pr(h(x_1) = h(x_2)) = 1/m$$

The class of hash functions
- $h_{ab}(x) = ((a \times x + b) \mod p) \mod m$

is universal (p ≥ m is a prime, $a = \{1\ldots p-1\}, b = \{0\ldots p-1\}$)

Some Theory: Minwise independent

**Minwise independent permutations:**
- $S_n$ : a finite collection (family) of permutations mapping $\{1\ldots n\} \rightarrow \{1\ldots n\}$
- $H$ is **minwise independent** if,
  - for $\pi \in S_n$ picked uniformly at random,
  - and for $X \subseteq \{1\ldots n\}$, and all $x \in X$
  $$\Pr(\min_{\pi(X)}(x)) = \pi(x)) \leq 1/|X|$$

It is actually hard to find a “compact” collection of hash functions that is minwise independent, but we can use an approximation.

Back to similarity

If $\pi \in S_n$ and $S_n$ is minwise independent then:

$$\Pr(\min(\pi(S(A))) = \min(\pi(S(B)))) = \frac{|S(A) \cap S(B)|}{|S(A) \cup S(B)|} = r(A,B)$$

This suggests we could just keep one minimum value as our "sketch", but our confidence would be low.

What we want for a sketch of size $k$ is either
- use $k\pi$'s,
- or keep the $k$ minimum values for one $\pi$