PageRank

CISC489/689-010, Lecture #20

Wednesday, April 29th

Ben Carterette

Web Search

- Problem:
  - Web search engines easily hacked
  - I want to sell something; I'll just add a few popular keywords to my page over and over and over again
  - All the retrieval models we've discussed will score that page higher for those keywords
- Other problems:
  - No hackers, but top-ranked pages are coming from deep within a site, or from pages that change often, or pages about very obscure topics
  - Not really useful

Possible Solution

- Leverage link structure
- Maybe if many pages are linking to a page, that page is more "important"
- Idea:
  - Count the number of inlinks to the page
  - Assign it "importance" based on that number
- Any problem with this?

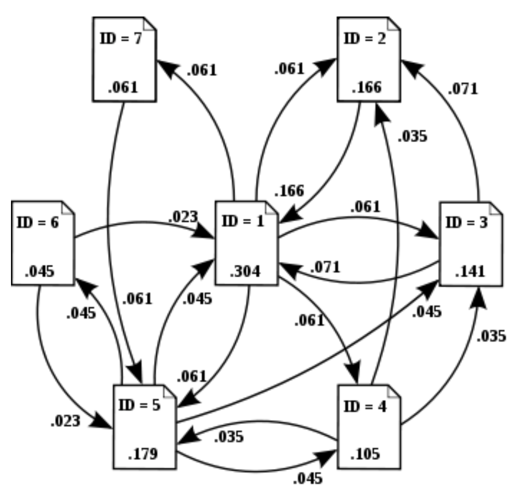
Hacking Link Counts

- I can just make a bunch of pages that link to my spam page
- Inlink count will be high even though my page is not important
- Better idea:
  - Recursively use the importance of the linking pages when calculating the importance of the page

PageRank

- Google's PageRank is probably the best known algorithm
- Intuitive idea: "random surfer" model
  - If you start on a random page on the internet and just start clicking links randomly,
  - What is the probability you will land on page u?
  - If one page has a higher landing probability, the pages it links to have higher landing probabilities as well
  - Higher probability = more authority = better PageRank

Illustration



PageRank Definition

$$R(u) = c \sum_{v \in B_u} \frac{R(v)}{N_v}$$

$R(u)$	is the PageRank of u
B_u	is a set of pages that link to u
v	is one of the pages in B_u
$R(v)$	is the PageRank of v
N_v	is the number of pages that v links to

Sinks

- Problem:
  - I have two pages that only link to each other, plus one page that links to one of them
  - When the "random surfer" gets to one of those pages, he will just keep alternating between them
  - Their PageRank will dominate everything else
- Solution: once in a while the random surfer just starts over at a new page
  - OK, but how do I put that in PageRank?

Random Restarts

$$R'(u) = c \sum_{v \in B_u} \frac{R'(v)}{N_v} + cE(u)$$

$E(u)$ is the probability that a random surfer jumps to u

Calculating PageRank

- A simple iterative algorithm:
  - First, assign a PageRank to every page
    - E.g. $R_0(u) = 1/N$
  - Initialize $E(u) = \alpha$ for all u (0.15/N in original paper)
  - Over iterations i=1…, do
    - Update each PageRank as: $R_{i+1}(u) \quad = \quad \sum_{v \in B_u} \frac{R_i(v)}{N_v}$
    - Calculate d as the sum of PageRanks from the previous iteration minus the sum of PageRanks from the current iteration
    - Update each PageRank as $R_{i+1}(u) = R_{i+1} + d\alpha$
    - Calculate δ as the sum of |PageRanks from the current iteration minus PageRanks from the previous iteration|
    - If δ > ε, PageRanks have converged

Scalability

- Calculating PageRank requires a vector for every URL
- Along with a list of the URLs that link to that URL and a list of URLs that page links to
- Space usage is $O(N^2)$
- Time complexity also $O(N^2)$
- Even for small collections (like Wikipedia), it is nearly impossible to keep all of this in memory

Calculating PageRank with MapReduce

- First: extract links
  - For every page, I need to know all the pages that link to it
  - MapReduce solution:
    - Map operator takes page u and outputs (v, u) for every URL v in page u
    - Reduce operator takes all tuples with key v and reduces them to a list of unique URLs B_u that link to v
      - $(v, (u_1, u_2, u_3, \ldots))$

Calculating PageRank with MapReduce

- Next: calculate PageRank iteratively
  - First, initialize all PageRanks to 1/N
  - Then iteratively MapReduce
    - Map operator takes a page u_i and the URLs v_j (j=1..n) that it links to, and outputs (v_j, $R(u_i)/n$)
    - Reduce operator takes pairs (v_j, $R(u_i)/n$) and outputs

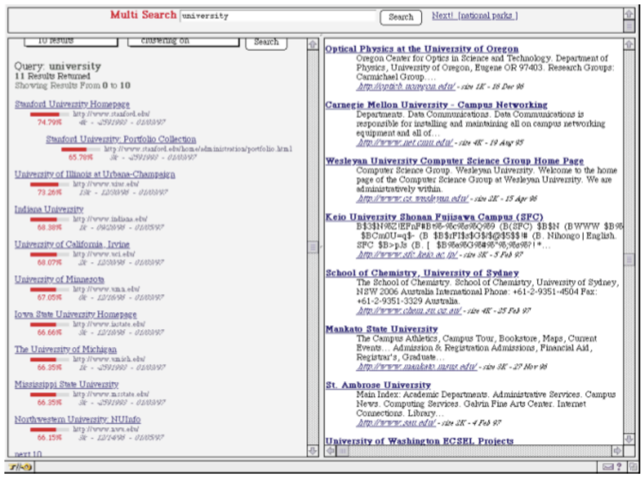
$$\left(v_j, (1-d) + d\sum_{i=1}^{m} \frac{R(u_i)}{n}\right)$$

    - Calculate delta and determine whether converged
    - If not, MapReduce again

PageRank for IR

- PageRank is query-independent
  - The importance of a page is not related to any query
  - We cannot simply rank pages by PageRank
- PageRank can be used to re-rank results that have been retrieved for a query
- It can also be used as a feature in the ranking function
- Or as a weight on anchor text features

Example Use of PageRank



From "The PageRank Citation Algorithm: Bringing Order to the Web", Page et al.

Wikipedia PageRanks

Page Title	PageRank
United States	2.9×10^{-3}
France	1.3×10^{-3}
United Kingdom	1.2×10^{-3}
England	1.0×10^{-3}
Germany	1.0×10^{-3}
Canada	0.9×10^{-3}
2007	0.8×10^{-3}
World War II	0.8×10^{-3}
Australia	0.7×10^{-3}
2008	0.7×10^{-3}

Based on links between Wikipedia pages

A Bit of Theory

- Markov chain:
  - N states with transition probability matrix P
  - At any time we are in exactly one state
  - P_{ij} indicates probability of moving from state i to state j
  - For all i, $\sum_{j=1}^{n} P_{ij} = 1$

A Bit of Theory

- Ergodic Markov chains
  - *Ergodic* means there is a path between any two states
  - No matter what state you start in, the probability of being in any other state after T steps is greater than zero (after a *burn-in* time T_0)
  - Over many steps T, each state has some “visit rate”: starting from any state, we will visit each state according to its visit rate
  - This is the *steady state* for the chain

A Bit of Theory

- Let x_0 be a vector representing our current state

$$x_0 = \begin{bmatrix} 0 & 0 & 0 & \cdots & 1 & \cdots & 0 & 0 \end{bmatrix}$$

1 in position i, 0s everywhere else

- What is the probability of each possible state we can transition to from x_0?

$$x_1 = x_0 P$$

- And the probability of each possible state from x_1 (two steps from x_0)?

$$x_2 = x_1 P = (x_0 P) P = x_0 P^2$$

A Bit of Theory

- After k steps, $x_k = x_0 P^k$
- As k goes to infinity, x_k converges to the steady state
- When x_k is the steady state, $x_k P = x_k$
- The steady state is an *eigenvector* of P
  - As it turns out, it is the *principle eigenvector*
  - Eigenvalue = 1
- If P is a matrix of links between documents, the principle eigenvector holds the PageRanks

PageRank Modifications

- The E(u) quantities solve the sink problem, but can also be used to adjust PageRanks
- Usually, E(u) assigned uniformly
  - Equal probability to jump to any page
- Instead, bias to certain pages
  - One possibility: assign one page E(u) = α, all other pages 0
  - For example, Yahoo home page
  - Then when the "random surfer" restarts, she always restarts at the same place
  - Result: Yahoo gets highest PageRank, followed by pages Yahoo links to

Topic-Based PageRank

- A different kind of random surfer:
  - First picks a category randomly
  - Then jumps to a page randomly within that category
- Instead of calculating a single PageRank for each page, calculate M PageRanks, one for each category
  - Category PageRank = PageRank among other pages in the same category

Topic-Based PageRank for Personalization

- Each individual user is more interested in some categories than others
- Calculate the probability that a user is interested in a category based on the frequency they visit pages in that category
  - E.g. sports = 0.7, finance = 0.2, health = 0.1, all others = 0
- Then the personalized PageRank for a page u is the weighted sum of category PageRanks for that page

$$R(u) = \sum_{i=1}^{M} p_i R_i(u)$$

p_i is the user's category i probability
$R_i(u)$ is the PageRank of u for category i

Hyperlink-Induced Topic Search (HITS)

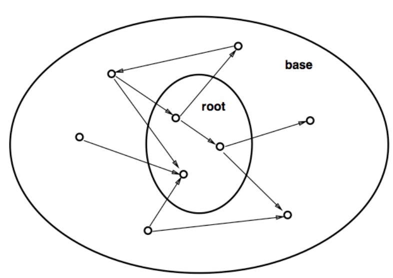
- Another link-graph algorithm
- Idea:
  - Some pages are *authoritative*: they are very informative about a topic
  - Other pages are *hubs* for a topic: they link to a lot of pages on the topic
- Example:
  - CiteSeer links to a lot of computer science research papers—it's a computer science resesarch hub
  - The papers it links to are computer science research authorities
- Find both hubs and authorities

Hubs and Authorities

- Hubs are pages that link to a lot of authorities
- Authorities are pages that are linked to by a lot of hubs
- Another recursive definition

HITS Algorithm

- First, get a *root set* of pages
  - Pages that match a query, for example
- From the root set, construct a *base set*
  - Pages that link to the root set and pages that the root set link to



From "Authoritative Sources in a Hyperlinked Environment", J. Kleinberg

Figure 1: Expanding the root set into a base set.

HITS Algorithm

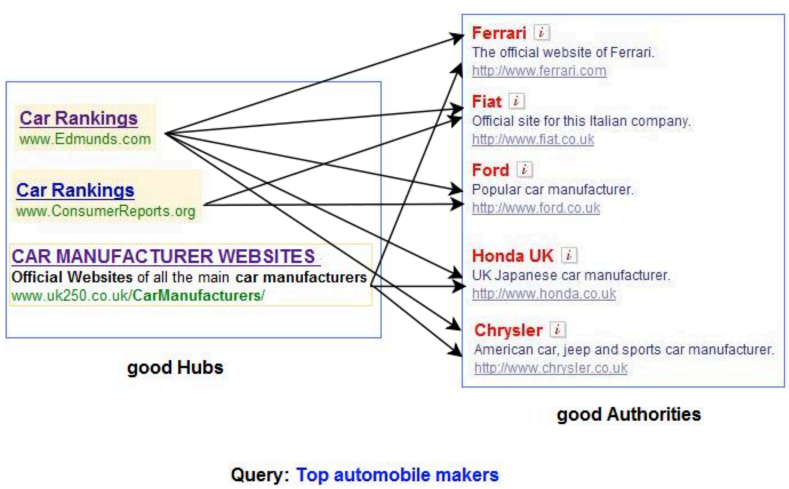
- Initialize "hub score" h(u)=1 and "authority score" a(u)=1 for each page u in the base set
- Then iteratively update h(u) and a(u) for all u:

$$h(u) = \sum_{v \in F_u} a(v)$$
$$a(u) = \sum_{v \in B_u} h(v)$$

F_u = set of pages that u links to
B_u = set of pages that link to u

- After each iteration, divide h(u) and a(u) by some constant
- After only a few iterations, scores converge

HITS Example Results



From http://www.math.cornell.edu/~mec/Winter2009/RalucaRemus/Lecture4/Images/cars1.png

Matrix Form

- P is the transition matrix
- PP’ is a sort of “similarity” matrix in terms of links to other pages
  – Entry i,j is higher if pages i and j link to the same pages
- P’P is a sort of “similarity” matrix in terms of links from other pages
  – Entry i,j is higher is pages i and j are linked to from the same pages

Matrix Form

- We can write the hub score as a matrix-vector product:
  – h = Pa (transition matrix times authority score)
- We can write authority score as
  – a = P’h (transpose of the transition matrix times hub score)
- Substituting, we get
  – h = PP’h
  – a = P’Pa

Matrix Eigenvectors

- If h = PP’h, then h is an eigenvector of the “outlink similarity matrix”
  - Hub scores are a basis vector of a space defined by outlinks
- If a = P’Pa, then a is an eigenvector of the “inlink similarity matrix”
  - Authority scores are a basis vector of a space defined by inlinks