# Relevance Ranking

Manish Gupta

# Today’s Agenda

- Need for Relevance Ranking
- TF and IDF
- Vector Space Model

# Today’s Agenda

- **Need for Relevance Ranking**
- TF and IDF
- Vector Space Model

# Ranked Retrieval

- Thus far, our queries have all been Boolean.
  - Documents either match or don't.
- Good for expert users with precise understanding of their needs and the collection.
  - Also good for applications: Applications can easily consume 1000s of results.
- Not good for the majority of users.
  - Most users are incapable of writing Boolean queries.
  - Most users don't want to wade through 1000s of results.
    - This is particularly true of web search.

# Problem with Boolean search: Feast or Famine

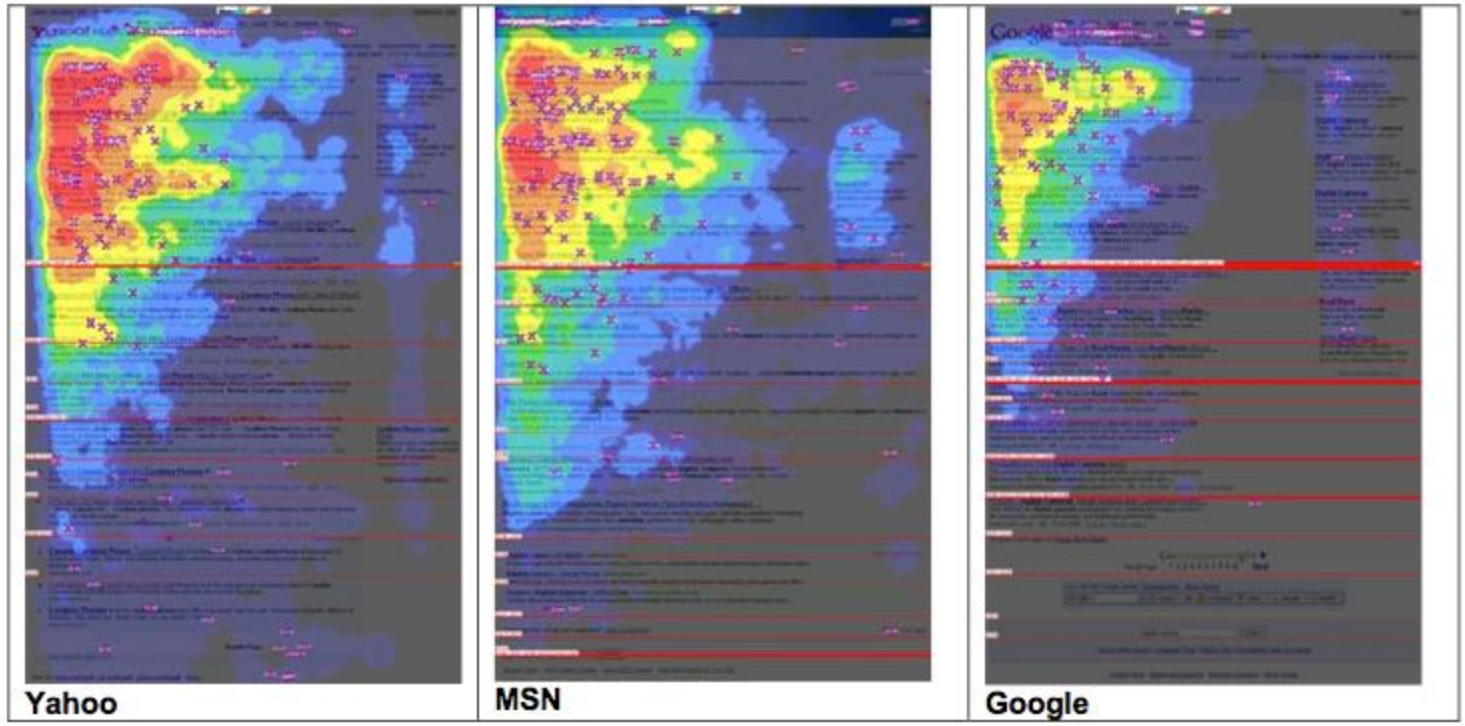
- Boolean queries often result in either too few (=0) or too many (1000s) results.
- Query 1: "*standard user dlink 650*" → 200,000 hits
- Query 2: "*standard user dlink 650 no card found*": 0 hits
- It takes a lot of skill to come up with a query that produces a manageable number of hits.
  - AND gives too few; OR gives too many

# Feast or Famine: OK for Ranked Retrieval

- Rather than a set of documents satisfying a query expression, in ranked retrieval, the system returns an ordering over the (top) documents in the collection for a query
- When a system produces a ranked result set, large result sets are not an issue
  - Indeed, the size of the result set is not an issue
  - We just show the top $k$ ( ≈ 10) results
  - We don't overwhelm the user

  - Premise: the ranking algorithm works. Is it true?

# Eye Tracking Study on Search Results




2006

http://www.mediative.com/eye-tracking-google-through-the-years/

# Scoring as the Basis of Ranked Retrieval

- We wish to return in order the documents most likely to be useful to the searcher
- How can we rank-order the documents in the collection with respect to a query?
- Assign a score – say in [0, 1] – to each document
- This score measures how well document and query "match."

# Query-Document Matching Scores

- We need a way of assigning a score to a query/document pair
- Let's start with a one-term query
- If the query term does not occur in the document: score should be 0
- The more frequent the query term in the document, the higher the score (should be)
- We will look at a number of alternatives for this.
- First take: Jaccard coefficient?

# Issues with Jaccard for Scoring

- It doesn't consider *term frequency* (how many times a term occurs in a document)
- Rare terms in a collection are more informative than frequent terms. Jaccard doesn't consider this information
- We need a more sophisticated way of normalizing for length

# Binary Term-Document Incidence Matrix

| | Antony and Cleopatra | Julius Caesar | The Tempest | Hamlet | Othello | Macbeth |
|---|---|---|---|---|---|---|
| Antony | 1 | 1 | 0 | 0 | 0 | 1 |
| Brutus | 1 | 1 | 0 | 1 | 0 | 0 |
| Caesar | 1 | 1 | 0 | 1 | 1 | 1 |
| Calpurnia | 0 | 1 | 0 | 0 | 0 | 0 |
| Cleopatra | 1 | 0 | 0 | 0 | 0 | 0 |
| mercy | 1 | 0 | 1 | 1 | 1 | 1 |
| worser | 1 | 0 | 1 | 1 | 1 | 0 |

Each document is represented by a binary vector $\in \{0,1\}^{|V|}$

# Term-Document Count Matrices

- Consider the number of occurrences of a term in a document
  - Each document is a count vector in $\mathbb{N}^v$: a column below

| | Antony and Cleopatra | Julius Caesar | The Tempest | Hamlet | Othello | Macbeth |
|---|---|---|---|---|---|---|
| Antony | 157 | 73 | 0 | 0 | 0 | 0 |
| Brutus | 4 | 157 | 0 | 1 | 0 | 0 |
| Caesar | 232 | 227 | 0 | 2 | 1 | 1 |
| Calpurnia | 0 | 10 | 0 | 0 | 0 | 0 |
| Cleopatra | 57 | 0 | 0 | 0 | 0 | 0 |
| mercy | 2 | 0 | 3 | 5 | 5 | 1 |
| worser | 2 | 0 | 1 | 1 | 1 | 0 |

# Today's Agenda

- Need for Relevance Ranking
- **TF and IDF**
- Vector Space Model

# Term Frequency TF

- The term frequency $\text{tf}_{t,d}$ of term *t* in document *d* is defined as the number of times that *t* occurs in *d*.
- Relevance does not increase proportionally with term frequency.
- So use log frequency weighting
- The log frequency weight of term t in d is $w_{td} = 1 + \log_{10} tf_{td}$ if $tf_{td} > 0$; else it is 0.
- Score for a document-query pair: sum over terms *t* in both *q* and *d*
  - $score = \sum_{t \in q \cap d} (1 + \log_{10} tf_{td})$

# Inverse Document Frequency IDF

- Frequent terms are less informative than rare terms
- $df_t$ is the document frequency of $t$: the number of documents that contain $t$
  - $df_t \leq N$
- $idf_t = \log_{10}\left(\frac{N}{df_t}\right)$
- IDF has no effect on ranking one term queries
  - IDF affects the ranking of documents for queries with at least two terms
  - For the query capricious person, IDF weighting makes occurrences of capricious count for much more in the final document ranking than occurrences of person.

# TF-IDF Weighting

- The tf-idf weight of a term is the product of its tf weight and its idf weight.

$$w_{t,d} = \log(1+\mathrm{tf}_{t,d}) \times \log_{10}(N/\mathrm{df}_t)$$

$$\mathrm{Score}(q,d) = \sum_{t \in q \cap d} \mathrm{tf.idf}_{t,d}$$

- Score for a document given a query

- There are many variants
  - How “tf” is computed (with/without logs)
  - Whether the terms in the query are also weighted
  - …

# Binary → Count → Weight Matrix

| | Antony and Cleopatra | Julius Caesar | The Tempest | Hamlet | Othello | Macbeth |
|---|---|---|---|---|---|---|
| Antony | 5.25 | 3.18 | 0 | 0 | 0 | 0.35 |
| Brutus | 1.21 | 6.1 | 0 | 1 | 0 | 0 |
| Caesar | 8.59 | 2.54 | 0 | 1.51 | 0.25 | 0 |
| Calpurnia | 0 | 1.54 | 0 | 0 | 0 | 0 |
| Cleopatra | 2.85 | 0 | 0 | 0 | 0 | 0 |
| mercy | 1.51 | 0 | 1.9 | 0.12 | 5.25 | 0.88 |
| worser | 1.37 | 0 | 0.11 | 4.15 | 0.25 | 1.95 |

Each document is now represented by a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$

# Ranking in Vector Space Model

- Represent both query and document as vectors in the |V|-dimensional space
- Use cosine similarity as the similarity measure
  - Incorporates length normalization automatically (longer vs shorter documents)

$$\cos(\vec{q},\vec{d}) = \frac{\vec{q}\bullet\vec{d}}{\left|\vec{q}\right|\left|\vec{d}\right|} = \frac{\vec{q}}{\left|\vec{q}\right|}\bullet\frac{\vec{d}}{\left|\vec{d}\right|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

- $q_i$ is the tf-idf weight of term *i* in the query
- $d_i$ is the tf-idf weight of term *i* in the document

# TF-IDF Weighting has many Variants

| Term frequency | | Document frequency | | Normalization | |
|---|---|---|---|---|---|
| n (natural) | $\mathrm{tf}_{t,d}$ | n (no) | 1 | n (none) | 1 |
| l (logarithm) | $1 + \log(\mathrm{tf}_{t,d})$ | t (idf) | $\log \frac{N}{\mathrm{df}_t}$ | c (cosine) | $\frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_M^2}}$ |
| a (augmented) | $0.5 + \frac{0.5 \times \mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$ | p (prob idf) | $\max\{0, \log \frac{N-\mathrm{df}_t}{\mathrm{df}_t}\}$ | u (pivoted unique) | $1/u$ |
| b (boolean) | $\begin{cases} 1 & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$ | | | b (byte size) | $1/CharLength^{\alpha}$, $\alpha < 1$ |
| L (log ave) | $\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t \in d}(\mathrm{tf}_{t,d}))}$ | | | | |

Columns headed ‘n’ are acronyms for weight schemes.

SMART Notation: denotes the combination in use in an engine, with the notation *ddd.qqq*, using the acronyms from the previous table

A very standard weighting scheme is: lnc.ltc

# Okapi BM25

- Given a query Q, containing keywords $q_1, \dots, q_n$, the BM25 score of a document D is:

$$score(D,Q) = \sum_{i=1}^{n} IDF(q_i).\frac{tf_{q_i,D}.(k_1+1)}{tf_{q_i,D} + k_1.\left(1 - b + b.\frac{|D|}{avgdl}\right)}$$

- |D| is the length of the document D in words
- avgdl is the average document length in the text collection from which documents are drawn
- $k_1$ and b are free parameters, usually chosen, in absence of an advanced optimization, as $k_1 \in [1.2,2.0]$ and b = 0.75.
- $IDF(q_i) = log\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5}$
  - $N$ = total number of documents in the collection
  - $N(q_i)$ = documents containing $q_i$

# Summary – Vector Space Ranking

- Represent the query as a weighted tf-idf vector
- Represent each document as a weighted tf-idf vector
- Compute the cosine similarity score for the query vector and each document vector
- Rank documents with respect to the query by score
- Return the top *K* (e.g., *K* = 10) to the user

# Today's Agenda

- Need for Relevance Ranking
- TF and IDF
- **Vector Space Model**

# Efficient Cosine Ranking

- Find the *K* docs in the collection "nearest" to the query $\Rightarrow$ *K* largest query-doc cosines.
- Efficient ranking
  - Computing a single cosine efficiently.
  - Choosing the *K* largest cosine values efficiently.
    - Can we do this without computing all *N* cosines?
    - We don't need to totally order all docs in the collection
    - Let *J* = number of docs with nonzero cosines
      - We seek the *K* best of these *J*
    - Use heap for selecting top $K$
    - Exact topK is difficult, but approx-topK is feasible and acceptable

# Generic Approach

- Find a set *A* of *contenders*, with $K < |A| << N$
  - *A* does not necessarily contain the top *K*, but has many docs from among the top *K*
  - Return the top *K* docs in *A*
- Think of *A* as pruning non-contenders

# Index Elimination

- Basic algorithm only considers docs containing at least one query term
- Take this further
  - Only consider high-idf query terms
  - Only consider docs containing many query terms

# Champion Lists

- Precompute for each dictionary term *t,* the *r* docs of highest weight in *t*'s postings
  - Call this the champion list for *t*
  - (aka fancy list or top docs for *t*)
- Note that *r* has to be chosen at index time
- At query time, only compute scores for docs in the champion list of some query term
  - Pick the *K* top-scoring docs from amongst these

# Static Quality Scores

- We want top-ranking documents to be both *relevant* and *authoritative*
- *Relevance* is being modeled by cosine scores
- *Authority* is typically a query-independent property of a document
- Examples of authority signals
  - Wikipedia among websites
  - Articles in certain newspapers
  - A paper with many citations
  - Many diggs, Y!buzzes or del.icio.us marks
  - (Pagerank)
- Assign to each document a *query-independent* quality score in [0,1] to each document *d*
  - Denote this by *g(d)*

# Net Score

- Consider a simple total score combining cosine relevance and authority
- net-score($q,d$) = $g(d)$ + cosine($q,d$)
  - Can use some other linear combination than an equal weighting
  - Indeed, any function of the two “signals” of user happiness

# PageRank

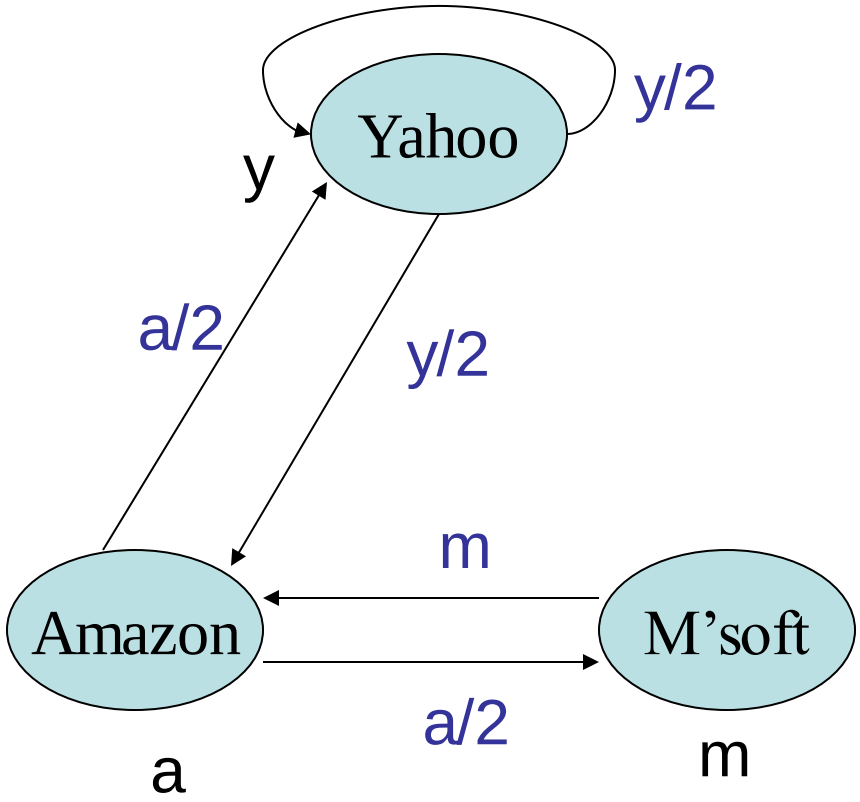
Manish Gupta

Slides borrowed (and modified) from
http://infolab.stanford.edu/~ullman/mining/2009/index.html
http://www.mpi-inf.mpg.de/departments/d5/teaching/ws11_12/irdm/slides/irdm-4-2-4.pptx

# Ranking Web Pages

- Web pages are not equally "important"
  - www.joe-schmoe.com vs www.stanford.edu
- Inlinks as votes
  - www.stanford.edu has 23,400 inlinks
  - www.joe-schmoe.com has 1 inlink
- Are all inlinks equal?
  - Recursive question
  - Each link's vote is proportional to the importance of its source page
  - If page P with importance x has n outlinks, each link gets x/n votes
  - Page P's own importance is the sum of the votes on its inlinks

# Simple “Flow” Model




$$y = y/2 + a/2$$
$$a = y/2 + m$$
$$m = a/2$$

# Solving the Flow Equations

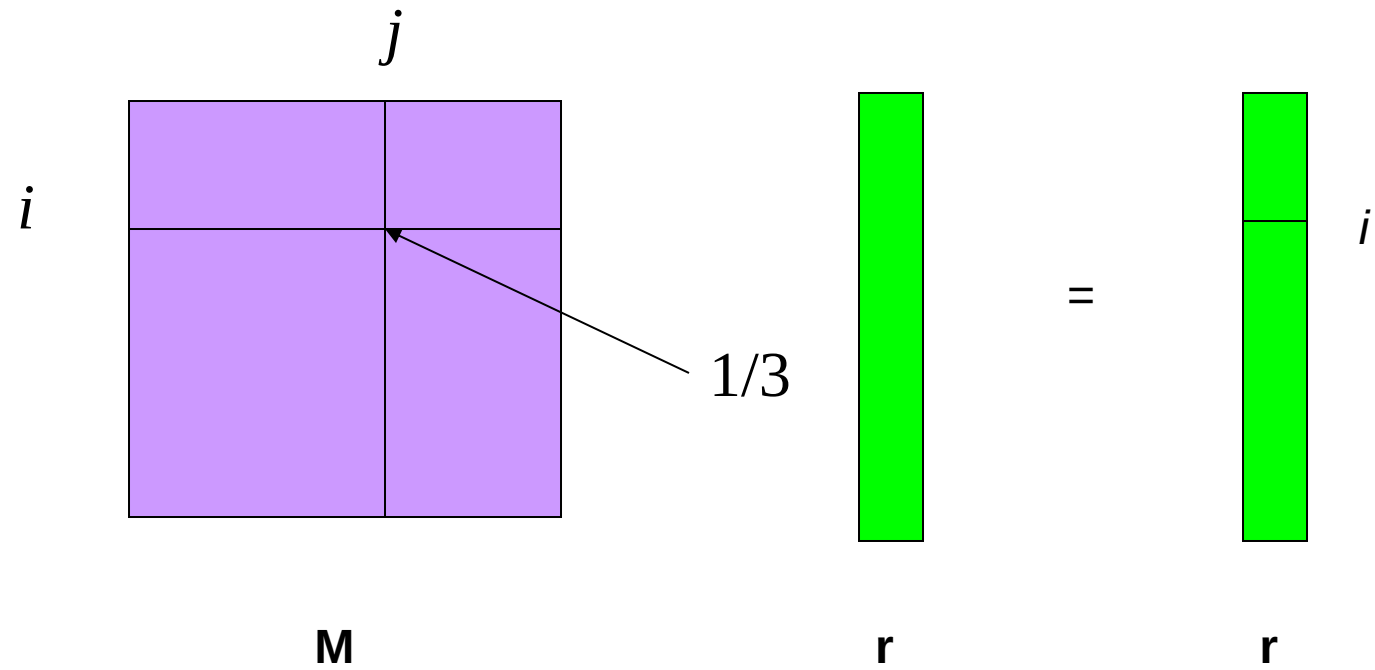
- 3 equations, 3 unknowns, no constants
  - No unique solution
  - All solutions equivalent modulo scale factor
- Additional constraint forces uniqueness
  - y+a+m = 1
  - y = 2/5, a = 2/5, m = 1/5
- Gaussian elimination method works for small examples, but we need a better method for large graphs

# Matrix Formulation

- Matrix **M** has one row and one column for each web page
- Suppose page j has n outlinks
  - If j != i, then $M_{ij}=1/n$
  - Else $M_{ij}=0$
- **M** is a column stochastic matrix
  - Columns sum to 1
- Suppose **r** is a vector with one entry per web page
  - $r_i$ is the importance score of page i
  - Call it the rank vector
  - $|\mathbf{r}| = 1$

# Example

Suppose page $j$ links to 3 pages, including $i$




$$y = y/2 + a/2$$

$$a = y/2 + m$$

$$m = a/2$$

# Eigenvector Formulation

- The flow equations can be written as

$$\mathbf{r} = \mathbf{M}\mathbf{r}$$

- So the rank vector is an eigenvector of the stochastic web matrix
  - In fact, its first or principal eigenvector, with corresponding eigenvalue 1

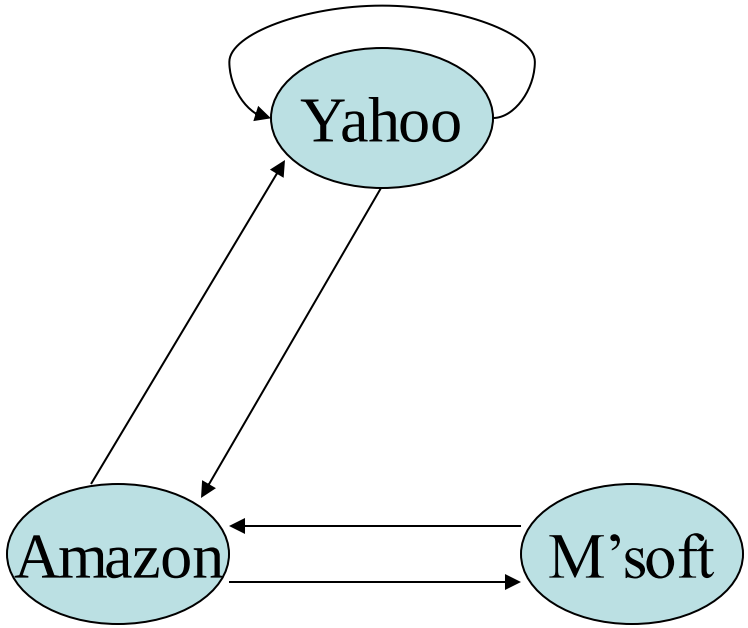
$$y = y/2 + a/2$$
$$a = y/2 + m$$
$$m = a/2$$

$$\begin{bmatrix} y \\ a \\ m \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix} \begin{bmatrix} y \\ a \\ m \end{bmatrix}$$

# Power Iteration Method

- Simple iterative scheme (aka relaxation)
- Suppose there are N web pages
- Initialize: $\mathbf{r}^0 = [1/N,\ldots.,1/N]^T$
- Iterate: $\mathbf{r}^{k+1} = \mathbf{M}\mathbf{r}^k$
- Stop when $|\mathbf{r}^{k+1} - \mathbf{r}^k|_1 < \varepsilon$
  - $|\mathbf{x}|_1 = \sum_{1 \le i \le N} |x_i|$ is the $L_1$ norm
  - Can use any other vector norm e.g., Euclidean

# Power Iteration Example




| | y | a | m |
|---|---|---|---|
| y | 1/2 | 1/2 | 0 |
| a | 1/2 | 0 | 1 |
| m | 0 | 1/2 | 0 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| y | | 1/3 | 1/3 | 5/12 | 3/8 | | 2/5 |
| a | = | 1/3 | 1/2 | 1/3 | 11/24 | . . . | 2/5 |
| m | | 1/3 | 1/6 | 1/4 | 1/6 | | 1/5 |

# Further Reading

- Chapters 6,7,8 of Manning-Raghavan-Schuetze book
  - http://nlp.stanford.edu/IR-book/
- S. E. Robertson and K. Spärck Jones. 1976. Relevance Weighting of Search Terms. *Journal of the American Society for Information Sciences* 27(3): 129–146.
- C. J. van Rijsbergen. 1979. *Information Retrieval.* 2nd ed. London: Butterworths, chapter 6. [Most details of math] http://www.dcs.gla.ac.uk/Keith/Preface.html
- N. Fuhr. 1992. Probabilistic Models in Information Retrieval. *The Computer Journal*, 35(3),243–255. [Easiest read, with BNs]
- F. Crestani, M. Lalmas, C. J. van Rijsbergen, and I. Campbell. 1998. Is This Document Relevant? ... Probably: A Survey of Probabilistic Models in Information Retrieval. *ACM Computing Surveys* 30(4): 528–552.
  - http://www.acm.org/pubs/citations/journals/surveys/1998-30-4/p528-crestani/

# Further Reading

- A nice summary of link analysis algorithms from John Kleinberg
  - http://dl.acm.org/citation.cfm?id=345982
- Chapter 5: "Link Analysis" from Mining of Massive Datasets
  - http://infolab.stanford.edu/~ullman/mmds.html
- Chapter 7 (Social Network Analysis) from Mining the Web
  - http://www.cse.iitb.ac.in/soumen/mining-the-web/
- J.M. Kleinberg: Authoritative Sources in a Hyperlinked Environment, JACM 46(5), 1999
- S Brin, L. Page: The Anatomy of a Large-Scale Hypertextual Web Search Engine, WWW 1998
- M. Najork, H. Zaragoza, M. Taylor: HITS on the Web: How does it Compare?, SIGIR 2007
- R. Lempel, S. Moran: SALSA: The Stochastic Approach for Link-Structure Analysis, ACM TOIS 19(2), 2001.
- G. Jeh, J. Widom: SimRank: a Measure of Structural-Context Similarity, KDD 2002
- Taher Haveliwala: Topic-Sensitive PageRank: A Context-Sensitive Ranking Algorithm for Web Search, IEEE Trans. on Knowledge and Data Engineering, 2003.
- G. Jeh, J. Widom: Scaling personalized web search, WWW 2003.
- D. Fogaras, B. Racz, K. Csalogany, A. Benczur: Towards Scaling Fully Personalized PageRank: Algorithms, Lower Bounds, and Experiments, Internet Mathematics 2(3): 333-358, 2006.