Web Search and Information Retrieval

Based on: C. D. Manning, P. Raghavan and H. Schütze. Introduction to Information Retrieval

Crawling

- Start from some root site e.g., Yahoo directories.
- Traverse the HREF links.

```
Search(initialLink)
    fringe.Insert( initialLink );
    loop do
        link ← fringe.Remove();
        page = RetrievePage(link);
        Index(page)
        fringe.InsertAll ( ExtractLinks(page) );
```

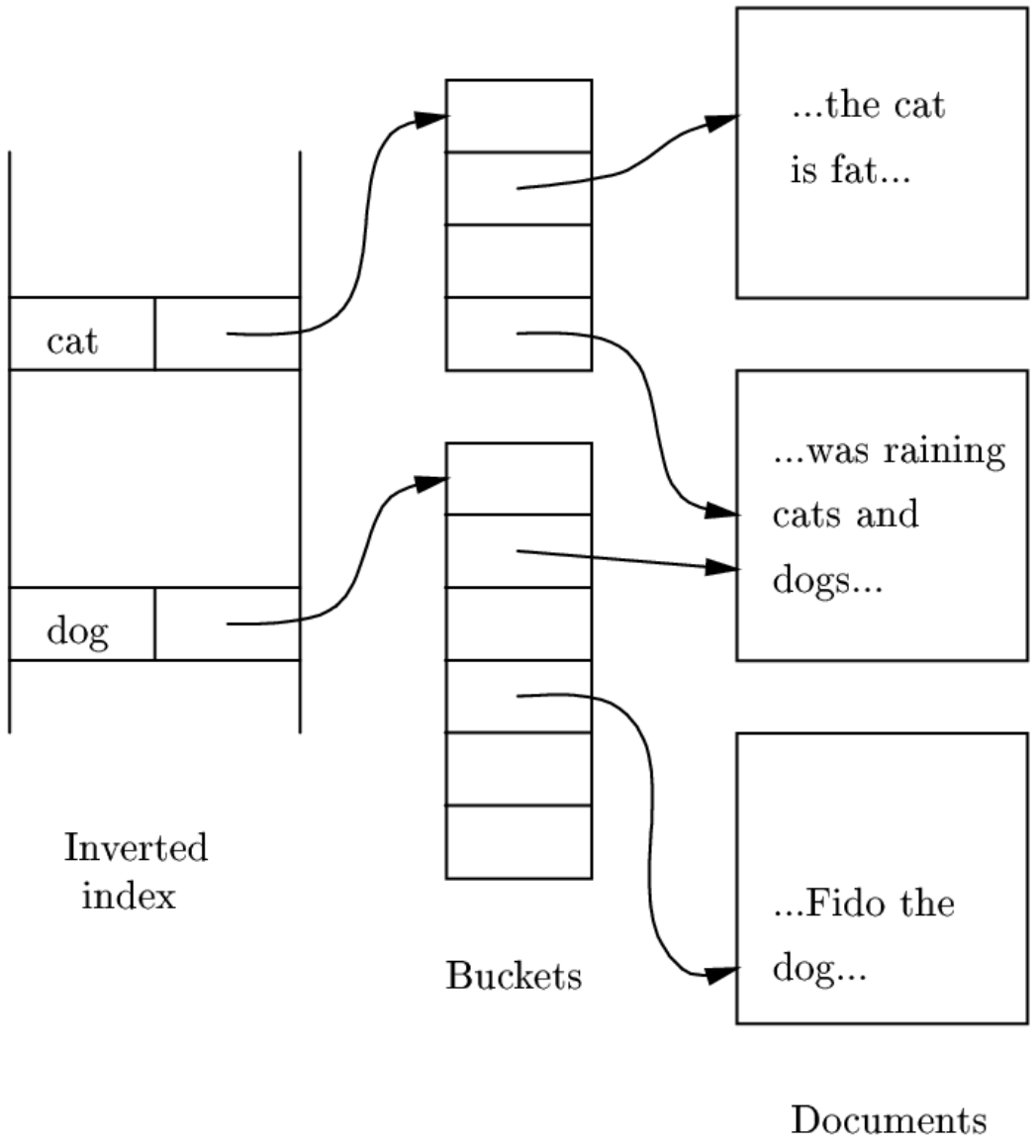
- Consequence:

  If there isn't an HREF path from some Yahoo like directory, then your page **probably** isn't indexed by any search engine.

Information Retrieval (IR)

- IR core problem: Find documents relevant to user queries
- Web search has its root in IR.

Inverted Indexes



Text pre-processing

1. Word (term) extraction: easy
2. Stopwords removal
3. Stemming
4. Frequency counts and computing TF-IDF term weights.

Stopwords removal

- Some of the most frequently used words aren't useful in IR and text mining – these words are called *stop words*.
  - the, of, and, to, ….
  - Typically about 400 to 500 such words
  - For an application, an additional domain specific stopwords list may be constructed

- Why do we need to remove stopwords?
  - Reduce indexing (or data) file size
    - stopwords account for 20-30% of total word counts.
  - Improve efficiency and effectiveness
    - stopwords are not useful for searching or text mining
    - they may also confuse the retrieval system.

Stemming

E.g.,

	user	engineering
	users	engineered
	used	engineer
	using	
stem:	use	engineer

Usefulness:

- improves the effectiveness of IR and text mining
- reduces index size
  - combing words with same roots may reduce indexing size as much as 40-50%.

Vector space model

- Documents are treated as a "bag" of words or terms.
  - Each document is represented as a vector.

- TF: (normalized) **term frequency**

$$tf_{ij} = \frac{f_{ij}}{\max\{f_{1j}, f_{2j}, \ldots, f_{|V|j}\}}$$

word *i*

document *j*

size of vocabulary

Retrieval in the vector space model

- Query **q** is represented in the same way as a document.
  - Weight of terms in **q** computed in the same way as for regular document.

- Relevance of **d** to **q**: Compare the similarity of query **q** and document **d**.
  - The bigger the cosine the smaller the angle and the higher the similarity

The sum has as many non-zero terms as there are words in the intersection of q and d_i.

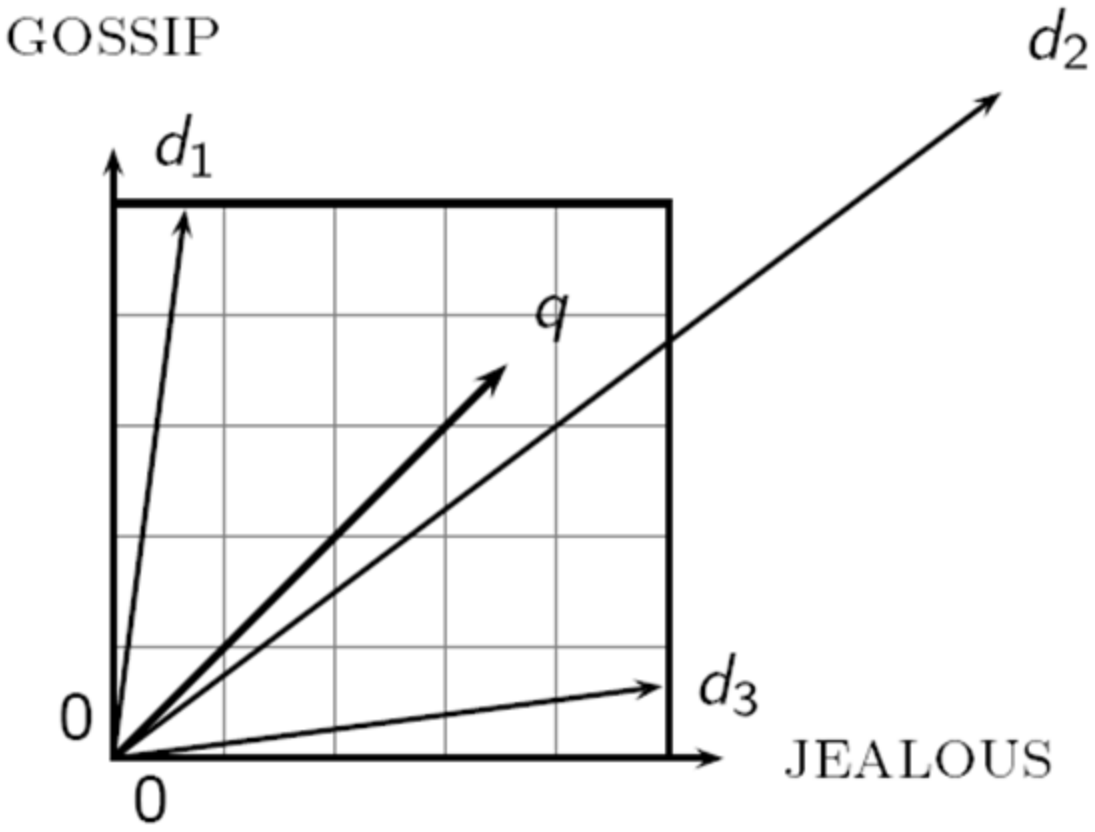
Dot product

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

We only need to retrieve from the inverted index the documents that contain at least one word of the query.

Documents that don't have some word in common with the query have a similarity of 0 with the query, so they don't need to be considered.

Cosine similarity



Example

d1: Most runners can recount some dog chasing them down on the street. The desire for interaction is a reason for dog chasing.

d2: Dogs and cats have antagonistic interactions.

d3: Interacting with others is vital.

d4: Human-animal interactions have positive effects on humans.

q: dog interaction

Example – non-stop words in blue

d1: Most runners can recount some dog chasing them down on the street. The desire for interaction is a reason for dog chasing.

d2: Dogs and cats have antagonistic interactions.

d3: Interacting with others is vital.

d4: Human-animal interactions have positive effects on humans.

q: dog interaction

See Excel file for calculations.

Document Frequency

term *i*	df_i
calpurnia	1
animal	100
sunday	1,000
fly	10,000
under	100,000
the	1,000,000

- Suppose query is: calpurnia animal

- An appearance of “calpurnia” should be much more important than an appearance of “animal” because “calpurnia” is a rare term.

TF-IDF term weighting scheme

- The most well known weighting scheme
  - TF: (normalized) **term frequency**
  - IDF: **inverse document frequency**.

$$tf_{ij} = \frac{f_{ij}}{\max\{ f_{1j}, f_{2j}, \ldots, f_{|V|j} \}}$$

N: total number of docs

df_i: the number of docs that t_i appears.

$$idf_i = \log \frac{N}{df_i}$$

- The final TF-IDF term weight is:

$$w_{ij} = tf_{ij} \times idf_j$$

Each document will be a vector of such numbers.

IDF

term *i*	**df_i**	**idf_i**
calpurnia	1	6
animal	100	4
sunday	1,000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

Precision and Recall

In information retrieval (search engines) community, system evaluation revolves around the notion of *relevant* and *not relevant* documents.

Precision is the fraction of retrieved documents that are relevant

$$\text{Precision} = \frac{\#(\text{relevant items retrieved})}{\#(\text{retrieved items})} = P(\text{relevant}|\text{retrieved})$$

Recall is the fraction of relevant documents that are retrieved

$$\text{Recall} = \frac{\#(\text{relevant items retrieved})}{\#(\text{relevant items})} = P(\text{retrieved}|\text{relevant})$$

How do we compute the precision and recall?

Precision at *k*

- The above measures precision at all recall levels.
- What matters is rather how many good results there are on the first page or the first three pages.
- This leads to measures of precision at fixed low levels of retrieved results, such as 10 or 30 documents.
- This is referred to as "Precision at *k*", for example "Precision at 10."

Web Search as a huge IR system

1. A Web crawler (robot) crawls the Web to collect all the pages.
2. Servers establish a huge inverted indexing database and other indexing databases
3. At query (search) time, search engines conduct different types of vector query matching.
    1. There is an *Information Retrieval score* coming out of this.
4. The documents have HREF links as well. They are used to compute a *reputation score*.
5. The two scores are combined together in order to produce a ranking of the returned documents.

Reputation Score: Google Page Ranking

“The Anatomy of a Large-Scale Hypertextual Web Search Engine”

by

Sergey Brin and Lawrence Page

http://www-db.stanford.edu/~backrub/google.html

Page Rank (PR)

Intuitively, we solve the recursive definition of "importance":

A page is important if important pages link to it.

- Page rank is the estimated page importance.

- In short PageRank is a "vote", by all the other pages on the Web, about how important a page is.

Page Rank Formula

$PR(A) = PR(T_1)/C(T_1) + \ldots + PR(T_n)/C(T_n)$

1. **$PR(T_n)$** - Each page has a notion of its own self-importance, which is **1** initially.
2. **$C(T_n)$** – Count of outgoing links from page T_n.

3. **$PR(T_n)/C(T_n)$** –
   a) Each page spreads its vote out evenly amongst all of it's outgoing links.
   b) So if our page (say page A) has a back link from page "*n*" the share of the vote page A will get from page "*n*" is "**$PR(T_n)/C(T_n)$.**"

How is Page Rank Calculated?

- The page rank (PR) of each page depends on the PR of the pages pointing to it.
  - But we won't know what PR those pages have until the pages pointing to them have their PR calculated and so on…

- Well, just go ahead and calculate a page's PR without knowing the final value of the PR of the other pages.
  - Each time we run the calculation we're getting a closer estimate of the final value.
  - Repeat the calculations lots of times until the numbers stop changing much.

Web Matrix

Capture the formula by the web matrix (M) that is:

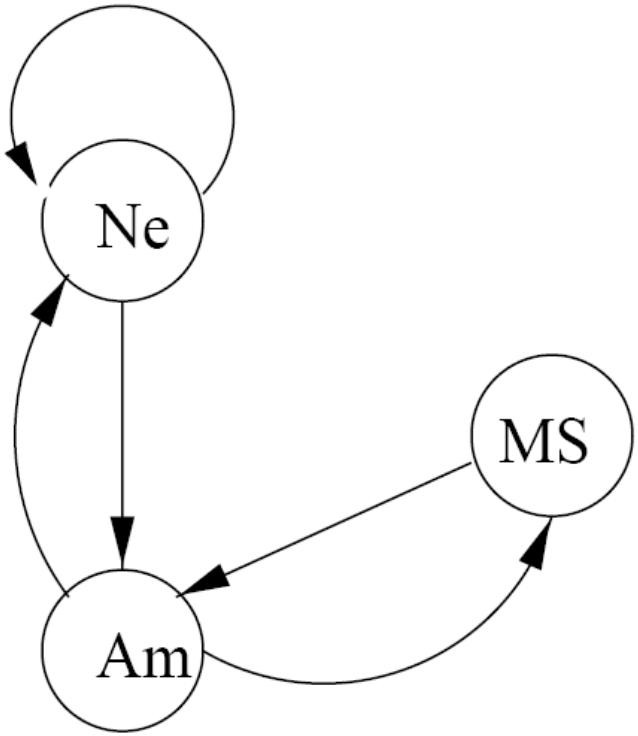
- If page *j* has *n* successors (links), then:
  - M[*i*, *j*] =1/*n* if page *i* is one of these *n* successors of page *j*, and
  - 0 otherwise.

Then, the importance vector containing the rank of each page is calculated by:

$$Rank_{new} = M \bullet Rank_{old}$$

Example

- In 1839, the Web consisted of only three pages: Netscape, Microsoft, and Amazon.



$$\begin{bmatrix} n_{new} \\ m_{new} \\ a_{new} \end{bmatrix} = \begin{bmatrix} 1/2 & 0 & 1/2 \\ 0 & 0 & 1/2 \\ 1/2 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} n_{old} \\ m_{old} \\ a_{old} \end{bmatrix}$$

For example, the first column of the Web matrix reflects the fact that Netscape divides its importance between itself and Amazon.

The second column indicates that Microsoft gives all its importance to Amazon.

Start with n = m = a = 1, then do rounds of improvements. We can also start with initial weights of 1/N (1/3 in this example).

Example

- The first four iterations give the following estimates:

n = 1	1	5/4	9/8	5/4
m = 1	1/2	3/4	1/2	11/16
a = 1	3/2	1	11/8	17/16

- In the limit, the solution is n = a = 6/5; m = 3/5.

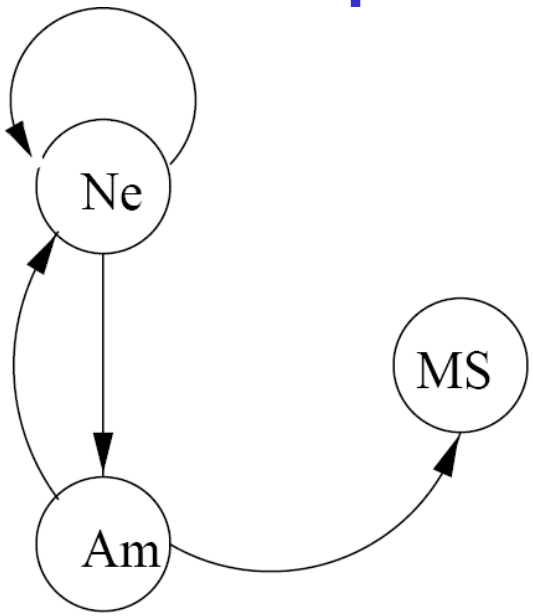
- That is, Netscape and Amazon each have the same importance, and twice the importance of Microsoft (well this was 1839).

Problems With Real Web Graphs

Dead ends: a page that has no successors has nowhere to send its importance.

Eventually, all importance will "leak out of" the Web.

Example: Suppose Microsoft tries to claim that it is a monopoly by removing all links from its site.



The new Web, and the rank vectors for the first 4 iterations are shown.

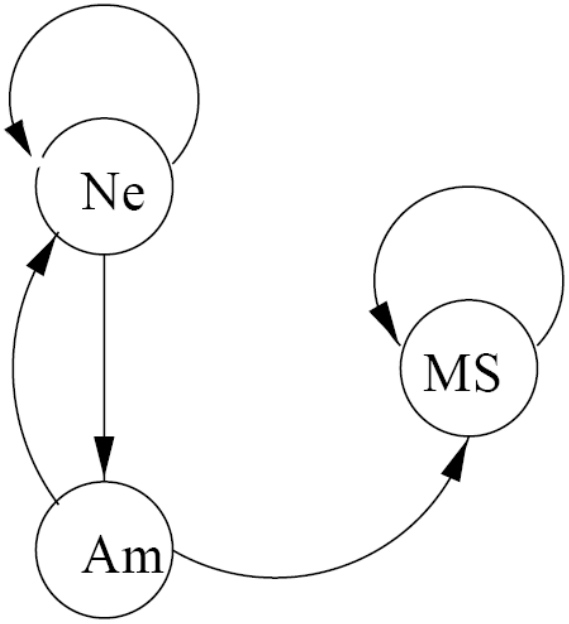
$$\begin{bmatrix} n_{new} \\ m_{new} \\ a_{new} \end{bmatrix} = \begin{bmatrix} 1/2 & 0 & 1/2 \\ 0 & 0 & 1/2 \\ 1/2 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} n_{old} \\ m_{old} \\ a_{old} \end{bmatrix}$$

Eventually, each of *n*, *m*, and *a* become 0; i.e., all the importance leaked out.

n =	1	1	3/4	5/8	1/2
m =	1	1/2	1/4	1/4	3/16
a =	1	1/2	1/2	3/8	5/16

Problems With Real Web Graphs

Spider traps: a group of one or more pages that have no links out of the group will eventually accumulate all the importance of the Web.



Example: Angered by the decision, Microsoft decides it will link only to itself from now on. Now, Microsoft has become a spider trap.

The new Web, and the rank vectors for the first 4 iterations are shown.

$$\begin{bmatrix} n_{new} \\ m_{new} \\ a_{new} \end{bmatrix} = \begin{bmatrix} 1/2 & 0 & 1/2 \\ 0 & 1 & 1/2 \\ 1/2 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} n_{old} \\ m_{old} \\ a_{old} \end{bmatrix}$$

n =	1	1	3/4	5/8	1/2
m =	1	3/2	7/4	2	35/16
a =	1	1/2	1/2	3/8	5/16

Now, m converges to 3, and $n = a = 0$.

Google Solution to Dead Ends and Spider Traps

Stop the other pages having too much influence.

This total vote is "damped down" by multiplying it by a factor.

Example: If we use a 20% damp-down, the equation of previous example becomes:

$$\begin{bmatrix} n_{new} \\ m_{new} \\ a_{new} \end{bmatrix} = 0.80 \cdot \begin{bmatrix} 1/2 & 0 & 1/2 \\ 0 & 1 & 1/2 \\ 1/2 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} n_{old} \\ m_{old} \\ a_{old} \end{bmatrix} + 0.20 \cdot \begin{bmatrix} n_{old} \\ m_{old} \\ a_{old} \end{bmatrix}$$

The solution to this equation is $n = 7/11$; $m = 21/11$; $a = 5/11$.

Appendix: NB Multinomial Model

$P(c)$: fraction of training documents that are of class c.

This slide is not related to Web Retrieval. It is here only to show another use of tfidf score

$$P(c) = \frac{N_c}{N}$$

$P(t|c)$: relative frequency of term t in documents of class c.

$$P(t|c) = \frac{T_{c,t}}{\sum_{t \in V} T_{c,t}}$$

V : vocabulary

Before:

$T_{c,t}$: number of occurrences of t in training documents of class c, including multiple occurrences of a term in a document.

Now:

$T_{c,t}$: sum of **tfidf** scores of t in training documents of class c.

Avoid zero frequency by: $P(t|c) = \frac{T_{c,t}+1}{\sum_{t \in V}(T_{c,t}+1)} = \frac{T_{c,t}+1}{(\sum_{t \in V} T_{c,t})+|V|}$

Classify a new document d by computing for each c

$$P(c \mid d) = \alpha * P(c) * \Pi_{t \in d} P(t \mid c)$$

Produce as classification result: $c_{map} = \text{argmax}_c \; P(c|d)$