# Web Information Retrieval

## MPRI 2.26.2: Web Data Managament

Bogdan Cautis
bogdan.cautis@u-psud.fr
(many thanks to Pierre Senellart)

February 1st, 2021

# Outline

# Brief history of Web search

Web search is based mainly on Information Retrieval (IR), a discipline has been around for 50-60 years

- June 11, 1994 – Brian Pinkerton: WebCrawler
- Dec 15, 1995 – AltaVista (crawled at 2.5 million pages per day, had 30 million pages)
- 1995, Yahoo! (Yet Another Hierarchical Officious Oracle)
- 1998 – Google (googol 10100), 2004 – Google IPO

Today: without search engines the Web would not exist.
Example: it makes any-level aggregation possible, unlimited "selection stores" possible (e.g., Amazon)

# IR basics

Discipline/principles for efficient large scale document search.

- Corpus: fixed document (textual) collection
- Goal: find documents whose content is relevant for a user's information need (query = sequence of words; short: avg. 2.7)
- Result: set of (few/most/top-k) relevant documents
- Relevance: for each query Q and stored document D in the corpus assume there exists relevance Score(Q, D)
- Usually the context / users are ignored

# Outline

# Information Retrieval, Search

## Problem

*How to index Web content so as to answer (keyword-based) queries efficiently?*

Context: set of text documents

| | |
|---|---|
| $d_1$ | The jaguar is a New World mammal of the Felidae family. |
| $d_2$ | Jaguar has designed four new engines. |
| $d_3$ | For Jaguar, Atari was keen to use a 68K family device. |
| $d_4$ | The Jacksonville Jaguars are a professional US football team. |
| $d_5$ | Mac OS X Jaguar is available at a price of US \$199 for Apple's new "family pack". |
| $d_6$ | One such ruling family to incorporate the jaguar into their name is Jaguar Paw. |
| $d_7$ | It is a big cat. |

# Text Preprocessing

Initial text preprocessing steps

- Number of optional steps
- Highly depends on the application
- Highly depends on the document language (illustrated with English)

# Language Identification

How to find the language used in a document?

- Meta-information about the document: often not reliable!
- Unambiguous scripts or letters: not very common!

한글
カタカナ
ދިވެހި
Għarbi
þorn

# Language Identification

How to find the language used in a document?

- Meta-information about the document: often not reliable!
- Unambiguous scripts or letters: not very common!

  한글
  カタカナ
  ދިވެހި
  Għarbi
  þorn

  Respectively: Korean Hangul, Japanese Katakana, Maldivian Dhivehi, Maltese, Icelandic
- Extension of this: frequent characters, or, better, frequent *k*-grams
- Use standard machine learning techniques (classifiers)

# Tokenization

## Principle

Separate text into tokens (words)

Not so easy!

- In some languages (Chinese, Japanese), words not separated by whitespace
- Deal consistently with acronyms, elisions, numbers, units, URLs, emails, etc.
- Compound words: *hostname*, *host-name* and *host name*. Break into two tokens or regroup them as one token? In any case, lexicon and linguistic analysis needed! Even more so in other languages as German.

Punctuation may be removed and case normalized at this point

# Tokenization: Example

$d_1$ the$_1$ jaguar$_2$ is$_3$ a$_4$ new$_5$ world$_6$ mammal$_7$ of$_8$ the$_9$ felidae$_{10}$ family$_{11}$

$d_2$ jaguar$_1$ has$_2$ designed$_3$ four$_4$ new$_5$ engines$_6$

$d_3$ for$_1$ jaguar$_2$ atari$_3$ was$_4$ keen$_5$ to$_6$ use$_7$ a$_8$ 68k$_9$ family$_{10}$ device$_{11}$

$d_4$ the$_1$ jacksonville$_2$ jaguars$_3$ are$_4$ a$_5$ professional$_6$ us$_7$ football$_8$ team$_9$

$d_5$ mac$_1$ os$_2$ x$_3$ jaguar$_4$ is$_5$ available$_6$ at$_7$ a$_8$ price$_9$ of$_{10}$ us$_{11}$ \$199$_{12}$ for$_{13}$ apple's$_{14}$ new$_{15}$ family$_{16}$ pack$_{17}$

$d_6$ one$_1$ such$_2$ ruling$_3$ family$_4$ to$_5$ incorporate$_6$ the$_7$ jaguar$_8$ into$_9$ their$_{10}$ name$_{11}$ is$_{12}$ jaguar$_{13}$ paw$_{14}$

$d_7$ it$_1$ is$_2$ a$_3$ big$_4$ cat$_5$

# Normalization (slide from [Manning et al., 2008])

- Need to "normalize" terms in indexed text as well as query terms into the same form.
- Example: We want to match *U.S.A.* and *USA*
- We most commonly implicitly define equivalence classes of terms.
- Alternatively: do asymmetric expansion
  - window → window, windows
  - windows → Windows, windows
  - Windows (no expansion)
- More powerful, but less efficient

## Exercise

*Why don't you want to put window, Window, windows, and Windows in the same equivalence class?*

# Normalization: Other Languages

(slide from [Manning et al., 2008])

- Accents: résumé vs. resume (simple omission of accent)
- Umlauts: Universität vs. Universitaet (substitution with special letter sequence "ae")
- Most important criterion: How are users likely to write their queries for these words?
- Even in languages that standardly have accents, users often do not type them. (Polish?)
- Normalization and language detection interact.
- *PETER WILL NICHT MIT.* $\rightarrow$ MIT $=$ mit
- *He got his PhD from MIT.* $\rightarrow$ MIT $\neq$ mit

# Stemming

### Principle

Merge different forms of the same word, or of closely related words, into a single stem

- Not in all applications!
- Useful for retrieving documents containing *geese* when searching for *goose*
- Various degrees of stemming
- Possibility of building different indexes, with different stemming

# Stemming schemes (1/2)

## Morphological stemming (lemmatization).

- Remove bound morphemes from words:
  - plural markers
  - gender markers
  - tense or mood inflections
  - etc.
- Can be linguistically very complex, cf:
  *Les poules du couvent couvent.*
  [The hens of the monastery brood.]
- In English, somewhat easy:
  - Remove final -s, -'s, -ed, -ing, -er, -est
  - Take care of semiregular forms (e.g., -y/-ies)
  - Take care of irregular forms (mouse/mice)
- But still some ambiguities: cf rose

# Stemming schemes (2/2)

Lexical stemming.

- Merge lexically related terms of various parts of speech, such as *policy*, *politics*, *political* or *politician*
- For English, Porter's stemming [Porter, 1980]; stems *university* and *universal* to *univers*: not perfect!
- Possibility of coupling this with lexicons to merge (near-)synonyms

Phonetic stemming.

- Merge phonetically related words: search proper names with different spellings!
- For English, Soundex [US National Archives and Records Administration, 2007] stems *Robert* and *Rupert* to *R163*. Very coarse!

# Stemming Example

- $d_1$ the$_1$ jaguar$_2$ be$_3$ a$_4$ new$_5$ world$_6$ mammal$_7$ of$_8$ the$_9$ felidae$_{10}$ family$_{11}$
- $d_2$ jaguar$_1$ have$_2$ design$_3$ four$_4$ new$_5$ engine$_6$
- $d_3$ for$_1$ jaguar$_2$ atari$_3$ be$_4$ keen$_5$ to$_6$ use$_7$ a$_8$ 68k$_9$ family$_{10}$ device$_{11}$
- $d_4$ the$_1$ jacksonville$_2$ jaguar$_3$ be$_4$ a$_5$ professional$_6$ us$_7$ football$_8$ team$_9$
- $d_5$ mac$_1$ os$_2$ x$_3$ jaguar$_4$ be$_5$ available$_6$ at$_7$ a$_8$ price$_9$ of$_{10}$ us$_{11}$ \$199$_{12}$ for$_{13}$ apple$_{14}$ new$_{15}$ family$_{16}$ pack$_{17}$
- $d_6$ one$_1$ such$_2$ rule$_3$ family$_4$ to$_5$ incorporate$_6$ the$_7$ jaguar$_8$ into$_9$ their$_{10}$ name$_{11}$ be$_{12}$ jaguar$_{13}$ paw$_{14}$
- $d_7$ it$_1$ be$_2$ a$_3$ big$_4$ cat$_5$

# Stop Word Removal

## Principle

Remove uninformative words from documents, in particular to lower the cost of storing the index; note: in English most common 30 words are 30% of the corpus tokens.

determiners: *a*, *the*, *this*, etc.

function verbs: *be*, *have*, *make*, etc.

conjunctions: *that*, *and*, etc.

etc.

# Stop Word Removal Example

$d_1$ jaguar$_2$ new$_5$ world$_6$ mammal$_7$ felidae$_{10}$ family$_{11}$

$d_2$ jaguar$_1$ design$_3$ four$_4$ new$_5$ engine$_6$

$d_3$ jaguar$_2$ atari$_3$ keen$_5$ 68k$_9$ family$_{10}$ device$_{11}$

$d_4$ jacksonville$_2$ jaguar$_3$ professional$_6$ us$_7$ football$_8$ team$_9$

$d_5$ mac$_1$ os$_2$ x$_3$ jaguar$_4$ available$_6$ price$_9$ us$_{11}$ \$199$_{12}$ apple$_{14}$ new$_{15}$ family$_{16}$ pack$_{17}$

$d_6$ one$_1$ such$_2$ rule$_3$ family$_4$ incorporate$_6$ jaguar$_8$ their$_{10}$ name$_{11}$ jaguar$_{13}$ paw$_{14}$

$d_7$ big$_4$ cat$_5$

# Outline

## Structure of an inverted index

Assume $D$ a collection of (text) documents. Create a matrix $M$ with one row for each document, one column for each token. Initialize the cells at 0.

Create the content of $M$: scan $D$, and extract for each document $d$ the tokens $t$ that can be found in $d$ (preprocessing); put 1 in $M[d][t]$

Invert $M$: one obtains the inverted index. Term appear as rows, with the list of document ids or *posting list*.

Problem: storage of the whole matrix is not feasible.

# Inverted Index

After all preprocessing, construction of an inverted index:

- Index of all terms, with the list of documents where this term occurs
- Small scale: disk storage, with memory mapping (cf. `mmap`) techniques; secondary index for offset of each term in main index
- Large scale: distributed on a cluster of machines; hashing gives the machine responsible for a given term
- Updating the index costly, so only batch operations (not one-by-one addition of term occurrences)

# Inverted Index Example

| | |
|---|---|
| family | $d_1$, $d_3$, $d_5$, $d_6$ |
| football | $d_4$ |
| jaguar | $d_1$, $d_2$, $d_3$, $d_4$, $d_5$, $d_6$ |
| new | $d_1$, $d_2$, $d_5$ |
| rule | $d_6$ |
| us | $d_4$, $d_5$ |
| world | $d_1$ |
| ... | |

Note:

- the length of an inverted (posting) list is highly variable – scanning short lists first is an important optimization.
- *entries* are homogeneous: this gives much room for compression.

# Index size matters

We want to index a collection of 1M emails. The average size of an email is 1,000 bytes and each email contains an average of 100 words. The number of distinct terms is 200,000.

1. size of the collection; number of words?
2. how many lists in the index?
3. we make the (rough) assumption that 20% of the terms in a document appear twice; a document appears in how many lists on average ?
4. how many entries in a list?
5. we represent document ids as 4-bytes unsigned integers, what is the index size ?

# Storing positions in the index

- phrase queries, NEAR operator: need to keep position information in the index
- just add it in the document list!

| | |
|---|---|
| family | $d_1/11$, $d_3/10$, $d_5/16$, $d_6/4$ |
| football | $d_4/8$ |
| jaguar | $d_1/2$, $d_2/1$, $d_3/2$, $d_4/3$, $d_5/4$, $d_6/8+13$ |
| new | $d_1/5$, $d_2/5$, $d_5/15$ |
| rule | $d_6/3$ |
| us | $d_4/7$, $d_5/11$ |
| world | $d_1/6$ |
| ... | |

$\Rightarrow$ so far, ok for Boolean queries: find the documents that contain a set of keywords; reject the other.

# Ranked search

Boolean search does not give an accurate result because it does not take account of the relevance of a document to a query.

If the search retrieves dozen or hundreds of documents, the most relevant must be shown in top position!

# Weighting terms occurrences

Relevance can be computed by giving a weight to term occurrences.

- Terms occurring frequently in a given document: more relevant. The *term frequency* is the number of occurrences of a term $t$ in a document $d$, divided by the total number of terms in $d$:

$$\mathrm{tf}(t,d) = \frac{n_{t,d}}{\sum_{t'} n_{t',d}}$$

where $n_{t',d}$ is the number of occurrences of $t'$ in $d$.

- Terms occurring rarely in the document collection as a whole: more informative. The *inverse document frequency* (idf) is obtained from the division of the total number of documents by the number of documents where $t$ occurs, as follows:

$$\mathrm{idf}(t) = \log \frac{|D|}{\left|\left\{d' \in D \mid n_{t,d'} > 0\right\}\right|}.$$

# TF-IDF Weighting

- Some term occurrences have more weight than others:
  - Terms occurring frequently in a given document: more relevant
  - Terms occurring rarely in the document collection as a whole: more informative
- Add Term Frequency—Inverse Document Frequency weighting to occurrences;

$$\mathrm{tfidf}(t,d) = \frac{n_{t,d}}{\sum_{t'} n_{t',d}} \cdot \log \frac{|D|}{\left|\left\{d' \in D \,\middle|\, n_{t,d'} > 0\right\}\right|}$$

| | |
|---|---|
| $n_{t,d}$ | number of occurrences of $t$ in $d$ |
| $D$ | set of all documents |

- Store documents (along with weight) in decreasing weight order in the index

# TF-IDF Weighting Example

| | |
|---|---|
| family | $d_1/11/.13$, $d_3/10/.13$, $d_6/4/.08$, $d_5/16/.07$ |
| football | $d_4/8/.47$ |
| jaguar | $d_1/2/.04$, $d_2/1/.04$, $d_3/2/.04$, $d_4/3/.04$, $d_6/8+13/.04$, $d_5/4/.02$ |
| new | $d_2/5/.24$, $d_1/5/.20$, $d_5/15/.10$ |
| rule | $d_6/3/.28$ |
| us | $d_4/7/.30$, $d_5/11/.15$ |
| world | $d_1/6/.47$ |
| ... | |

# Outline

# Answering Boolean Queries

- Single keyword query: just consult the index and return the documents in index order.
- Boolean multi-keyword query

  (*jaguar* AND *new* AND NOT *family*) OR *cat*

  Same way! Retrieve document lists from all keywords and apply adequate set operations:

  | | |
  |---|---|
  | AND | intersection |
  | OR | union |
  | AND NOT | difference |

- Global score: some function of the individual weight (e.g., addition for conjunctive queries)
- Position queries: consult the index, and filter by appropriate condition

## Exercise

Consider the following documents:

1. $d_1$ = I like to watch the sun set with my friend.
2. $d_2$ = The Best Places To Watch The Sunset.
3. $d_3$ = My friend watches the sun come up.

Construct an inverted index with tf/idf weights for terms 'best' and 'sun'. What would be the ranked result of the query 'best OR sun'?

# Answering Top-$k$ Queries

$$t_1 \text{ AND } \dots \text{ AND } t_n$$

$$t_1 \text{ OR } \dots \text{ OR } t_n$$

### Problem

Find the top-$k$ results (for some given $k$) to the query, without retrieving all documents matching it.

Notations:

$s(t,d)$ weight of $t$ in $d$ (e.g., tfidf)

$g(s_1,\dots,s_n)$ monotonous function that computes the global score (e.g., addition)

# Basic algorithm

First version of the top-$k$ algorithm: the inverted file contains entries sorted on the document id. The query is

$$t_1 \text{ AND } \ldots \text{ AND } t_n$$

1. Take the first entry of each list; one obtains a tuple $T = [e_1, \ldots e_n]$;
2. Let $d_1$ be the minimal doc id in the entries of $T$: compute the global score of $d_1$;
3. For each entry $e_i$ featuring $d_1$: advance on the inverted list $L_i$.

When *all* lists have been scanned: sort the documents on the global scores.

Not very efficient; cannot give the ranked result before a full scan on the lists.

# Fagin's Threshold Algorithm [Fagin et al., 2001]

(entries are sorted according to score, with an additional direct index giving $s(t,d)$)

1. Let $R$ be the empty list and $m = +\infty$.
2. For each $1 \leq i \leq n$:
   2.1 Retrieve the document $d^{(i)}$ containing term $t_i$ that has the next largest $s(t_i, d^{(i)})$.
   2.2 Compute its global score $g_{d^{(i)}} = g(s(t_1, d^{(i)}), \ldots, s(t_n, d^{(i)}))$ by retrieving all $s(t_j, d^{(i)})$ with $j \neq i$.
   2.3 If $R$ contains less than $k$ documents, or if $g_{d^{(i)}}$ is greater than the minimum of the score of documents in $R$, add $d^{(i)}$ to $R$ (and remove the worst element in $R$ if it is full).
3. Let $m = g(s(t_1, d^{(1)}), s(t_2, d^{(2)}), \ldots, s(t_n, d^{(n)}))$.
4. If $R$ contains $k$ documents, and the minimum of the score of the documents in $R$ is greater than or equal to $m$, return $R$.
5. Redo step 2.

# The TA, by example

$q =$ "*new* OR *family*", and $k = 3$.

| | |
|---|---|
| **family** | $d_1/11/.13$, $d_3/10/.13$, $d_6/4/.08$, $d_5/16/.07$ |
| **new** | $d_2/5/.24$, $d_1/5/.20$, $d_5/15/.10$ |
| **...** | |

Initially, $R = \varnothing$ and $\tau = +\infty$.

1. $d^{(1)}$ is the first entry in $L_{family}$, one finds $s(new, d_1) = .20$; the global score for $d_1$ is $.13 + .20 = .33$.
2. Next, $i = 2$, and one finds that the global score for $d_2$ is .24.
3. The algorithm quits the loop on $i$ with $R = \langle [d_1, .33], [d_2, .24] \rangle$ and $\tau = .13 + .24 = .37$.
4. We proceed with the loop again, taking $d_3$ with score .13 and $d_5$ with score .17. $[d_5, .17]$ is added to $R$ (at the end) and $\tau$ is now $.10 + .13 = .23$.

   A last loop concludes that the next candidate is $d_6$, with a global score of .08. Then we are done.

# Fagin's No Random Access Algorithm [Fagin et al., 2001]

(no additional direct index needed)

1. Let $R$ be the empty list and $m = +\infty$.
2. For each document $d$, maintain $W(d)$ as its worst possible score, and $B(d)$ as its best possible score.
3. At the beginning, $W(d) = 0$ and $B(d) = g(s(t_1, d^{(1)}) \dots s(t_n, d^{(n)})$.
4. Then, access the next best document for each token, in a round-robin way ($t_1$, $t_2 \dots t_n$, then $t_1$ again, etc.)
5. Update the $W(d)$ and $B(d)$ lists each time, and maintain $R$ as the list of $k$ documents with best $W(d)$ scores (solve ties with $B(d)$), and $m$ as the minimum value for $W(d)$ in $R$.
6. Stop when $R$ contains at least $k$ documents, and all documents outside of $R$ verify $B(d) \leq m$.

# Outline

# External Sort/Merge

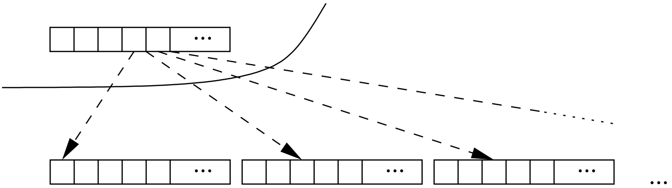
Building an inverted index from a document collection requires a sort/merge of the index entries.

- first extracts triplets $[d, t, f]$ from the collection;
- then sort the set of triplets on the term-docid pair $[t, d]$.
- contiguous inverted lists can be created from the sorted entries.

Note: ubiquitous operation; used in RDBMS for ORDER BY, GROUP BY, DISTINCT, and non-indexed joins.

# First phase: sort

Repeat: fill the memory with entries; sort in memory (with quicksort); flush the memory in a “run”.

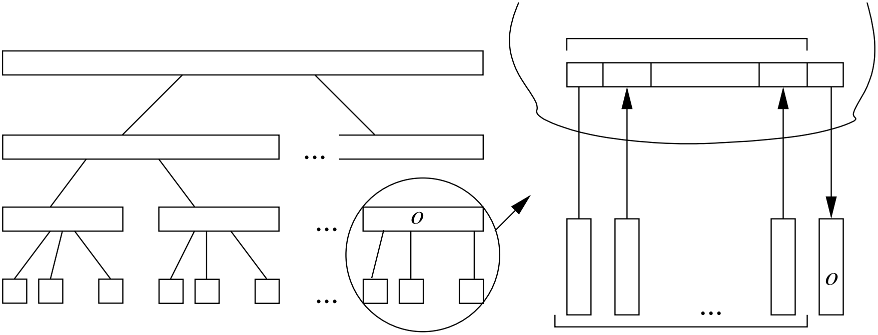


One obtains a list of sorted runs.

Cost: documents are read once; entries are written once.

# Second phase: merge

Group the runs and merge.




Cost: one read/write of all entries for each level of the merge tree.

# Compression of inverted lists

Without compression, an inverted index with positions and weights may be larger than the documents collection!

Compression is essential. The gain must be higher than the time spent to compress.

Key to compression in inverted lists: documents are ordered by id:

$$[87; 273; 365; 576; 810].$$

First step: use *delta-coding*:

$$[87; 186; 92; 211; 234].$$

Exercise: what is the minimal number of bytes for the first list? for the second?

# Variable bytes encoding

Idea: encode integers on 7 bits ($2^7 = 128$); use the leading bit for termination.

Let $v = 9$, encoded on one byte as 10001001 (note first bit set to 1).

Let $v = 137$.

1. the first byte encodes $v' = v \bmod 128 = 9$, thus $b = 10001001$ just as before;
2. next we encode $v/128 = 1$, in a byte $b' = 00000001$ (note first bit set to 0).

137 is therefore encoded on two bytes:

$$00000001\ 10001001.$$

Compression ratio: typically 1/4 to 1/2 of the fixed-length representation.

# Exercise

The inverted list of a term $t$ consists of the following document ids:

$$[345; 476; 698; 703].$$

Apply the VByte compression technique to this sequence. What is the amount of space gained by the method?

# Outline

# *k*-gram indexes for spelling correction

(slide from [Manning et al., 2008])

- Problem: able to deal with incorrectly spelled terms in documents, or variants in spelling
- Enumerate all *k*-grams in the query term
- Use the *k*-gram index to retrieve "correct" words that match query term *k*-grams
- Threshold by number of matching *k*-grams
- E.g., only vocabulary terms that differ by at most 3 *k*-grams
- Example: bigram index, misspelled word *bordroom*
- Bigrams: *bo, or, rd, dr, ro, oo, om*

# *k*-gram indexes for spelling correction: *bordroom*

BO → aboard → about → boardroom → border

OR → border → lord → morbid → sordid

RD → aboard → ardent → boardroom → border

# Example with trigrams

(slide from [Manning et al., 2008])

- Issue: Fixed number of *k*-grams that differ does not work for words of differing length.
- Suppose the correct word is NOVEMBER
- Trigrams: *nov*, *ove*, *vem*, *emb, mbe, ber*
- And the query term is DECEMBER
- Trigrams: *dec*, *ece*, *cem*, *emb, mbe, ber*
- So 3 trigrams overlap (out of 6 in each term)
- How can we turn this into a normalized measure of overlap?

# Example with trigrams

(slide from [Manning et al., 2008])

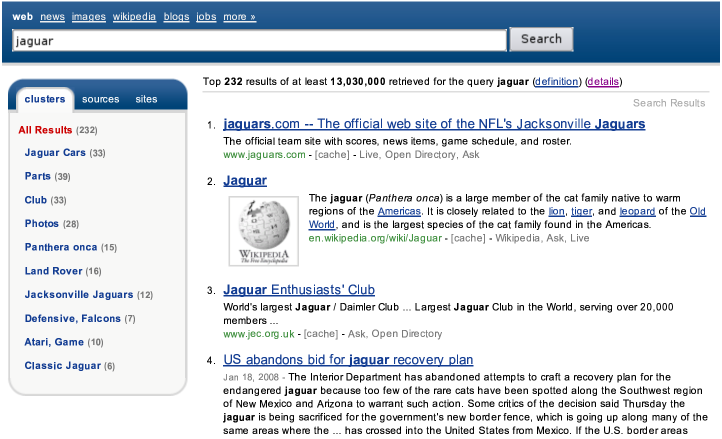
- Issue: Fixed number of *k*-grams that differ does not work for words of differing length.
- Suppose the correct word is NOVEMBER
- Trigrams: *nov*, *ove*, *vem*, *emb, mbe, ber*
- And the query term is DECEMBER
- Trigrams: *dec*, *ece*, *cem*, *emb, mbe, ber*
- So 3 trigrams overlap (out of 6 in each term)
- How can we turn this into a normalized measure of overlap? $\rightarrow$ Use Jaccard coefficent!

# Context-sensitive spelling correction (slide from [Manning et al., 2008])

- *an asteroid that fell form the sky*
- How can we correct *form* here?
- One idea: hit-based spelling correction
  - Retrieve "correct" terms close to each query term
  - for *flew form munich*: *flea* for *flew*, *from* for *form*, *munch* for *munich*
  - Now try all possible resulting phrases as queries with one word "fixed" at a time
  - Try query *"flea form munich"*
  - Try query *"flew from munich"*
  - Try query *"flew form munch"*
  - The correct query *"flew from munich"* has the most hits.
- The "hit-based" algorithm we just outlined is not very efficient.
- More efficient alternative: look at "collection" of queries, not documents

# Outline

# Clustering Example



## Cosine Similarity of Documents

- Document Vector Space model:

| | |
|---|---|
| terms | dimensions |
| documents | vectors |
| coordinates | weights |

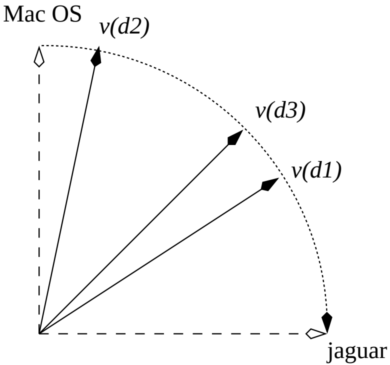
  (The projection of document $d$ along coordinate $t$ is the weight of $t$ in $d$, say $\mathrm{tf-idf(t,d)}$)
- Similarity between documents $d$ and $d'$: cosine of these two vectors

$$\cos(d,d') = \frac{d \cdot d'}{\|d\| \times \|d'\|}$$

| | |
|---|---|
| $d \cdot d'$ | scalar product of $d$ and $d'$ |
| $\|d\|$ | norm of vector $d$ |

- $\cos(d,d) = 1$
- $\cos(d,d') = 0$ if $d$ and $d'$ are orthogonal ( no common term)

# Similarity in a vector space



# Agglomerative Clustering of Documents

1. Initially, each document forms its own cluster.
2. The similarity between two clusters is defined as the maximal similarity between elements of each cluster.
3. Find the two clusters whose mutual similarity is highest. If it is lower than a given threshold, end the clustering. Otherwise, regroup these clusters. Repeat.

**Remark**

*Many other more refined algorithms for clustering exist.*

# Outline

# Indexing HTML

- HTML: text + meta-information + structure
- Possibly: separate index for meta-information (title, keywords)
- Increase weight of structurally emphasized content in index
- Tree structure can also be queried with XPath or XQuery, but not very useful on the Web as a whole, because of tag soup and lack of consistency.

# Indexing Multimedia Content

- Basic approach: index text from context of the media
  - surrounding text
  - text in or around the links pointing to the content
  - filenames
  - associated subtitles (hearing-impaired track on TV)
- Elaborate approach: index and search the media itself, with the help of speech recognition and sound, image, and video analysis. Becoming more and more performant!
  - TrackID, Shazam: identify a song played on the radio
  - Musipedia: look for music by whistling a tune, `http://www.musipedia.org/`, `http://www.midomi.com/`
  - Image search from a similar image, `http://images.google.com/`, Google Goggles, etc.
  - Voxalead, `http://voxaleadnews.labs.exalead.com/`

# Outline

# Precision and Recall

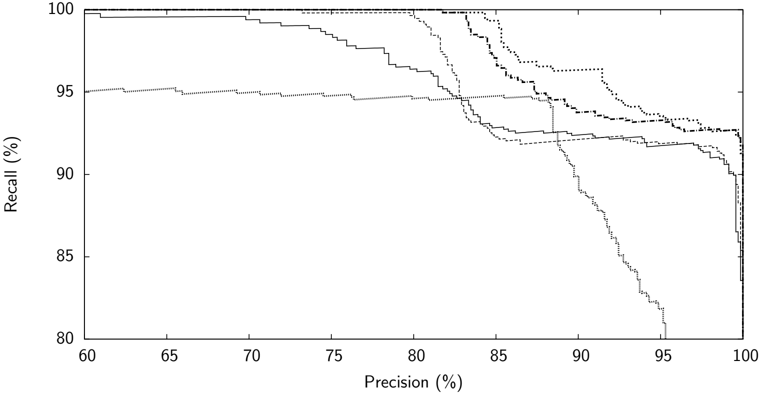
- Quality of search engines results evaluated with precision and recall

$$\text{precision} = \frac{\text{nb of correct results returned}}{\text{total nb of results}}$$

$$\text{recall} = \frac{\text{nb of correct results returned}}{\text{total nb of correct results}}$$

- "Correctness" usually given by human assessment
- Precision can be evaluated relatively reliably, much more difficult for recall! (Why?)
- Human judgment quite subjective! Agreement between human evaluators rarely go over 80%

# Precision-Recall Curve




- Computed with the precision-at-$k$, recall-at-$k$ for the $k$ top results
- Area under the curve: quality of a method
- Usually, interpolate to force the decreasing of the curve

# Outline

IR and the Inverted Index Model

Clustering

Indexing Other Media

Measuring the Quality of Results

Web Ranking

- The market of search engines
- PageRank
- Ranking Factors
- Spamdexing

# Web search engines

- A large number of different search engines, with market shares varying a lot from country to country.
- At the world level:

  **Google** vastly dominating (around 90% of the market; around 95% market share in France!)

  **Yahoo!+Bing** main competitor globally
- In some countries, local search engines dominate the market (Baidu and Shemna in Chinai) or are still present (Naver in Korea, Yandex in Russia)
- Other search engines mostly either use one of these as backend (e.g., Google for AOL) or combine the results of existing search engines (e.g., DuckDuckGo, which also has a small Web index)

# Yahoo!

In July 2009, Microsoft and Yahoo! announced a major agreement:

- Yahoo! stops developing its own search engine (launched in 2003, after the buyouts of Inktomi and Altavista) and will use Bing instead;
- Yahoo! provides the advertisement services used in Bing.

Does not concern Yahoo! Japan, which on the contrary uses Google as engine.

## Web search APIs

- Used to be plenty of free APIs to Web search engines... not the case any more
- Paid-for Web search APIs:

  **Yahoo! BOSS** 1.80 USD per 1,000 queries (uses Bing's index)
  `https://developer.yahoo.com/boss/search/`

  **Google Custom Search Engine** 100 free queries per day, 5 USD per further 1,000 queries, up to 10,000 queries per day
  `https://developers.google.com/custom-search/`

  **Bing Search API** $\approx$ 3 USD per 100,000 queries
  `https://docs.microsoft.com/en-us/azure/cognitive-services/bing-web-search/`
- Anything else?

# Outline

# Motivation

Putting together what we saw so far: Web crawls building collections of pages, listing the terms in an inverted index, answering queries with ranked results (tf–idf)

- This is what 1st generation search engines did (AltaVista) → Failed!

What made the Google search engine so successful ?

- The first to defeat spammers
- Used PageRank, a tool for estimating the importance of Web pages
- Used not only terms appearing in a page but also terms appearing in or near the links to the page.

# PageRank (Google's Ranking [Brin and Page, 1998])

## Idea

Important pages are pages pointed to by important pages.

$$\begin{cases} g_{ij} = 0 & \text{if there is no link between page } i \text{ and } j; \\ g_{ij} = \frac{1}{n_i} & \text{otherwise, with } n_i \text{ the number of outgoing links of page } i. \end{cases}$$

## Definition (Tentative)

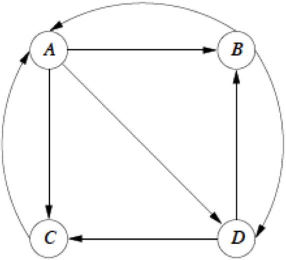
Probability that the surfer following the random walk in $G$ has arrived on page $i$ at some distant given point in the future.

$$\text{pr}(i) = \left( \lim_{k \to +\infty} (G^T)^k v \right)_i$$

where $v$ is some initial column vector.

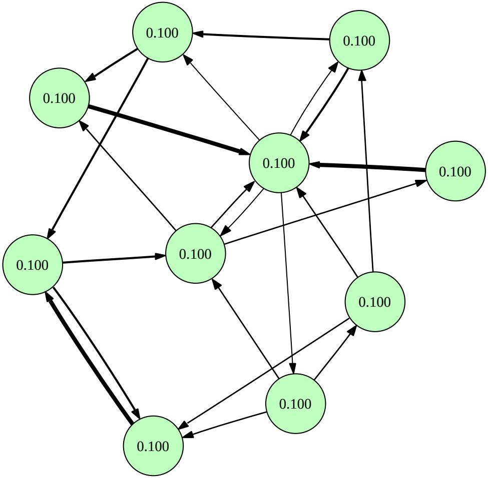
# An initial example

$$G^T = \begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix} \qquad v = \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}$$

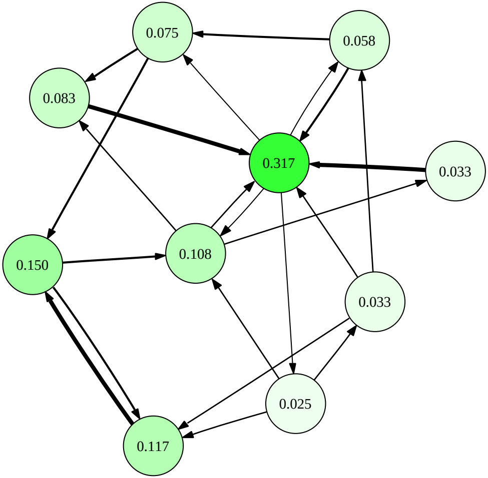



- A random surfer starting on page A, where can she be at step 1, step 2, etc ?
- What tells $v' = G^T \times v, v'' = G^T \times v' = (G^T)^2 \times v, \ldots$ ?
- Can we end with $v^{k+1} = v^k$ ?

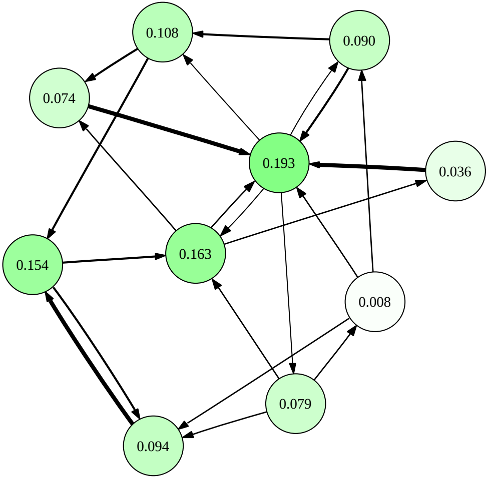
# Illustrating PageRank Computation



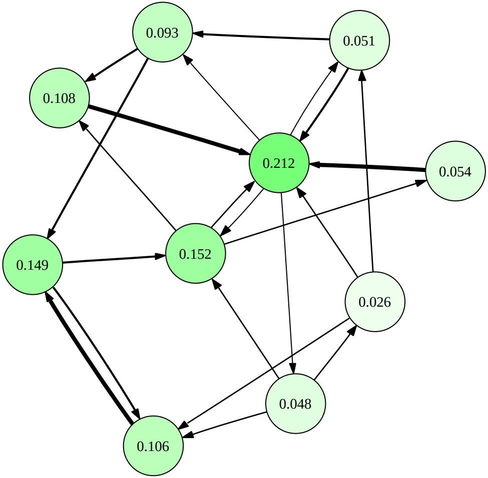
# Illustrating PageRank Computation



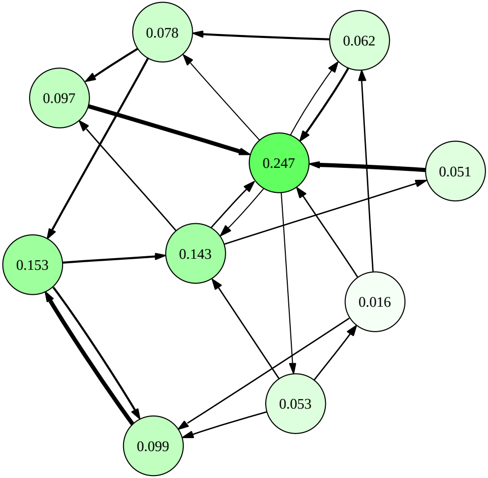
# Illustrating PageRank Computation



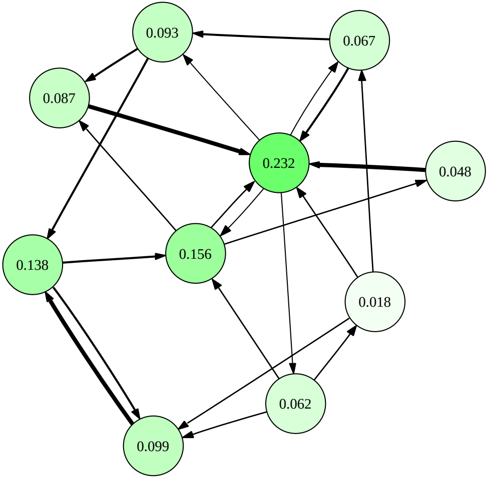
# Illustrating PageRank Computation



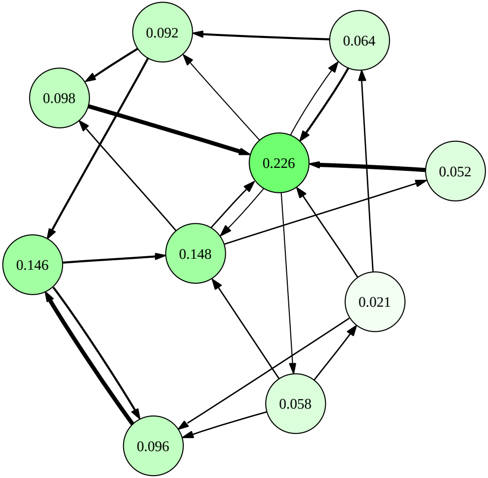
# Illustrating PageRank Computation



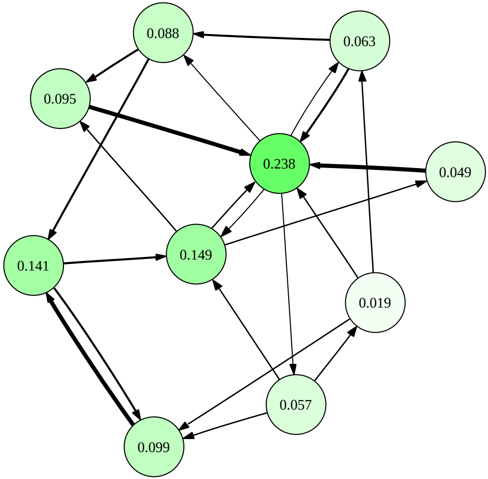
# Illustrating PageRank Computation



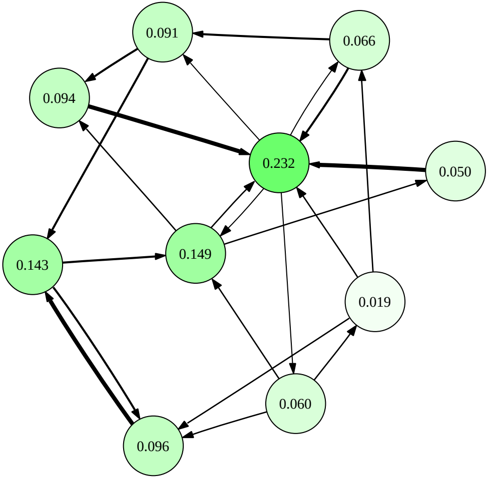
# Illustrating PageRank Computation



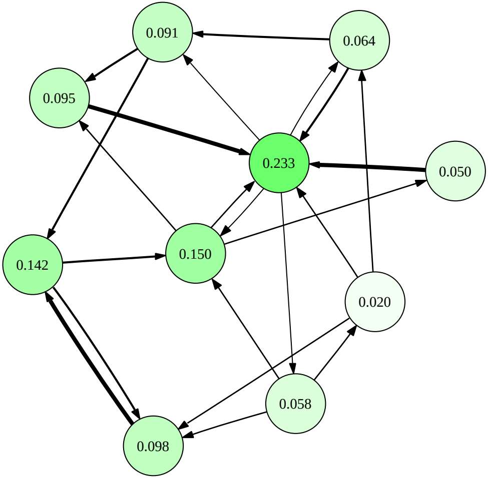
# Illustrating PageRank Computation



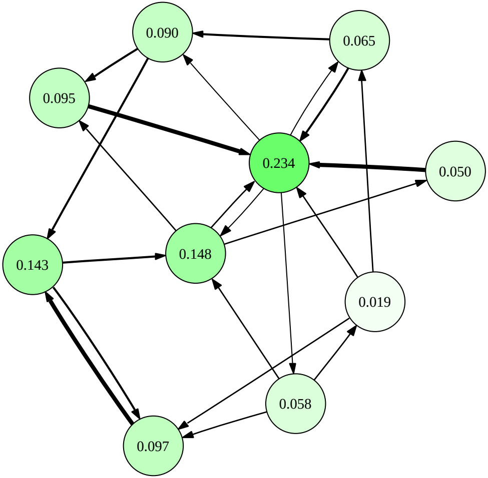
# Illustrating PageRank Computation



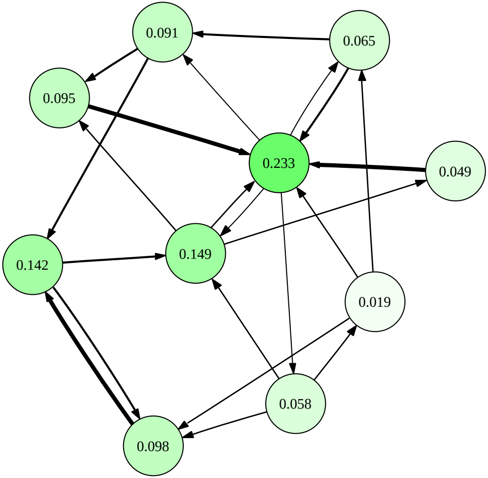
# Illustrating PageRank Computation



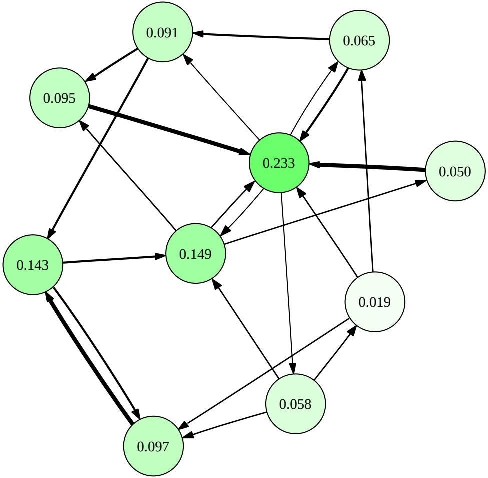
# Illustrating PageRank Computation



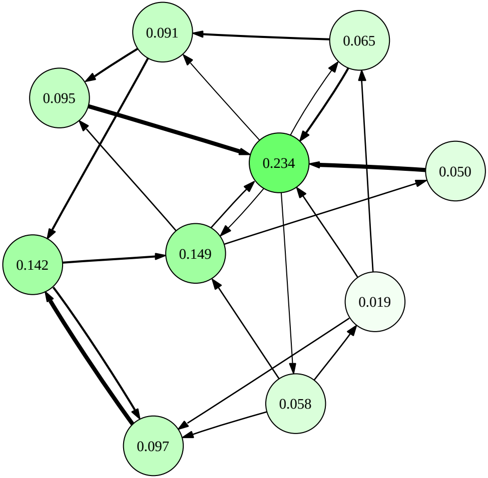
# Illustrating PageRank Computation



# Illustrating PageRank Computation



# Illustrating PageRank Computation



# PageRank with Damping

May not always converge (must be strongly connected, no dead ends, or convergence may not be unique.
To fix this, the random surfer can at each step randomly jump to any page of the Web with some probability $d$ ($1 - d$: damping factor).

$$\mathrm{pr}(i) = \left( \lim_{k \to +\infty} ((1-d)G^T + dU)^k v \right)_i$$

where $U$ is the matrix with all $\frac{1}{N}$ values with $N$ the number of vertices.

# Using PageRank to Score Query Results

- PageRank: global score, independent of the query
- Can be used to raise the weight of important pages:

$$\mathrm{weight}(t, d) = \mathrm{tfidf}(t, d) \times \mathrm{pr}(d),$$

- This can be directly incorporated in the index.

# HITS [Kleinberg, 1999]

## Idea

Two kinds of important pages: hubs and authorities. Hubs are pages that point to good authorities, whereas authorities are pages that are pointed to by good hubs.

$G'$ transition matrix (with 0 and 1 values) of a subgraph of the Web. We use the following iterative process (starting with $a$ and $h$ vectors of norm 1):

$$\begin{cases} a := \frac{1}{\|G'^T h\|} \; G'^T h \\ h := \frac{1}{\|G' a\|} \; G' a \end{cases}$$

Converges under some technical assumptions to authority and hub scores.

# Using HITS to Order Web Query Results

1. Retrieve the set $D$ of Web pages matching a keyword query.
2. Retrieve the set $D^*$ of Web pages obtained from $D$ by adding all linked pages, as well as all pages linking to pages of $D$.
3. Build from $D^*$ the corresponding subgraph $G'$ of the Web graph.
4. Compute iteratively hubs and authority scores.
5. Sort documents from $D$ by authority scores.

Less efficient than PageRank, because local scores.

# Outline

IR and the Inverted Index Model

Clustering

Indexing Other Media

Measuring the Quality of Results

Web Ranking

- The market of search engines
- PageRank
- Ranking Factors
- Spamdexing

# Ranking formula

- In modern search engines, Web query results are not just a combination of query relevance and PageRank (but these are most important)
- Instead, complex combination of dozens of components
- Can be integrated into the inverted index, or added on the fly when computing query results
- Simple way of combining: linear (or log-linear) combination of individual weights, with weights chosen in an ad-hoc manner, or, better, trained with machine learning
- Thereafter: collection of such components

# Traditional IR

Relevance weighting: tf-idf, OKAPI BM25, etc.

Position-aware scoring: rank higher terms that appear closer to each other

Metadata scoring: Use information from metadata (title, keywords, etc.); Not much used any more, too much abuse

Query rewriting and spell checking: Compare to query logs or the index to issue a similar, more popular, query

Diversification: Give results as diversified as possible

# Web graph mining

PageRank: important pages are pointed to by important pages

SiteRank: important sites are pointed to by important sites

TrustRank: to fight spam, assign initial trust to a seed of Web pages, and increase the score of neighboring pages

Link farm detection: lower score of subgraphs with dubious structures

# Relevance feedback

**Other user's feedback:** use previous clicks of other users as positive examples this link is relevant (or absence of click as negative examples)

**Own feedback:** use user's history to rank higher previously visited pages

**Manually crafted:** for common, important search terms, manually design the search result pages!

# Exploiting content and layout

**Structural emphasis:** raise score of section titles, emphasized words, etc.

**Layout emphasis:** render the page in a layout engine, and raise score of visually prominent items

**Invisible content detection:** static analysis of CSS/JS code, or layout rendering, to detect and penalize invisible content

# Quality factors

Standard-compliant: raise score of valid HTML pages

Speed of access: decrease score of slow servers

Visual appearance: decrease score of gaudy-looking pages or non-responsive designs

Up-to-date character: increase score of recently modified pages

Domain names: increase score of reputable domain names vs dubious-looking, lengthy, ones

URL structure: decrease score of lengthy or convoluted URLs

# User-centric factors

**Social search:** Bias by a users' social network (if the user is logged in)

**Location search:** Bias by a users' precise location (if available) or IP geolocation

**Language-specific search:** Bias by a user's language preferences (as reported by the browser, or as manually chosen)

**History-aware search:** Bias by a user's search history

# Outline

IR and the Inverted Index Model

Clustering

Indexing Other Media

Measuring the Quality of Results

Web Ranking
- The market of search engines
- PageRank
- Ranking Factors
- Spamdexing

Conclusion

# Spamdexing

## Definition

Fraudulent techniques that are used by unscrupulous webmasters to artificially raise the visibility of their website to users of search engines

Purpose: attracting visitors to websites to make profit.

Unceasing war between spamdexers and search engines

# Spamdexing: Lying about the Content

## Technique

Put unrelated terms in:

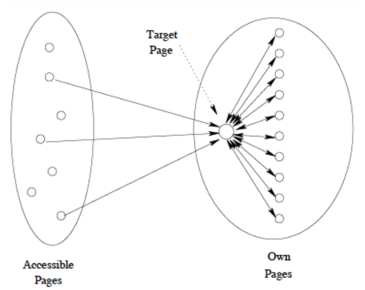
- meta-information (`<meta name="description">`, `<meta name="keywords">`)
- text content hidden to the user with JavaScript, CSS, or HTML presentational elements

## Countertechnique

- Ignore meta-information
- Try and detect invisible text

# Link Farm Attacks

Technique: Huge number of hosts on the Internet used for purpose of referencing each other, without any content in themselves, to raise importance of a given website or set of websites.




Countertechniques: Detection of websites with empty or duplicate content; heuristics to discover subgraphs that look like link farms.

# Link Pollution

### Technique

Pollute user-editable websites (blogs, wikis) or exploit security bugs to add artificial links to websites, in order to raise its importance.

### Countertechnique

`rel="nofollow"` attribute to <a> links not validated by a page's owner

# Outline

# Conclusion I

## What you should remember

- The inverted index model, associated tools and techniques
- Main ideas behind Fagin's TA and NRA
- The document vector space model
- PageRank and its iterative computation
- Complex formula for ranking Web query results

## Software

- Most DBMS have text indexing capabilities (e.g., MySQL's `FULLTEXT` indexes)
- Apache Lucene, free software library to build inverted indexes + Apache Solr, free software for building a keyword search engine (or Elasticsearch that integrates both)

# Conclusion II

## To go further

- A good textbook [Manning et al., 2008]. Available online, along with slides!
- A very influential paper on top-$k$ algorithms: [Fagin et al., 2001]
- The paper at the origins of Google [Brin and Page, 1998]

# Bibliography I

Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks*, 30(1–7): 107–117, April 1998.

Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *Proc. PODS*, Santa Barbara, USA, May 2001.

Jon M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. *Journal of the ACM*, 46(5):604–632, 1999.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, United Kingdom, 2008. Available online at http://informationretrieval.org.

Martin F. Porter. An algorithm for suffix stripping. *Program*, 14 (3):130–137, July 1980.

# Bibliography II

US National Archives and Records Administration. The Soundex indexing system. `http://www.archives.gov/genealogy/census/soundex.html`, May 2007.