Introduction to Information Retrieval

http://informationretrieval.org

IIR 7: Scores in a Complete Search System

Hinrich Schütze & Wiltrud Kessler

Institute for Natural Language Processing, University of Stuttgart

2012-11-23

Overview

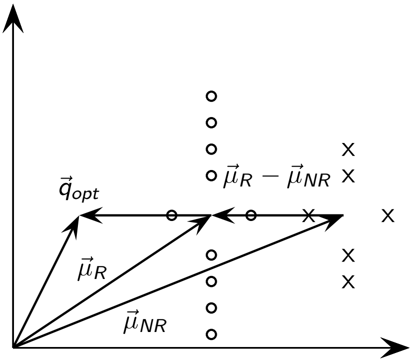
1. Recap
2. Why rank?
3. More on cosine
4. Implementation of ranking
5. The complete search system

Outline

Relevance feedback: Basic idea

- The user issues a (short, simple) query.
- The search engine returns a set of documents.
- User marks some docs as relevant, some as nonrelevant.
- Search engine computes a new representation of the information need – should be better than the initial query.
- Search engine runs new query and returns new results.
- New results have (hopefully) better recall.

Rocchio illustrated



Types of query expansion

- Manual thesaurus (maintained by editors, e.g., PubMed)
- Automatically derived thesaurus (e.g., based on co-occurrence statistics)
- Query-equivalence based on query log mining (common on the web as in the “palm” example)

Query expansion at search engines

- Main source of query expansion at search engines: query logs
- Example 1: After issuing the query [herbs], users frequently search for [herbal remedies].
  - → "herbal remedies" is potential expansion of "herb".
- Example 2: Users searching for [flower pix] frequently click on the URL photobucket.com/flower. Users searching for [flower clipart] frequently click on the same URL.
  - → "flower clipart" and "flower pix" are potential expansions of each other.

Take-away today

- The importance of ranking: User studies at Google
- Length normalization: Pivot normalization
- Implementation of ranking
- The complete search system

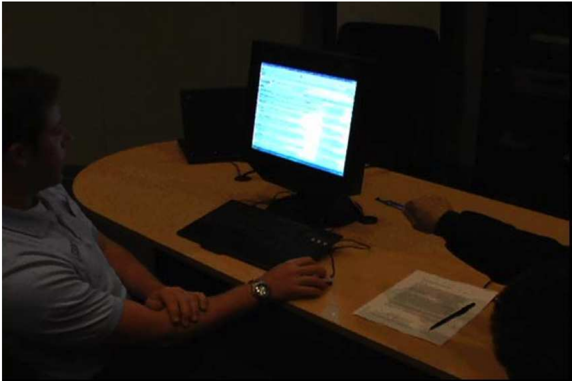
Outline

Why is ranking so important?

- Last lectures: Problems with unranked retrieval
  - Users want to look at a few results – not thousands.
  - It's very hard to write queries that produce a few results.
  - Even for expert searchers
  - $\rightarrow$ Ranking is important because it effectively reduces a large set of results to a very small one.
- Next: More data on "users only look at a few results"
- Actually, in the vast majority of cases they only examine 1, 2, or 3 results.

Empirical investigation of the effect of ranking

- How can we measure how important ranking is?
- Observe what searchers do when they are searching in a controlled setting
  - Videotape them
  - Ask them to "think aloud"
  - Interview them
  - Eye-track them
  - Time them
  - Record and count their clicks
- The following slides are from Dan Russell's JCDL talk
- Dan Russell is the "Über Tech Lead for Search Quality & User Happiness" at Google.



So.. Did you notice the FTD official site?

To be honest, I didn't even look at that.

At first I saw "from $20" and $20 is what I was looking for.

To be honest, 1800-flowers is what I'm familiar with and why I went there next even though I kind of assumed they wouldn't have $20 flowers

And you knew they were expensive?

I knew they were expensive but I thought "hey, maybe they've got some flowers for under $20 here…"

But you didn't notice the FTD?

No I didn't, actually… that's really funny.

Interview video

Rapidly scanning the results

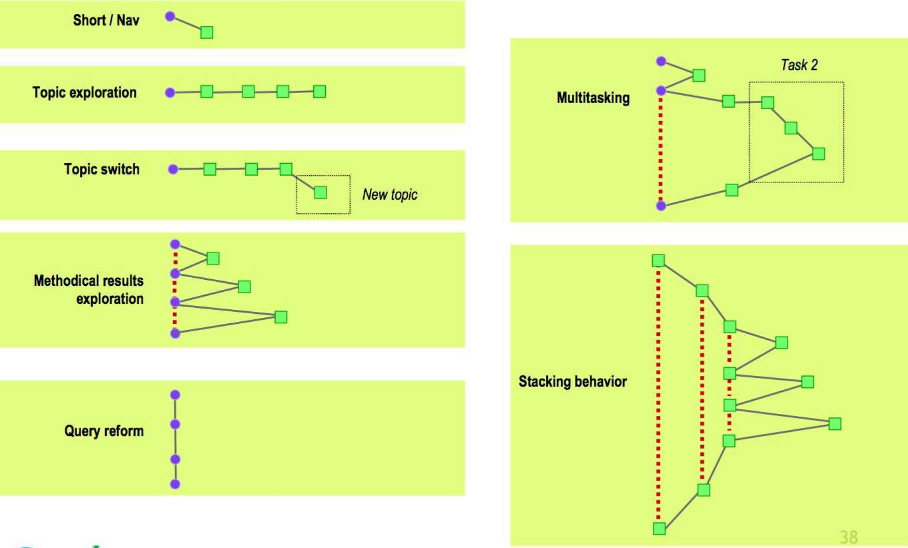
Note scan pattern:

Page 3:	Result 1
	Result 2
	Result 3
	Result 4
	Result 3
	Result 2
	Result 4
	Result 5
	Result 6 <click>

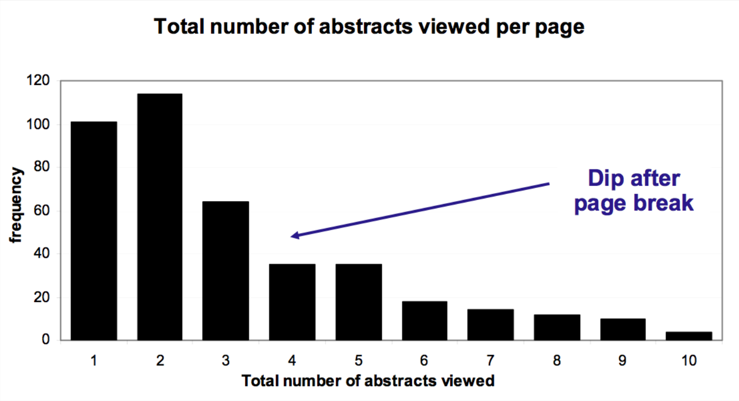
Q: Why do this?

A: What's learned later influences judgment of earlier content.

Kinds of behaviors we see in the data



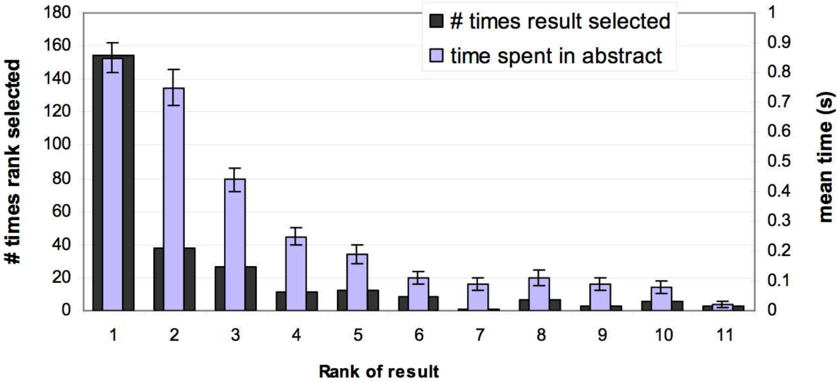
How many links do users view?



Mean: 3.07   Median/Mode: 2.00

Google

Looking vs. Clicking

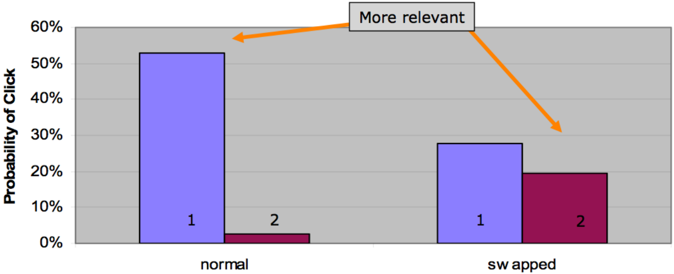


- Users view results one and two more often / thoroughly
- Users click most frequently on result one

Google

Presentation bias – reversed results

- Order of presentation influences where users look
  AND where they click



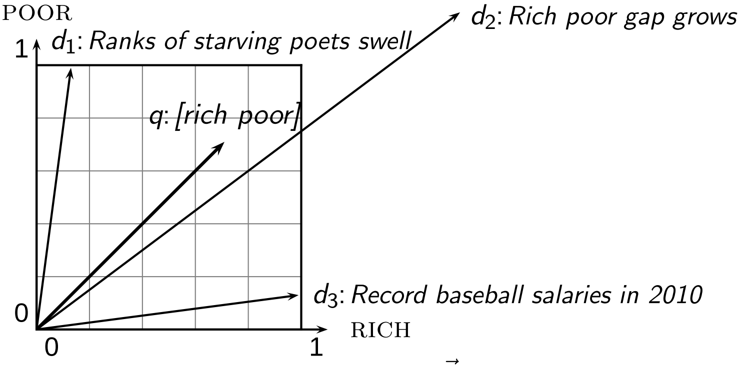
Google

Importance of ranking: Summary

- Viewing abstracts: Users are a lot more likely to read the abstracts of the top-ranked pages (1, 2, 3, 4) than the abstracts of the lower ranked pages (7, 8, 9, 10).
- Clicking: Distribution is even more skewed for clicking
- In 1 out of 2 cases, users click on the top-ranked page.
- Even if the top-ranked page is not relevant, 30% of users will click on it.
- $\rightarrow$ Getting the ranking right is very important.
- $\rightarrow$ Getting the top-ranked page right is most important.

Outline

Why distance is a bad idea



The Euclidean distance of $\vec{q}$ and $\vec{d_2}$ is large although the distribution of terms in the query q and the distribution of terms in the document d_2 are very similar.

That's why we do length normalization or, equivalently, use cosine to compute query-document matching scores.

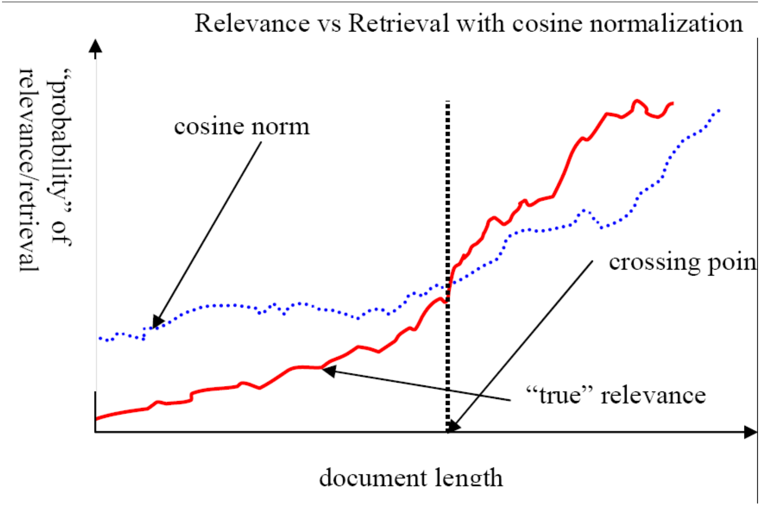
Exercise: A problem for cosine normalization

- Query q: "anti-doping rules Beijing 2008 olympics"
- Compare three documents
  - d_1: a short document on anti-doping rules at 2008 Olympics
  - d_2: a long document that consists of a copy of d_1 and 5 other news stories, all on topics different from Olympics/anti-doping
  - d_3: a short document on anti-doping rules at the 2004 Athens Olympics
- What ranking do we expect in the vector space model?
- What can we do about this?

Pivot normalization

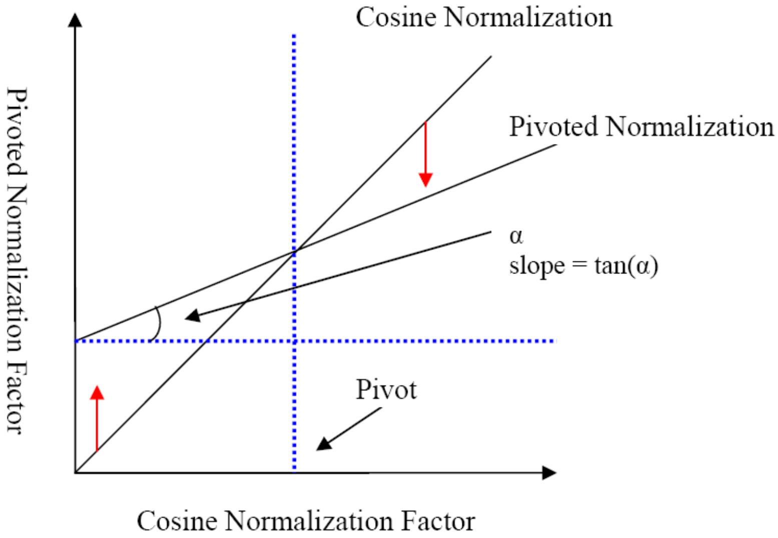
- Cosine normalization produces weights that are too large for short documents and too small for long documents (on average).
- Adjust cosine normalization by linear adjustment: “turning” the average normalization on the pivot
- Effect: Similarities of short documents with query decrease; similarities of long documents with query increase.
- This removes the unfair advantage that short documents have.

Predicted and true probability of relevance



source: Lillian Lee

Pivot normalization



source:
Lillian Le

Pivoted normalization: Amit Singhal's experiments

Cosine	Pivoted Cosine Normalization				
	Slope				
	0.60	0.65	0.70	**0.75**	0.80
6,526 0.2840	6,342 0.3024	6,458 0.3097	6,574 0.3144	**6,629** **0.3171**	6,671 0.3162
Improvement	+ 6.5%	+ 9.0%	+10.7%	**+11.7%**	+11.3%

(relevant documents retrieved and (change in) average precision)

Outline

Now we also need term frequencies in the index

BRUTUS	⟶	1,2	7,3	83,1	87,2	...
CAESAR	⟶	1,1	5,1	13,1	17,1	...
CALPURNIA	⟶	7,1	8,2	40,1	97,3	

term frequencies

We also need positions. Not shown here.

Term frequencies in the inverted index

- In each posting, store $\text{tf}_{t,d}$ in addition to docID d
- As an integer frequency, not as a (log-)weighted real number . . .
- . . . because real numbers are difficult to compress.
- Overall, additional space requirements are small: a byte per posting or less

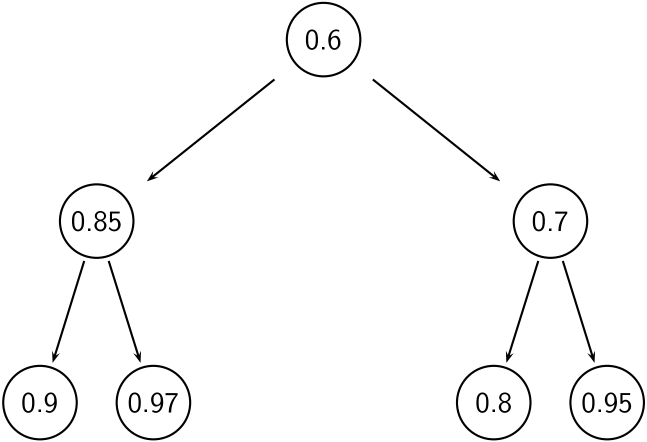
How do we compute the top k in ranking?

- In many applications, we don't need a complete ranking.
- We just need the top k for a small k (e.g., $k = 100$).
- If we don't need a complete ranking, is there an efficient way of computing just the top k?
- Naive:
  - Compute scores for all N documents
  - Sort
  - Return the top k
- Not very efficient
- Alternative: min heap

Use min heap for selecting top k ouf of N

- A binary min heap is a binary tree in which each node's value is less than the values of its children.
- Takes $O(N \log k)$ operations to construct (where N is the number of documents) ...
- ...then read off k winners in $O(k \log k)$ steps

Binary min heap



Selecting top k scoring documents in $O(N \log k)$

- Goal: Keep the top k documents seen so far
- Use a binary min heap
- To process a new document d' with score s':
  - Get current minimum h_m of heap ($O(1)$)
  - If $s' \leq h_m$ skip to next document
  - If $s' > h_m$ heap-delete-root ($O(\log k)$)
  - Heap-add d'/s' ($O(\log k)$)

Even more efficient computation of top k?

- Ranking has time complexity $O(N)$ where N is the number of documents.
- Optimizations reduce the constant factor, but they are still $O(N)$, $N > 10^{10}$
- Are there sublinear algorithms?
- What we're doing in effect: solving the k-nearest neighbor (kNN) problem for the query vector ($=$ query point).
- There are no general solutions to this problem that are sublinear.

More efficient computation of top k: Heuristics

- Idea 1: Reorder postings lists
  - Instead of ordering according to docID . . .
  - . . . order according to some measure of "expected relevance".
- Idea 2: Heuristics to prune the search space
  - Not guaranteed to be correct . . .
  - . . . but fails rarely.
  - In practice, close to constant time.
  - For this, we'll need the concepts of document-at-a-time processing and term-at-a-time processing.

Non-docID ordering of postings lists

- So far: postings lists have been ordered according to docID.
- Alternative: a query-independent measure of "goodness" of a page
- Example: PageRank $g(d)$ of page d, a measure of how many "good" pages hyperlink to d (chapter 21)
- Order documents in postings lists according to PageRank: $g(d_1) > g(d_2) > g(d_3) > \dots$
- Define composite score of a document:

$$\text{net-score}(q, d) = g(d) + \cos(q, d)$$

- This scheme supports early termination: We do not have to process postings lists in their entirety to find top k.

Non-docID ordering of postings lists (2)

- Order documents in postings lists according to PageRank: $g(d_1) > g(d_2) > g(d_3) > \dots$
- Define composite score of a document:

$$\text{net-score}(q, d) = g(d) + \cos(q, d)$$

- Suppose: (i) $g \rightarrow [0, 1]$; (ii) $g(d) < 0.1$ for the document d we're currently processing; (iii) smallest top k score we've found so far is 1.2
- Then all subsequent scores will be < 1.1.
- So we've already found the top k and can stop processing the remainder of postings lists.
- Questions?

Document-at-a-time processing

- Both docID-ordering and PageRank-ordering impose a consistent ordering on documents in postings lists.
- Computing cosines in this scheme is document-at-a-time.
- We complete computation of the query-document similarity score of document d_i before starting to compute the query-document similarity score of d_{i+1}.
- Alternative: term-at-a-time processing

Weight-sorted postings lists

- Idea: don't process postings that contribute little to final score
- Order documents in postings list according to weight
- Simplest case: normalized tf-idf weight (rarely done: hard to compress)
- Documents in the top k are likely to occur early in these ordered lists.
- $\rightarrow$ Early termination while processing postings lists is unlikely to change the top k.
- But:
  - We no longer have a consistent ordering of documents in postings lists.
  - We no longer can employ document-at-a-time processing.

Term-at-a-time processing

- Simplest case: completely process the postings list of the first query term
- Create an accumulator for each docID you encounter
- Then completely process the postings list of the second query term
- . . . and so forth

Term-at-a-time processing

CosineScore(q)

```
float Scores[N] = 0
float Length[N]
for each query term t
do calculate w_{t,q} and fetch postings list for t
    for each pair(d, tf_{t,d}) in postings list
    do Scores[d] += w_{t,d} × w_{t,q}
Read the array Length
for each d
do Scores[d] = Scores[d]/Length[d]
return Top k components of Scores[]
```

The elements of the array “Scores” are called accumulators.

Accumulators

- For the web (20 billion documents), an array of accumulators A in memory is infeasible.
- Thus: Only create accumulators for docs occurring in postings lists
- This is equivalent to: Do not create accumulators for docs with zero scores (i.e., docs that do not contain any of the query terms)

Accumulators: Example

BRUTUS	⟶	1,2	7,3	83,1	87,2	. . .
CAESAR	⟶	1,1	5,1	13,1	17,1	. . .
CALPURNIA	⟶	7,1	8,2	40,1	97,3	

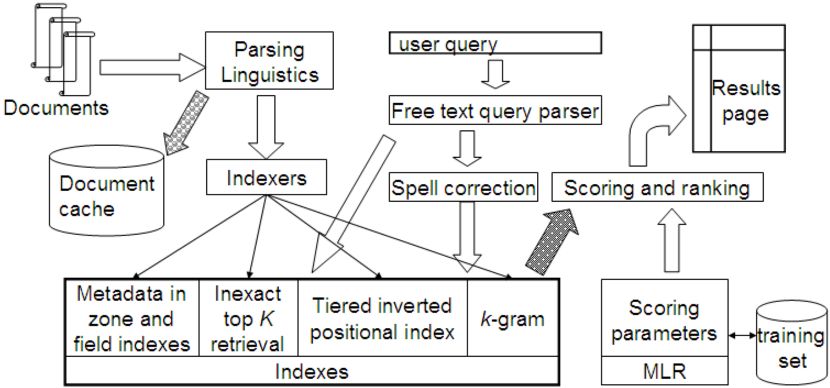
- For query: [Brutus Caesar]:
- Only need accumulators for 1, 5, 7, 13, 17, 83, 87
- Don't need accumulators for 3, 8 etc.

Enforcing conjunctive search

- We can enforce conjunctive search (a la Google): only consider documents (and create accumulators) if all terms occur.
- Example: just one accumulator for [Brutus Caesar] in the example above . . .
- . . . because only d_1 contains both words.

Outline

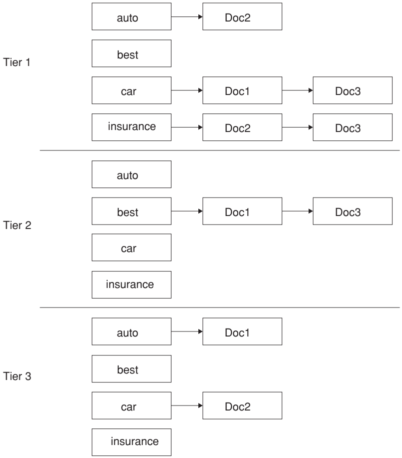
Complete search system



Tiered indexes

- Basic idea:
  - Create several tiers of indexes, corresponding to importance of indexing terms
  - During query processing, start with highest-tier index
  - If highest-tier index returns at least k (e.g., $k = 100$) results: stop and return results to user
  - If we've only found $< k$ hits: repeat for next index in tier cascade
- Example: two-tier system
  - Tier 1: Index of all titles
  - Tier 2: Index of the rest of documents
  - Pages containing the search words in the title are better hits than pages containing the search words in the body of the text.

Tiered index



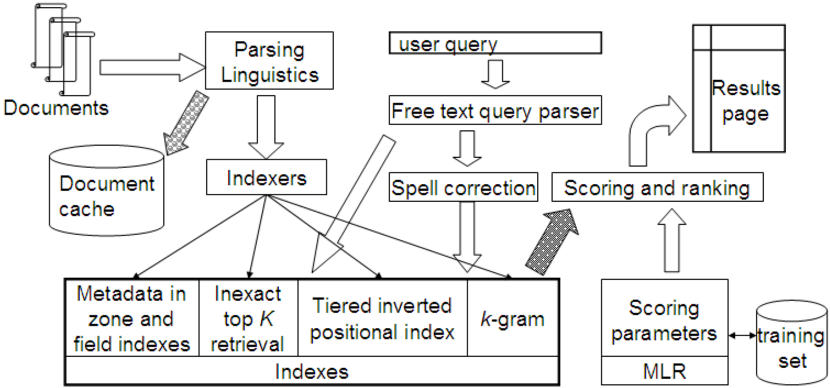
Tiered indexes

- The use of tiered indexes is believed to be one of the reasons that Google search quality was significantly higher initially (2000/01) than that of competitors.
- (along with PageRank, use of anchor text and proximity constraints)

Exercise

- Design criteria for tiered system
  - Each tier should be an order of magnitude smaller than the next tier.
  - The top 100 hits for most queries should be in tier 1, the top 100 hits for most of the remaining queries in tier 2 etc.
  - We need a simple test for "can I stop at this tier or do I have to go to the next one?"
    - There is no advantage to tiering if we have to hit most tiers for most queries anyway.
- Consider a two-tier system where the first tier indexes titles and the second tier everything.
- Question: Can you think of a better way of setting up a multitier system? Which "zones" of a document should be indexed in the different tiers (title, body of document, others?)? What criterion do you want to use for including a document in tier 1?

Complete search system



Components we have introduced thus far

- Document preprocessing (linguistic and otherwise)
- Positional indexes
- Tiered indexes
- Spelling correction
- k-gram indexes for wildcard queries and spelling correction
- Query processing
- Document scoring
- Term-at-a-time processing

Components we haven't covered yet

- Document cache: we need this for generating snippets (= dynamic summaries)
- Zone indexes: They separate the indexes for different zones: the body of the document, all highlighted text in the document, anchor text, text in metadata fields etc
- Machine-learned ranking functions
- Proximity ranking (e.g., rank documents in which the query terms occur in the same local window higher than documents in which the query terms occur far from each other)
- Query parser

Vector space retrieval: Interactions

- How do we combine phrase retrieval with vector space retrieval?
- We do not want to compute document frequency / idf for every possible phrase. Why?
- How do we combine Boolean retrieval with vector space retrieval?
- For example: “+”-constraints and “-”-constraints
- Postfiltering is simple, but can be very inefficient – no easy answer.
- How do we combine wild cards with vector space retrieval?
- Again, no easy answer

Take-away today

- The importance of ranking: User studies at Google
- Length normalization: Pivot normalization
- Implementation of ranking
- The complete search system

Resources

- Chapters 6 and 7 of IIR
- Resources at `http://ifnlp.org/ir`
  - How Google tweaks its ranking function
  - Interview with Google search guru Udi Manber
  - Amit Singhal on Google ranking
  - SEO perspective: ranking factors
  - Yahoo Search BOSS: Opens up the search engine to developers. For example, you can rerank search results.
  - Compare Google and Yahoo ranking for a query
  - How Google uses eye tracking for improving search