NPFL103: Information Retrieval (5)

Ranking, Complete search system, Evaluation, Benchmarks

Pavel Pecina

pecina@ufal.mff.cuni.cz

Institute of Formal and Applied Linguistics
Faculty of Mathematics and Physics
Charles University

Original slides are courtesy of Hinrich Schütze, University of Stuttgart.

Contents

Ranking

Why is ranking so important?

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 effectively reduces a large set of results to a very small one.

Empirical investigation of the effect of ranking

- How can we measure how important ranking is?
- Observe what searchers do while 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 from Google.

Rapidly scanning the results

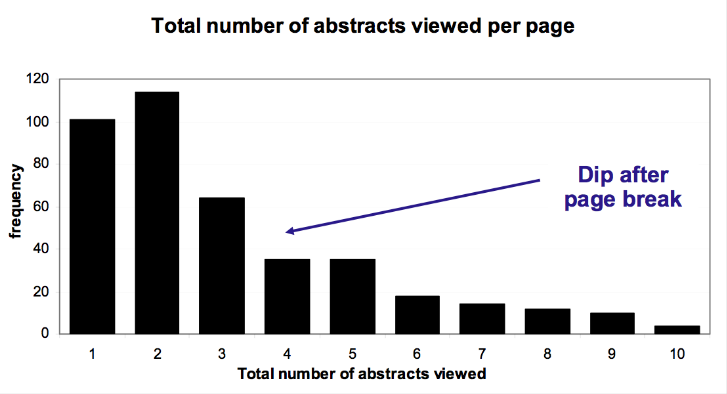
Note scan pattern:

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.

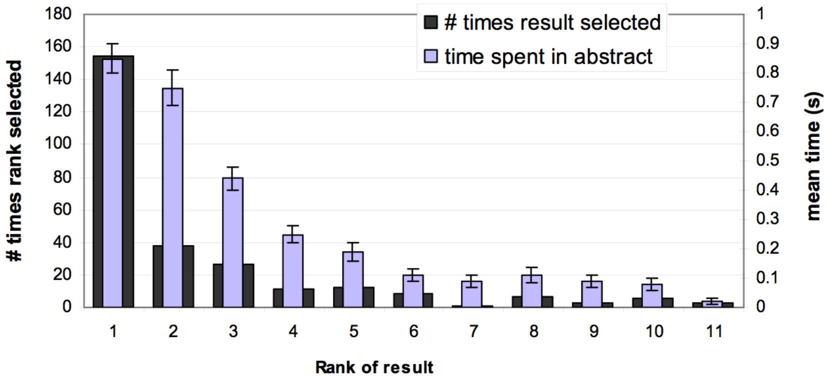
How many links do users view?



Mean: 3.07 Median/Mode: 2.00

Google

Looking vs. Clicking

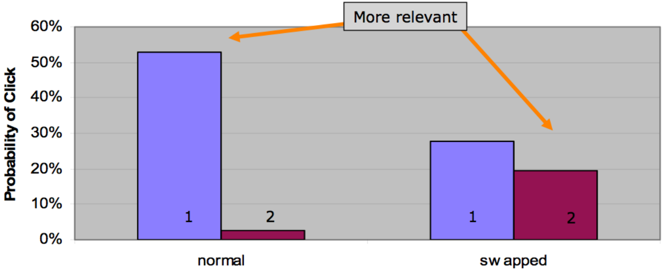


- Users view results one and two more often / thoroughly
- Users click most frequently on result one

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 (50%!), users click on the top-ranked page.
- Even if the top-ranked page is not relevant, 30% of users click on it.

$\rightarrow$ Getting the ranking right is very important.

$\rightarrow$ Getting the top-ranked page right is most important.

We need term frequencies in the index

BRUTUS	$\longrightarrow$	1,2	7,3	83,1	87,2	...
CAESAR	$\longrightarrow$	1,1	5,1	13,1	17,1	...
CALPURNIA	$\longrightarrow$	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 $\mathrm{tf}_{t,d}$ in addition to docID of d.
- Use an integer frequency, not as a (log-)weighted real number ...

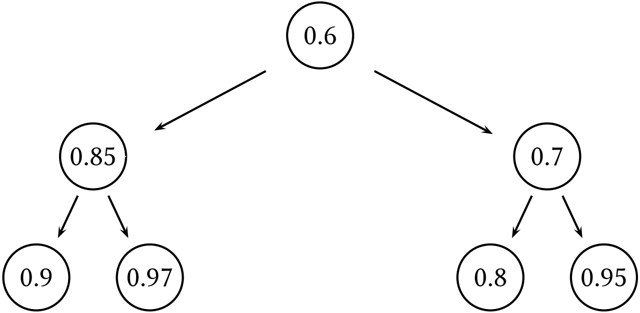
  ... because real numbers are difficult to compress.
- Additional space requirements are small: a byte per posting or less.

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., $k = 100$).
- Is there an efficient way of computing just the top k?
- Naive (not very efficient):
  - Compute scores for all N documents
  - Sort
  - Return the top k
- 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 build (N – number of documents)
- And then $O(k \log k)$ steps to read off k winners.

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':
  1. Get current minimum h_m of heap ($O(1)$)
  2. If $s' \leq h_m$ skip to next document
  3. If $s' > h_m$ heap-delete-root ($O(\log k)$)
  4. Heap-add d'/s' ($O(\log k)$)

Even more efficient computation of top *k*?

- Ranking has time complexity $O(N)$, N is the number of documents.
- Optimizations reduce the constant factor, but 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 (later in this course)
- Order documents in postings lists according to PageRank: $g(d_1) > g(d_2) > g(d_3) > \dots$
- Define composite score of a document: $s(q, d) = g(d) + \cos(q, d)$
- 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: $s(q, d) = g(d) + \cos(q, d)$
- Suppose:
  (i) $g \to [0, 1]$;
  (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.

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 (rarely done: hard to compress).
- Top-k documents are likely to occur early in these ordered lists.

  $\rightarrow$ Early termination is unlikely to change the top k.
- But:
  - no consistent ordering of documents in postings lists.
  - no way to employ document-at-a-time processing.

Term-at-a-time processing

- Simplest case: completely process 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 $\mathrm{w}_{t,q}$ and fetch postings list for t

    for each pair$(d, \mathrm{tf}_{t,d})$ in postings list

    do $Scores[d] += \mathrm{w}_{t,d} \times \mathrm{w}_{t,q}$

Read the array *Length*

for each d

do $Scores[d] = Scores[d]/Length[d]$

return Top k components of $Scores[]$

- Accumulators ("Scores[]") as an array not optimal (or even infeasible).
- Thus: Only create accumulators for docs occurring in postings lists.

Accumulators: Example

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

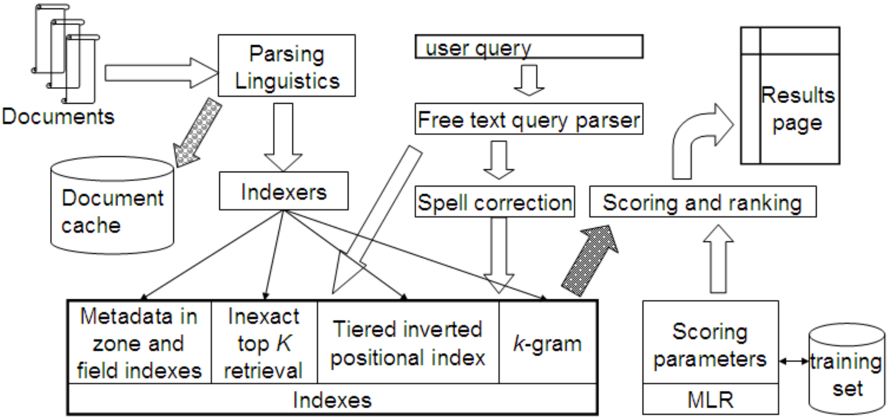
- For query: [Brutus Caesar]:
- Only need accumulators for 1, 5, 7, 13, 17, 83, 87
- Don't need accumulators for 3, 8 etc.

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.

Complete search system

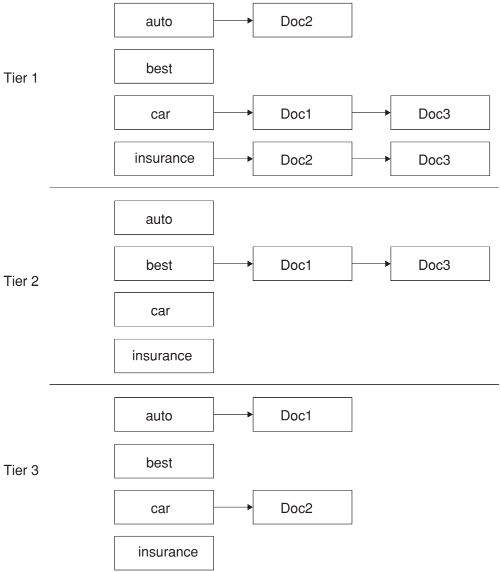
Complete search system



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
  - Motivation: 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



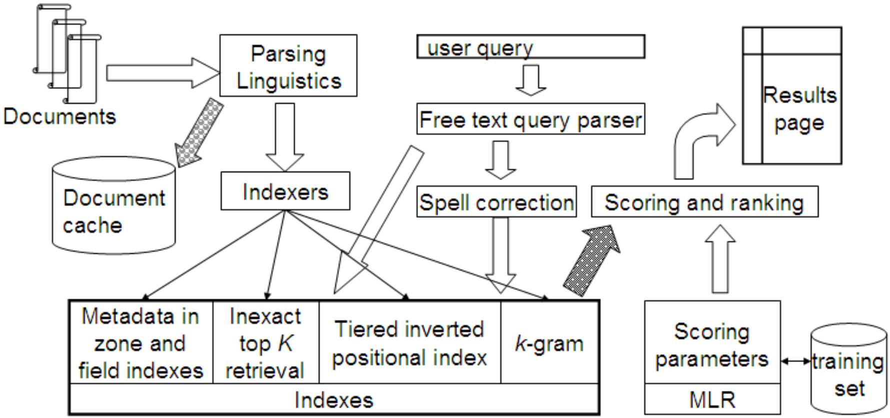
Tiered indexes

- 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)

Query parser

- IR systems often guess what the user intended.
- The two-term query *London tower* (without quotes) may be interpreted as the phrase query *"London tower"*.
- The query *100 Madison Avenue, New York* may be interpreted as a request for a map.
- How do we "parse" the query and translate it into a formal specification containing phrase operators, proximity operators, indexes to search etc.?

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: 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)

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.

Evaluation

Measures for a search engine

1. How fast does it index?
   - e.g., number of bytes per hour
2. How fast does it search?
   - e.g., latency as a function of queries per second
3. What is the cost per query?
   - in dollars

Measures for a search engine

- All of the preceding criteria are measurable: speed / size / money
- However, the key measure for a search engine is user happiness.
- Factors of user happiness include:
  - Speed of response
  - Size of index
  - Uncluttered UI
  - Most important: relevance
  - (actually, maybe even more important: it's free)
- Note that none of these is sufficient: blindingly fast, but useless answers won't make a user happy.
- How can we quantify user happiness?

Who is the user?

- Web search engine: searcher
  - Success: Searcher finds what she was looking for
  - Measure: rate of return to this search engine
- Web search engine: advertiser
  - Success: Searcher clicks on ad
  - Measure: clickthrough rate
- Ecommerce: buyer
  - Success: Buyer buys something
  - Measures: purchase time, fraction of searchers-to-buyers "conversions"
- Ecommerce: seller
  - Success: Seller sells something
  - Measure: profit per item sold
- Enterprise: CEO
  - Success: Employees are more productive (because of effective search)
  - Measure: profit of the company

Most common definition of user happiness: Relevance

- User happiness equated with relevance of search results to the query.
- But how do you measure relevance?
- Standard methodology in IR consists of three elements:
  1. A benchmark document collection.
  2. A benchmark suite of queries.
  3. An assessment of the relevance of each query-document pair.

Relevance to what?

"Relevance to the query" is very problematic:

- Information need i: "I am looking for information on whether drinking red wine is more effective at reducing your risk of heart attacks than white wine."

  (This is an information need, not a query.)

- Query q: [red wine white wine heart attack]

- Consider document d':
  At heart of his speech was an attack on the wine industry lobby for downplaying the role of red and white wine in drunk driving.

- d' is an excellent match for query q but not relevant to the information need i.

Relevance: query vs. information need

- User happiness can only be measured by relevance to an information need, not by relevance to queries.
- Our terminology is sloppy – though we mean information-need-document relevance judgments.

Precision and recall

- Precision (P) 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 (R) is the fraction of relevant documents that are retrieved:

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

Precision and recall: confusion matrix

	Relevant	Nonrelevant
Retrieved	true positives (TP)	false positives (FP)
Not retrieved	false negatives (FN)	true negatives (TN)

$$P = \frac{TP}{TP + FP} \qquad P = \frac{TP}{TP + FN}$$

Precision/recall tradeoff

- You can increase recall by returning more docs.
- Recall is a non-decreasing function of the number of docs retrieved.
- A system that returns all docs has 100% recall!
- The converse is also true (usually): It's easy to get high precision for very low recall.
- Suppose the document with the largest score is relevant. How can we maximize precision?

A combined measure: F

- The F measure allows us to trade off precision against recall.

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R} \quad \text{where} \quad \beta^2 = \frac{1-\alpha}{\alpha}$$

- $\alpha \in [0,1]$ and thus $\beta^2 \in [0,\infty]$
- Most frequently used: balanced F with $\beta = 1$ or $\alpha = 0.5$
- This is the harmonic mean of P and R: $\frac{1}{F} = \frac{1}{2}(\frac{1}{P} + \frac{1}{R})$
- What value range of β weights recall higher than precision?

F measure: Example

	relevant	not relevant	
retrieved	20	40	60
not retrieved	60	1,000,000	1,000,060
	80	1,000,040	1,000,120

- $P = 20/(20+40) = 1/3$
- $R = 20/(20+60) = 1/4$
- $F_1 = 2\frac{1}{\frac{1}{\frac{1}{3}}+\frac{1}{\frac{1}{4}}} = 2/7$

Accuracy

- Why do we use complex measures like precision, recall, and F?
- Why not something simple like accuracy?
- Accuracy is the fraction of correct decisions (relevant/nonrelevant)
- In terms of the contingency table above:

$$A = \frac{TP + TN}{TP + FP + FN + TN}$$

- Why is accuracy not a useful measure for web information retrieval?

Why accuracy is a useless measure in IR

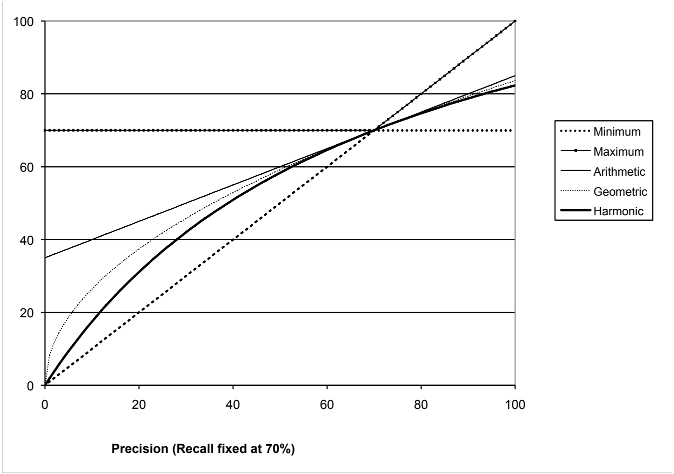
- The number of relavant and non-relevant documents is unbalanced.
- A trick to maximize accuracy in IR: always say no and return nothing.
- You then get 99.99% accuracy on most queries (0.01% docs relevant).
- Searchers on the web (and in IR in general) want to find something and have a certain tolerance for junk.
- It's better to return some bad hits as long as you return something.

  $\rightarrow$ We use precision, recall, and F for evaluation, not accuracy.

F measure: Why harmonic mean?

- Why don't we use a different mean of P and R as a measure?
  - e.g., the arithmetic mean
- The simple (arithmetic) mean is 50% for "return-everything" search engine ($P = 0\%$, $R = 100\%$), which is too high.
- Desideratum: Punish bad performance on either precision or recall.
- Taking the minimum achieves this.
- But minimum is not smooth and hard to weight.
- F (harmonic mean) is a kind of smooth minimum.

F_1 and other averages



- We can view the harmonic mean as a kind of soft minimum.

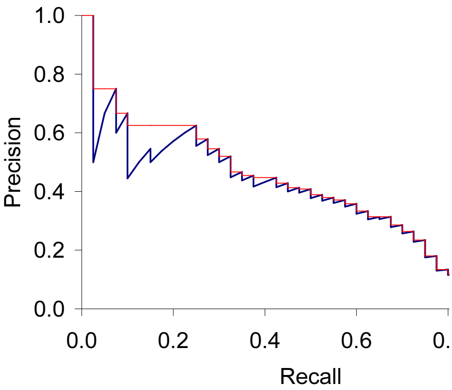
Difficulties in using precision, recall and F measure

- We should always average over a large set of queries.
- We need relevance judgments for information-need-document pairs – but they are expensive to produce.
- Alternatives to using precision/recall and having to produce relevance judgments exists (A/B testing).

Precision-recall curve

- Precision/recall/F are measures for unranked sets.
- We can easily turn set measures into measures of ranked lists.
- Just compute the set measure for each "prefix" of the ranked list: the top 1, top 2, top 3, top 4 etc. results.
- Doing this for precision and recall gives you a precision-recall curve.

A precision-recall curve



- Each point corresponds to a result for top *k* ranked hits (*k*=1,2,3,...)
- Interpolation (in red): Take maximum of all future points.
- Rationale for interpolation: The user is willing to look at more stuff if both precision and recall get better.

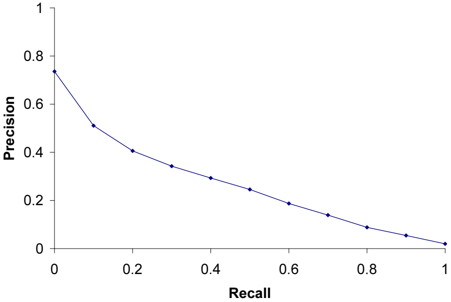
11-point interpolated average precision

Recall	Interpolated Precision
0.0	1.00
0.1	0.67
0.2	0.63
0.3	0.55
0.4	0.45
0.5	0.41
0.6	0.36
0.7	0.29
0.8	0.13
0.9	0.10
1.0	0.08

11-point average: ≈ 0.425

How can precision at 0.0 be > 0?

Averaged 11-point precision/recall graph



- Compute interpolated precision at recall levels 0.0, 0.1, 0.2, ...
- Do this for each of the queries in the evaluation benchmark.
- Average over queries.
- This measure measures performance at all recall levels.

Evaluation at large search engines

- Recall is difficult to measure on the web.
- Search engines often use precision at top k, e.g., $k = 10$ or use measures that reward you more for getting rank 1 right than for getting rank 10 right.
- Search engines also use non-relevance-based measures.
  - Example 1: clickthrough on first result

    Not very reliable if you look at a single clickthrough (you may realize after clicking that the summary was misleading and the document is nonrelevant but pretty reliable in the aggregate.)
  - Example 2: Ongoing studies of user behavior in the lab
  - Example 3: A/B testing

A/B testing

- Purpose: Test a single innovation
- Prerequisite: You have a large search engine up and running
- Steps:
  1. Have most users use the old system.
  2. Divert a small proportion of traffic (e.g., 1%) to the new system.
  3. Evaluate with an automatic measure like clickthrough on first result.
  4. Directly see if the innovation does improve user happiness.
- Probably the eval. methodology that large search engines trust most.
- Variant: Give users the option to switch to new algorithm/interface.

Benchmarks

What we need for a benchmark

1. A collection of documents
   - Must be representative of the documents we expect to see in reality.
2. A collection of information needs
   - (which we will often incorrectly refer to as queries)
   - Must be representative of the inform. needs we expect to see in reality.
3. Human relevance assessments
   - We need to hire/pay “judges” or assessors to do this.
   - Expensive, time-consuming.
   - Judges must be representative of the users we expect to see in reality.

Standard relevance benchmark: Cranfield

- Pioneering: first testbed allowing precise quantitative measures of information retrieval effectiveness.
- Late 1950s, UK.
- 1398 abstracts of aerodynamics journal articles, a set of 225 queries, exhaustive relevance judgments of all query-document-pairs.
- Too small, too untypical for serious IR evaluation today.

Standard relevance benchmark: TREC

- TREC = Text Retrieval Conference (TREC)
- Organized by National Institute of Standards and Technology (NIST)
- TREC is actually a set of several different relevance benchmarks.
- Best known: TREC Ad Hoc, used for TREC evaluations in 1992 – 1999
- 1.89 M documents, mainly newswire articles, 450 information needs
- No exhaustive relevance judgments – too expensive
- Rather, NIST assessors' relevance judgments are available only for the documents that were among the top *k* returned for some system which was entered in the TREC evaluation for which the information need was developed.

Standard relevance benchmarks: Others

- GOV2
  - Another TREC/NIST collection
  - 25 million web pages
  - Used to be largest collection that is easily available
  - But still 3 orders of magnitude smaller than what Google/Bing index
- NTCIR
  - East Asian language and cross-language information retrieval
- Cross Language Evaluation Forum (CLEF)
  - This evaluation series has concentrated on European languages and cross-language information retrieval.
- Many others