ADFOCS 2004

Prabhakar Raghavan

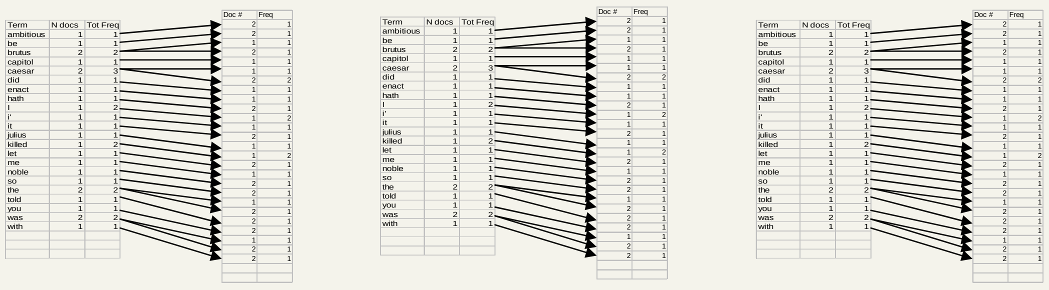
Lecture 3

Zones

- A zone is an identified region within a doc
  - E.g., Title, Abstract, Bibliography
  - Generally culled from marked-up input or document metadata (e.g., powerpoint)
- Contents of a zone are free text
  - Not a "finite" vocabulary
- Indexes for each zone - allow queries like
  - ***sorting*** in Title AND ***smith*** in Bibliography AND ***recur**** in Body
- Not queries like "all papers whose authors cite themselves"

Why?

Zone indexes – simple view



Title Author Body etc.

Scoring

- Thus far, our queries have all been Boolean
  - Docs either match or not
- Good for expert users with precise understanding of their needs and the corpus
- Applications can consume 1000's of results
- Not good for (the majority of) users with poor Boolean formulation of their needs
- Most users don't want to wade through 1000's of results – cf. altavista

Scoring

- *We wish to return in order the documents most likely to be useful to the searcher*
- How can we rank order the docs in the corpus with respect to a query?
- Assign a score – say in [0,1]
  - for each doc on each query
- Begin with a perfect world – no spammers
  - Nobody stuffing keywords into a doc to make it match queries
  - More on this in 276B under web search

Linear zone combinations

- First generation of scoring methods: use a linear combination of Booleans:
  - E.g.,

Score = 0.6*<***sorting*** in <u>Title></u> + 0.3*<***sorting*** in <u>Abstract</u>> + 0.1*<***sorting*** in <u>Body</u>>

  - Each expression such as <***sorting*** in <u>Title</u>> takes on a value in {0,1}.
  - Then the overall score is in [0,1].

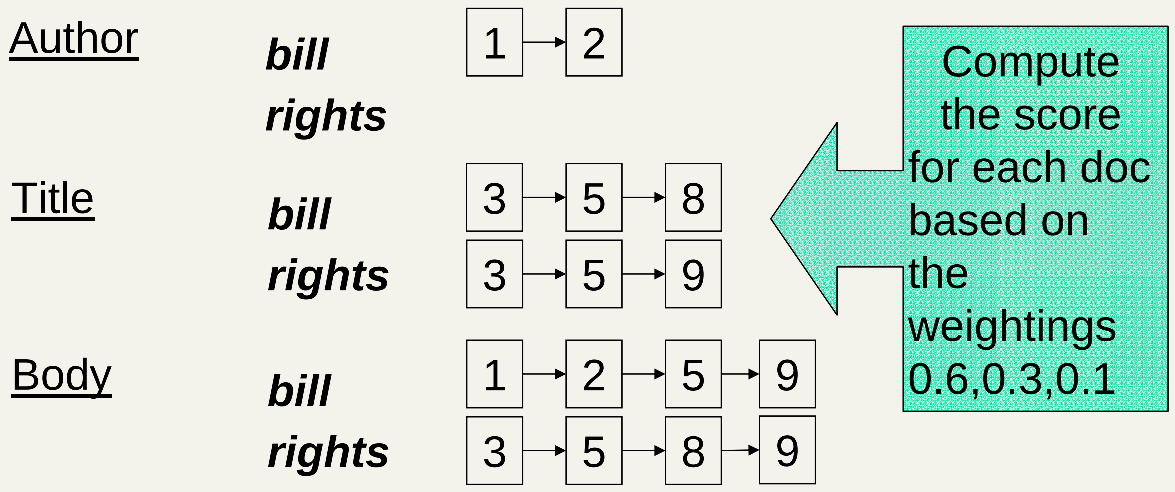
For this example the scores can only take on a finite set of values – what are they?

Linear zone combinations

- In fact, the expressions between <> on the last slide could be *any* Boolean query
- Who generates the Score expression (with weights such as 0.6 etc.)?
  - In uncommon cases – the user through the UI
  - Most commonly, a query parser that takes the user's Boolean query and runs it on the indexes for each zone
  - Weights determined from user studies and hard-coded into the query parser

Exercise

- On the query ***bill** OR **rights*** suppose that we retrieve the following docs from the various zone indexes:



Author	***bill***	1 → 2
	rights	
Title	***bill***	3 → 5 → 8
	rights	3 → 5 → 9
Body	***bill***	1 → 2 → 5 → 9
	rights	3 → 5 → 8 → 9

Compute the score for each doc based on the weightings 0.6,0.3,0.1

General idea

- We are given a <u>weight vector</u> whose components sum up to 1.
  - There is a weight for each zone/field.
- Given a Boolean query, we assign a score to each doc by adding up the weighted contributions of the zones/fields.
- Typically – users want to see the *K* highest-scoring docs.

Index support for zone combinations

- In the simplest version we have a separate inverted index for each zone
- Variant: have a single index with a separate dictionary entry for each term and zone
- E.g.,

bill.author $$\begin{tikzcd}\boxed{1} \arrow[r] & \boxed{2}\end{tikzcd}$$

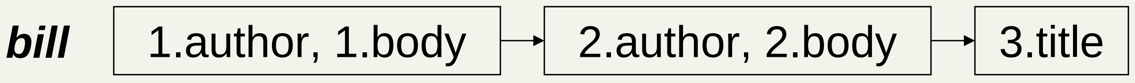
bill.title $$\begin{tikzcd}\boxed{3} \arrow[r] & \boxed{5} \arrow[r] & \boxed{8}\end{tikzcd}$$

bill.body $$\begin{tikzcd}\boxed{1} \arrow[r] & \boxed{2} \arrow[r] & \boxed{5} \arrow[r] & \boxed{9}\end{tikzcd}$$

Of course, compress zone names like author/title/body.

Zone combinations index

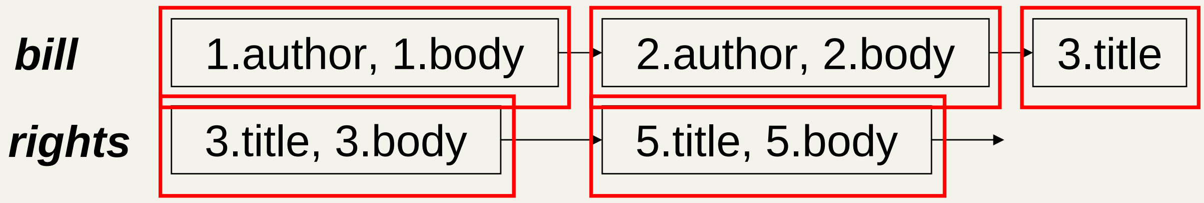
- The above scheme is still wasteful: each term is potentially replicated for each zone
- In a slightly better scheme, we encode the zone in the postings:



As before, the zone names get compressed.

- At query time, accumulate contributions to the total score of a document from the various postings, e.g.,

Score accumulation



- As we walk the postings for the query ***bill*** *OR* ***rights***, we accumulate scores for each doc in a linear merge as before.
- Note: we get both ***bill*** and ***rights*** in the Title field of doc 3, but score it no higher.
- Should we give more weight to more hits?

Scoring: density-based

- Zone combinations relied on the <u>position</u> of terms in a doc – title, author etc.
- Obvious next: idea if a document talks about a topic *more,* then it is a better match
- This applies even when we only have a single query term.
- A query should then just specify terms that are relevant to the information need
  - Document relevant if it has a lot of the terms
  - Boolean syntax not required – more web-style

Counts vs. frequencies

- Consider again the ***ides of march*** query.
  - *Julius Caesar* has 5 occurrences of ***ides***
  - No other play has ***ides***
  - ***march*** occurs in over a dozen
  - All the plays contain ***of***
- By this scoring measure, the top-scoring play is likely to be the one with the most ***of***s

Term frequency *tf*

- Further, long docs are favored because they're more likely to contain query terms
- We can fix this to some extent by replacing each term count by term frequency
  - $tf_{t,d}$ = the count of term *t* in doc *d* divided by the total number of words in *d*.
- Good news – all *tf*'s for a doc add up to 1
  - Technically, the doc vector has unit L_1 norm
- But is raw *tf* the right measure?

Weighting term frequency: *tf*

- What is the relative importance of
  - 0 vs. 1 occurrence of a term in a doc
  - 1 vs. 2 occurrences
  - 2 vs. 3 occurrences …
- Unclear: while it seems that more is better, a lot isn't proportionally better than a few
  - Can just use raw *tf*
  - Another option commonly used in practice:

$$wf_{t,d} = \max(1 + \log count_{t,d}, 0)$$

Digression: terminology

- WARNING: In a lot of IR literature, “frequency” is used to mean “count”

Dot product matching

- Match is dot product of query and document

$$q \cdot d = \sum_i tf_{i,q} \times tf_{i,d}$$

- [Note: 0 if orthogonal (no words in common)]
- Rank by match
- Can use *wf* instead of *tf* in above dot product
- It still doesn’t consider:
  - Term scarcity in collection (***ides*** is rarer than ***of***)

Weighting should depend on the term overall

- Which of these tells you more about a doc?
  - 10 occurrences of *hernia*?
  - 10 occurrences of *the*?
- Would like to attenuate the weight of a common term
  - But what is "common"?
- Suggest looking at collection frequency (*cf*)
  - The total number of occurrence of the term in the entire collection of documents

Document frequency

- But document frequency (*df*) may be better:

Word	cf	df
try	10422	8760
insurance	10440	3997

- Document/collection frequency weighting is only possible in known (static) collection.
- So how do we make use of *df* ?

tf x idf term weights

- Assign a tf.idf weight to each term *i* in each document *d*

$$w_{i,d} = tf_{i,d} \times \log(n / df_i)$$

What is the wt of a term that occurs in all of the docs?

$$tf_{i,d} = \text{frequency of term } i \text{ in document } j$$

$$n = \text{total number of documents}$$

$$df_i = \text{the number of documents that contain term } i$$

- Increases with the number of occurrences *within* a doc
- Increases with the rarity of the term *across* the whole corpus

- See Kishore Papineni, NAACL 2, 2002 for theoretical justification

Real-valued term-document matrices

- Function (scaling) of count of a word in a document:
  - Bag of words model
  - Each is a vector in $\mathbb{R}^v$
  - Here log-scaled *tf.idf*

Note can be >1!

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	$\boxed{13.1}$	11.4	0.0	0.0	0.0	0.0
Brutus	3.0	8.3	0.0	1.0	0.0	0.0
Caesar	2.3	2.3	0.0	0.5	0.3	0.3
Calpurnia	0.0	11.2	0.0	0.0	0.0	0.0
Cleopatra	17.7	0.0	0.0	0.0	0.0	0.0
mercy	0.5	0.0	0.7	0.9	0.9	0.3
worser	1.2	0.0	0.6	0.6	0.6	0.0

Bag of words view of a doc

- Thus the doc
  - ***John is quicker than Mary***.

is indistinguishable from the doc

  - ***Mary is quicker than John***.

Documents as vectors

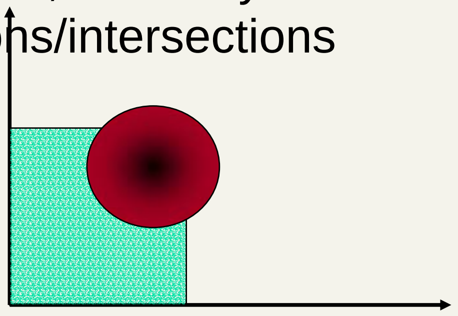
- Each doc *j* can now be viewed as a vector of *wf*×*idf* values, one component for each term
- So we have a vector space
  - terms are axes
  - docs live in this space
  - even with stemming, may have 20,000+ dimensions

Documents as vectors

- Each query *q* can be viewed as a vector in this space
- We need a notion of *proximity* between vectors
  - Cosine of angle between vectors: assign a score in [0,1] to each doc, with respect to *q*
- Allows score for a doc with respect to a doc!

Vectors and Boolean queries

- Vectors and Boolean queries really don't work together very well
- In the space of terms, vector proximity selects by <u>spheres</u>: e.g., all docs having cosine similarity $\geq$0.5 to the query
- Boolean queries on the other hand, select by (hyper-)rectangles and their unions/intersections
- Round peg - square hole



Vectors and phrases

- Phrases don’t fit naturally into the vector space world:
  - ***“tangerine trees” “marmalade skies”***
  - Positional indexes don’t capture tf/idf information for ***“tangerine trees”***
- Biword indexes (lecture 2) treat certain phrases as terms
  - For these, can pre-compute tf/idf.
- A hack: cannot expect end-user formulating queries to know what phrases are indexed

Vectors and wild cards

- How about the query ***tan* marm****?
  - Can we view this as a bag of words?
  - Thought: expand each wild-card into the matching set of dictionary terms.
- Danger – unlike the Boolean case, we now have *tf*s and *idf*s to deal with.
- Net – not a good idea.

Vector spaces and other operators

- Vector space queries are apt for no-syntax, bag-of-words queries
  - Clean metaphor for similar-document queries
- Not a good combination with Boolean, wild-card, positional query operators

Exercises

- How would you augment the inverted index built in lectures 1–2 to support cosine ranking computations?
- Walk through the steps of serving a query.
- *The math of the vector space model is quite straightforward, but being able to do cosine ranking efficiently at runtime is nontrivial*

Efficient cosine ranking

- Find the *k* docs in the corpus “nearest” to the query $\Rightarrow$ *k* largest query-doc cosines.
- Efficient ranking:
  - Computing a single cosine efficiently.
  - Choosing the *k* largest cosine values efficiently.
    - Can we do this without computing all *n* cosines?

Computing a single cosine

- For every term *i*, with each doc *j*, store term frequency tf_{ij}.
  - Some tradeoffs on whether to store term count, term weight, or weighted by idf_i.
- Accumulate component-wise sum

$$sim(\vec{d}_j, \vec{d}_k) = \sum_{i=1}^{m} w_{i,j} \times w_{i,k}$$

- If you’re indexing 5 billion documents (web search) an array of accumulators is infeasible

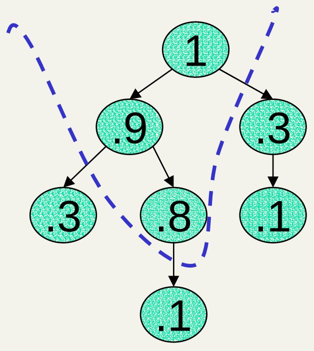
Ideas?

Computing the *k* largest cosines: selection vs. sorting

- Typically we want to retrieve the top *k* docs (in the cosine ranking for the query)
  - not totally order all docs in the corpus
  - can we pick off docs with *k* highest cosines?

Use heap for selecting top *k*

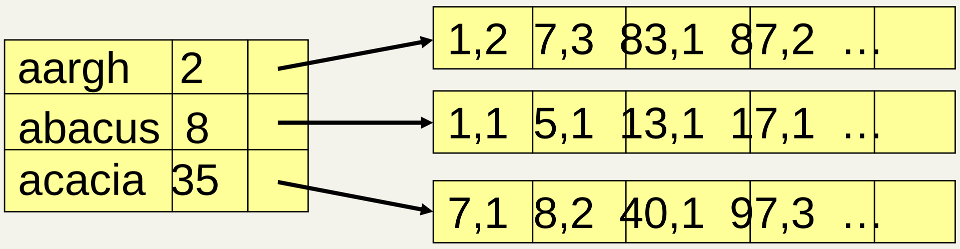
- Binary tree in which each node's value > values of children
- Takes *2n* operations to construct, then each of *k* log *n* "winners" read off in 2log *n* steps.
- For *n*=1M, *k*=100, this is about 10% of the cost of sorting.



Bottleneck

- Still need to first compute cosines from query to each of n docs → several seconds for n = 1M.
- Can select from only non-zero cosines
  - Need union of postings lists accumulators (<<1M): on the query ***aargh abacus*** would only do accumulators 1,5,7,13,17,83,87 (below).

Why do skip pointers help?



Removing bottlenecks

- Can further limit to documents with non-zero cosines on rare (high idf) words
- Enforce conjunctive search (a la Google): non-zero cosines on *all* words in query
  - Get # accumulators down to {min of postings lists sizes}
- But still potentially expensive
  - Sometimes have to fall back to (expensive) soft-conjunctive search:
  - If no docs match a 4-term query, look for 3-term subsets, etc.

Can we avoid this?

- Yes, but may occasionally get an answer wrong
  - a doc *not* in the top *k* may creep into the answer.

Term-wise candidates

- Preprocess: Pre-compute, for each term, its *m* nearest docs.
  - (Treat each term as a 1-term query.)
  - lots of preprocessing.
  - Result: “preferred list” for each term.
- Search:
  - For a *t*-term query, take the union of their *t* preferred lists – call this set *S,* where $|S| \leq mt$.
  - Compute cosines from the query to only the docs in *S*, and choose top *k*.

Need to pick *m>k* to work well empirically.

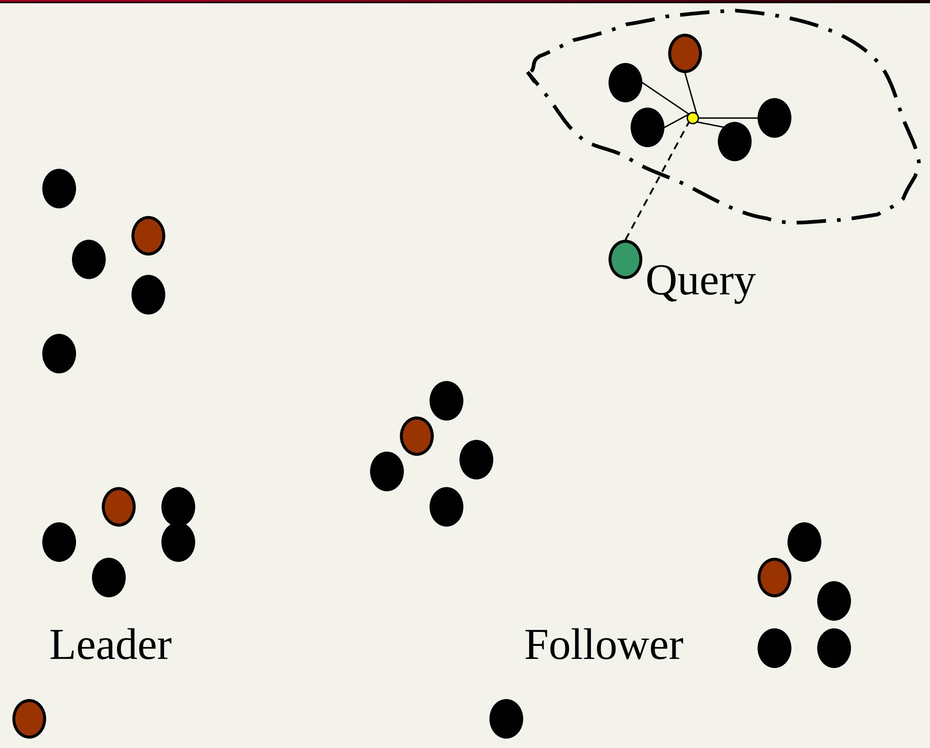
Exercises

- Fill in the details of the calculation:
  - Which docs go into the preferred list for a term?
- Devise a small example where this method gives an incorrect ranking.

Cluster pruning

- First run a pre-processing phase:
  - pick √n *docs* at random: call these *leaders*
  - For each other doc, pre-compute nearest leader
    - Docs attached to a leader: its *followers;*
    - Likely: each leader has ~ √*n* followers.
- Process a query as follows:
  - Given query *Q*, find its nearest *leader L.*
  - Seek *k* nearest docs from among *L*'s followers.

Visualization



Why use random sampling

- Fast
- Leaders reflect data distribution

General variants

- Have each follower attached to *a*=3 (say) nearest leaders.
- From query, find *b*=4 (say) nearest leaders and their followers.
- Can recur on leader/follower construction.

Exercises

- To find the nearest leader in step 1, how many cosine computations do we do?
  - Why did we have $\sqrt{n}$ in the first place?
- What is the effect of the constants *a*,*b* on the previous slide?
- Devise an example where this is *likely to* fail – i.e., we miss one of the *k* nearest docs.
  - *Likely* under random sampling.

Measures for a search engine

- How fast does it index
  - Number of documents/hour
  - (Average document size)
- How fast does it search
  - Latency as a function of index size
- Expressiveness of query language
  - Speed on complex queries

Measures for a search engine

- All of the preceding criteria are *measurable*: we can quantify speed/size; we can make expressiveness precise
- The key measure: user happiness
  - What is this?
  - Speed of response/size of index are factors
  - But blindingly fast, useless answers won’t make a user happy
- Need a way of quantifying user happiness

Measuring user happiness

- Issue: who is the user we are trying to make happy?
  - Depends on the setting
- Web engine: user finds what they want and return to the engine
  - Can measure rate of return users
- eCommerce site: user finds what they want and make a purchase
  - Is it the end-user, or the eCommerce site, whose happiness we measure?
  - Measure time to purchase, or fraction of searchers who become buyers?

Measuring user happiness

- Enterprise (company/govt/academic): Care about "user productivity"
  - How much time do my users save when looking for information?
  - Many other criteria having to do with breadth of access, secure access … more later

Happiness: elusive to measure

- Commonest proxy: *relevance* of search results
- But how do you measure relevance?
- Will detail a methodology here, then examine its issues
- Requires 3 elements:
  1. A benchmark document collection
  2. A benchmark suite of queries
  3. A binary assessment of either Relevant or Irrelevant for each query-doc pair

Evaluating an IR system

- Note: **information need** is translated into a **query**
- Relevance is assessed relative to the **information need** *not* the **query**

Standard relevance benchmarks

- TREC - National Institute of Standards and Testing (NIST) has run large IR testbed for many years
- Reuters and other benchmark doc collections used
- "Retrieval tasks" specified
  - sometimes as queries
- Human experts mark, for each query and for each doc, <u>Relevant</u> or <u>Irrelevant</u>
  - or at least for subset of docs that some system returned for that query

Precision and Recall

- **Precision**: fraction of retrieved docs that are relevant = P(relevant|retrieved)
- **Recall**: fraction of relevant docs that are retrieved = P(retrieved|relevant)

	Relevant	Not Relevant
Retrieved	tp	fp
Not Retrieved	fn	tn

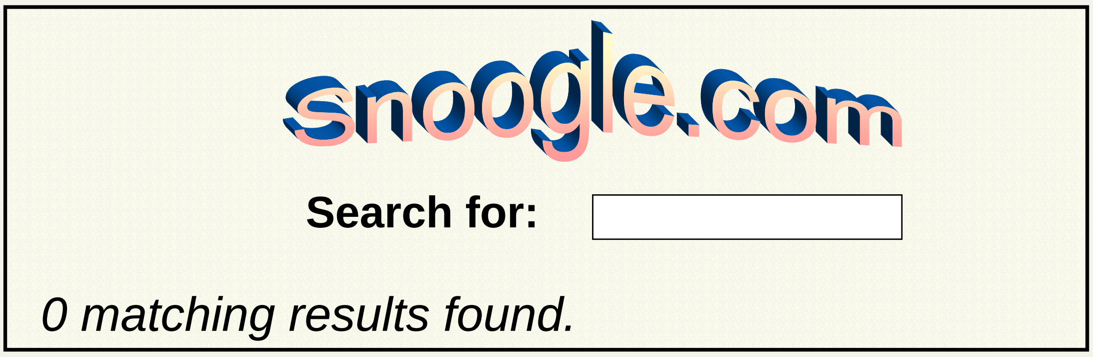
- Precision P = tp/(tp + fp)
- Recall R = tp/(tp + fn)

Accuracy

- Given a query an engine classifies each doc as “Relevant” or “Irrelevant”.
- Accuracy of an engine: the fraction of these classifications that is correct.

Why not just use accuracy?

- How to build a 99.9999% accurate search engine on a low budget….



snoogle.com

Search for:

0 matching results found.

- People doing information retrieval want to find *something* and have a certain tolerance for junk.

Precision/Recall

- Can get high recall (but low precision) by retrieving all docs for all queries!
- Recall is a non-decreasing function of the number of docs retrieved
  - Precision usually decreases (in a good system)

Difficulties in using precision/recall

- Should average over large corpus/query ensembles
- Need human relevance assessments
  - People aren't reliable assessors
- Assessments have to be binary
  - Nuanced assessments?
- Heavily skewed by corpus/authorship
  - Results may not translate from one domain to another

A combined measure: *F*

- Combined measure that assesses this tradeoff is F measure (weighted harmonic mean):

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

- People usually use balanced F_1 measure
  - i.e., with $\beta = 1$ or $\alpha = ½$
- Harmonic mean is conservative average
  - See CJ van Rijsbergen, *Information Retrieval*

Ranked results

- Evaluation of ranked results:
  - You can return any number of results
  - By taking various numbers of returned documents (levels of recall), you can produce a *precision-recall curve*

Precision-recall curves

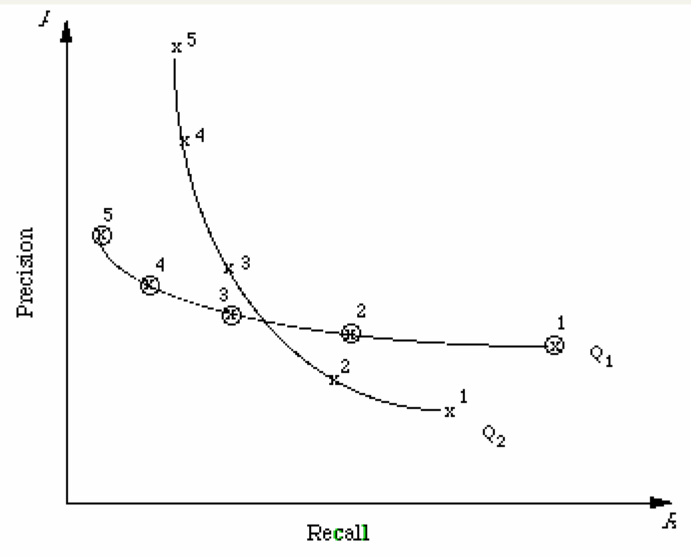
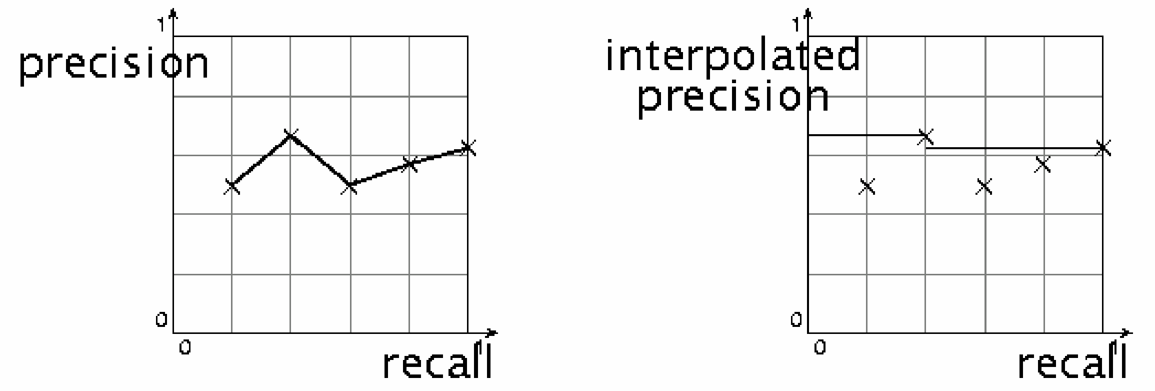


Figure 7.1. The precision-recall curves for two queries. The ordinals indicate the values of the control parameter λ.

Interpolated precision

- If you can increase precision by increasing recall, then you should get to count that…



Evaluation

- There are various other measures
  - Precision at fixed recall
    - Perhaps most appropriate for web search: all people want are good matches on the first one or two results pages
  - 11-point interpolated average precision
    - The standard measure in the TREC competitions: you take the precision at 11 levels of recall varying from 0 to 1 by tenths of the documents, using interpolation (the value for 0 is always interpolated!), and average them

Critique of Pure Relevance

- Relevance vs Marginal Relevance
  - A document can be redundant even if it is highly relevant
  - Duplicates
  - The same information from different sources
  - Marginal relevance is a better measure of utility for the user.
- Using facts/entities as evaluation units more directly measures true relevance.
- But harder to create evaluation set

Can we avoid human judgements?

- Not really
- Makes experimental work hard
  - Especially on a large scale
- In some very specific settings, can use proxies
- Example below, approximate vector space retrieval

Approximate vector retrieval

- Given *n* document vectors and a query, find the *k* doc vectors closest to the query.
- Exact retrieval – we know of no better way than to compute cosines from the query to every doc
- Approximate retrieval schemes – such as cluster pruning in lecture 6
- Given such an approximate retrieval scheme, how do we measure its goodness?

Approximate vector retrieval

- Let *G(q)* be the “ground truth” of the actual *k* closest docs on query *q*
- Let *A(q)* be the *k* docs returned by approximate algorithm *A* on query *q*
- For precision and recall we would measure $A(q) \cap G(q)$
  - Is this the right measure?

Alternative proposal

- Focus instead on how *A(q)* compares to *G(q).*
- Goodness can be measured here in cosine proximity to *q:* we sum up $q \bullet d$ over $d \in A(q)$.
- Compare this to the sum of $q \bullet d$ over $d \in G(q)$.
  - Yields a measure of the relative “goodness” of *A* vis-à-vis *G*.
  - Thus *A* may be 90% “as good as” the ground-truth *G*, without finding 90% of the docs in *G*.
  - For scored retrieval, this may be acceptable:
  - Most web engines don’t always return the same answers for a given query.

Resources for this lecture

- MG 4.5, 4.4
- New Retrieval Approaches Using SMART: TREC 4
  Gerard Salton and Chris Buckley. Improving Retrieval Performance by Relevance Feedback. Journal of the American Society for Information Science, 41(4):288-297, 1990.