

Relevance Ranking

CS635

Soumen Chakrabarti

(+ many slides from MRS book)

Ranked retrieval

- Thus far, our queries have all been Boolean.
 - Documents either match or don't.
- Works for expert users with precise understanding of their needs and the collection.
 - Also good for applications: Applications can easily consume 1000s of results.
- Not good for the majority of users.
 - Most users incapable of writing Boolean queries (or they are, but they think it's too much work).
 - Most users don't want to wade through 1000s of results.
 - This is particularly true of web search.

The need for relevance ranking

- Boolean search returns a *set* of docs without any scores or ranking
- Average Web query is 2–3 words long
- Bits of entropy in query ≈ 10 which ‘addresses’ 1024 items
- Meanwhile corpus has > 10 G docs
- Typical query results in > 10 M hits

Problem with Boolean search: feast or famine

- Boolean queries often result in either too few (=0) or too many (1000s) results.
- Query 1: “*windows dlink 650*” → 200,000 hits
- Query 2: “*windows dlink 650 no card found*” → 0 hits
- It takes a lot of skill to come up with a query that produces a manageable number of hits.
 - AND gives too few; OR gives too many

Ranked retrieval models

- Rather than a set of documents satisfying a query expression, in ranked retrieval, the system returns an ordering over the (top) documents in the collection for a query
- Free text queries: Rather than a query language of operators and expressions, the user's query is just one or more words in a human language
- In principle, there are two separate choices here, but in practice, ranked retrieval has normally been associated with free text queries and vice versa

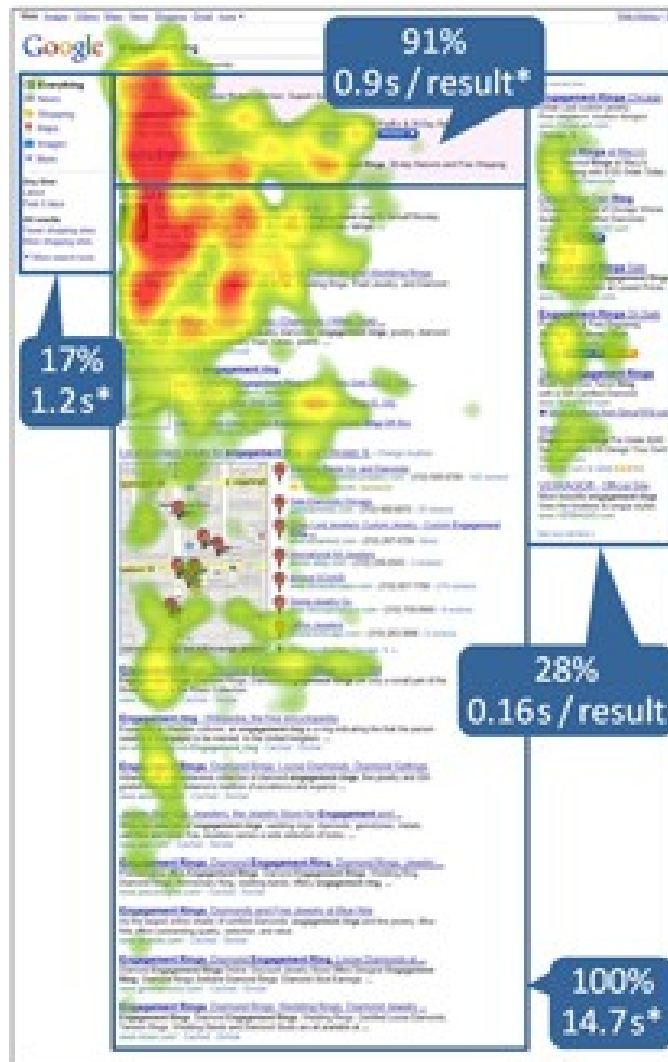
Scoring as the basis of ranked retrieval

- We wish to return in order the documents most likely to be useful to the searcher
- How can we rank-order the documents in the collection with respect to a query?
- Assign a score – say in $[0, 1]$ – to each document
- This score measures how well document and query “match”.

Feast or famine: not a problem in ranked retrieval

- When a system produces a ranked result set, large result sets are not an issue
 - Indeed, the size of the result set is not an issue
 - We just show the top k (≈ 10) results
 - We don't overwhelm the user
 - *Premise: the ranking algorithm works*
- How do users react to ranked responses from search engines?

Eye tracking study on search results



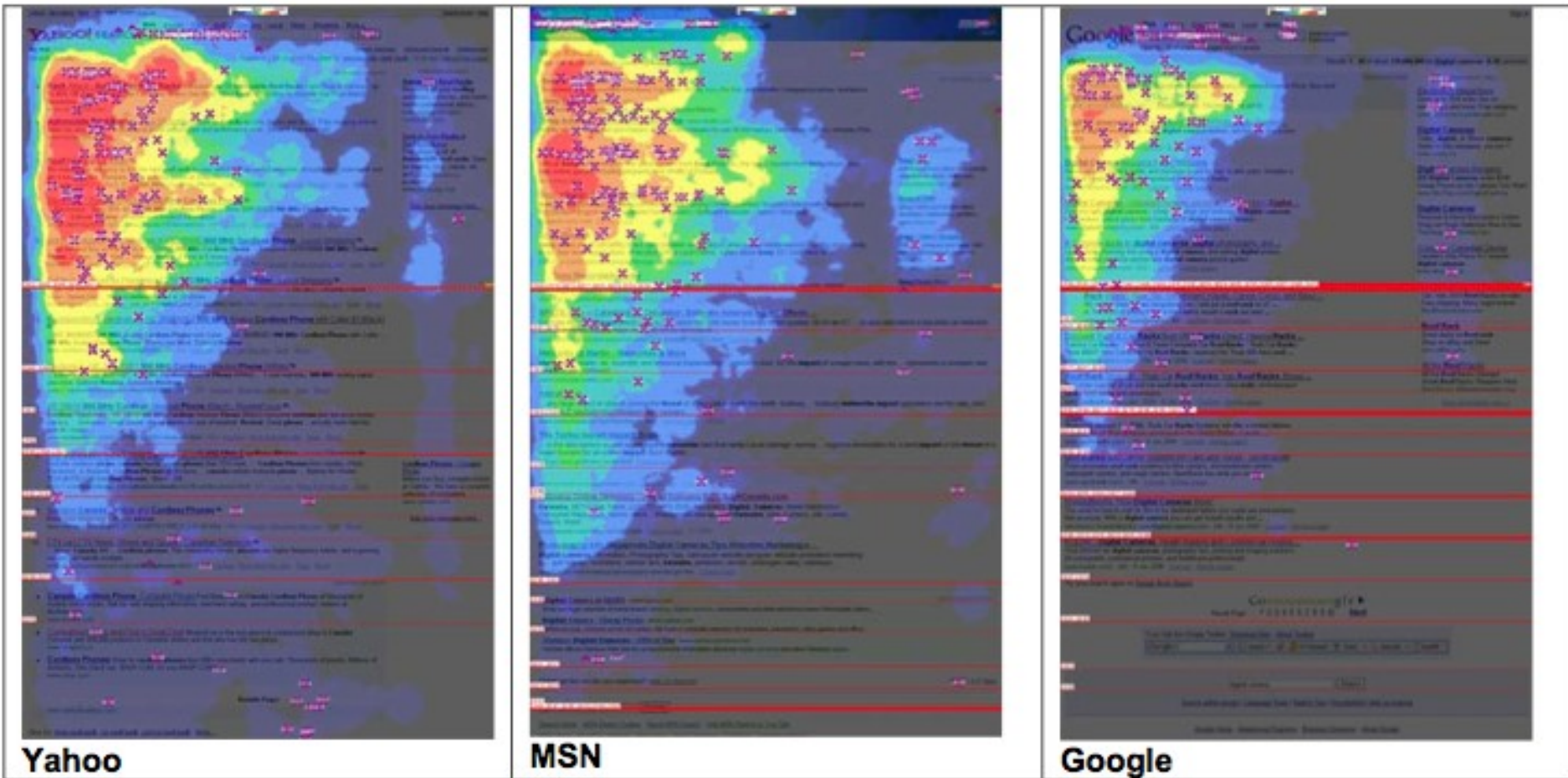
Google =



= Bing

Heatmaps showing the aggregate gaze time of all 24 participants on Google (left) and Bing (right) for one of the transactional tasks. The red color indicates areas that received the most total gaze time (4.5 seconds and above). Each callout includes the percentage of participants who looked at the area and the time (in seconds) they spent looking there. The numerical data are an average across all four tasks. Asterisks indicate values that were significantly different between Google and Bing at $\alpha = .1$.

Yahoo, MSN, Google



<https://www.forbes.com/sites/roberthof/2015/03/03/do-you-google-new-eye-tracking-study-reveals-huge-change>

Eye tracking summary

- Small differences in gaze at ad panels, but
- Overall, users spend a lot of time gazing at “organic” results
- Mostly scanning down from rank 1 in search of search satisfaction
- Spending more time inspecting top ranks
- There are very sophisticated models of summary inspection, skip, click, satisfaction, backoff, etc.
- For now, the clear first principle is: **present results most relevant first**
- **How to measure relevance?**

Forms of supervision

Regression: for each doc, relevance score
(unrealistic to collect absolute scores)

Ordinal regression: discretize into K-point
relevance scale (somewhat more realistic)

Complete rank order: Total order on docs but
no scores (still quite unrealistic)

Prefix of rank order: Slightly more realistic

Pairwise preferences: (possibly inconsistent)
partial order between docs “u less preferred
than v” (inferred from click-log and eye-
tracking)

Loss and reward

L1 or L2: If y, \hat{y} are gold, predicted scores, can use $|\hat{y} - y|^2$ or $|\hat{y} - y|$

Let T_k, \widehat{T}_k be gold and system top- k sets

Total gold score of $T_k = \sum_{v \in T_k} y(v)$

Total gold (note, not system)

score of $\widehat{T}_k = \sum_{v \in \widehat{T}_k} y(v)$

Relative aggregated goodness (RAG):

$$\frac{\sum_{v \in \widehat{T}_k} y(v)}{\sum_{v \in T_k} y(v)} \in [0,1]$$

Useful to give credit to “good enough” hits

Losses and rewards, cont'd

Pair preference violation: If $u \prec v$ but $y(u) > y(v)$, this is a violated pair. Count the number of pair violations as loss.

Rank correlation: Order docs by decreasing y and compute rank correlation with (unrealistic) ground truth ranking

Prefix rank correlation:

m pairs v, w from $T_k \cup \widehat{T}_k$

c concordant pairs v, w where

$$(y(v) - y(w))(\hat{y}(v) - \hat{y}(w)) > 0$$

d discordant pairs where above quantity < 0

Losses and rewards, cont'd

Prefix rank correlation:

a approximate ties where $\hat{y}(v) = \hat{y}(w)$

e exact ties where $y(v) = y(w)$

Kendall's tau is defined as

$$\frac{c - d}{\sqrt{(m - e)(m - a)}} \in [-1, +1]$$

What matters most in practice is the density of relevant results at the top ranks

Binary relevance

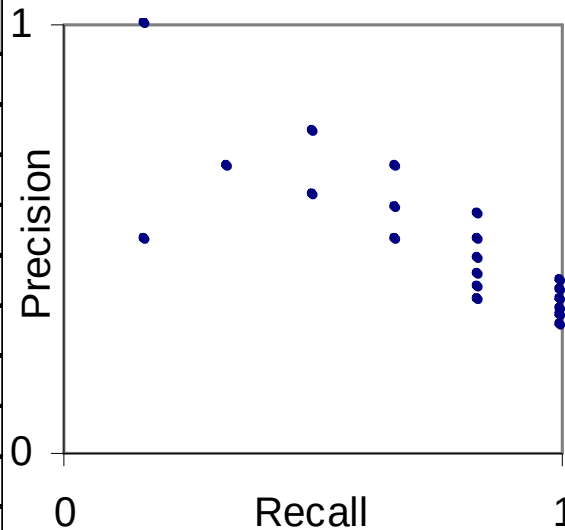
- Suppose each doc in corpus *is known as* relevant or irrelevant (big assumption!)
- Ideal search engine should rank all relevant documents above any irrelevant document
- In practice, an imperfect search engine may mix them up a little
- If we went far enough down the list, we would collect all relevant docs (“total recall”)
- But many collected docs would be irrelevant
- *Recall vs. precision* tradeoff

Recall and precision

- Suppose for a query there are R relevant documents
- We inspect the response list up to rank k
- $\text{Precision@}k$ = fraction of docs within rank k that are relevant
 - Rest are false positives
- $\text{Recall@}k$ = fraction of R relevant docs that is included in top k ranks
 - Rest are false negatives
- “Silent and correct” vs. “verbose and wrong”

Recall-precision tradeoff

k	r_k
1	1
2	
3	1
4	1
5	
6	1
7	
8	
9	1
10	
11	
12	
13	
14	
15	1
16	
17	
18	
19	
20	

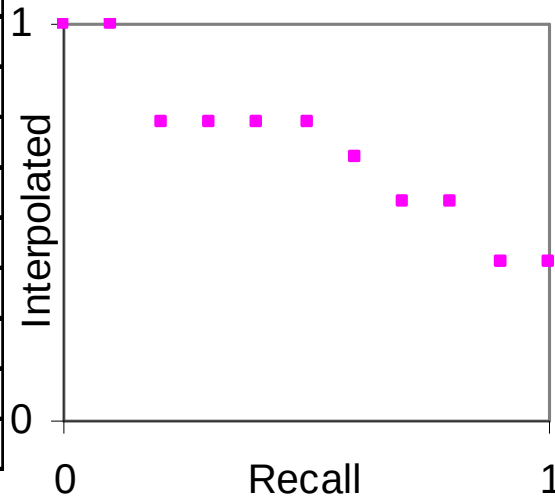


■ As recall is increased, precision generally (but not always) decreases

■ Interpolated precision fixes this anomaly

■ Many ways to reduce to single number

- R, P, F1
- Break-even point
- MRR
- AUC
- MAP
- NDCG



“Nice thing about standards is there’re so many to choose from!

Mean reciprocal rank (MRR)

- Query set Q ; one query q ; first relevant doc at rank r_q


$$\text{MRR} = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{r_q}$$

- Sometimes truncated at fixed “patience runs out” rank k (often $k=10$)

$$\text{MRR} = \frac{1}{|Q|} \sum_{q \in Q, r_q \leq k} \frac{1}{r_q}$$

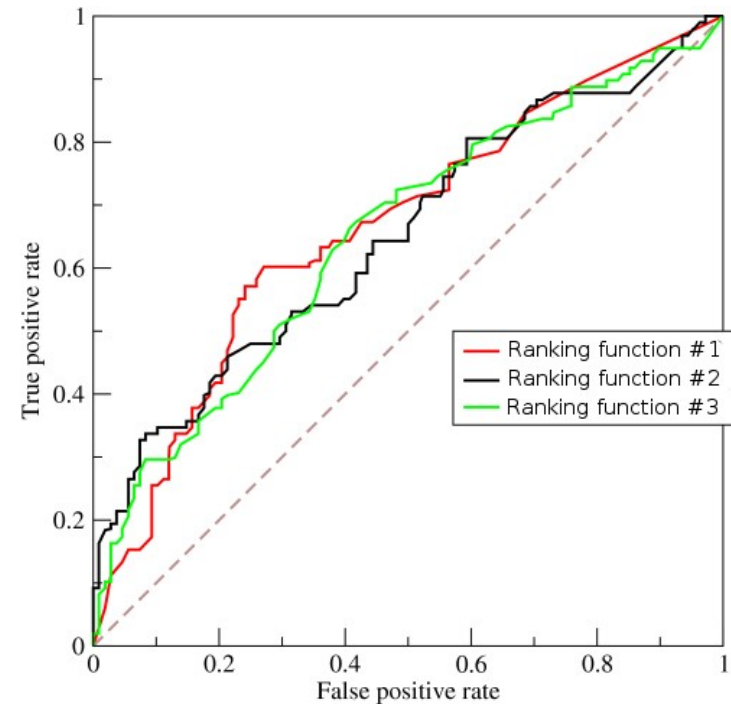
- Dropping from 1 to 2 as bad as 2 to ∞
- “Mean” reciprocal rank
- Good for navigational queries
 - E.g. jet airways

ROC curve

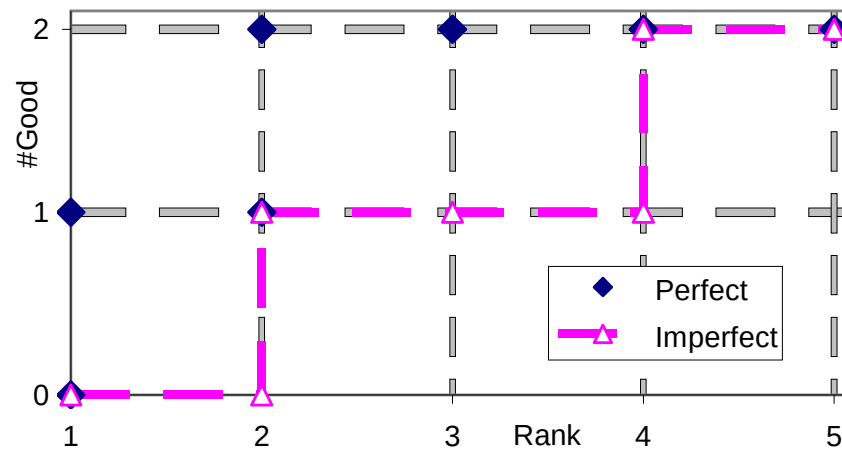
- Shorthand: good=relevant, bad=irrelevant
- Fix query, suppose n^+ good, n^- bad docs
- Algorithm sorts all (n^++n^-) docs 
- For $i = 0, 1, 2, \dots, (n^++n^-)$
 - Suppose algorithm marks top i as relevant
 - Rest as irrelevant
 - In top i , say n_i^+ actually good, $n_i^- = i - n_i^+$ bad
 - true positive@ $i = (n_i^+ / n^+)$
 - false positive@ $i = (n_i^- / n^-)$
- Plot $y = \text{true positive}$ vs. $x = \text{false positive}$

Area under ROC curve (AUC)

- Called Receiver Operating Characteristic curve
- Suppose ranking is random
 - Will get roughly diagonal line
- Good ranking functions
 - Will get very high true positive at very small false positive rates
 - Large lift above diagonal
- Measure area under the curve
 - $\frac{1}{2} \Rightarrow$ effectively random
 - Close to 1 \Rightarrow good ranking algorithm



Another AUC example



- Good = 1, bad = 0; two good, three bad docs
- For perfect ordering 1,1,0,0,0, plot passes through (1,0), (1,1), (2,1), (2,2), (3,2), (4,2), (5,2)
- For imperfect ordering 0,1,0,1,0, plot passes through (1,0), (2,0), (2,1), (3,1), (4,1), (4,2), (5,2)
- Area between two plots related to number of **discordant pairs** Q

Concordant and discordant pairs

- ~~Relevant~~ **Relevant** docs for a given query
- Search engine creates ranking r_{engine} which lists them at ranks $1 \leq p_1 < p_2 < \dots < p_R$
- Ideal system creates ranking r_{ideal} that lists all good docs before any bad doc
- But keeps relative order with good and bad unaffected

$$r_{\text{engine}} = d_1^+, d_2^-, d_3^+, d_4^+, d_5^-, d_6^-, d_7^+, d_8^-$$

$$r_{\text{ideal}} = d_1^+, d_3^+, d_4^+, d_7^+; d_2^-, d_5^-, d_6^-, d_8^-$$

- Say Q discordant pairs between these two

Relating ranks p_i and discordant pairs Q

- Account for Q as follows: First consider the relevant document at position p_1 in r_{engine} . Because it has been pushed out from position 1 to position p_1 , the number of inversions introduced is $p_1 - 1$.
- For the document at position p_2 in r_{engine} , the number of inversions introduced is $p_2 - 1 - 1$, the last “-1” thanks to having the first relevant document ahead of it.
- Summing up, we get

$$\sum_{i=1}^R p_i - 1 - (i - 1) = Q, \quad \text{or}$$

$$\sum_{i=1}^R p_i = Q + \sum_{i=1}^R i = Q + \frac{R(R+1)}{2} = Q + \binom{R+1}{2}.$$

Average precision

- “Informational” queries
- High-ranking relevant hits matter a lot
- But user continues exploring (with increasing satiation or fatigue)
- Accrues reward for each additional relevant doc, but reward decreases with rank (fatigue)
- Fix one query, suppose there are R relevant documents at ranks $1 \leq p_1 < p_2 < \dots < p_R$
- Precision at rank p_i is (i/p_i)
- Over all relevant ranks:
$$\text{AvgPrec} = \frac{1}{R} \sum_{i=1}^R \frac{i}{p_i}$$

Bounding average precision given Q

- If Q is small, average precision must be large
- Minimize average precision subject to Q
- Relax integer rank and inequalities among p_i

$$\mathcal{L}(p_1, \dots, p_R; \lambda) = \frac{1}{R} \sum_{i=1}^R \frac{i}{p_i} + \lambda \left(\sum_{i=1}^R p_i - Q - \binom{R+1}{2} \right)$$

$$\therefore \frac{\partial \mathcal{L}}{\partial p_i} = -\frac{i}{R p_i^2} + \lambda \stackrel{\text{set}}{=} 0 \quad \text{to get} \quad p_i^* = \sqrt{\frac{i}{R\lambda}}.$$

- From which we get

$$\text{AvgPrec}(r_{\text{engine}}, r_{\text{ideal}}) \geq \frac{\left(\sum_{i=1}^R \sqrt{i} \right)^2}{R \left(Q + \binom{R+1}{2} \right)}$$

Normalized discounted cumulative gain

- Fix query q
- Relevance level of document ranked j wrt q be $r_q(j)$
- $r_q(j)=0$ means totally irrelevant
- Response list is inspected up to rank L
- Discounted cumulative gain for query q is

$$\text{NDCG}_q = \underbrace{Z_q}_{\text{normalized}} \sum_{j=1}^L \frac{\underbrace{2^{r_q(j)} - 1}_{\text{gain}}}{\underbrace{\log(1 + j)}_{\text{rank discount}}}$$

- Z_q is a normalization factor that ensures the perfect ordering has $\text{NDCG}_q = 1$
- Overall NDCG is average of NDCG_q over all q
- No notion of recall, only precision at low ranks

This is all fine, but...

- ...how does a search and ranking system optimize one or more of these criteria?
- Early IR systems—time-tested heuristics
 - Coming up next
- Later, some probabilistic justification
 - Will visit when we study corpus models
- Then, direct optimization based on machine learning
 - After we cover a bit of ground on basic ML

The “vector space model”

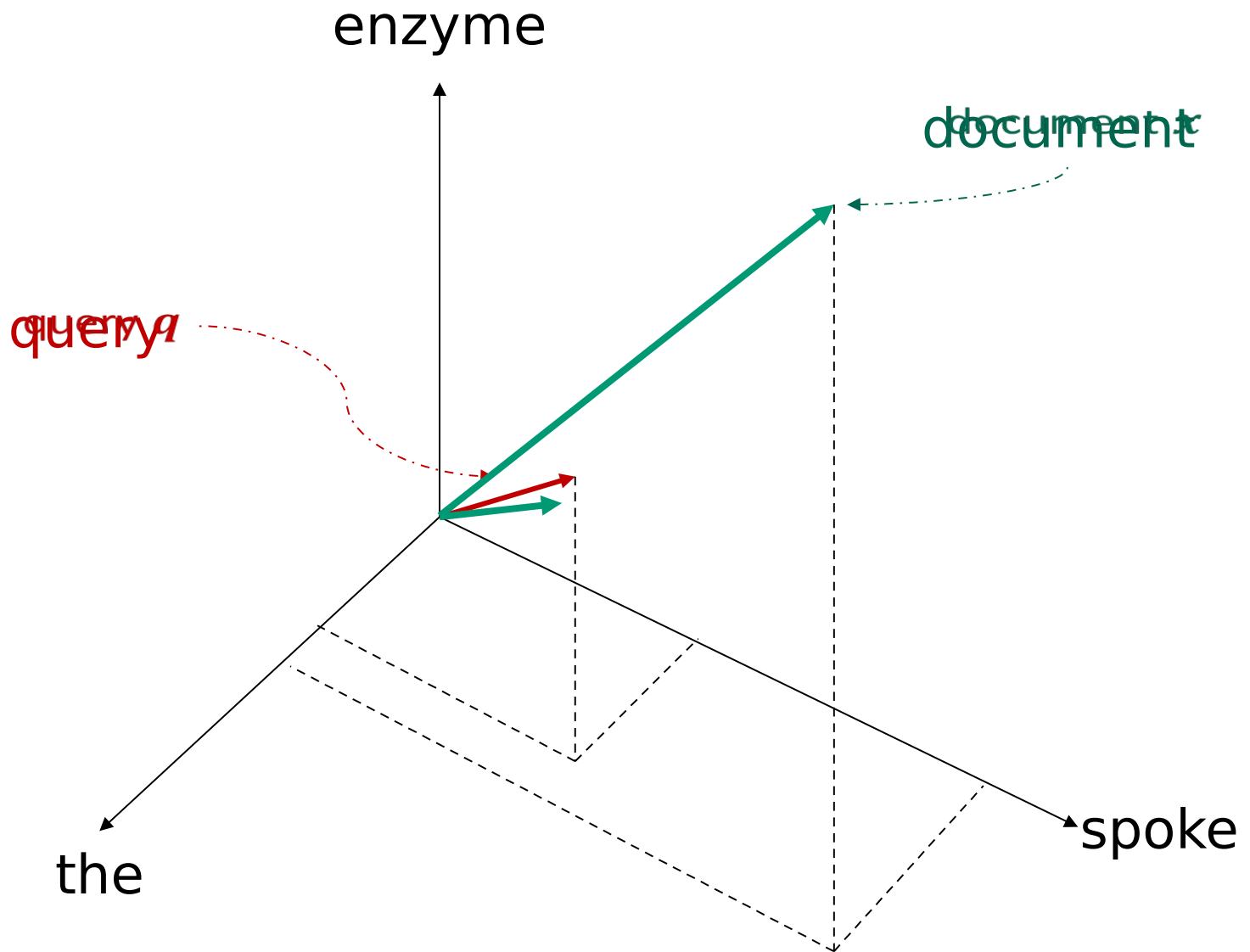
- Let $w = 1, \dots, W$ define the corpus vocab
- Let $\mathbf{x} \in \mathbb{R}^W$ denote a vector representing a document (i.e., one axis per vocab word)
- Query $\mathbf{q} \in \mathbb{R}^W$ is in the same space
- Similarity between \mathbf{q} and \mathbf{x} is defined as

$$\cos(\mathbf{q}, \mathbf{x}) = \frac{\mathbf{q} \cdot \mathbf{x}}{\|\mathbf{q}\|_2 \|\mathbf{x}\|_2}$$

Why length
normalization

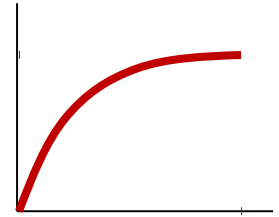
- In which component x_w depends on?
 - **Term frequency**: number of times w occurs in x
 - **Utility** of word w as a search term

Comparing query to docs



TF×IDF representation

- Use word rarity as a surrogate
 - ‘The’ appears in almost all docs
 - ‘Calculus’ is rare and possibly useful for search
- Rarity → inverse document frequency (IDF)
- “Squashing functions”
 - $TF(x, w) = \log(1 + \#(x, w))$
 - Here $\#(x, w)$ is the raw word count in doc
 - $IDF(w) = \log(1 + \#_{\max} / \#(w))$
 - $\#(w)$ is #docs in which w occurs at least once
 - $\#_{\max} = \max_w \#(w)$



Property of
IDF?

TFxIDF representation

- Multiply TF and IDF together

$$\ell_w(\mathbf{x}) = \text{TF}(\mathbf{x}, w) \cdot \text{IDF}(w)$$

- Raw 'length' of doc

- Normalize to final vector components
- $$L(\mathbf{x}) = \sqrt{\sum_w \ell_w(\mathbf{x})}$$

- Normalize to final vector components
- Queries rarely repeat words, only TF?

- Double-include IDF for doc and query?
- Queries rarely repeat words, only TF?
- Double-include IDF for doc and query?

Query processing using inverted lists

How exactly does TFIDF retrieval happen using inverted lists?

- Interested in top-k documents only

- Speed up using various pruning and termination heuristics

A general framework started by Fagin et al.

- Worst case (pessimistic) guarantees

- Depends on posting list ordering

Later, probabilistic bounds

- Incorrect ranking with small probability

Basic TFIDF vector space scoring

(Assume no phrases or Boolean clauses)

Init empty accumulator map: $\text{score}[\text{docid}]$

In decreasing IDF order of query words t

Scan posting list for word to get $(x, \text{TF}(x, w))$

$\text{score}[x] += \text{TF}(x, w) * \text{IDF}(w)$

Divide each $\text{score}[x]$ by (function of) length of x
(implement cosine)

Many score accumulators, need top-k docids

Partial sort (how?) and report top-k

Where is time spent?

For queries with relatively rare terms

- Accumulator management

- Sparse or dense map?

For queries with some frequent terms

- Bit processing for decompressing postings

- Arithmetic to update accumulators

- Wasted effort in computing scores to throw away

You can't really afford (updating) a billion accumulators for 300 million queries a day

- Be sloppy when no one is looking

Score/impact ordering

- Thus far we have ordered postings by document ID...
- ... which are assigned arbitrarily
- For any query we must scan to the end of posting lists
- Because the best doc may be at the end
- Instead we can order postings by decreasing **impact**: how much the doc's score can be affected by that term
- Docs in different order in different lists \nrightarrow

Quit and continue heuristics

Quit: Once $|\text{score}|$ exceeds some size just quit and report answers

Continue: Stop creating new score accumulators but continue processing and accumulating scores for remaining words

Critical to process in decreasing IDF order

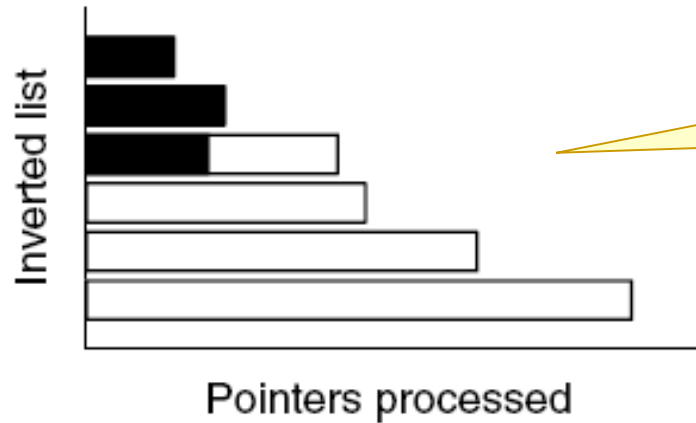
Quit is crude but continue is reasonable

Prescale by document length to avoid final division (that can upset many ranks)

Could destroy compressibility of index

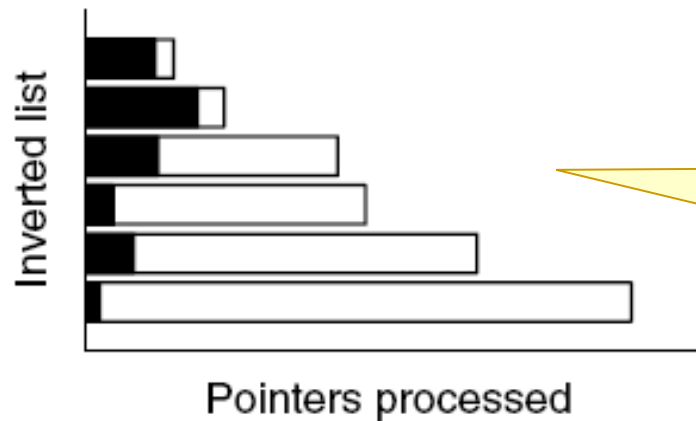
Limited precision doc lengths (6 bits adequate)

Quit, continue, prune



(a)

IDF-ordered quit
heuristic



(b)

Impact-ordered
continuation and/or
pruning heuristic

Term impact

Worth/impact of updating accumulator of x in response to word w depends on

Term frequency $TF(x,w)$

Inverse document frequency $IDF(w)$

(Function of) document length $L(x)$

Store $TF(x,w) IDF(w) / L(x)$

= “term impact”

Quantize to b bits

Uniform/geometric?

Effect on retrieval
accuracy and
index size

TREC,
precision
at rank 10

TREC

Web

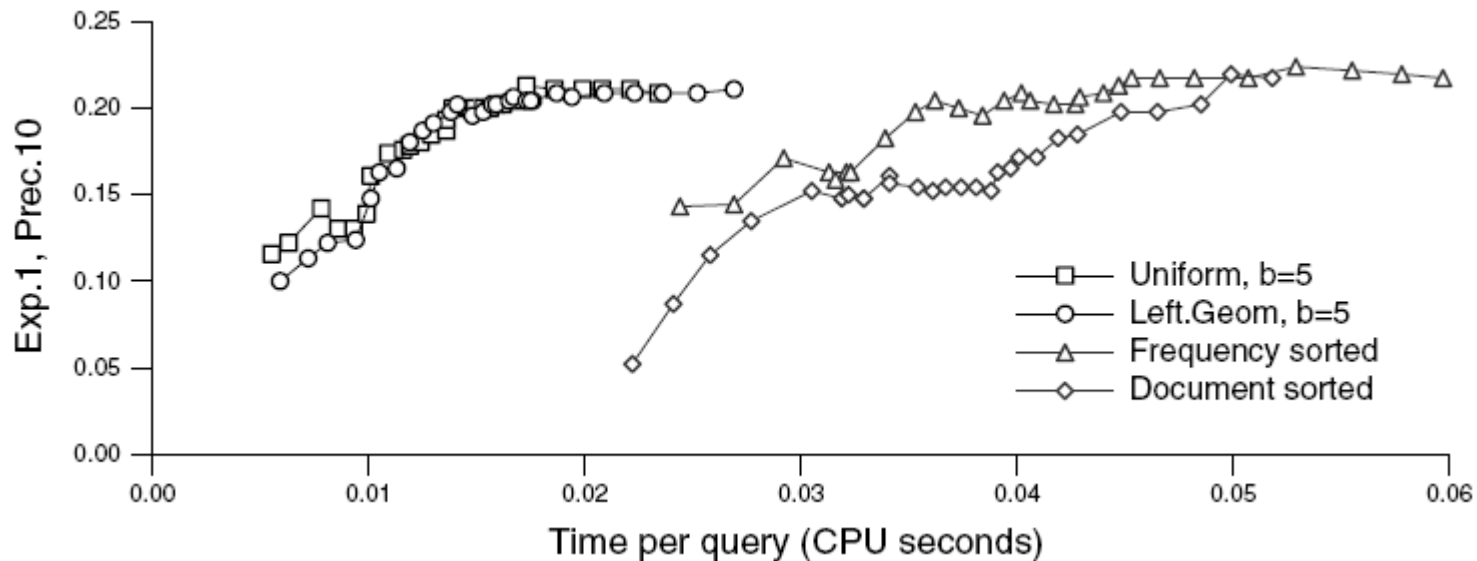
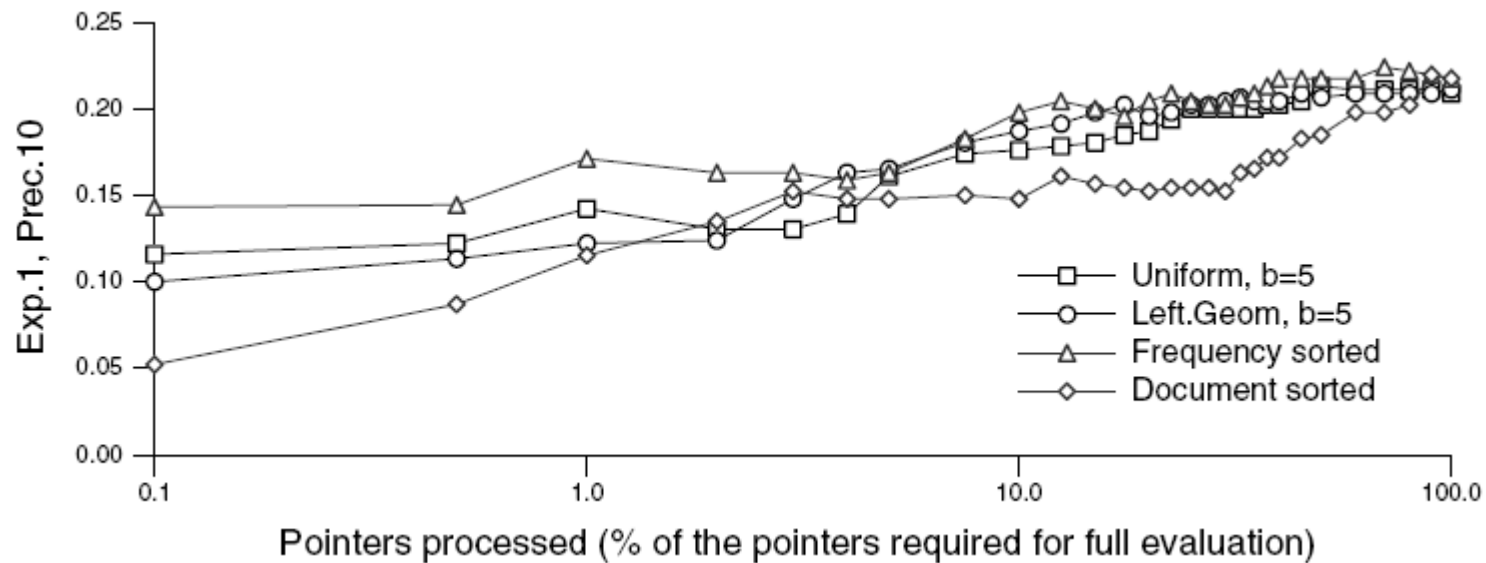
%corpus

9.4
7.8
7.0
6.3
5.6
5.1

6.0
5.0
4.5
4.0
3.6
3.3

	<i>Left.Geom</i>	<i>Uniform</i>
$b = 16$.2174	.2174
$b = 12$.2174	.2174
$b = 10$.2174	.2174
$b = 8$.2196	.2174
$b = 6$.2130	.2109
$b = 5$.2109	.2087
$b = 4$.1891	.1935
$b = 3$.1674	.1891
$b = 2$.1609	.1174

Term impact order -- results



Formalize as branch and bound

Cartesian space $D_1 \times D_2 \times \dots \times D_m$

m-dimensional data points and queries

Similarity function $s_i: D_i \times D_i \rightarrow [0,1]$ for each dimension

Global similarity = $\text{aggr}_i s_i(q,d)$

Sorted access to each dimension in order of s_i as in impact-sorted IR

We are lucky that queries are composed of terms

What if they were term pairs? Small graphs?

Keep ub and lb on scores of candidate points

Generic pseudocode

i ranges
over dims

Any future
record will
have a lower
score

Explored
dimensions
of item d

Eviction

Note: need to
fully evaluate
winner scores

TA-sorted:

top-k := {dummy₁, ..., dummy_k}; // with $s(\text{dummy}_v) = 0$
min-k := 0;

candidates := \emptyset ;

scan all lists L_i ($i = 1..m$) in parallel:

// e.g., round-robin or merged in descending order of s_i values

consider item d at position pos_i in L_i ;

if $d \notin \text{candidates}$ then

candidates := candidates \cup {d};

E(d) := {i};

high_i := $s_i(q_i, d)$; // current score in L_i

E(d) := E(d) \cup {i};

bestscore(d) := $\text{aggr} \{ \text{aggr} \{ s_v(q_v, d) \mid v \in E(d) \},$
 $\text{aggr} \{ \text{high}_v \mid v \notin E(d) \} \}$

worstscore(d) := $\text{aggr} \{ s_v(q_v, d) \mid v \in E(d) \}$;

if worstscore(d) > min-k then

if $d \notin \text{top-k}$ then

remove $\text{argmin}_{d'} \{ \text{worstscore}(d') \mid d' \in \text{top-k} \}$ from top-k;

candidates := candidates \cup {d'};

add d to top-k

min-k := $\min \{ \text{worstscore}(d') \mid d' \in \text{top-k} \}$;

if bestscore(d) \leq min-k then candidates := candidates - {d};

threshold := $\max \{ \text{bestscore}(d') \mid d' \in \text{candidates} \}$;

if threshold \leq min-k then exit;

Upper bound

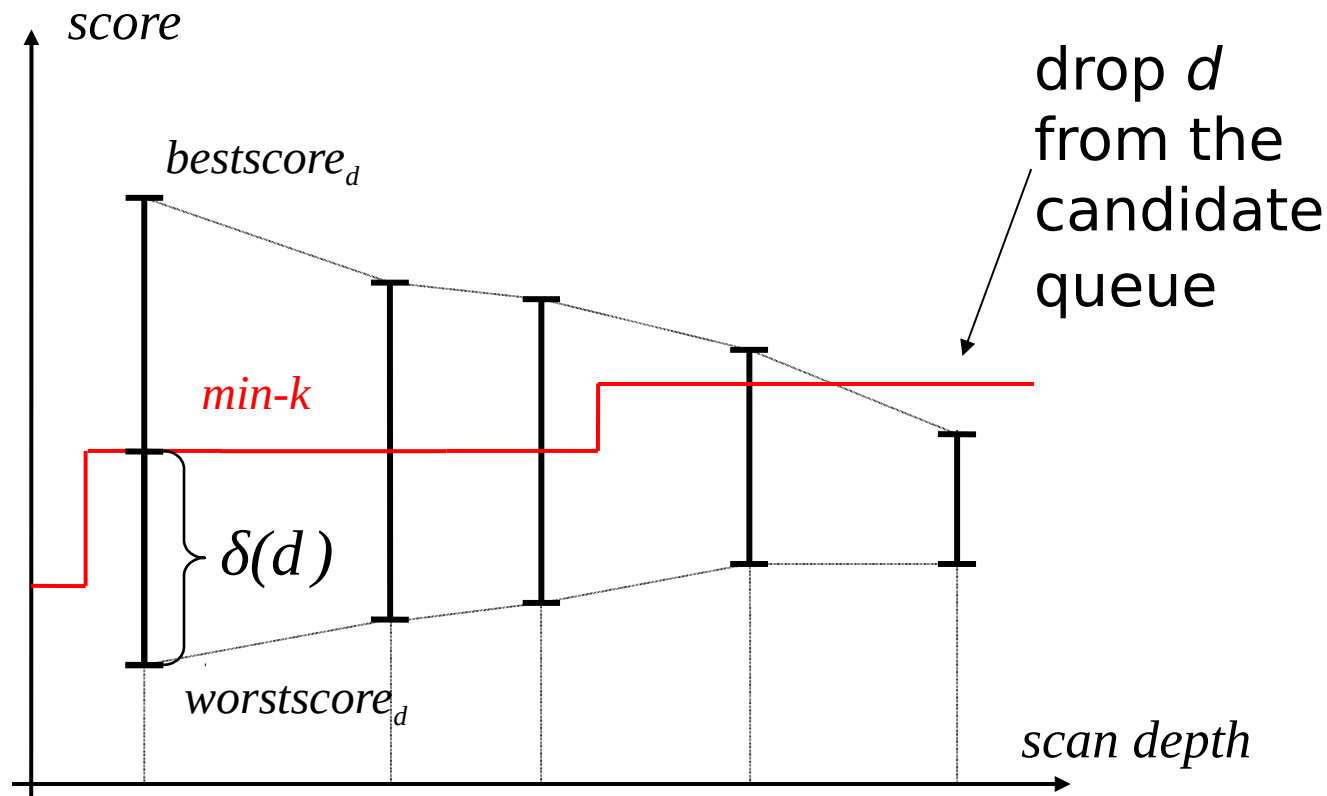
LB assuming
all else are
zeros

Some properties

$$\text{worstscore}(q, d) \leq \text{score}(q, d) \leq \text{bestscore}(q, d)$$

$$\text{agg}(\text{worstscore}(q, d), \text{agg}\{\text{high_il} \in \mathcal{E}(d)\}) = \text{bestscore}(q, d)$$

Test $\text{bestscore}(q, d) < \text{min_k}$ may be conservative



Guaranteed vs. probabilistic pruning

Guaranteed to be correct:

$$\sum_{i \in E(d)} s_i(d) \leq s(d) \leq \sum_{i \in E(d)} s_i(d) + \sum_{i \notin E(d)} \text{high}_i$$

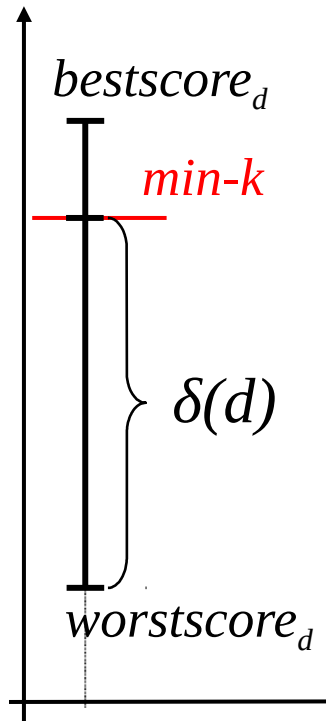
If $\text{min_k} > \text{bestscore}(q, d)$ drop d

More adventurous drop test:

$$p(d) := P \left[\sum_{i \in E(d)} s_i(d) + \sum_{i \notin E(d)} s_i(d) > \text{min}_k \right] \leq \varepsilon$$

$$p(d) = P \left[\sum_{i \notin E(d)} s_i(d) > \delta(d) \right] \leq \varepsilon$$

How to guess this?



Guessing the remaining accumulation

Model distributions over each dimension

Conditioned on seeing latest value $high_i$

Remaining distribution bounded in $[0, high_i]$

For two random variables S_1 and S_2

Density functions $f_1(x) = 1/high_1$ and $f_2(x) = 1/high_2$

Consider convolution

$$f(x) = \int_0^x f_1(z) f_2(x-z) dz$$

Each factor is non-zero in $0 \leq z \leq high_1$ and $0 \leq x-z \leq high_2$

⇒ copious case differentiations

Instead, consider moment-generation functions

$$M_i(s) = \int_0^s e^{sx} f_i(x) dx = E \left[e^{sS_i} \right]$$

Of the form

$$M(s) = \prod_i M_i(s)$$

Consider convolution

Apply Chernoff-Hoeffding bounds

$$P \left[\sum_i S_i > \delta \right] \leq \inf_{s \geq 0} \{ e^{-s\delta} M(s) \}$$