

Information Retrieval and Text Mining

WS 2004/05, Dec 15

Hinrich Schütze

Today's lecture

- Creating test collections for IR
- Parametric search; Zones
- Scoring
- Weighting: tf.idf
- Documents as vectors

Creating Test Collections for IR Evaluation

Relevance gold standard

- Docs: Set of documents
- Queries: Set of queries
- Rels: Judges who establish „truth“ for all document-query pairs (ideally, not possible in practice)
- We can then compute average precision, recall, F for a system on this triple (docs, queries, rels)

Test Corpora

TABLE 4.3 Common Test Corpora

<i>Collection</i>	<i>NDocs</i>	<i>NQrys</i>	<i>Size (MB)</i>	<i>Term/Doc</i>	<i>Q-D RelAss</i>
ADI	82	35			
AIT	2109	14	2	400	>10,000
CACM	3204	64	2	24.5	
CISI	1460	112	2	46.5	
Cranfield	1400	225	2	53.1	
LISA	5872	35	3		
Medline	1033	30	1		
NPL	11,429	93	3		
OSHMED	34,8566	106	400	250	16,140
Reuters	21,578	672	28	131	
TREC	740,000	200	2000	89-3543	» 100,000

From corpora to test collections

- Still need
 - Test queries
 - Relevance assessments
- Test queries
 - Must be germane to docs available
 - Best designed by domain experts
 - Random query terms generally not a good idea
- Relevance assessments
 - Human judges, time-consuming
 - Are human panels perfect?

Kappa measure for inter-judge (dis) agreement

- Kappa measure
 - Agreement among judges
 - Designed for categorical judgments
 - Corrects for chance agreement
- $\text{Kappa} = [P(A) - P(E)] / [1 - P(E)]$
- $P(A)$ – proportion of time coders agree
- $P(E)$ – what agreement would be by chance
- $\text{Kappa} = 0$ for chance agreement, 1 for total agreement.

P(A)? P(E)?

Kappa Measure: Example

Number of docs	Judge 1	Judge 2
300	Relevant	Relevant
70	Nonrelevant	Nonrelevant
20	Relevant	Nonrelevant
10	Nonrelevant	relevant

Kappa Example

- $P(A) = 370/400 = 0.925$
- $P(\text{nonrelevant}) = (10+20+70+70)/800 = 0.2125$
- $P(\text{relevant}) = (10+20+300+300)/800 = 0.7878$
- $P(E) = 0.2125^2 + 0.7878^2 = 0.665$
- $\text{Kappa} = (0.925 - 0.665)/(1-0.665) = 0.776$

- For >2 judges: average pairwise kappas

Kappa Measure

- $\text{Kappa} > 0.8$ = good agreement
- $0.67 < \text{Kappa} < 0.8$ -> “tentative conclusions” (Carletta 96)
- Depends on purpose of study

Interjudge Agreement: TREC 3

Analysis of Consistency of Relevance Judgments

Topic	Judged	Diff	NR	R
51	211	6	4	2
62	400	157	149	8
67	400	68	37	31
70	235	29	25	4
71	400	114	97	17
76	366	97	87	10
78	283	25	22	3
83	400	110	92	18
84	308	95	93	2
95	400	110	108	2
105	259	60	4	56
111	400	76	59	17
122	320	60	43	17
127	400	106	12	94
129	400	28	15	13
131	228	26	6	20

Impact of Inter-judge Agreement

- Impact on **absolute** performance measure can be significant (0.32 vs 0.39)
- Little impact on ranking of different systems or **relative** performance

Unit of Evaluation

- We can compute precision, recall, F, and ROC curve for different units.
- Possible units
 - Documents (most common)
 - Facts (used in some TREC evaluations)
 - Entities (e.g., car companies)
- May produce different results. Why?

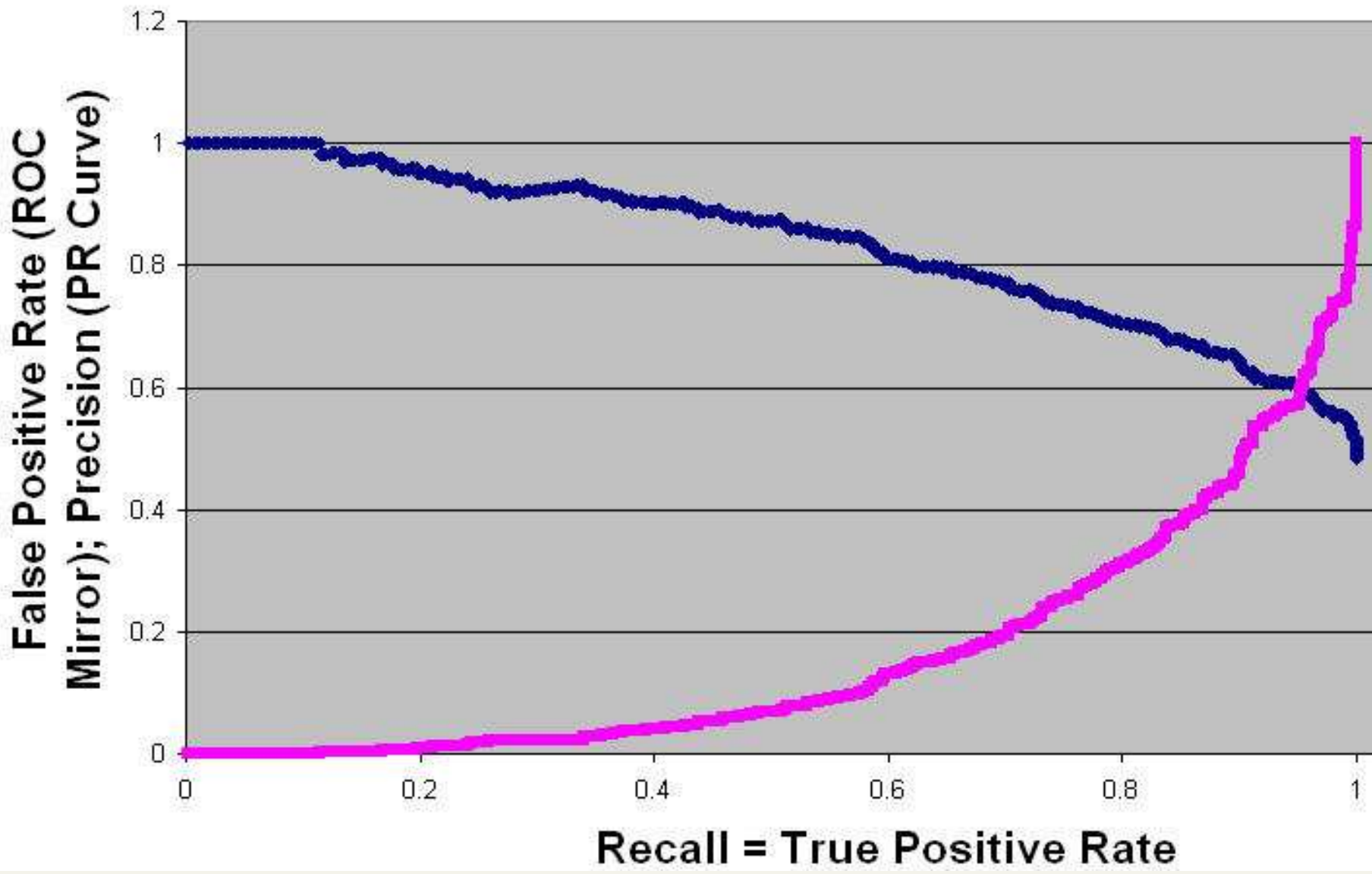
Critique of pure relevance

- 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
- See Carbonell reference

Recap: Evaluation

- Ideally: User happiness
 - Hard to measure directly
- Surrogate: Relevance
 - Is this document relevant to query?
- Precision: True positives / all positives
- Recall: True positives / all relevant
- F: harmonic mean of precision and recall
- Accuracy is meaningless. (Snoogle)
- Methodology: Relevance gold standard

Precision Recall Graph (blue) and
Mirrored ROC Curve (violet)



January 14 lecture

- Link analysis?

Parametric Search Zones

Parametric search

- Each document has, in addition to text, some “meta-data” in fields e.g.,

- Make = BMW

Fields → Model = 5-series ← Values

- Year = 1995

- Mileage = 20,000

- A parametric search interface allows the user to combine a full-text query with selections on these field values e.g.,
 - language, date range, etc.

Parametric search example

CarFinder.com 

Over one million fictional vehicles to choose from!

Choose your search criteria from the drop down menus:

Number of results to display:

Make Model Category Year
City Color Price



Reset Filters

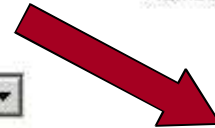
Reset Sorts

Notice that the output is a (large) table. Various parameters in the table (column headings) may be clicked on to effect a sort.

Make	Model	Year	City	Mileage	Price	Category	Description	Color
BMW	5-Series	1995	San Francisco	16100	11100	Luxury	Never driven in winter conditions. Body work makes it look like new. Keyless entry and security features. This is a bargain.	Silver
BMW	5-Series	1995	San Francisco	16600	11100	Luxury	Great first car for your teen-aged kid. Solid, dependable, affordable with 0% down and owner financing.	Blue
BMW	5-Series	1995	San Francisco	16800	11200	Luxury	Upgraded sound system really rocks. Customized interior features wood grain dash and beige leather seats. Power locks, windows, steering. Price firm.	White
BMW	5-Series	1995	San Francisco	16100	11300	Luxury	Safe choice for a young family: ABS, driver and passenger air bags. Roomy interior with power everything. Low mileage driving kids back and forth to soccer.	Maroon
BMW	5-Series	1995	San Francisco	16300	11400	Luxury	This baby's got it all: power steering, cruise, power locks, power windows, remote entry, leather interior, security alarm, AM/FM/CD/Cassette. Priced to sell!	Brown

Parametric search example

We can add text search.



CarFinder.com 

Over one million fictional vehicles to choose from!

Choose your search criteria from the drop down menus:

Number of results to display: 50

Make: BMW | Model: 5-Series | Category: Any | Year: 1997
City: San Francisco | Color: Any | Price: From \$10,100 to \$15,000 | Description:



Reset Filters | Reset Sorts

Make	Model	Year	City	Mileage	Price	Category	Description	Color
BMW	5-Series	1997	San Francisco	14300	13100	Luxury	5-speed, heavy-duty suspension, extra wide tires. Well-maintained by mechanic-owner. Cloth seats and upgraded stereo system.	White
BMW	5-Series	1997	San Francisco	14600	13100	Luxury	Is that price for real? You bet it is. Fully loaded with all factory options. Former floor model.	Beige
BMW	5-Series	1997	San Francisco	14900	13100	Luxury	Fun to drive. Manual 5-speed transmission, turbo charger. Garaged all winter and pampered the rest of the year. This is a steal!	Orange
BMW	5-Series	1997	San Francisco	14800	13200	Luxury	Fully loaded, automatic transmission. Power everything. Anti-lock brakes and full safety features. Must test drive. Price firm.	Green
BMW	5-Series	1997	San Francisco	14300	13200	Luxury	Formerly an executive's vehicle. Interior has been professionally maintained, engine factory serviced every 3000 miles. Great gas mileage. Price negotiable.	Maroon
BMW	5-Series	1997	San Francisco	15000	13200	Luxury	Sun roof, air, CD player, driver side air bag. 10% deposit required. Owner financing available. Best offer by end of weekend buy it	Red

Index support for parametric search

- Must be able to support queries of the form
 - Find BMW cars, model 5-series, with description containing „automatic“
 - A field selection and a phrase query
- Field selection – use inverted index of field values
→ docids
 - Organized by field name
 - Use compression etc. as before
- Recall Lucene

Parametric index support

- Range search – find docs authored between September and December
 - Inverted index doesn't work (as well)
 - Use techniques from database range search
 - See for instance www.bluerwhite.org/btree/ for a summary of B-trees
- Solutions to range search are similar to solutions to wildcard search. Compare:
 - kolmogoro*
 - Mileage interval: 50,000 - 100,000

Field retrieval

- In some cases, must retrieve field values
 - E.g., ISBN numbers of books by **Chomsky**
- Maintain “forward” index – for each doc, those field values that are “retrievable”
 - Indexing control file specifies which fields are retrievable (and can be updated)
 - Storing primary data here, not just an index

(as opposed
to “inverted”)



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

Zone indexes – simple view

Term	N docs	Tot Freq	Doc #	Freq
ambitious	1	1	2	1
be	1	1	2	1
brutus	2	2	2	1
capitol	1	1	1	1
caesar	2	3	1	1
did	1	1	2	2
enact	1	1	1	1
hath	1	1	1	1
i	1	2	2	1
i'	1	1	1	2
it	1	1	1	1
julius	1	1	2	1
killed	1	2	1	1
let	1	1	1	2
me	1	1	2	1
noble	1	1	1	1
so	1	1	2	1
the	2	2	2	1
told	1	1	1	1
you	1	1	2	1
was	2	2	2	1
with	1	1	2	1
			1	1
			2	1
			2	1

Title

Term	N docs	Tot Freq	Doc #	Freq
ambitious	1	1	2	1
be	1	1	1	1
brutus	2	2	2	1
capitol	1	1	1	1
caesar	2	3	1	1
did	1	1	2	2
enact	1	1	1	1
hath	1	1	1	1
i	1	2	2	1
i'	1	1	1	2
it	1	1	1	1
julius	1	1	2	1
killed	1	2	1	1
let	1	1	1	2
me	1	1	2	1
noble	1	1	1	1
so	1	1	2	1
the	2	2	2	1
told	1	1	1	1
you	1	1	2	1
was	2	2	2	1
with	1	1	2	1
			1	1
			2	1
			2	1

Author

Term	N docs	Tot Freq	Doc #	Freq
ambitious	1	1	2	1
be	1	1	1	1
brutus	2	2	2	1
capitol	1	1	1	1
caesar	2	3	1	1
did	1	1	2	2
enact	1	1	1	1
hath	1	1	1	1
i	1	2	2	1
i'	1	1	1	2
it	1	1	1	1
julius	1	1	2	1
killed	1	2	1	1
let	1	1	1	2
me	1	1	2	1
noble	1	1	1	1
so	1	1	2	1
the	2	2	2	1
told	1	1	1	1
you	1	1	2	1
was	2	2	2	1
with	1	1	2	1
			1	1
			2	1
			2	1

Body

etc.

So we have a database now?

- Not really.
- Databases do lots of things we don't need
 - Transactions
 - Recovery (our index is not the system of record; if it breaks, simply reconstruct from the original source)
 - Indeed, we never have to store text in a search engine – only indexes
- Databases don't do inverted indexes.
 - Actually, they do ...
 - But focus is different: optimized for data vs text
- Not queries like “all papers whose authors cite themselves”. Why?

Scoring

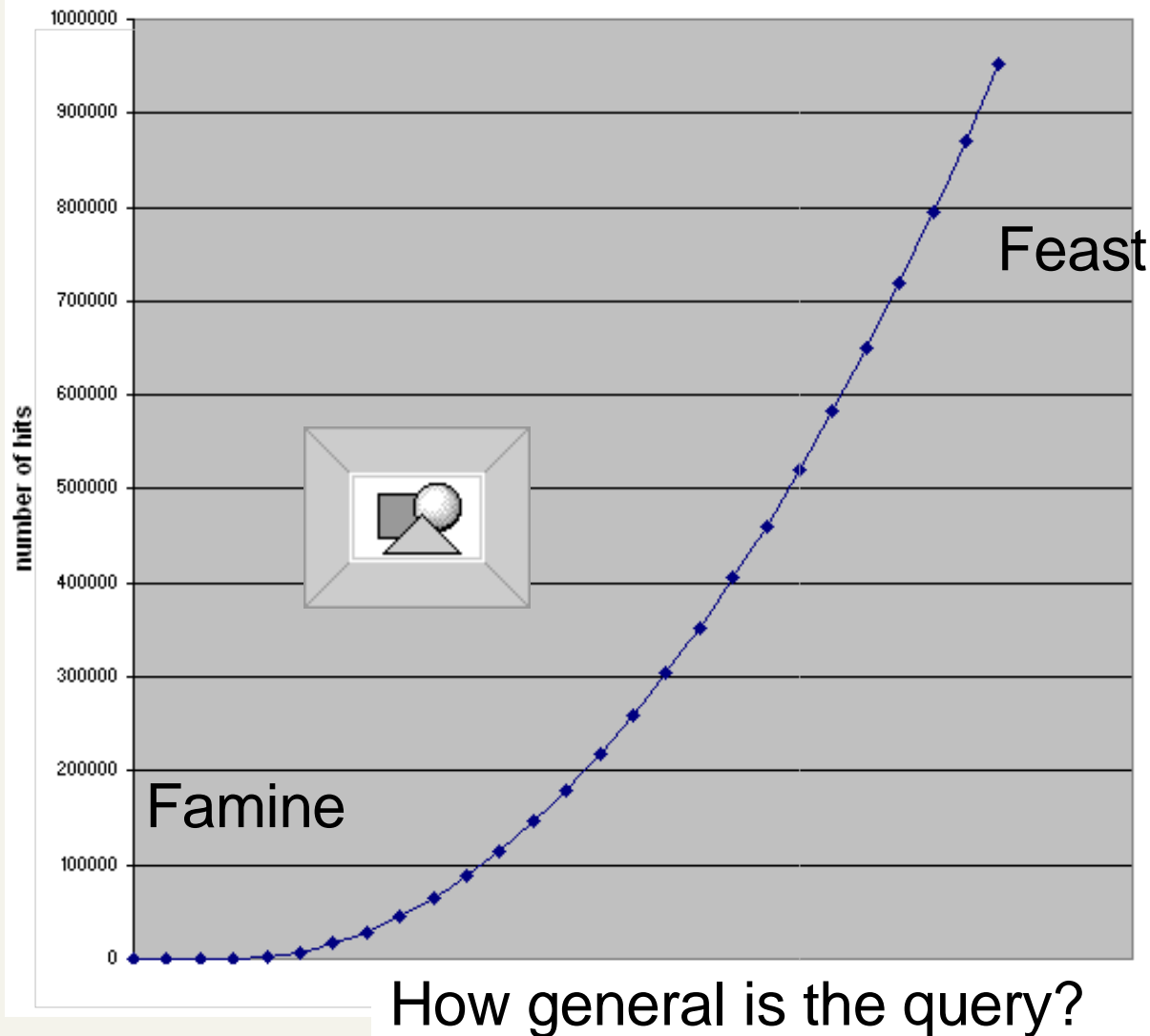
Scoring

- 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
- Not good for (the majority of) users with poor Boolean formulation of their needs
-

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
- Order docs according to score

One Problem With Boolean Queries: Feast or Famine



Specifying
a well targeted
query is hard.

Google: 1860 hits
for: *standard user*
dlink 650

0 hits after adding
no card found

Linear zone combinations

- First generation of scoring methods: use a linear combination of Booleans:

- E.g.,

$$\text{Score} = 0.6 * \langle \textit{sorting} \text{ in } \underline{\text{Title}} \rangle + 0.3 * \langle \textit{sorting} \text{ in } \underline{\text{Abstract}} \rangle + 0.05 * \langle \textit{sorting} \text{ in } \underline{\text{Body}} \rangle + 0.05 * \langle \textit{sorting} \text{ in } \text{Boldface} \rangle$$

- Each expression such as $\langle \textit{sorting} \text{ in } \underline{\text{Title}} \rangle$ 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

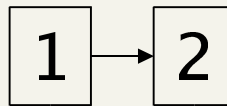
- In fact, the expressions between $\langle \rangle$ on the last slide could be *any* Boolean query
- Who generates the Score expression (with weights such as 0.6 etc.)?
 - In uncommon cases – the user through the UI
 - Weights determined from user studies and hard-coded into the weighting
 - Weights learned by machine learning

Exercise

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

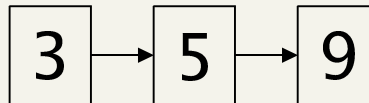
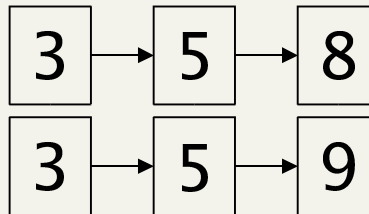
Author

bill
rights



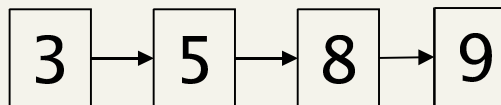
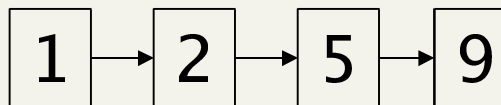
Title

bill
rights



Body

bill
rights



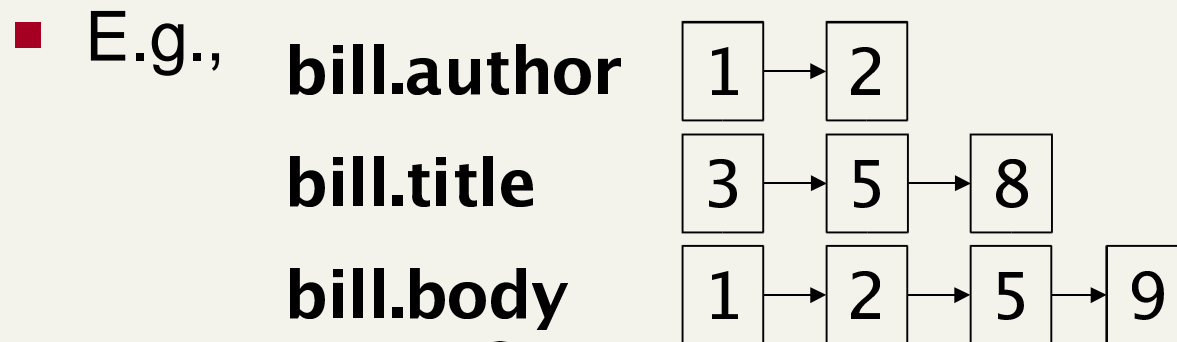
Compute the score for each doc based on the weightings 0.6,0.3,0.1

General idea

- We are given a weight vector 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



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:

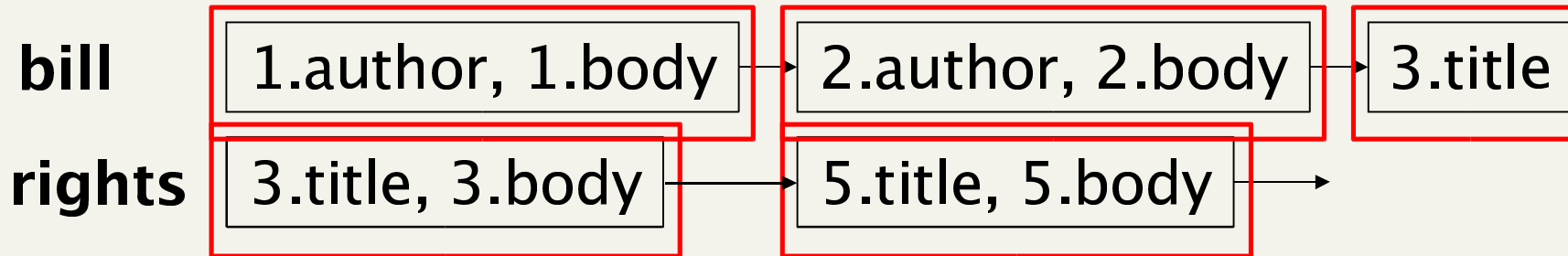
bill 1.author, 1.body → 2.author, 2.body → 3.title

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

1	0.7
2	0.7
3	0.4
5	0.4



- 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?

Zones for web search engines

Free Text Queries

Term weighting

Free text queries

- We want to raise the score for more hits
 - 3 occurrences of BMW are better than one
- Most users don't use Boolean connectives
 - They will enter: ***bill rights*** or ***bill of rights***
 - Instead of: ***bill OR rights***
- How do we interpret these “free text” queries?
 - No Boolean connectives
 - Of several query terms some may be missing in a doc
 - Only some query terms may occur in the title, etc.

Free text queries

- Desiderata for free text queries
 - A way of assigning a score to a pair <free text query, document>
 - Zero query terms in the document should mean a zero score
 - More query terms in the zone should mean a higher score
 - Scores don't have to be Boolean
- Will look at an alternatives now
 - Vector space scoring
 - Zone scoring and Vector space scoring are orthogonal

Incidence matrices

- Recall: Document (or a zone in it) is binary vector X in $\{0,1\}^v$
 - Query is a vector
- Score: Overlap measure:

$$|X \cap Y|$$

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Example

- On the query *ides of march*, Shakespeare's *Julius Caesar* has a score of 3
- All other Shakespeare plays have a score of 2 (because they contain *march*) or 1
- Thus in a rank order, *Julius Caesar* would come out tops

Overlap matching

- What's wrong with the overlap measure?
- It doesn't consider:
 - Term frequency in document
 - Term scarcity in collection (document mention frequency)
 - *of* is more common than *ides* or *march*
 - Length of documents
 - (And queries: score not normalized)

Overlap matching

- One can normalize in various ways:

- Jaccard coefficient:

$$|X \cap Y| / |X \cup Y|$$

- Cosine measure:

$$|X \cap Y| / \sqrt{|X| \times |Y|}$$

- What documents would score best using Jaccard against a typical query?
 - Does the cosine measure fix this problem?

Scoring: density-based

- Thus far: position and overlap 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.
- Document relevant if it has a lot of the terms
- This leads to the idea of term weighting.


Term-document count matrices

- Consider the number of occurrences of a term in a document:
 - Bag of words model
 - Document is a vector in \mathbb{N}^v : a column below

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	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.*



Which of the indexes discussed so far distinguish these two docs?

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 *ofs*

Digression: terminology

- WARNING: In a lot of IR literature, “frequency” is used to mean “count”, not „relative frequency“
 - Thus *term frequency* in IR literature is used to mean *number of occurrences* in a doc
 - Not divided by document length (which is the meaning of relative frequency)
- We will conform to this convention
 - In saying term frequency we mean the number of occurrences of a term in a document.

Term frequency *tf*

- Long docs are favored because they're more likely to contain query terms
- Can fix this to some extent by normalizing for document length
- 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:

$$wf_{t,d} = 0 \text{ if } tf_{t,d} = 0, 1 + \log tf_{t,d} \text{ otherwise}$$

Score computation

- Score for a query q = sum over terms t in q :

$$= \sum_{t \in q} tf_{t,d}$$

- [Note: 0 if no query terms in document]
- This score can be zone-combined
- Can use wf instead of tf in the above
- 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 occurrences of the term in the entire collection of documents

Document frequency

- But document frequency (*df*) may be better:
- *df* = number of docs in the corpus containing the term

Word	<i>cf</i>	<i>df</i>
<i>try</i>	10422	8760
<i>insurance</i>	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

- tf x idf measure combines:
 - term frequency (*tf*)
 - or *wf*, measure of term density in a doc
 - inverse document frequency (*idf*)
 - measure of informativeness of a term: its rarity across the whole corpus
 - could just be raw count of number of documents the term occurs in ($idf_i = 1/df_i$)
 - but by far the most commonly used version is:

$$idf_i = \log\left(\frac{n}{df_i}\right)$$

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

Summary: tf x idf (or tf.idf)

- 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?

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

n = total number of documents

df_i = 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

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	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

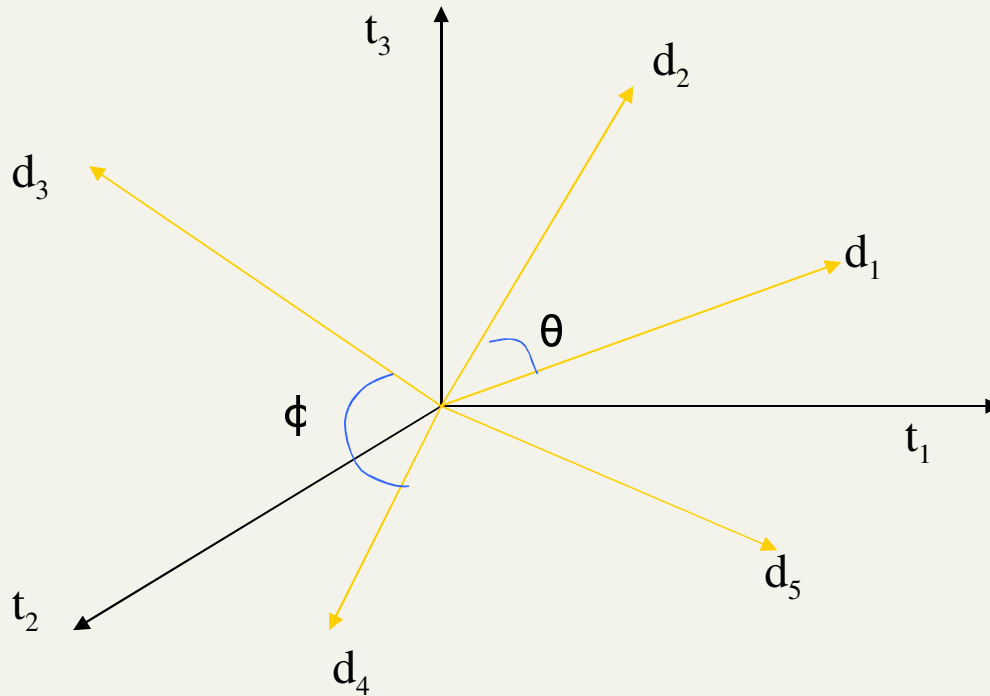
Documents as vectors

- Each doc j can now be viewed as a vector of $wf \times idf$ 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
- (The corpus of documents gives us a matrix, which we could also view as a vector space in which words live – transposable data)

Why turn docs into vectors?

- First application: Query-by-example
 - Given a doc D , find others “like” it.
- Now that D is a vector, find vectors (docs) “near” it.

Intuition



Postulate: Documents that are “close together” in the vector space talk about the same things.

The vector space model

Query as vector:

- We regard query as short document
- We return the documents ranked by the closeness of their vectors to the query, also represented as a vector.

Desiderata for proximity

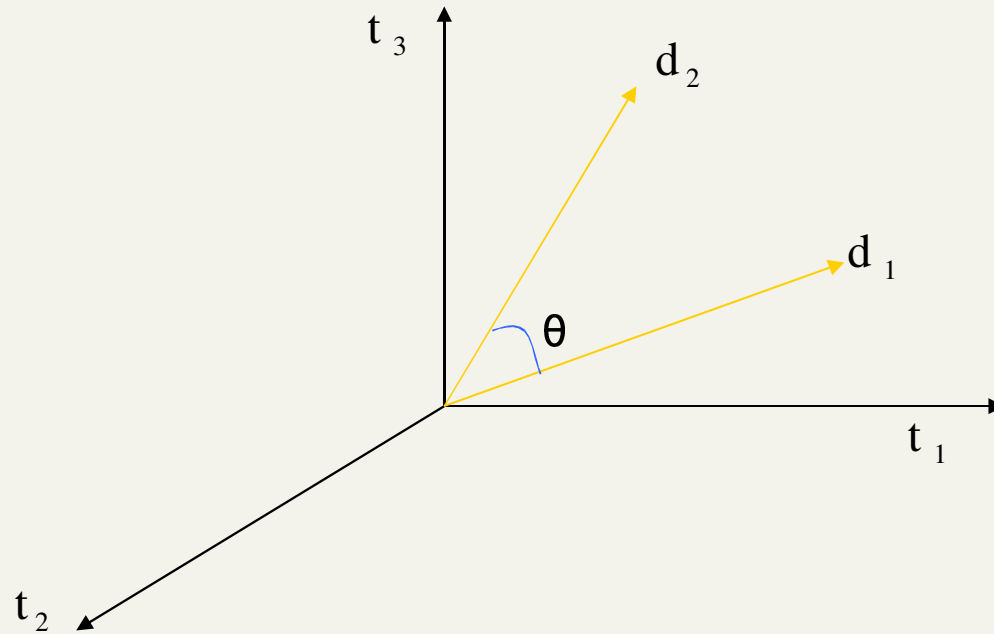
- If d_1 is near d_2 , then d_2 is near d_1 .
- If d_1 near d_2 , and d_2 near d_3 , then d_1 is not far from d_3 .
- No doc is closer to d than d itself.

First cut

- Distance between d_1 and d_2 is the length of the vector $|d_1 - d_2|$.
 - Euclidean distance
- Why is this not a great idea?
- We still haven't dealt with the issue of length normalization
 - Long documents would be more similar to each other by virtue of length, not topic
- However, we can implicitly normalize by looking at *angles* instead

Cosine similarity

- Distance between vectors d_1 and d_2 captured by the cosine of the angle x between them.
- Note – this is *similarity*, not distance
 - No triangle inequality for similarity.



Cosine similarity

- A vector can be *normalized* (given a length of 1) by dividing each of its components by its length – here we use the L_2 norm

$$\|\mathbf{x}\|_2 = \sqrt{\sum_i x_i^2}$$

- This maps vectors onto the unit sphere:
- Then, $|d_j| = \sqrt{\sum_{i=1}^n w_{i,j}^2} = 1$
- Longer documents don't get more weight

Cosine similarity

$$\text{sim}(d_j, d_k) = \frac{\vec{d}_j \cdot \vec{d}_k}{\|\vec{d}_j\| \|\vec{d}_k\|} = \frac{\sum_{i=1}^n w_{i,j} w_{i,k}}{\sqrt{\sum_{i=1}^n w_{i,j}^2} \sqrt{\sum_{i=1}^n w_{i,k}^2}}$$

- Cosine of angle between two vectors
- The denominator involves the lengths of the vectors.

Normalization

Normalized vectors

- For normalized vectors, the cosine is simply the dot product:

$$\cos(d_j, d_k) = d_j \cdot d_k$$

Cosine similarity exercises

- *Exercise: Rank the following by decreasing cosine similarity:*
 - Two docs that have only frequent words (***the, a, an, of***) in common.
 - Two docs that have no words in common.
 - Two docs that have many rare words in common (***wingspan, tailfin***).

Exercise

- Euclidean distance between vectors:

$$\left| d_j - d_k \right| = \sqrt{\sum_{i=1}^n \left(d_{i,j} - d_{i,k} \right)^2}$$

- Show that, for normalized vectors, Euclidean distance gives the same proximity ordering as the cosine measure

Example

- Docs: Austen's *Sense and Sensibility*, *Pride and Prejudice*; Bronte's *Wuthering Heights*

	SaS	PaP	WH
<i>affection</i>	115	58	20
<i>jealous</i>	10	7	11
<i>gossip</i>	2	0	6
	SaS	PaP	WH
<i>affection</i>	0.996	0.993	0.847
<i>jealous</i>	0.087	0.120	0.466
<i>gossip</i>	0.017	0.000	0.254

- $\cos(\text{SAS}, \text{PAP}) = .996 \times .993 + .087 \times .120 + .017 \times 0.0 = 0.999$
- $\cos(\text{SAS}, \text{WH}) = .996 \times .847 + .087 \times .466 + .017 \times .254 = 0.929$

Digression: spamming indices

- This was all invented before the days when people were in the business of spamming web search engines:
 - Indexing a sensible passive document collection vs.
 - An active document collection, where people (and indeed, service companies) are shaping documents in order to maximize scores

Summary: What's the real point of using vector spaces?

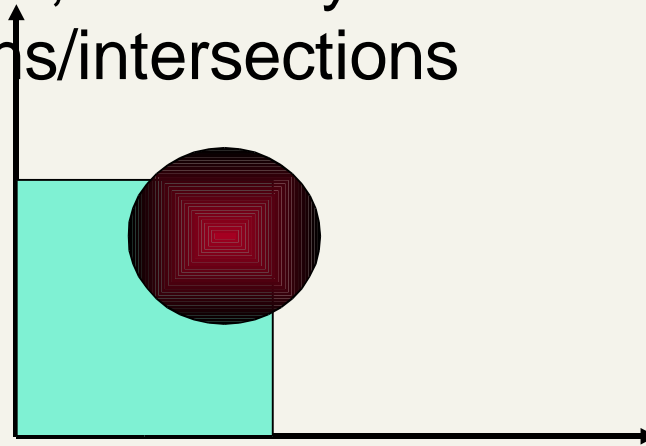
- Key: A user's query can be viewed as a (very) short document.
- Query becomes a vector in the same space as the docs.
- Can measure each doc's proximity to it.
- Natural measure of scores/ranking – no longer Boolean.
 - Queries are expressed as bags of words
- Other similarity measures: see <http://www.lans.ece.utexas.edu/~strehl/diss/node52.html> for a survey

Interaction: 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: we cannot expect end-user formulating queries to know what phrases are indexed
- Indexing all biwords is too expensive

Vectors and Boolean queries

- Vectors and Boolean queries really don't work together very well
- In the space of terms, vector proximity selects by spheres: e.g., all docs having cosine similarity ≥ 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 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 *tfs* and *idfs* 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
- But ...

Query language vs. scoring

- May allow user a certain query language, say
 - Freetext basic queries
 - Phrase, wildcard etc. in Advanced Queries.
- For scoring (oblivious to user) may use all of the above, e.g. for a freetext query
 - Highest-ranked hits have query as a phrase
 - Next, docs that have all query terms near each other
 - Then, docs that have some query terms, or all of them spread out, with tfxidf weights for scoring

Exercises

- How would you augment the inverted index built in lectures 1–3 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?

Efficient cosine ranking

- What we're doing in effect: solving the k -nearest neighbor problem for a query vector
- In general, do not know how to do this efficiently for high-dimensional spaces
- But it is solvable for short queries, and standard indexes are optimized to do this

Computing a single cosine

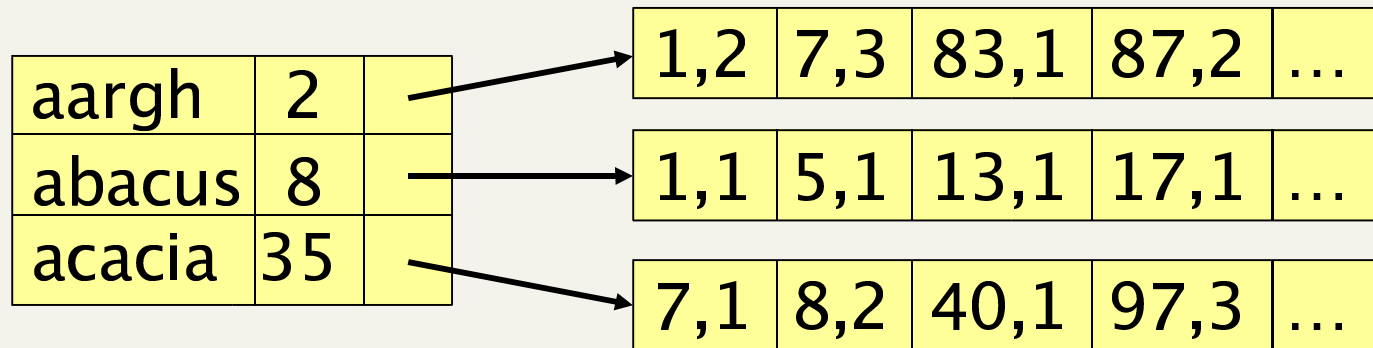
- For every term i , with each doc j , store term frequency tf_{ij} .
 - Some tradeoffs on whether to store term count, term weight, or weighted by idf_i .
- At query time, accumulate component-wise sum

$$sim(d_j, d_k) = \sum_{i=1}^m w_{i,j} \times w_{i,k}$$

- If you're indexing 8 billion documents (web search) an array of accumulators is infeasible



Encoding document frequencies



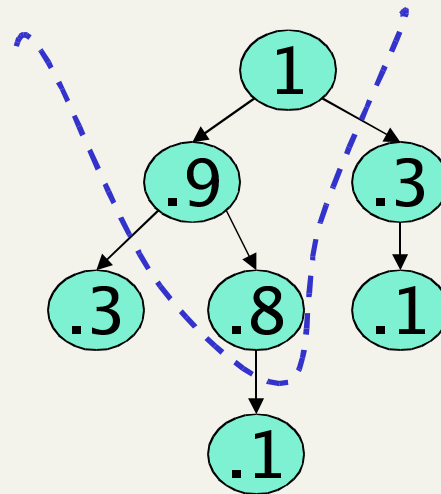
- Add $tf_{t,d}$ to postings lists
 - Almost always as frequency – scale at runtime
 - Unary code is very effective here ← Why?
 - γ code (Lecture 3) is an even better choice
 - Overall, requires little additional space

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?

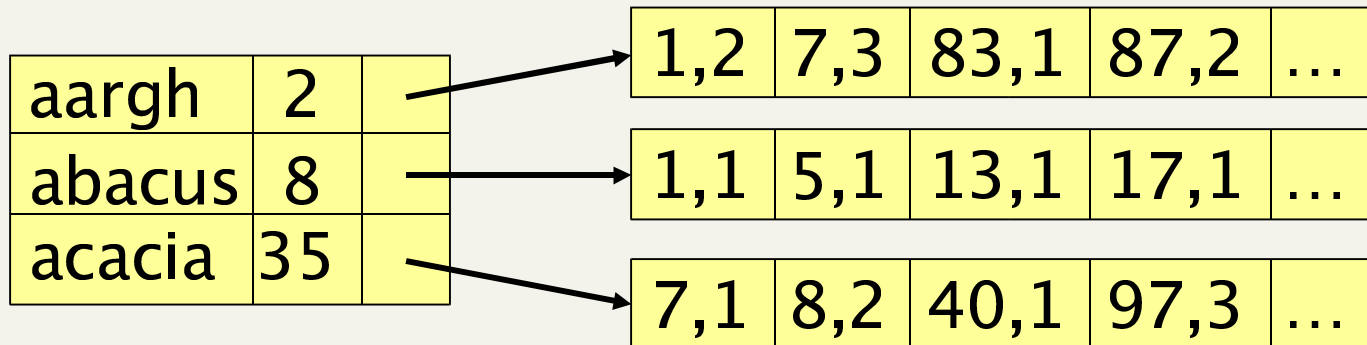
Use heap for selecting top k

- 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 $2\log n$ steps.
- For $n=1\text{M}$, $k=100$, this is about 10% of the cost of sorting.



Bottleneck

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



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.

Best m 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 the 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.

Term weighting vs. Boolean

- Summary
 - Complex boolean queries are difficult for average user
 - Feast or famine problem
- Prior to google, many IR researchers thought boolean queries were a bad idea.
- Google queries are strict conjunctions.
- Why is this working well?

Resources for this lecture

- MIR Chapter 3
- MG 4.5
- MG Ch. 3; Ch. 4.4-4.6; MIR 2.5, 2.7.2