

What's wrong with Boolean?

- 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
- We want to raise the score for more hits
 - 3 occurrences of BMW are better than one

Information Retrieval and Text Mining

WS 2004/05, Dec 17

Hinrich Schütze

Ranking

- *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

Today's lecture

- Free text queries
- Ranking
- Tf.idf weighting
- Documents as vectors

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

Free text vs. Boolean queries

- No Boolean connectives
- Of several query terms some may be missing in a doc
- How do we interpret these "free text" queries?

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 document should mean a higher score
- Vector space models
 - First model that met these desiderata
 - Zone scoring and Vector space scoring are orthogonal

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.

What's wrong with overlap?

- 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)

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}^n : 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

Overlap matching

- One can normalize in various ways:
 - Jaccard coefficient:

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

- Cosine measure:

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

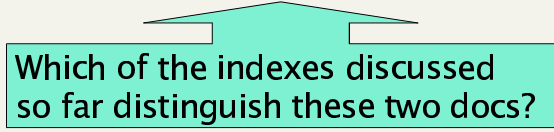
- What documents would score best using Jaccard against a typical query?
 - Does the cosine measure fix this problem?

Digression: terminology

- 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.

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?

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?

Counts vs. frequencies

- Consider again the ***ides of march*** query.
 - *Julius Caesar* has 5 occurrences of ***ides***
 - No other play has ***ides***
 - Most (all?) plays contain ***march***
- By this scoring measure, the top-scoring play is likely to be the one with the most ***march***'s

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”?
- Assumption: content words are rare, function words are frequent
- 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

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

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
- Still doesn't consider term scarcity in collection (*ides* is rarer than *march*)

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* × *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)

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
 - Most commonly used version is:

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

- n is the number of documents in the collection.

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 none 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

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.

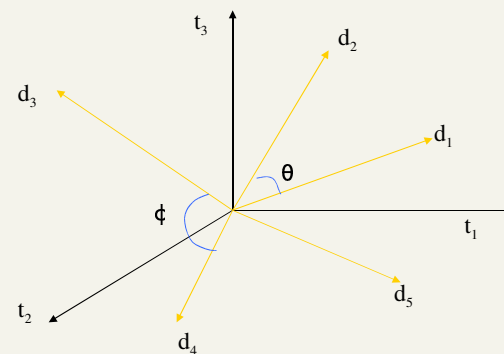
Why turn docs into vectors?

- Query can also be represented as a vector in this high-dimensional space
- We can view querying as searching for close neighbors
- Also: Query-by-example
 - Given a doc D , find others “like” it.

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.

Intuition



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

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, $\|d_j\| = \sqrt{\sum_{i=1}^n w_{i,j}} = 1$

- Longer documents don't get more weight

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
- Picture
- However, we can implicitly normalize by looking at *angles* instead

Cosine similarity

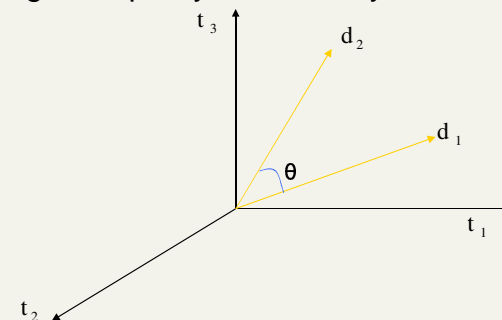
$$\text{sim}(d_j, d_k) = \frac{d_j \cdot d_k}{\|d_j\| \|d_k\|} = \frac{\sum_{i=1}^n w_{i,j} w_{i,k}}{\sqrt{\sum_{i=1}^n w_{i,j}^2} \sqrt{\sum_{i=1}^n w_{i,k}^2}}$$

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

Normalization

Cosine similarity

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



Exercise

- Euclidean distance between vectors:

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

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

Normalized vectors

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

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

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

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

Cosine similarity exercises

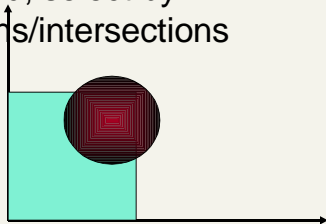
- Exercise: Rank the following by decreasing cosine similarity:*
 - Two docs that have only frequent words (**the, a, an, of**) in common.
 - Two docs that have no words in common.
 - Two docs that have many rare words in common (**wingspan, tailfin**).

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 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
- Violates independence assumptions even more than usual

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



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
- Example: Altavista

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

Combining methods vs. results

- Direct combination of methods hard:
 - Phrase, wildcards, Boolean or/not
- Alternative: Combination of results
 - 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 tfidf weights for scoring

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.

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*

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 ...

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 frequency (tf) weight, weighted by idf (tf.idf), or length-normalized tf.idf.
- At query time, accumulate component-wise sum

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

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?

Encoding document frequencies

aargh	2	→	1,2	7,3	83,1	87,2	...
abacus	8	→	1,1	5,1	13,1	17,1	...
acacia	35	→	7,1	8,2	40,1	97,3	...

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

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

Bottleneck

- Still need to first compute cosines from query to each of n docs \rightarrow several seconds for $n = 1M$.
- Completely impossible for 8 billion documents.
- 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).

aargh	2		1,2	7,3	83,1	87,2	...
abacus	8		1,1	5,1	13,1	17,1	...
acacia	35		7,1	8,2	40,1	97,3	...

Removing bottlenecks

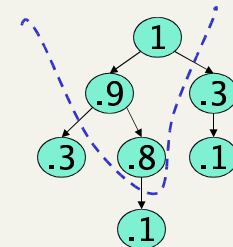
- Can further limit to documents with non-zero cosines on rare (high idf) words
- Enforce conjunctive search (a la Google): non-zero cosines on *all* words in query
 - Get # accumulators down to {min of postings lists sizes}
- But still potentially expensive
 - Sometimes have to fall back to (expensive)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

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 $n=1M$, $k=100$, this is about 10% of the cost of sorting.



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.

Can we avoid this?

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

But aren't Google queries Boolean?

- Prior to google, many IR researchers thought boolean queries were a bad idea.
 - Example: „turkey beach vacation resort snorkeling“
 - Many relevant examples will lack one of these terms.
- Google queries are (usually) strict conjunctions.
- Why is this working well?

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.

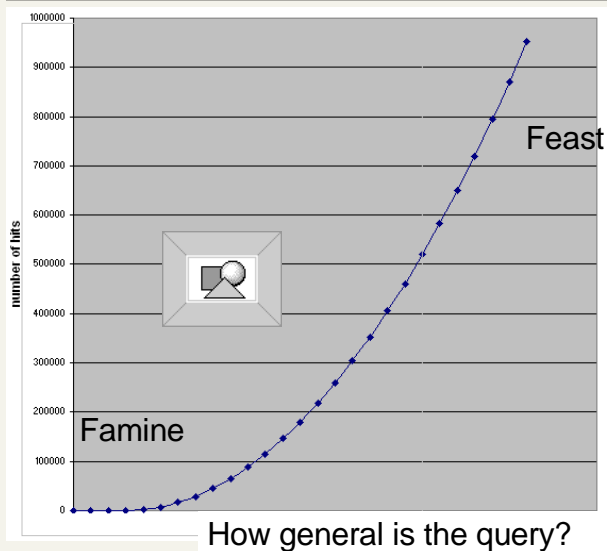
Recap: Gold St., Metadata, Zones

- Gold standards in information retrieval
 - Docs, Queries, Relevance judgements
 - Variability: Absolute vs. Relative evaluation
- Metadata and zones
 - Modified inverted index or several inverted indices
- Boolean vs Ranked retrieval
 - Feast or famine problem
 - Most users can't do Boolean logic
- Weighting zones
 - High-weight zones: title, abstract, anchor text
 - Low-weight zones: body of document

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)

One Problem With Boolean Queries: Feast or Famine

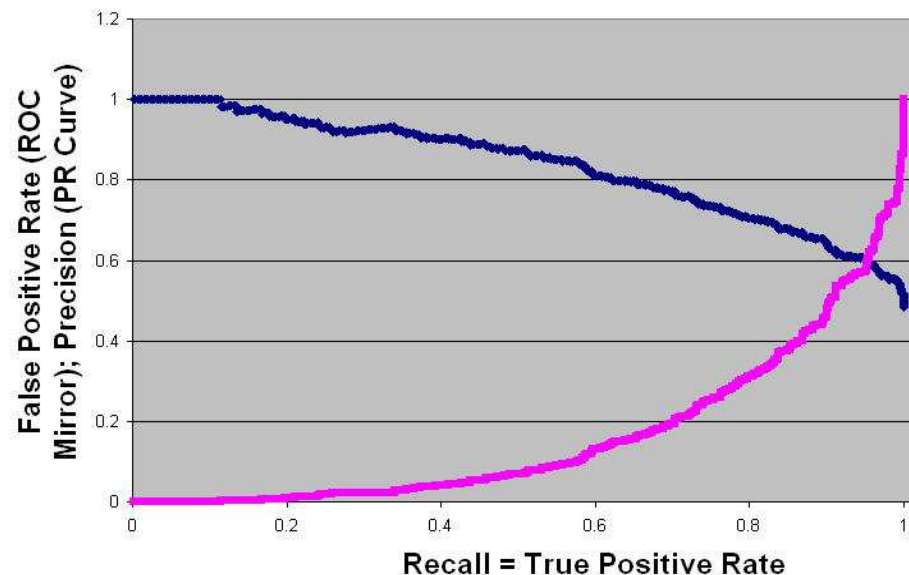


Specifying a well targeted query is hard.

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

0 hits after adding *no card found*

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



January 14 lecture

- Link analysis?

Resources for this lecture

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