Introduction to Information Retrieval

http://informationretrieval.org

IIR 1: Boolean Retrieval

Hinrich Schütze

Institute for Natural Language Processing, University of Stuttgart

2011-08-29

Models and Methods

1. Boolean model and its limitations (30)
2. Vector space model (30)
3. Probabilistic models (30)
4. Language model-based retrieval (30)
5. Latent semantic indexing (30)
6. Learning to rank (30)

Models and Methods

1. Boolean model and its limitations (30)
2. Vector space model (30)
3. Probabilistic models (30)
4. Language model-based retrieval (30)
5. Latent semantic indexing (30)
6. Learning to rank (30)

Take-away

- Boolean model and Inverted index: The Boolean model and the basic data structure of most IR systems
- Processing Boolean queries
- Why is Boolean retrieval not enough? or Why do we need ranked retrieval?

Outline

Definition of *information retrieval*

Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).

Definition of *information retrieval*

Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).

The adhoc retrieval problem: Given a user information need and a collection of documents, the IR system determines how well the documents satisfy the query and returns a subset of relevant documents to the user. □

Boolean retrieval

- The Boolean model is arguably the simplest model to base an information retrieval system on.
- Queries are Boolean expressions, e.g., CAESAR AND BRUTUS
- The seach engine returns all documents that satisfy the Boolean expression.

Boolean retrieval

- The Boolean model is arguably the simplest model to base an information retrieval system on.
- Queries are Boolean expressions, e.g., CAESAR AND BRUTUS
- The seach engine returns all documents that satisfy the Boolean expression.

Model collection: The works of Shakespeare

Each of Shakespeare's tragedies, comedies etc is a document in this collection.

Term-document incidence matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	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	

...

Entry is 1 if term occurs. Example: CALPURNIA occurs in *Julius Caesar*.

Entry is 0 if term doesn't occur. Example: CALPURNIA doesn't occur in *The tempest*.

Term-document incidence matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	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	

...

Entry is 1 if term occurs. Example: CALPURNIA occurs in *Julius Caesar*.

Entry is 0 if term doesn't occur. Example: CALPURNIA doesn't occur in *The tempest*.

Term-document incidence matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	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	

...

Entry is 1 if term occurs. Example: CALPURNIA occurs in *Julius Caesar*.

Entry is 0 if term doesn't occur. Example: CALPURNIA doesn't occur in *The tempest*.

We will return to this matrix many times in this class. □

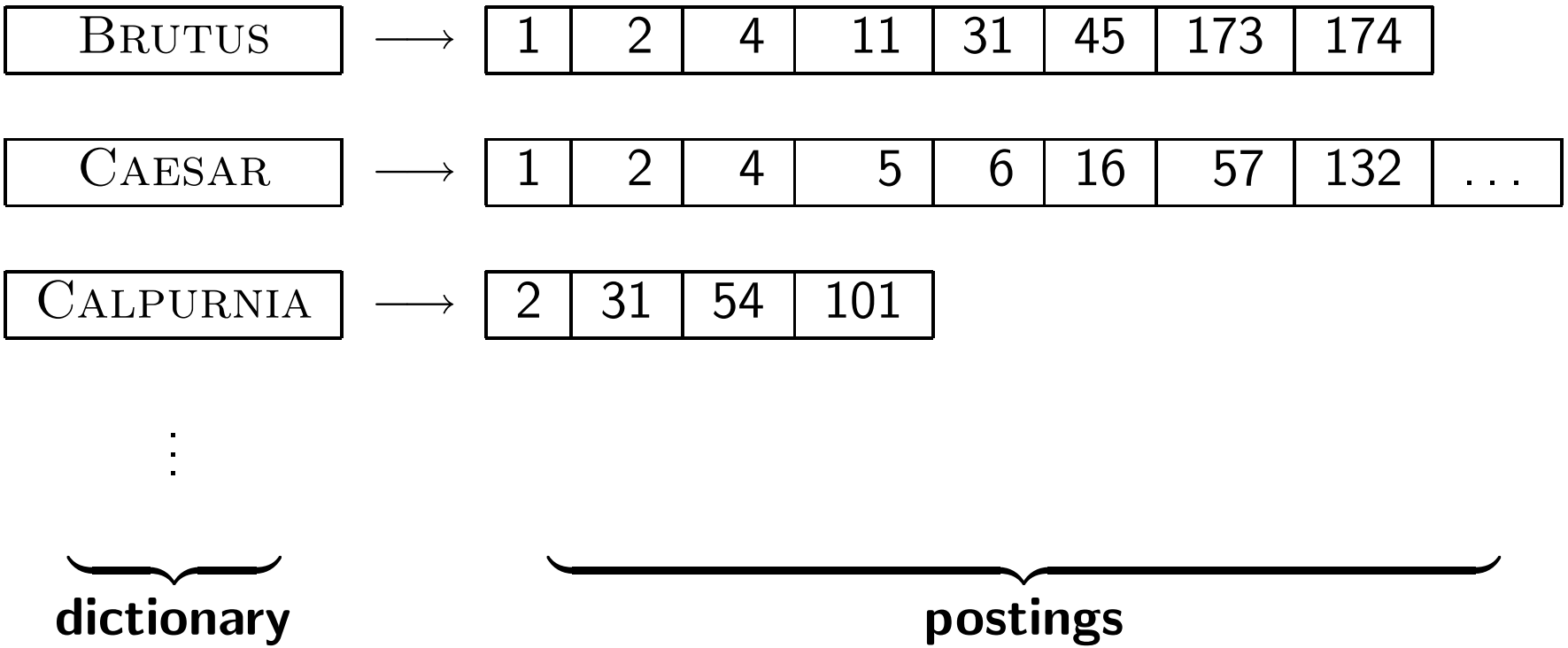
We can't build the incidence matrix for large collections

- Size of incidence matrix: number of documents times number terms $\longrightarrow$ too large for large collections
- But the matrix is very sparse – mostly 0s, few 1s.
- Inverted index: We only record the 1s.

Inverted Index

For each term t, we store a list of all documents that contain t.

= For each term t, we store the 1s in its row in the incidence matrix

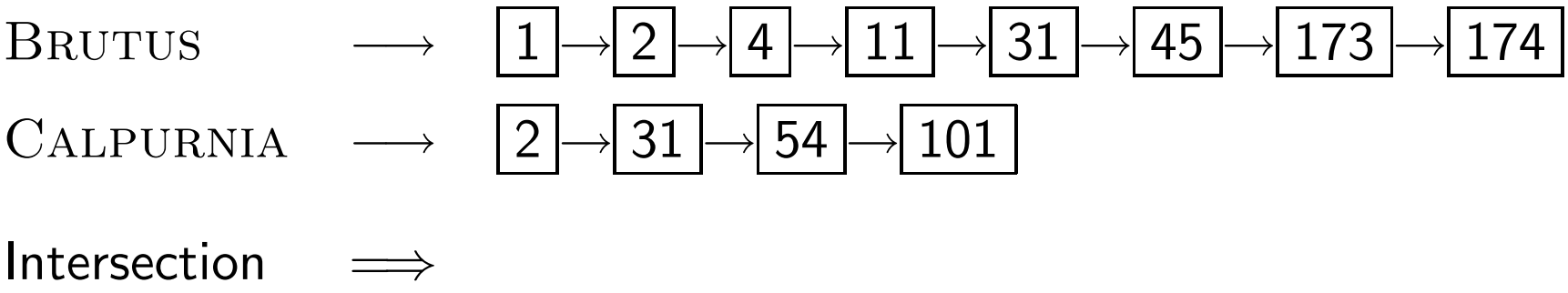


Outline

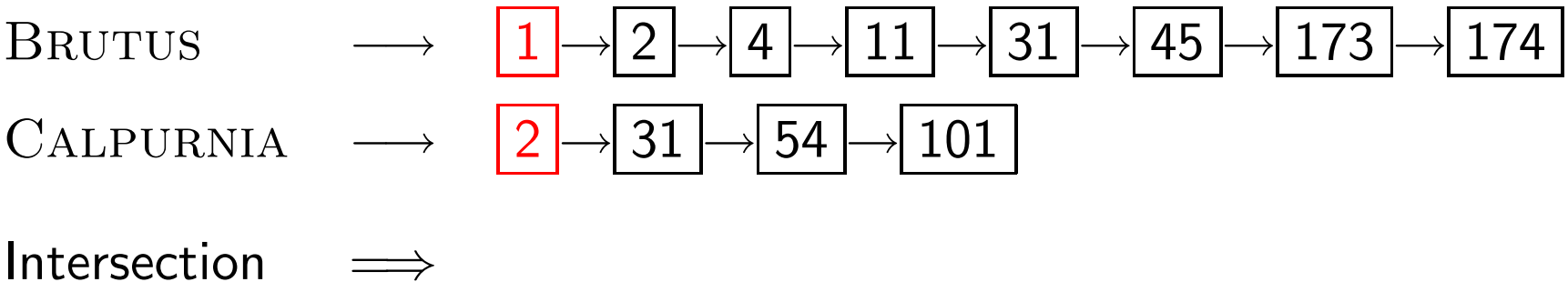
Simple conjunctive query (two terms)

- Consider the query: BRUTUS AND CALPURNIA
- To find all matching documents using inverted index:
  1. Locate BRUTUS in the dictionary
  2. Retrieve its postings list from the postings file
  3. Locate CALPURNIA in the dictionary
  4. Retrieve its postings list from the postings file
  5. Intersect the two postings lists
  6. Return intersection to user

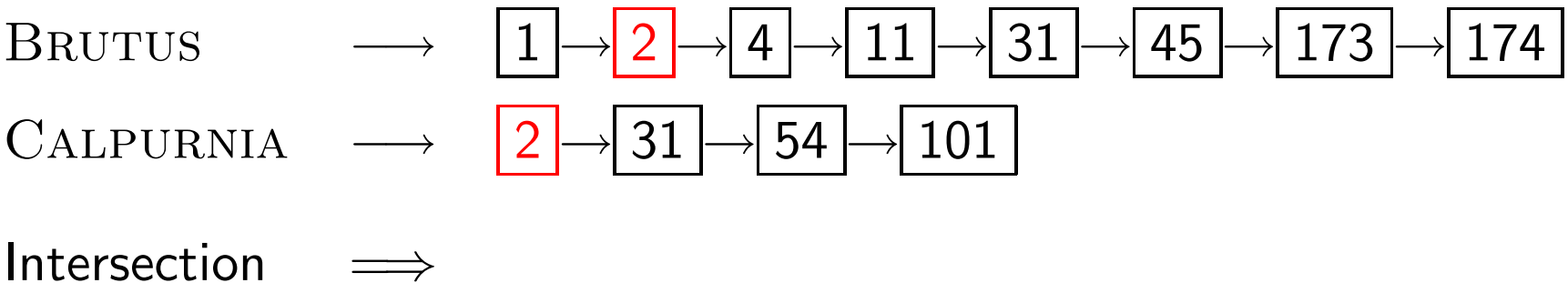
Intersecting two postings lists



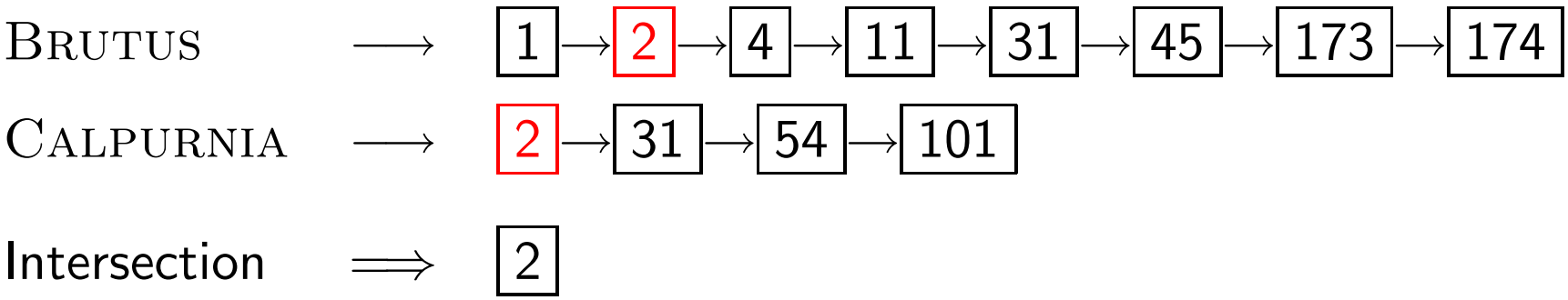
Intersecting two postings lists



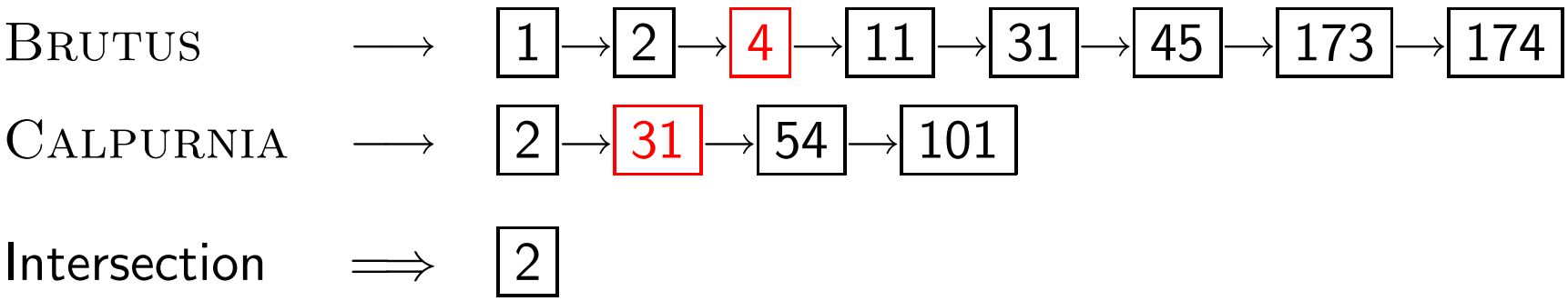
Intersecting two postings lists



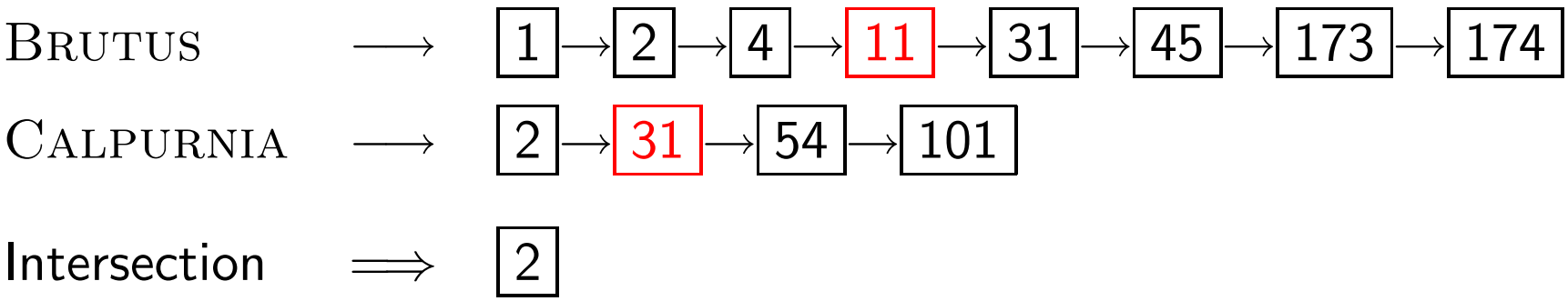
Intersecting two postings lists



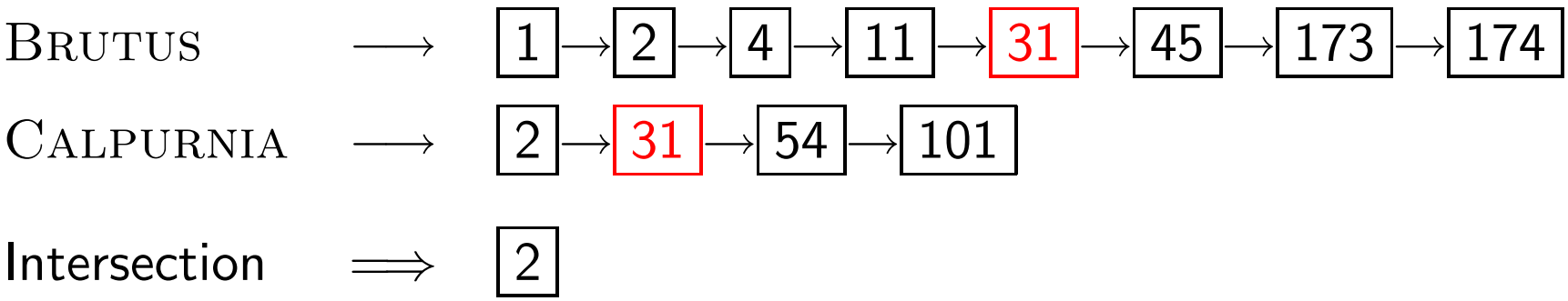
Intersecting two postings lists



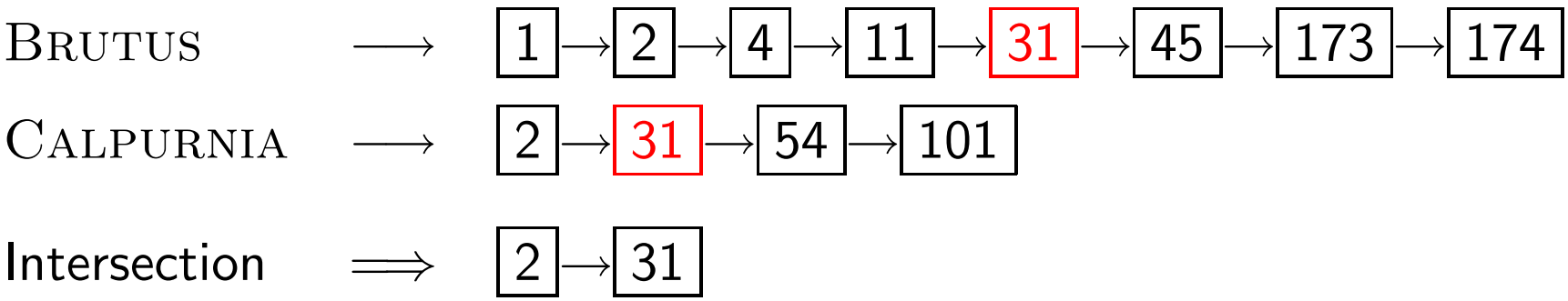
Intersecting two postings lists



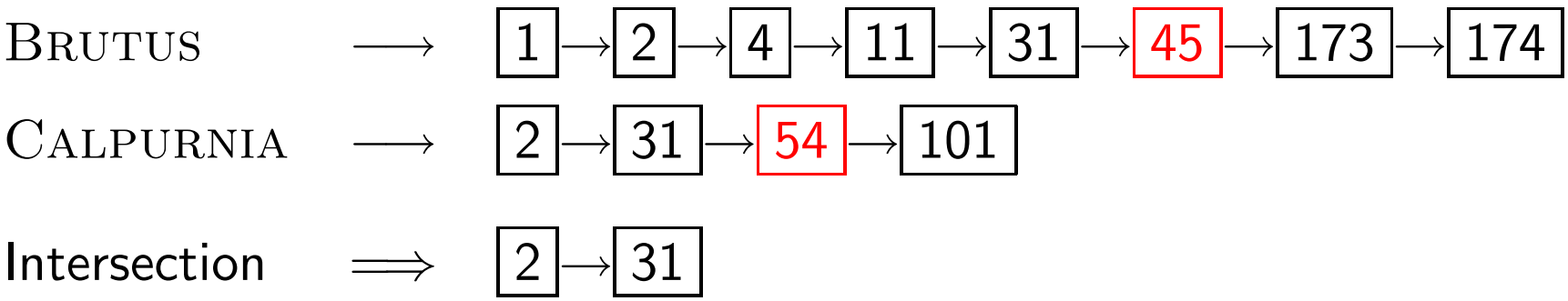
Intersecting two postings lists



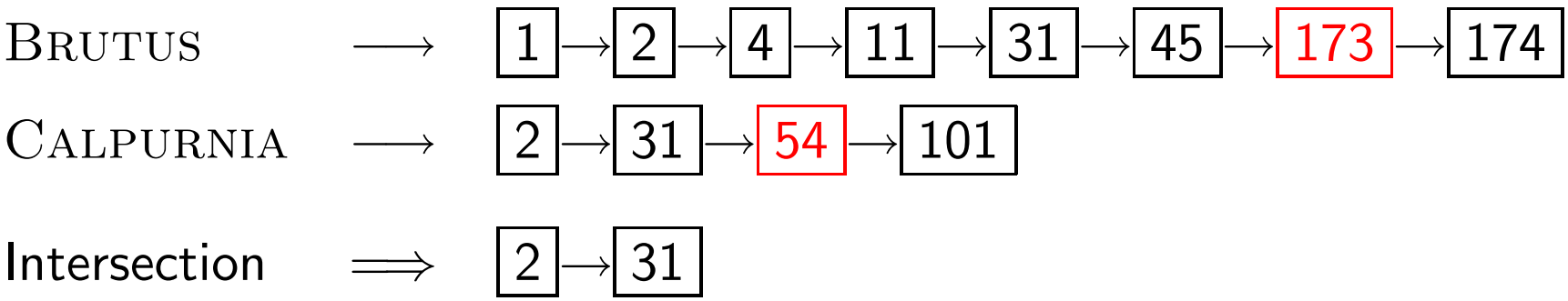
Intersecting two postings lists



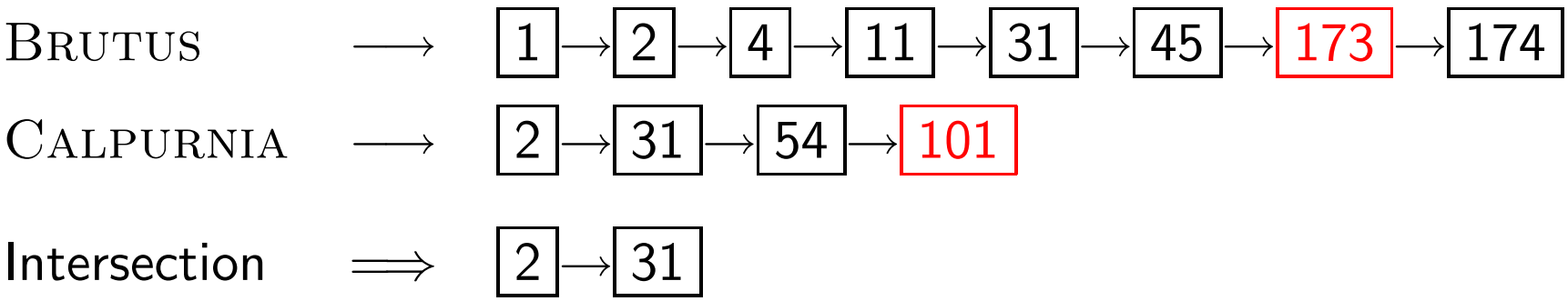
Intersecting two postings lists



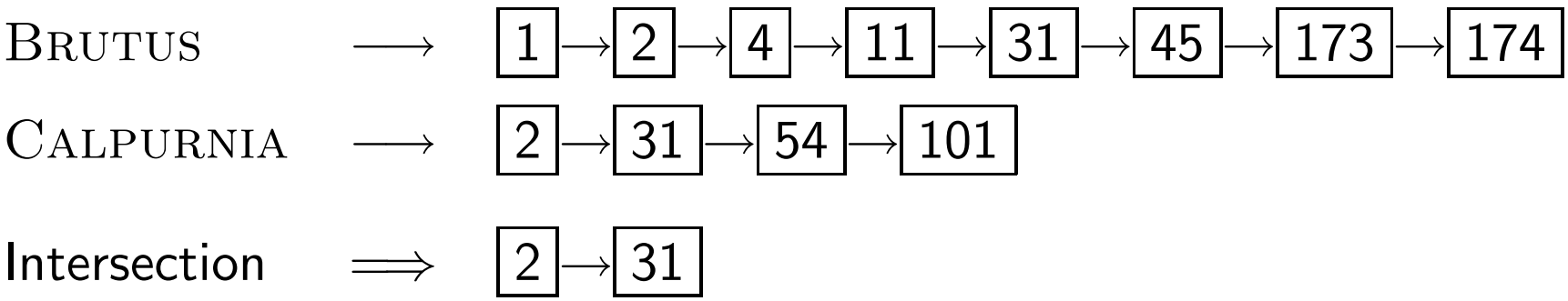
Intersecting two postings lists



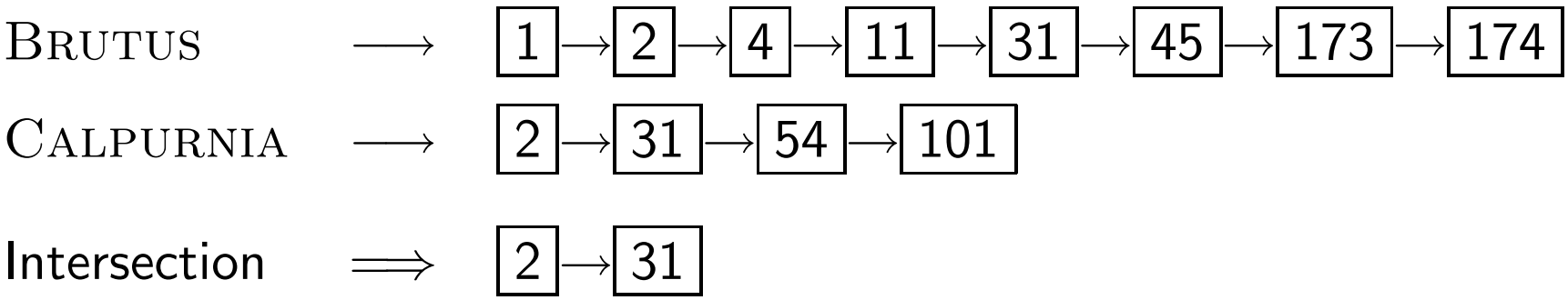
Intersecting two postings lists



Intersecting two postings lists



Intersecting two postings lists



- This is linear in the length of the postings lists.

Boolean queries

- The example was a simple conjunctive query . . .
- . . . the Boolean retrieval model can answer any query that is a Boolean expression.
  - Boolean queries are queries that use AND, OR and NOT to join query terms.
  - Views each document as a set of terms.
  - Is precise: Document matches condition or not.
- Primary commercial retrieval tool for 3 decades
- Many professional searchers (e.g., lawyers) still like Boolean queries.
  - You know exactly what you are getting.
- Many search systems you use are also Boolean: search system on your laptop, in your email reader, on the intranet etc
- So are we done? □

Outline

The Boolean model: Pros and Cons

- Key property: Documents either match or don't.
- Good for expert users with precise understanding of their needs and of the collection.
- Also good for applications: Applications can easily consume 1000s of results.
- Not good for the majority of users
- Most users are not capable 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.

Problem with Boolean search: Feast or famine

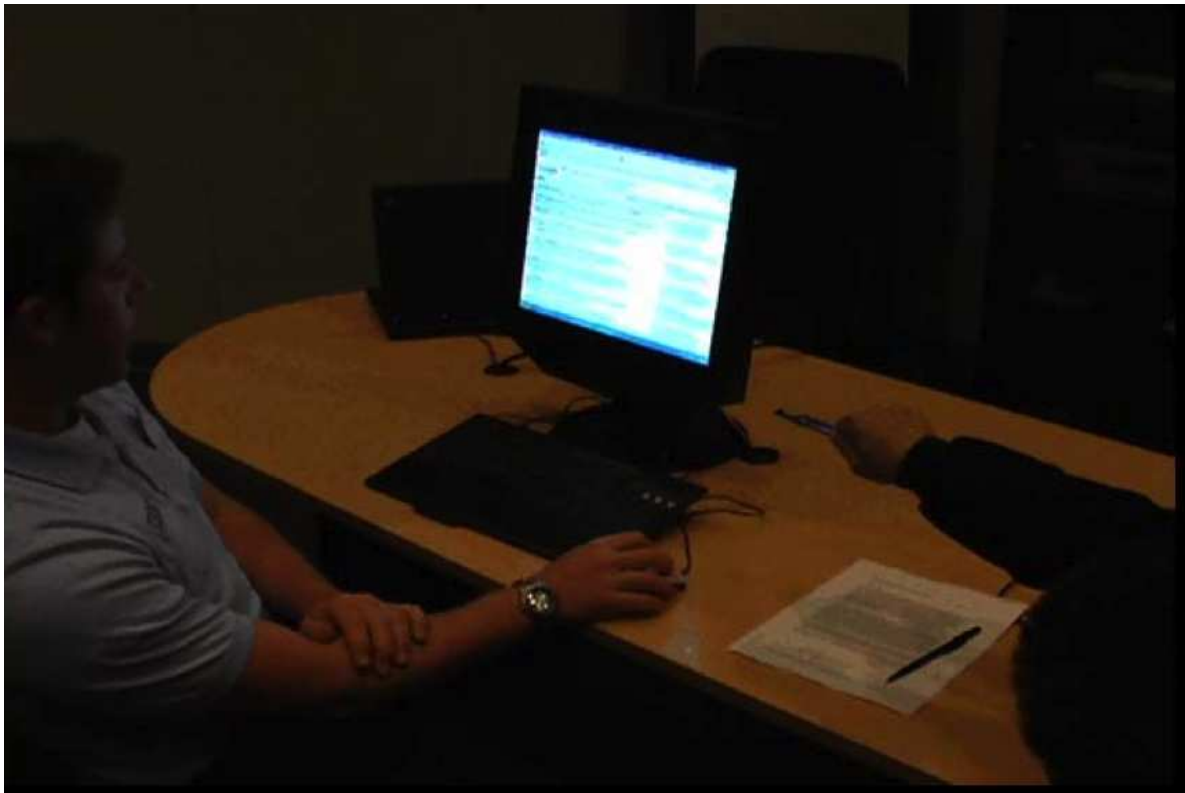
- Boolean queries often result in either too few (=0) or too many (1000s) results.
- Query 1 (boolean conjunction): [standard user dlink 650]
  - → 200,000 hits – feast
- Query 2 (boolean conjunction): [standard user dlink 650 no card found]
  - → 0 hits – famine
- In Boolean retrieval, it takes a lot of skill to come up with a query that produces a manageable number of hits.

Feast or famine: No problem in ranked retrieval

- With ranking, large result sets are not an issue.
- Just show the top 10 results and the user won't be overwhelmed
- Premise: the ranking algorithm works: More relevant results are ranked higher than less relevant results.

Empirical investigation of the effect of ranking

- How can we measure how important ranking is?
- Observe what searchers do when they are searching in a controlled setting
  - Videotape them
  - Ask them to "think aloud"
  - Interview them
  - Eye-track them
  - Time them
  - Record and count their clicks
- The following slides are from Dan Russell's 2007 JCDL talk
- Dan Russell was at the "Über Tech Lead for Search Quality & User Happiness" at Google. □



So.. Did you notice the FTD official site?

To be honest, I didn't even look at that.

At first I saw "from $20" and $20 is what I was looking for.

To be honest, 1800-flowers is what I'm familiar with and why I went there next even though I kind of assumed they wouldn't have $20 flowers

And you knew they were expensive?

I knew they were expensive but I thought "hey, maybe they've got some flowers for under $20 here…"

But you didn't notice the FTD?

No I didn't, actually… that's really funny.

Interview video

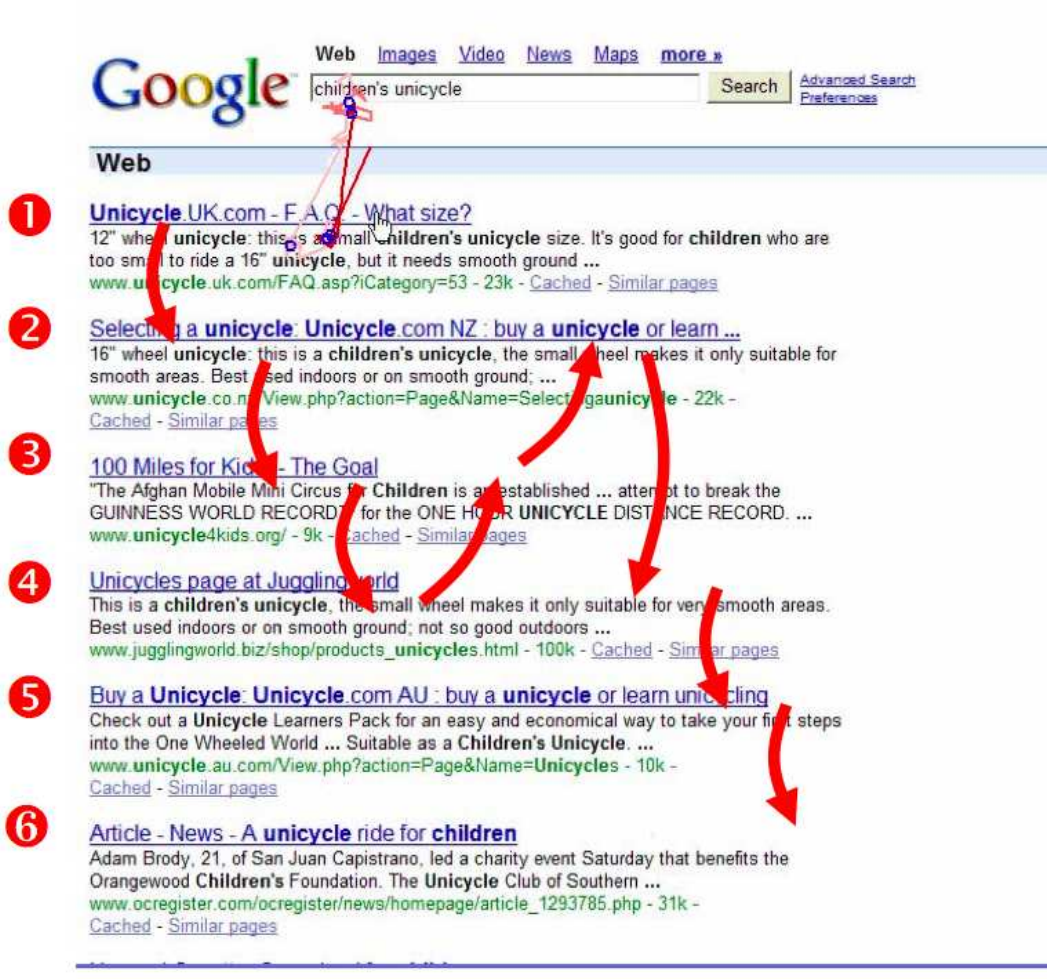
Rapidly scanning the results

Note scan pattern:

Page 3:
Result 1
Result 2
Result 3
Result 4
Result 3
Result 2
Result 4
Result 5
Result 6 <click>

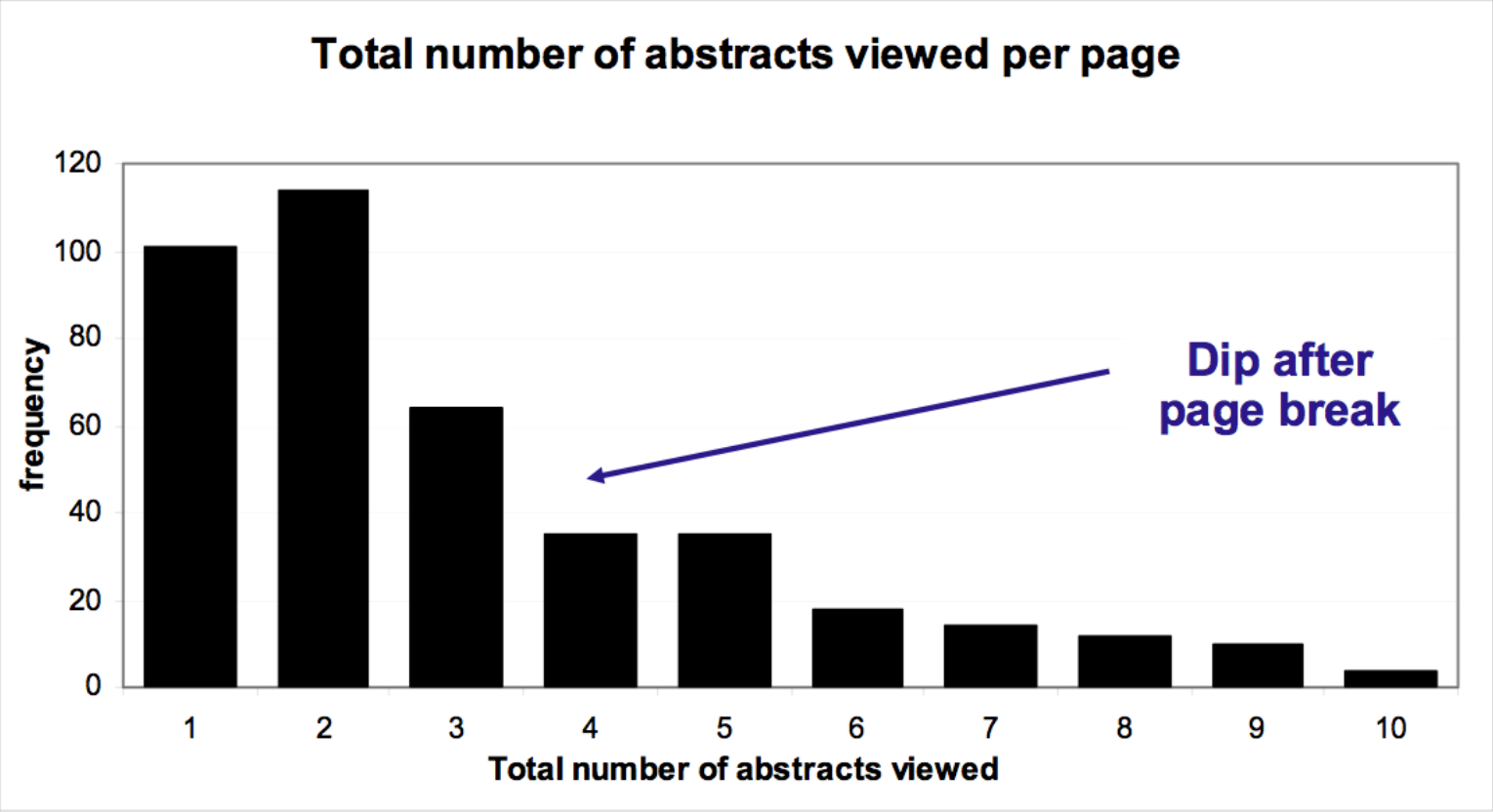
Q: Why do this?

A: What's learned later influences judgment of earlier content.



Google

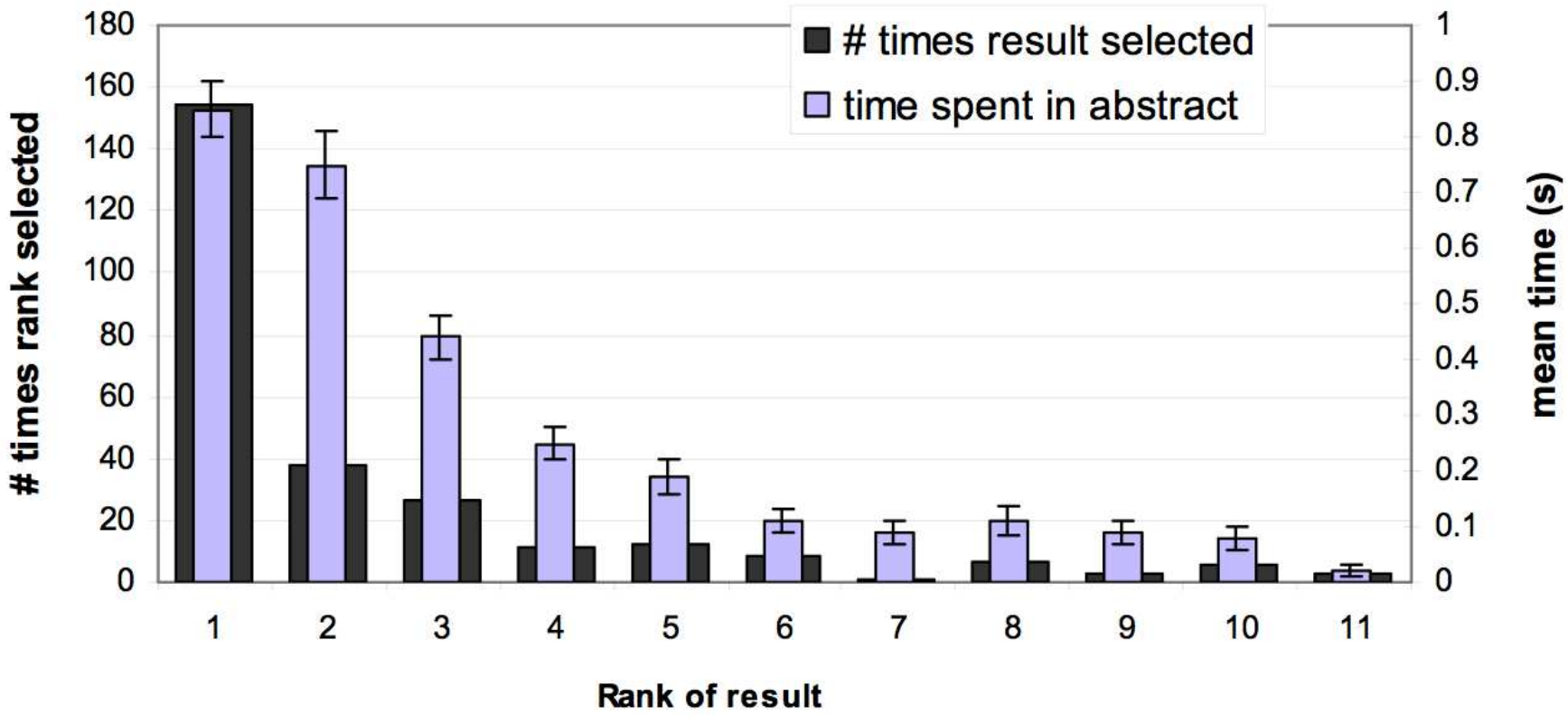
How many links do users view?



Mean: 3.07 Median/Mode: 2.00

Google

Looking vs. Clicking

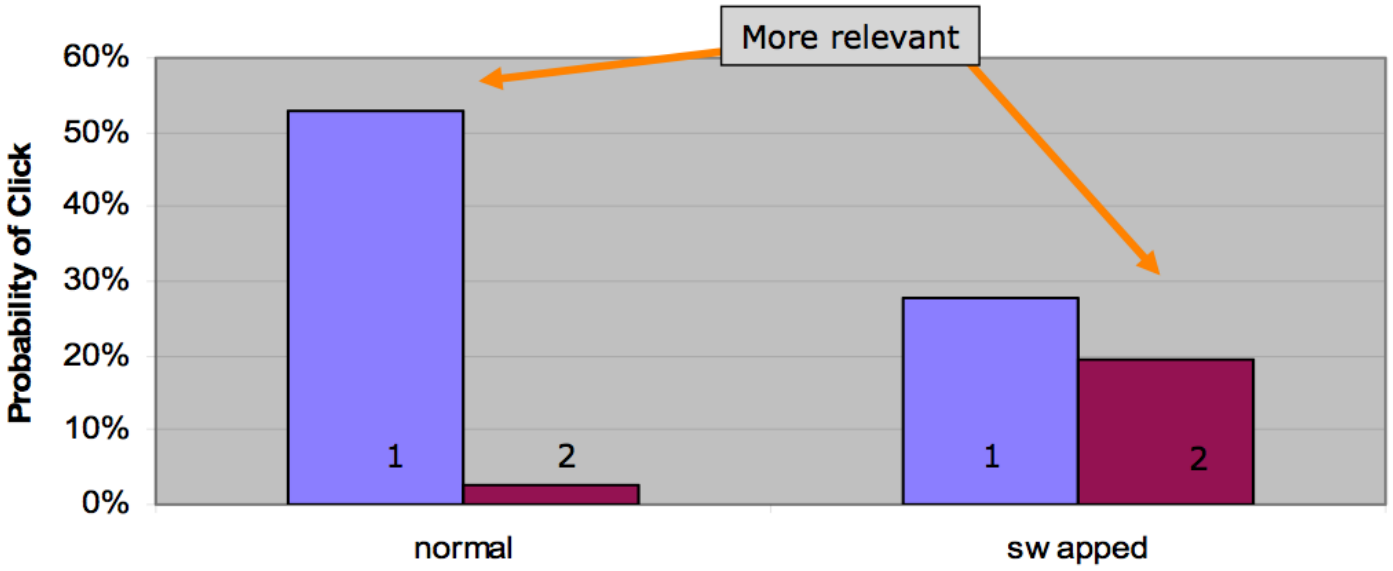


- Users view results one and two more often / thoroughly
- Users click most frequently on result one

Google

Presentation bias – reversed results

- Order of presentation influences where users look **AND** where they click



Google

Importance of ranking: Summary

- Viewing abstracts: Users are a lot more likely to read the abstracts of the top-ranked pages (1, 2, 3, 4) than the abstracts of the lower ranked pages (7, 8, 9, 10).
- Clicking: Distribution is even more skewed for clicking
- There is a very strong bias to click on the top-ranked page.
- Even if the top-ranked page is not relevant, 30% of users will click on it.
- $\rightarrow$ Getting the ranking right is very important.
- $\rightarrow$ Getting the top-ranked page right is most important. □

Take-away

- Boolean model and Inverted index: The Boolean model and the basic data structure of most IR systems
- Processing Boolean queries
- Why is Boolean retrieval not enough? or Why do we need ranked retrieval?

Resources

- Chapter 1 of Introduction to Information Retrieval
- Resources at http://informationretrieval.org/essir2011
  - List of useful information retrieval resources
  - Shakespeare search engine
  - Daniel Russell's home page

Exercise

Does Bing/Google use the Boolean model?
Does Spotlight use the Boolean model?

Does web search engines use the Boolean model?

- Default interpretation of a query by web search engines: $[w_1\ w_2\ \ldots w_n]$ is w_1 AND w_2 AND . . . AND w_n
- Cases where you get hits that do not contain one of the w_i:
  - anchor text
  - page contains variant of w_i (morphology, spelling correction, synonym)
  - long queries (n large)
  - conjunctive boolean query generates very few hits
- Simple Boolean vs. Ranking of result set
  - Simple Boolean retrieval returns matching documents in no particular order.
  - Google (and most well designed Boolean engines) rank the result set – they rank good hits (according to some estimator of relevance) higher than bad hits.

Introduction to Information Retrieval

http://informationretrieval.org

IIR 6&7: Vector Space Model

Hinrich Schütze

Institute for Natural Language Processing, University of Stuttgart

2011-08-29

Models and Methods

1. Boolean model and its limitations (30)
2. Vector space model (30)
3. Probabilistic models (30)
4. Language model-based retrieval (30)
5. Latent semantic indexing (30)
6. Learning to rank (30)

Take-away

- tf-idf weighting: Quick review of tf-idf weighting
- Vector space model – represents queries and documents in a high-dimensional space.
- Pivot normalization (or "pivoted document length normalization"): alternative to cosine normalization that removes a bias inherent in standard length normalization

Outline

Binary incidence matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	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	
...							

Each document is represented as a binary vector $\in \{0,1\}^{|V|}$. □

Binary incidence matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	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	
...							

Each document is represented as a binary vector $\in \{0,1\}^{|V|}$.

Count matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	157	73	0	0	0	1	
BRUTUS	4	157	0	2	0	0	
CAESAR	232	227	0	2	1	0	
CALPURNIA	0	10	0	0	0	0	
CLEOPATRA	57	0	0	0	0	0	
MERCY	2	0	3	8	5	8	
WORSER	2	0	1	1	1	5	
...							

Each document is now represented as a count vector $\in \mathbb{N}^{|V|}$. □

Count matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	157	73	0	0	0	1	
BRUTUS	4	157	0	2	0	0	
CAESAR	232	227	0	2	1	0	
CALPURNIA	0	10	0	0	0	0	
CLEOPATRA	57	0	0	0	0	0	
MERCY	2	0	3	8	5	8	
WORSER	2	0	1	1	1	5	
...							

Each document is now represented as a count vector $\in \mathbb{N}^{|V|}$.

Term frequency tf

- The term frequency $\text{tf}_{t,d}$ of term t in document d is defined as the number of times that t occurs in d.
- We want to rank documents according to query-document matching scores and use tf as a component in these matching scores.
- But how?
- Raw term frequency is not what we want because:
- A document with $\text{tf} = 10$ occurrences of the term is more relevant than a document with $\text{tf} = 1$ occurrence of the term.
- But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

Instead of raw frequency: Log frequency weighting

- The log frequency weight of term t in d is defined as follows

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d} & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

- $\mathrm{tf}_{t,d} \rightarrow w_{t,d}$:
  $0 \rightarrow 0$, $1 \rightarrow 1$, $2 \rightarrow 1.3$, $10 \rightarrow 2$, $1000 \rightarrow 4$, etc.
- Matching score for a document-query pair: sum over terms t in both q and d:
  tf-matching-score$(q, d) = \sum_{t \in q \cap d} (1 + \log \mathrm{tf}_{t,d})$

Frequency in document vs. frequency in collection

- In addition, to term frequency (the frequency of the term in the document) ...
- ...we also want to use the frequency of the term in the collection for weighting and ranking.

idf weight

- df_t is the document frequency, the number of documents that t occurs in.
- df_t is an inverse measure of the informativeness of term t.
- Inverse document frequency, idf_t, is a direct measure of the informativeness of the term.
- The idf weight of term t is defined as follows:

$$\text{idf}_t = \log_{10} \frac{N}{\text{df}_t}$$

(N is the number of documents in the collection.)

- $[\log N/\text{df}_t]$ instead of $[N/\text{df}_t]$ to "dampen" the effect of idf

Examples for idf

$$\text{idf}_t = \log_{10} \frac{1{,}000{,}000}{\text{df}_t}$$

term	df_t	idf_t
calpurnia	1	6
animal	100	4
sunday	1000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

Effect of idf on ranking

- idf gives high weights to rare terms like ARACHNOCENTRIC.
- idf gives low weights to frequent words like GOOD, INCREASE, and LINE.
- idf affects the ranking of documents for queries with at least two terms.
- For example, in the query "arachnocentric line", idf weighting increases the relative weight of ARACHNOCENTRIC and decreases the relative weight of LINE.
- idf has little effect on ranking for one-term queries. □

Summary: tf-idf weighting

- Assign a tf-idf weight for each term t in each document d: $w_{t,d} = (1 + \log \text{tf}_{t,d}) \cdot \log \frac{N}{\text{df}_t}$
- The tf-idf weight ...
  - ... increases with the number of occurrences within a document. (term frequency component)
  - ... increases with the rarity of the term in the collection. (inverse document frequency component)

Outline

Binary incidence matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	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	
...							

Each document is represented as a binary vector $\in \{0,1\}^{|V|}$.

Count matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	157	73	0	0	0	1	
BRUTUS	4	157	0	2	0	0	
CAESAR	232	227	0	2	1	0	
CALPURNIA	0	10	0	0	0	0	
CLEOPATRA	57	0	0	0	0	0	
MERCY	2	0	3	8	5	8	
WORSER	2	0	1	1	1	5	
...							

Each document is now represented as a count vector $\in \mathbb{N}^{|V|}$.

Binary → count → weight matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	5.25	3.18	0.0	0.0	0.0	0.35	
BRUTUS	1.21	6.10	0.0	1.0	0.0	0.0	
CAESAR	8.59	2.54	0.0	1.51	0.25	0.0	
CALPURNIA	0.0	1.54	0.0	0.0	0.0	0.0	
CLEOPATRA	2.85	0.0	0.0	0.0	0.0	0.0	
MERCY	1.51	0.0	1.90	0.12	5.25	0.88	
WORSER	1.37	0.0	0.11	4.15	0.25	1.95	
...							

Each document is now represented as a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$.

Binary → count → weight matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	5.25	3.18	0.0	0.0	0.0	0.35	
BRUTUS	1.21	6.10	0.0	1.0	0.0	0.0	
CAESAR	8.59	2.54	0.0	1.51	0.25	0.0	
CALPURNIA	0.0	1.54	0.0	0.0	0.0	0.0	
CLEOPATRA	2.85	0.0	0.0	0.0	0.0	0.0	
MERCY	1.51	0.0	1.90	0.12	5.25	0.88	
WORSER	1.37	0.0	0.11	4.15	0.25	1.95	
...							

Each document is now represented as a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$.

Documents as vectors

- Each document is now represented as a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$.
- So we have a $|V|$-dimensional real-valued vector space.
- Terms are axes of the space.
- Documents are points or vectors in this space.
- Very high-dimensional: tens of millions of dimensions when you apply this to web search engines
- Each vector is very sparse - most entries are zero.

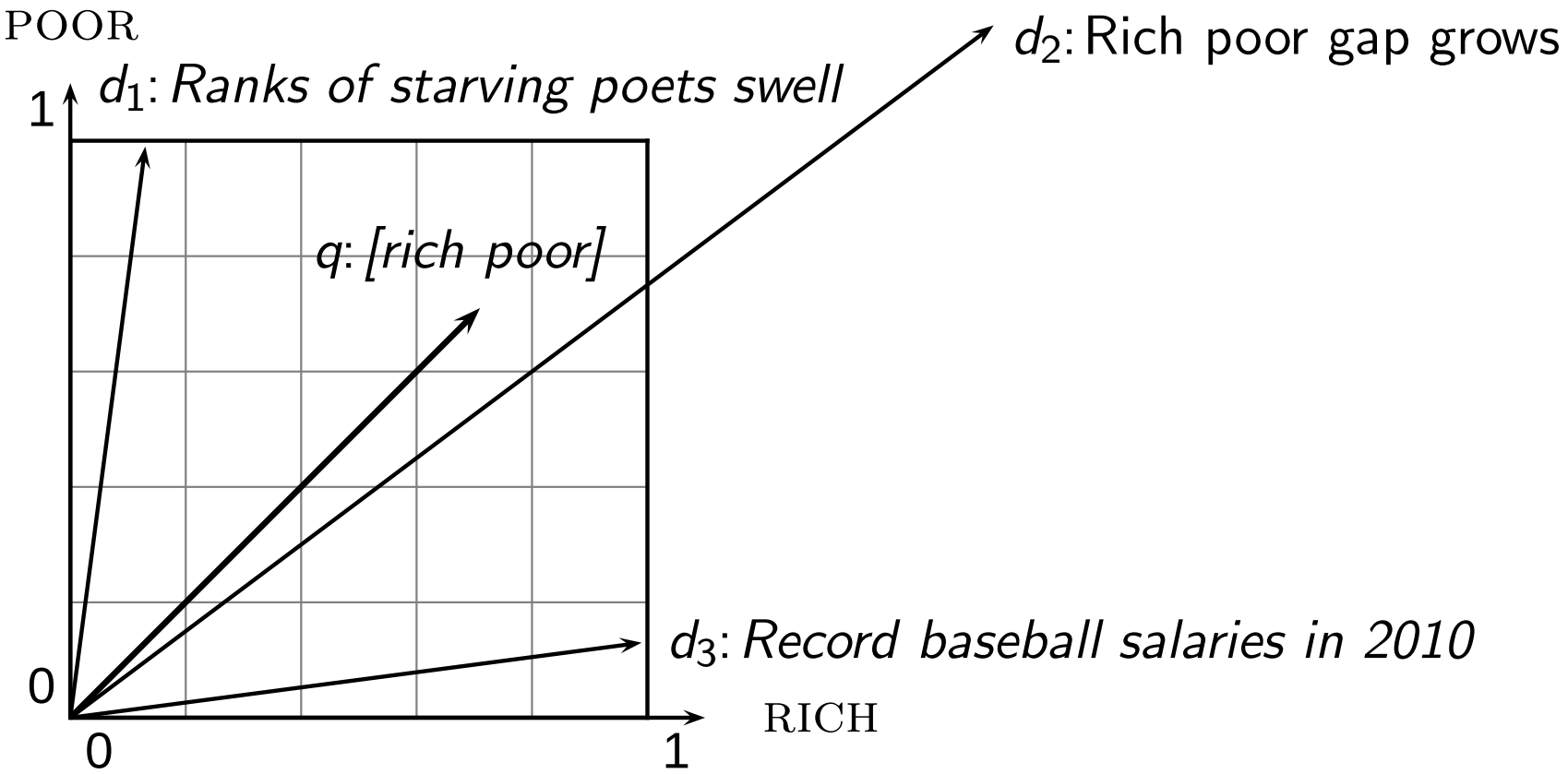
Queries as vectors

- Key idea 1: do the same for queries: represent them as vectors in the high-dimensional space
- Key idea 2: Rank documents according to their proximity to the query
- proximity = similarity
- proximity $\approx$ negative distance
- Recall: We're doing this because we want to get away from the you're-either-in-or-out, feast-or-famine Boolean model.
- Instead: rank relevant documents higher than nonrelevant documents

How do we formalize vector space similarity?

- First cut: (negative) distance between two points
- (= distance between the end points of the two vectors)
- Euclidean distance?
- Euclidean distance is a bad idea . . .
- . . . because Euclidean distance is large for vectors of different lengths.

Why distance is a bad idea



The Euclidean distance of $\vec{q}$ and $\vec{d}_2$ is large although the distribution of terms in the query q and the distribution of terms in the document d_2 are very similar. □

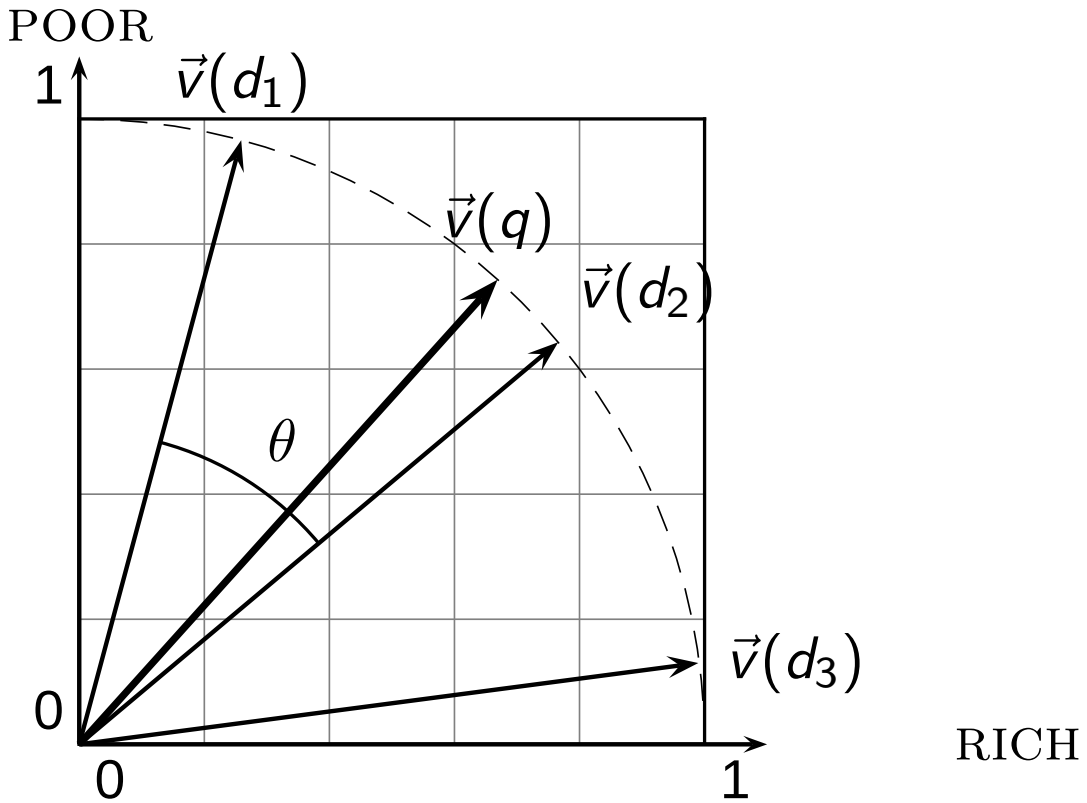
Use angle instead of distance

- Rank documents according to angle with query
- The following two notions are equivalent.
  - Rank documents according to the angle between query and document in decreasing order
  - Rank documents according to cosine(query,document) in increasing order
- Cosine is a monotonically decreasing function of the angle for the interval $[0°, 180°]$
- $\longrightarrow$ do ranking according to cosine

Cosine similarity between query and document

- $$\cos(\vec{q}, \vec{d}) = \text{SIM}(\vec{q}, \vec{d}) = \frac{\vec{q}}{|\vec{q}|} \cdot \frac{\vec{d}}{|\vec{d}|} = \sum_{i=1}^{|V|} \frac{q_i}{|\vec{q}|} \cdot \frac{d_i}{|\vec{d}|}$$
- q_i is the tf-idf weight of term i in the query.
- d_i is the tf-idf weight of term i in the document.
- $|\vec{q}|$ and $|\vec{d}|$ are the lengths of $\vec{q}$ and $\vec{d}$.
- This is the cosine similarity of $\vec{q}$ and $\vec{d}$ . . . . . or, equivalently, the cosine of the angle between $\vec{q}$ and $\vec{d}$.
- cosine similarity = dot product of length-normalized vectors

Cosine similarity illustrated



Components of tf-idf weighting

Term frequency		Document frequency		Normalization	
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1 + \log(\mathrm{tf}_{t,d})$	t (idf)	$\log \frac{N}{\mathrm{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_M^2}}$
a (augmented)	$0.5 + \frac{0.5 \times \mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N-\mathrm{df}_t}{\mathrm{df}_t}\}$	u (pivoted unique)	$1/u$
b (boolean)	$\begin{cases} 1 & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/CharLength^{\alpha}$, $\alpha < 1$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t \in d}(\mathrm{tf}_{t,d}))}$				

Components of tf-idf weighting

Term frequency		Document frequency		Normalization	
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1 + \log(\mathrm{tf}_{t,d})$	t (idf)	$\log \frac{N}{\mathrm{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_M^2}}$
a (augmented)	$0.5 + \frac{0.5 \times \mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N-\mathrm{df}_t}{\mathrm{df}_t}\}$	u (pivoted unique)	$1/u$
b (boolean)	$\begin{cases} 1 & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/\mathit{CharLength}^{\alpha}$, $\alpha < 1$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t \in d}(\mathrm{tf}_{t,d}))}$				

Best known combination of weighting options

tf-idf example

- We often use different weightings for queries and documents.
- Notation: ddd.qqq
- Example: lnc.ltn
- document: logarithmic tf, no df weighting, cosine normalization
- query: logarithmic tf, idf, no normalization
- Isn't it bad to not idf-weight the document?
- Example query: "best car insurance"
- Example document: "car insurance auto insurance"

tf-idf example: lnc.ltn

Query: "best car insurance". Document: "car insurance auto insurance".

word	query					document				product
	tf-raw	tf-wght	df	idf	weight	tf-raw	tf-wght	weight	n'lized	
auto										
best										
car										
insurance										

Key to columns: tf-raw: raw (unweighted) term frequency, tf-wght: logarithmically weighted term frequency, df: document frequency, idf: inverse document frequency, weight: the final weight of the term in the query or document, n'lized: document weights after cosine normalization, product: the product of final query weight and final document weight

tf-idf example: lnc.ltn

Query: "best car insurance". Document: "car insurance auto insurance".

word	query					document				product
	tf-raw	tf-wght	df	idf	weight	tf-raw	tf-wght	weight	n'lized	
auto	0									
best	1									
car	1									
insurance	1									

Key to columns: tf-raw: raw (unweighted) term frequency, tf-wght: logarithmically weighted term frequency, df: document frequency, idf: inverse document frequency, weight: the final weight of the term in the query or document, n'lized: document weights after cosine normalization, product: the product of final query weight and final document weight

tf-idf example: lnc.ltn

Query: "best car insurance". Document: "car insurance auto insurance".

word	query					document				product
	tf-raw	tf-wght	df	idf	weight	tf-raw	tf-wght	weight	n'lized	
auto	0					1				
best	1					0				
car	1					1				
insurance	1					2				

Key to columns: tf-raw: raw (unweighted) term frequency, tf-wght: logarithmically weighted term frequency, df: document frequency, idf: inverse document frequency, weight: the final weight of the term in the query or document, n'lized: document weights after cosine normalization, product: the product of final query weight and final document weight

tf-idf example: lnc.ltn

Query: "best car insurance". Document: "car insurance auto insurance".

word	query					document				product
	tf-raw	tf-wght	df	idf	weight	tf-raw	tf-wght	weight	n'lized	
auto	0	0				1				
best	1	1				0				
car	1	1				1				
insurance	1	1				2				

Key to columns: tf-raw: raw (unweighted) term frequency, tf-wght: logarithmically weighted term frequency, df: document frequency, idf: inverse document frequency, weight: the final weight of the term in the query or document, n'lized: document weights after cosine normalization, product: the product of final query weight and final document weight

tf-idf example: lnc.ltn

Query: "best car insurance". Document: "car insurance auto insurance".

word	query					document				product
	tf-raw	tf-wght	df	idf	weight	tf-raw	tf-wght	weight	n'lized	
auto	0	0				1	1			
best	1	1				0	0			
car	1	1				1	1			
insurance	1	1				2	1.3			

Key to columns: tf-raw: raw (unweighted) term frequency, tf-wght: logarithmically weighted term frequency, df: document frequency, idf: inverse document frequency, weight: the final weight of the term in the query or document, n'lized: document weights after cosine normalization, product: the product of final query weight and final document weight

tf-idf example: lnc.ltn

Query: "best car insurance". Document: "car insurance auto insurance".

word	query					document				product
	tf-raw	tf-wght	df	idf	weight	tf-raw	tf-wght	weight	n'lized	
auto	0	0	5000			1	1			
best	1	1	50000			0	0			
car	1	1	10000			1	1			
insurance	1	1	1000			2	1.3			

Key to columns: tf-raw: raw (unweighted) term frequency, tf-wght: logarithmically weighted term frequency, df: document frequency, idf: inverse document frequency, weight: the final weight of the term in the query or document, n'lized: document weights after cosine normalization, product: the product of final query weight and final document weight

tf-idf example: lnc.ltn

Query: "best car insurance". Document: "car insurance auto insurance".

word	query					document				product
	tf-raw	tf-wght	df	idf	weight	tf-raw	tf-wght	weight	n'lized	
auto	0	0	5000	2.3		1	1			
best	1	1	50000	1.3		0	0			
car	1	1	10000	2.0		1	1			
insurance	1	1	1000	3.0		2	1.3			

Key to columns: tf-raw: raw (unweighted) term frequency, tf-wght: logarithmically weighted term frequency, df: document frequency, idf: inverse document frequency, weight: the final weight of the term in the query or document, n'lized: document weights after cosine normalization, product: the product of final query weight and final document weight

tf-idf example: lnc.ltn

Query: "best car insurance". Document: "car insurance auto insurance".

word	query					document				product
	tf-raw	tf-wght	df	idf	weight	tf-raw	tf-wght	weight	n'lized	
auto	0	0	5000	2.3	0	1	1			
best	1	1	50000	1.3	1.3	0	0			
car	1	1	10000	2.0	2.0	1	1			
insurance	1	1	1000	3.0	3.0	2	1.3			

Key to columns: tf-raw: raw (unweighted) term frequency, tf-wght: logarithmically weighted term frequency, df: document frequency, idf: inverse document frequency, weight: the final weight of the term in the query or document, n'lized: document weights after cosine normalization, product: the product of final query weight and final document weight

tf-idf example: lnc.ltn

Query: "best car insurance". Document: "car insurance auto insurance".

word	query					document				product
	tf-raw	tf-wght	df	idf	weight	tf-raw	tf-wght	weight	n'lized	
auto	0	0	5000	2.3	0	1	1			
best	1	1	50000	1.3	1.3	0	0			
car	1	1	10000	2.0	2.0	1	1			
insurance	1	1	1000	3.0	3.0	2	1.3			

Key to columns: tf-raw: raw (unweighted) term frequency, tf-wght: logarithmically weighted term frequency, df: document frequency, idf: inverse document frequency, weight: the final weight of the term in the query or document, n'lized: document weights after cosine normalization, product: the product of final query weight and final document weight

tf-idf example: lnc.ltn

Query: "best car insurance". Document: "car insurance auto insurance".

word	query					document				product
	tf-raw	tf-wght	df	idf	weight	tf-raw	tf-wght	weight	n'lized	
auto	0	0	5000	2.3	0	1	1	1		
best	1	1	50000	1.3	1.3	0	0	0		
car	1	1	10000	2.0	2.0	1	1	1		
insurance	1	1	1000	3.0	3.0	2	1.3	1.3		

Key to columns: tf-raw: raw (unweighted) term frequency, tf-wght: logarithmically weighted term frequency, df: document frequency, idf: inverse document frequency, weight: the final weight of the term in the query or document, n'lized: document weights after cosine normalization, product: the product of final query weight and final document weight

tf-idf example: lnc.ltn

Query: "best car insurance". Document: "car insurance auto insurance".

word	query					document				product
	tf-raw	tf-wght	df	idf	weight	tf-raw	tf-wght	weight	n'lized	
auto	0	0	5000	2.3	0	1	1	1	0.52	
best	1	1	50000	1.3	1.3	0	0	0	0	
car	1	1	10000	2.0	2.0	1	1	1	0.52	
insurance	1	1	1000	3.0	3.0	2	1.3	1.3	0.68	

Key to columns: tf-raw: raw (unweighted) term frequency, tf-wght: logarithmically weighted term frequency, df: document frequency, idf: inverse document frequency, weight: the final weight of the term in the query or document, n'lized: document weights after cosine normalization, product: the product of final query weight and final document weight

$\sqrt{1^2 + 0^2 + 1^2 + 1.3^2} \approx 1.92$
$1/1.92 \approx 0.52$
$1.3/1.92 \approx 0.68$

tf-idf example: lnc.ltn

Query: "best car insurance". Document: "car insurance auto insurance".

word	query					document				product
	tf-raw	tf-wght	df	idf	weight	tf-raw	tf-wght	weight	n'lized	
auto	0	0	5000	2.3	0	1	1	1	0.52	0
best	1	1	50000	1.3	1.3	0	0	0	0	0
car	1	1	10000	2.0	2.0	1	1	1	0.52	1.04
insurance	1	1	1000	3.0	3.0	2	1.3	1.3	0.68	2.04

Key to columns: tf-raw: raw (unweighted) term frequency, tf-wght: logarithmically weighted term frequency, df: document frequency, idf: inverse document frequency, weight: the final weight of the term in the query or document, n'lized: document weights after cosine normalization, product: the product of final query weight and final document weight

tf-idf example: lnc.ltn

Query: "best car insurance". Document: "car insurance auto insurance".

word	query					document				product
	tf-raw	tf-wght	df	idf	weight	tf-raw	tf-wght	weight	n'lized	
auto	0	0	5000	2.3	0	1	1	1	0.52	0
best	1	1	50000	1.3	1.3	0	0	0	0	0
car	1	1	10000	2.0	2.0	1	1	1	0.52	1.04
insurance	1	1	1000	3.0	3.0	2	1.3	1.3	0.68	2.04

Key to columns: tf-raw: raw (unweighted) term frequency, tf-wght: logarithmically weighted term frequency, df: document frequency, idf: inverse document frequency, weight: the final weight of the term in the query or document, n'lized: document weights after cosine normalization, product: the product of final query weight and final document weight

Final similarity score between query and document: $\sum_i w_{qi} \cdot w_{di} = 0 + 0 + 1.04 + 2.04 = 3.08$

Outline

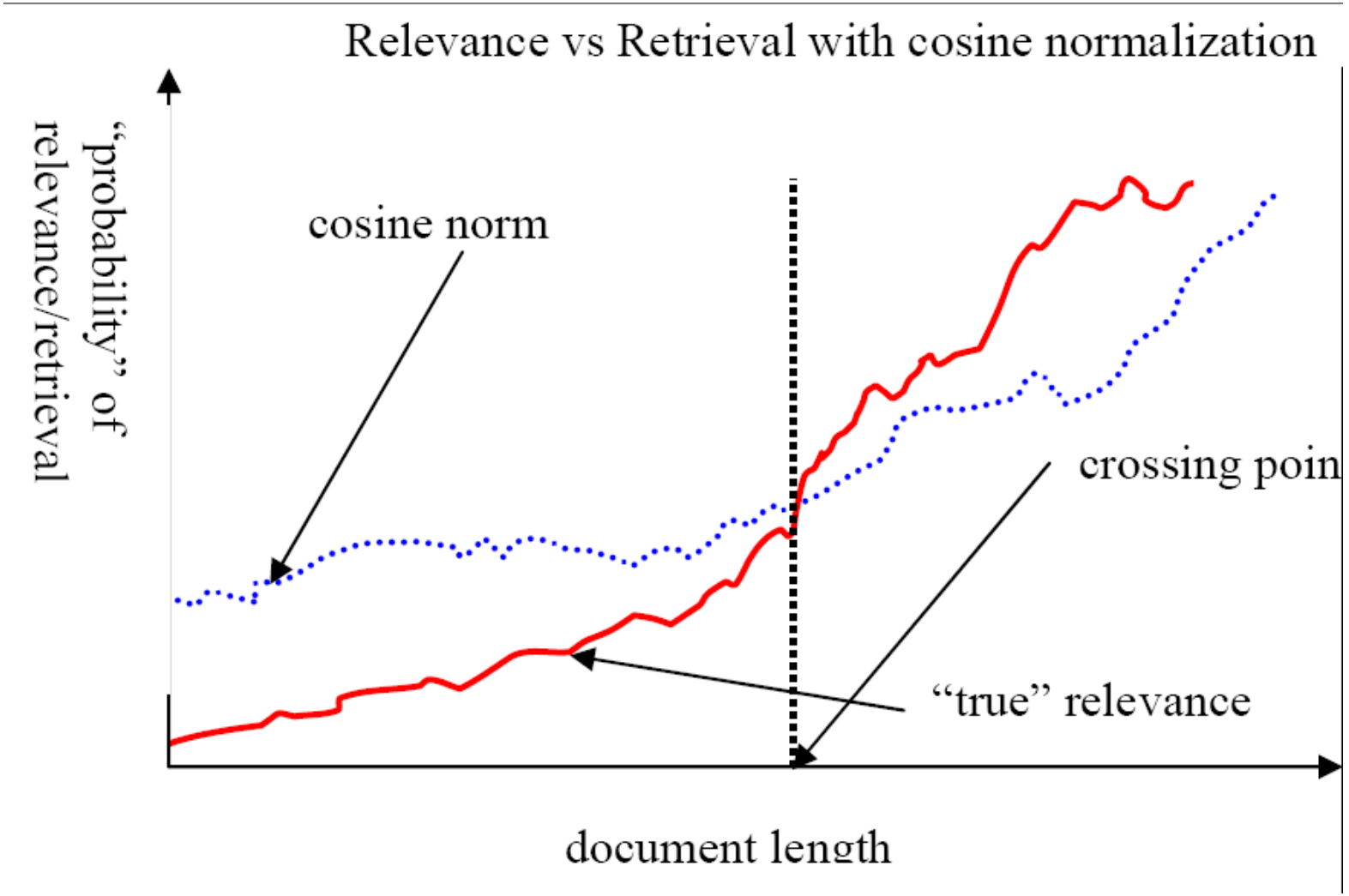
A problem for cosine normalization

- Query q: "anti-doping rules Beijing 2008 olympics"
- Compare three documents
  - d_1: a short document on anti-doping rules at 2008 Olympics
  - d_2: a long document that consists of a copy of d_1 and 5 other news stories, all on topics different from Olympics/anti-doping
  - d_3: a short document on anti-doping rules at the 2004 Athens Olympics
- What ranking do we expect in the vector space model? □

Pivot normalization

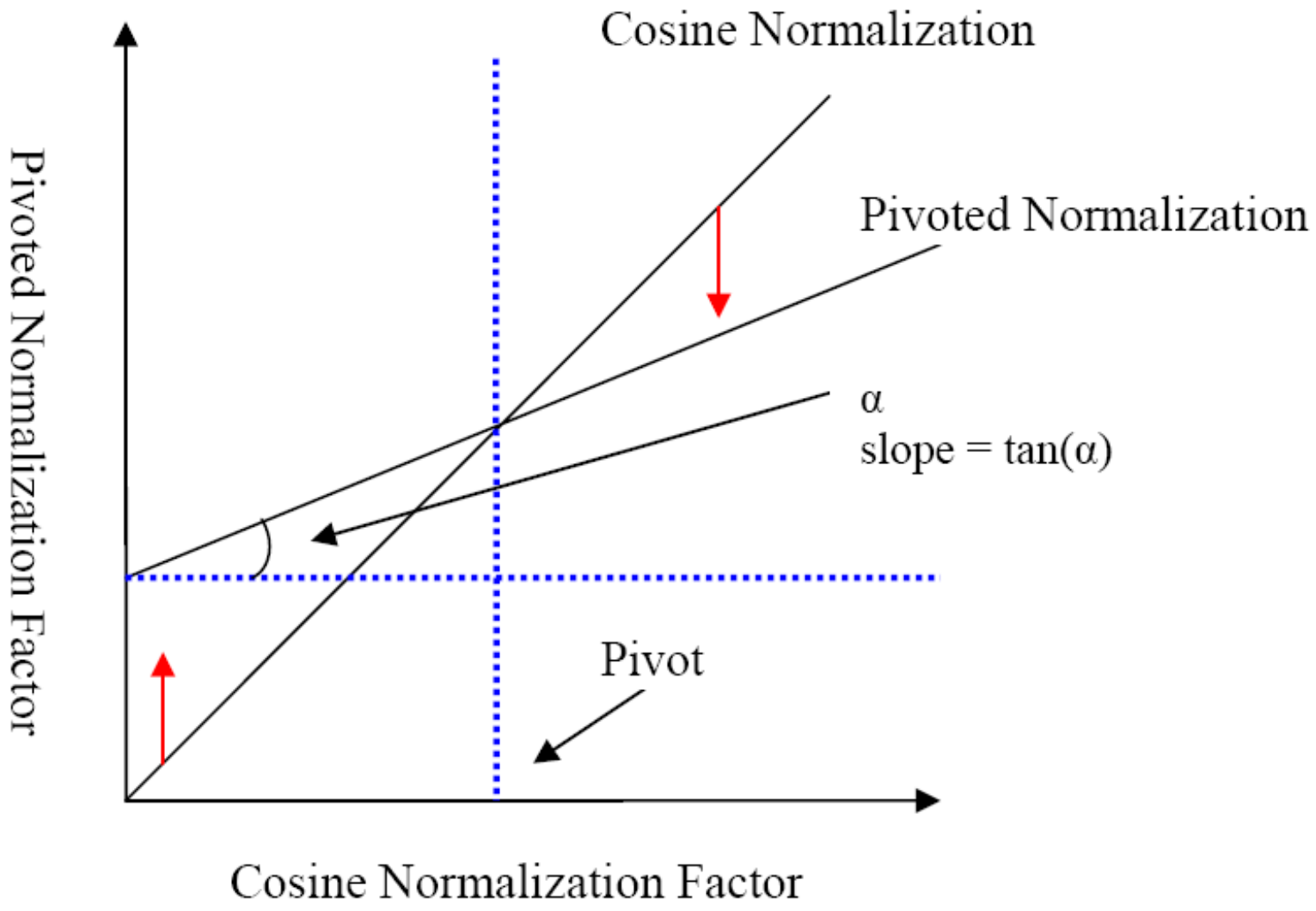
- Cosine normalization produces weights that are too large for short documents and too small for long documents (on average).
- Adjust cosine normalization by linear adjustment: "turning" the average normalization on the pivot
- Effect: Similarities of short documents with query decrease; similarities of long documents with query increase.
- This removes the unfair advantage that short documents have.
- Singhal's study is also interesting from the point of view of methodology.

Predicted and true probability of relevance



source: Lillian Lee

Pivot normalization



source:
Lillian Lee

Pivoted normalization: Amit Singhal's experiments

Cosine	Pivoted Cosine Normalization				
	Slope				
	0.60	0.65	0.70	**0.75**	0.80
6,526	6,342	6,458	6,574	**6,629**	6,671
0.2840	0.3024	0.3097	0.3144	**0.3171**	0.3162
Improvement	+ 6.5%	+ 9.0%	+10.7%	**+11.7%**	+11.3%

(relevant documents retrieved and (change in) average precision)

Summary: Ranked retrieval in the vector space model

- Represent each document as a weighted tf-idf vector
- Represent the query as a weighted tf-idf vector
- Compute the cosine similarity between the query vector and each document vector
  - Alternatively, use pivot normalization
- Rank documents with respect to the query
- Return the top K (e.g., $K = 10$) to the user

Take-away

- tf-idf weighting: Quick review of tf-idf weighting
- Vector space model – represents queries and documents in a high-dimensional space.
- Pivot normalization (or "pivoted document length normalization"): alternative to cosine normalization that removes a bias inherent in standard length normalization

Resources

- Chapters 6 and 7 of Introduction to Information Retrieval
- Resources at http://informationretrieval.org/essir2011
  - Gerard Salton (main proponent of vector space model in 70s, 80s, 90s)
  - Exploring the similarity space (Moffat and Zobel, 2005)
  - Pivot normalization (original paper)

Introduction to Information Retrieval

http://informationretrieval.org

IIR 11: Probabilistic Information Retrieval

Hinrich Schütze

Institute for Natural Language Processing, Universität Stuttgart

2011-08-29

Models and Methods

1. Boolean model and its limitations (30)
2. Vector space model (30)
3. Probabilistic models (30)
4. Language model-based retrieval (30)
5. Latent semantic indexing (30)
6. Learning to rank (30)

Take-away

- Probabilistic approach to IR: Introduction
- Binary independence model or BIM – the first influential probabilistic model

Take-away

- Probabilistic approach to IR: Introduction
- Binary independence model or BIM – the first influential probabilistic model
- Okapi BM25, a more modern, better performing probabilistic model

Outline

Probabilistic approach to IR

- The adhoc retrieval problem: Given a user information need and a collection of documents, the IR system must determine how well the documents satisfy the query.
- The IR system has an uncertain understanding of the user query . . .
- . . . and makes an uncertain guess of whether a document satisfies the query.
- Probability theory provides a principled foundation for such reasoning under uncertainty.
- Probabilistic IR models exploit this foundation to estimate how likely it is that a document is relevant to a query.

Probabilistic vs. vector space model

- Vector space model: rank documents according to similarity to query.
- The notion of similarity does not translate directly into an assessment of "is the document a good document to give to the user or not?"
- The most similar document can be highly relevant or completely nonrelevant.
- Probability theory is arguably a cleaner formalization of what we really want an IR system to do: give relevant documents to the user.

Probabilistic IR models at a glance

- Classical probabilistic retrieval models
  - Binary Independence Model
  - Okapi BM25
- Bayesian networks for text retrieval
  - Don't have time for this
- Language model approach to IR
  - Important recent work, will be covered in the next lecture

Probabilistic IR and ranking

- Ranked retrieval setup: the user issues a query, and a ranked list of documents is returned.
- How can we rank probabilistically?
- Let $R_{d,q}$ be a random dichotomous variable, such that
  - $R_{d,q} = 1$ if document d is relevant w.r.t query q
  - $R_{d,q} = 0$ otherwise
- (This is a binary notion of relevance.)
- Probabilistic ranking orders documents decreasingly by their estimated probability of relevance w.r.t. query: $P(R = 1|d, q)$
- How can we justify this way of proceeding? □

Probability Ranking Principle (PRP)

If the retrieved documents are ranked decreasingly on their probability of relevance (w.r.t a query), then the effectiveness of the system will be the best that is obtainable.

Fundamental assumption: the relevance of each document is independent of the relevance of other documents. □

Outline

Binary Independence Model (BIM)

- Binary: documents and queries represented as binary term incidence vectors
- Independence: terms are independent of each other (not true, but works in practice – naive assumption of Naive Bayes models)

Binary incidence matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	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	
...							

Each document is represented as a binary vector $\in \{0,1\}^{|V|}$.

Bayes' rule

- $$P(R=1|\vec{x},\vec{q}) = \frac{P(\vec{x}|R=1,\vec{q})P(R=1|\vec{q})}{P(\vec{x}|\vec{q})}$$
  $$P(R=0|\vec{x},\vec{q}) = \frac{P(\vec{x}|R=0,\vec{q})P(R=0|\vec{q})}{P(\vec{x}|\vec{q})}$$
- (Recall that document and query are modeled as term incidence vectors: $\vec{x}$ and $\vec{q}$.)
- $P(\vec{x}|R=1,\vec{q})$ and $P(\vec{x}|R=0,\vec{q})$: probability that if a relevant or nonrelevant document is retrieved, then that document's representation is $\vec{x}$
- Use statistics about the document collection to estimate these probabilities

Priors

$P(R|d,q)$ is modeled using term incidence vectors as $P(R|\vec{x},\vec{q})$

$$P(R=1|\vec{x},\vec{q}) = \frac{P(\vec{x}|R=1,\vec{q})P(R=1|\vec{q})}{P(\vec{x}|\vec{q})}$$

$$P(R=0|\vec{x},\vec{q}) = \frac{P(\vec{x}|R=0,\vec{q})P(R=0|\vec{q})}{P(\vec{x}|\vec{q})}$$

- $P(R=1|\vec{q})$ and $P(R=0|\vec{q})$: prior probability of retrieving a relevant or nonrelevant document for a query $\vec{q}$
- Estimate $P(R=1|\vec{q})$ and $P(R=0|\vec{q})$ from percentage of relevant documents in the collection

Ranking according to odds

- We said that we're going to rank documents according to $P(R = 1|\vec{x}, \vec{q})$
- Easier: rank documents by their odds of relevance (gives same ranking)

$$O(R|\vec{x}, \vec{q}) = \frac{P(R = 1|\vec{x}, \vec{q})}{P(R = 0|\vec{x}, \vec{q})} = \frac{\frac{P(R=1|\vec{q})P(\vec{x}|R=1,\vec{q})}{P(\vec{x}|\vec{q})}}{\frac{P(R=0|\vec{q})P(\vec{x}|R=0,\vec{q})}{P(\vec{x}|\vec{q})}}$$

$$= \frac{P(R = 1|\vec{q})}{P(R = 0|\vec{q})} \cdot \frac{P(\vec{x}|R = 1, \vec{q})}{P(\vec{x}|R = 0, \vec{q})}$$

- $\frac{P(R=1|\vec{q})}{P(R=0|\vec{q})}$ is a constant for a given query - can be ignored □

Naive Bayes conditional independence assumption

Now we make the Naive Bayes conditional independence assumption that the presence or absence of a word in a document is independent of the presence or absence of any other word (given the query):

$$\frac{P(\vec{x}|R=1,\vec{q})}{P(\vec{x}|R=0,\vec{q})} = \frac{\prod_{t=1}^{M} P(x_t|R=1,\vec{q})}{\prod_{t=1}^{M} P(x_t|R=0,\vec{q})}$$

So:

$$O(R|\vec{x},\vec{q}) \propto \prod_{t=1}^{M} \frac{P(x_t|R=1,\vec{q})}{P(x_t|R=0,\vec{q})}$$

Separating terms in the document vs. not

Since each x_t is either 0 or 1, we can separate the terms:

$$O(R|\vec{x},\vec{q}) \propto \prod_{t:x_t=1} \frac{P(x_t=1|R=1,\vec{q})}{P(x_t=1|R=0,\vec{q})} \prod_{t:x_t=0} \frac{P(x_t=0|R=1,\vec{q})}{P(x_t=0|R=0,\vec{q})}$$

Definition of p_t and u_t

- Let $p_t = P(x_t = 1|R = 1, \vec{q})$ be the probability of a term appearing in relevant document.
- Let $u_t = P(x_t = 1|R = 0, \vec{q})$ be the probability of a term appearing in a nonrelevant document.
- Can be displayed as contingency table:

		$R = 1$	$R = 0$
term present	$x_t = 1$	p_t	u_t
term absent	$x_t = 0$	$1 - p_t$	$1 - u_t$

- $$O(R|\vec{x}, \vec{q}) \propto \prod_{t:x_t=1} \frac{p_t}{u_t} \prod_{t:x_t=0} \frac{1 - p_t}{1 - u_t}$$

Dropping terms that don't occur in the query

- Additional simplifying assumption: If $q_t = 0$, then $p_t = u_t$
  - A term not occurring in the query is equally likely to occur in relevant and nonrelevant documents.
- Now we need only to consider terms in the products that appear in the query:

$$O(R|\vec{x},\vec{q}) \propto \prod_{t:x_t=1} \frac{p_t}{u_t} \prod_{t:x_t=0} \frac{1-p_t}{1-u_t} \approx \prod_{t:x_t=q_t=1} \frac{p_t}{u_t} \prod_{t:x_t=0,q_t=1} \frac{1-p_t}{1-u_t}$$

BIM retrieval status value

- Including the query terms found in the document into the right product, but simultaneously dividing by them in the left product, gives:

$$O(R|\vec{x},\vec{q}) \propto \prod_{t:x_t=q_t=1} \frac{p_t(1-u_t)}{u_t(1-p_t)} \cdot \prod_{t:q_t=1} \frac{1-p_t}{1-u_t}$$

- The right product is now over all query terms, hence constant for a particular query and can be ignored.
- $\longrightarrow$ The only quantity that needs to be estimated to rank documents w.r.t a query is the left product.
- Hence the Retrieval Status Value (RSV) in this model:

$$RSV_d = \log \prod_{t:x_t=q_t=1} \frac{p_t(1-u_t)}{u_t(1-p_t)} = \sum_{t:x_t=q_t=1} \log \frac{p_t(1-u_t)}{u_t(1-p_t)}$$

BIM retrieval status value (2)

Equivalent: rank documents using the log odds ratios for the terms in the query c_t:

$$c_t = \log \frac{p_t(1-u_t)}{u_t(1-p_t)} = \log \frac{p_t}{(1-p_t)} - \log \frac{u_t}{1-u_t}$$

- The odds ratio is the ratio of two odds: (i) the odds of the term appearing if the document is relevant ($p_t/(1-p_t)$), and (ii) the odds of the term appearing if the document is nonrelevant ($u_t/(1-u_t)$)
- $c_t = 0$: term has equal odds of appearing in relevant and nonrelevant docs
- c_t positive: higher odds to appear in relevant documents
- c_t negative: higher odds to appear in nonrelevant documents

Term weight c_t in BIM

- $c_t = \log \frac{p_t}{(1-p_t)} - \log \frac{u_t}{1-u_t}$ functions as a term weight.
- Retrieval status value for document d: $RSV_d = \sum_{x_t=q_t=1} c_t$.
- So BIM and vector space model are similar on an operational level.
- In particular: we can use the same data structures (inverted index etc) for the two models. □

Computing term weights c_t

For each term t in a query, estimate c_t in the whole collection using a contingency table of counts of documents in the collection, where df_t is the number of documents that contain term t:

	documents	relevant	nonrelevant	Total
Term present	$x_t = 1$	s	$\mathrm{df}_t - s$	df_t
Term absent	$x_t = 0$	$S - s$	$(N - \mathrm{df}_t) - (S - s)$	$N - \mathrm{df}_t$
	Total	S	$N - S$	N

$$p_t = s/S$$

$$u_t = (\mathrm{df}_t - s)/(N - S)$$

$$c_t = K(N, \mathrm{df}_t, S, s) = \log \frac{s/(S-s)}{(\mathrm{df}_t - s)/((N - \mathrm{df}_t) - (S - s))}$$

Avoiding zeros

- If any of the counts is a zero, then the term weight is not well-defined.
- Maximum likelihood estimates do not work for rare events.
- To avoid zeros: add 0.5 to each count (expected likelihood estimation = ELE) or use a different type of smoothing □

More simplifying assumptions

- Assume that relevant documents are a very small percentage of the collection ...
- ...then we can approximate statistics for nonrelevant documents by statistics from the whole collection:

$$\log[(1 - u_t)/u_t] = \log[(N - \mathrm{df}_t)/\mathrm{df}_t] \approx \log N/\mathrm{df}_t$$

- This should look familiar to you ...

Probability estimates in relevance feedback

- For relevance feedback, we can directly compute term weights c_t based on the contingency table (using an appropriate smoothing method like ELE).

Computing term weights c_t for relevance feedback

For each term t in a query, estimate c_t in the whole collection using a contingency table of counts of documents in the collection, where df_t is the number of documents that contain term t:

	documents	relevant	nonrelevant	Total
Term present	$x_t = 1$	s	$\mathrm{df}_t - s$	df_t
Term absent	$x_t = 0$	$S - s$	$(N - \mathrm{df}_t) - (S - s)$	$N - \mathrm{df}_t$
	Total	S	$N - S$	N

$$p_t = s/S$$

$$u_t = (\mathrm{df}_t - s)/(N - S)$$

$$c_t = K(N, \mathrm{df}_t, S, s) = \log \frac{s/(S-s)}{(\mathrm{df}_t - s)/((N - \mathrm{df}_t) - (S - s))}$$

Probability estimates in adhoc retrieval

- Ad-hoc retrieval: no user-supplied relevance judgments available
- In this case: assume constant $p_t = 0.5$ for all terms x_t in the query
- Each query term is equally likely to occur in a relevant document, and so the p_t and $(1 - p_t)$ factors cancel out in the expression for RSV.
- Weak estimate, but doesn't disagree violently with expectation that query terms appear in many but not all relevant documents.
- Weight c_t in this case: $c_t = \log \frac{p_t}{(1-p_t)} - \log \frac{u_t}{1-u_t} \approx \log N/\mathrm{df}_t$
- For short documents (titles or abstracts), this simple version of BIM works well.

Outline

Okapi BM25: Overview

- Okapi BM25 is a probabilistic model that incorporates term frequency (i.e., it's nonbinary) and length normalization.
- BIM was originally designed for short catalog records of fairly consistent length, and it works reasonably in these contexts.
- For modern full-text search collections, a model should pay attention to term frequency and document length.
- BM25 (BestMatch25) is sensitive to these quantities. □

Okapi BM25: Starting point

- In the simplest version of BIM, the score for document d is just idf weighting of the query terms present in the document:
- $$RSV_d = \sum_{t \in q \cap d} \log \frac{N}{\mathrm{df}_t}$$

Okapi BM25 basic weighting

- Improve idf term [log N/df] by factoring in term frequency and document length.

$$RSV_d = \sum_{t \in q} \log \left[\frac{N}{\mathrm{df}_t} \right] \cdot \frac{(k_1 + 1)\mathrm{tf}_{td}}{k_1((1-b) + b \times (L_d/L_{\mathrm{ave}})) + \mathrm{tf}_{td}}$$

- tf_{td}: term frequency in document d
- L_d (L_{ave}): length of document d (average document length in the whole collection)
- k_1: tuning parameter controlling scaling of term frequency
- b: tuning parameter controlling the scaling by document length

Take-away

- Probabilistic approach to IR: Introduction
- Binary independence model or BIM – the first influential probabilistic model
- Okapi BM25, a more modern, better performing probabilistic model

Resources

- Chapter 11 of Introduction to Information Retrieval
- Resources at http://informationretrieval.org/essir2011
  - Binary independence model (original paper)
  - More details on Okapi BM25
  - Why the Naive Bayes independence assumption often works (paper)

Exercise

Naive Bayes conditional independence assumption: the presence or absence of a word in a document is independent of the presence or absence of any other word (given the query).

Why is this wrong? Good example?

PRP assumes that the relevance of each document is independent of the relevance of other documents.

Why is this wrong? Good example?

Introduction to Information Retrieval

http://informationretrieval.org

IIR 12: Language Models for IR

Hinrich Schütze

Institute for Natural Language Processing, Universität Stuttgart

2011-08-29

Models and Methods

1. Boolean model and its limitations (30)
2. Vector space model (30)
3. Probabilistic models (30)
4. Language model-based retrieval (30)
5. Latent semantic indexing (30)
6. Learning to rank (30)

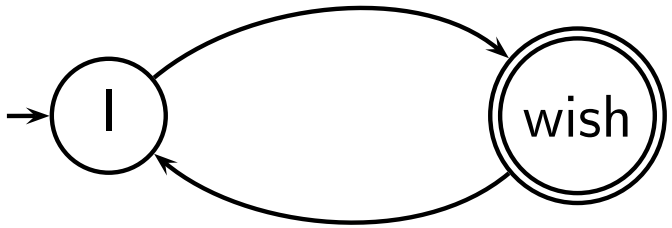
Take-away

- Statistical language models: Introduction
- Statistical language models in IR
- Discussion: Properties of different probabilistic models in use in IR

Outline

What is a language model?

We can view a finite state automaton as a deterministic language model.

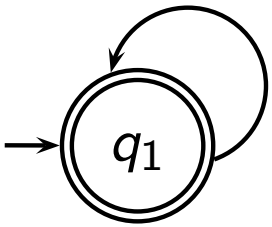


I wish I wish I wish I wish . . .

Cannot generate: "wish I wish" or "I wish I"

Our basic model: each document was generated by a different automaton like this except that these automata are probabilistic.

A probabilistic language model



w	$P(w\|q_1)$	w	$P(w\|q_1)$
STOP	0.2	toad	0.01
the	0.2	said	0.03
a	0.1	likes	0.02
frog	0.01	that	0.04
		...	...

This is a one-state probabilistic finite-state automaton – a unigram language model – and the state emission distribution for its one state q_1.

STOP is not a word, but a special symbol indicating that the automaton stops.

frog said that toad likes frog STOP

$P(\text{string}) = 0.01 \cdot 0.03 \cdot 0.04 \cdot 0.01 \cdot 0.02 \cdot 0.01 \cdot 0.2$
$= 0.0000000000048$

A different language model for each document

language model of d_1

w	$P(w\|.)$	w	$P(w\|.)$
STOP	.2	toad	.01
the	.2	said	.03
a	.1	likes	.02
frog	.01	that	.04
		...	...

language model of d_2

w	$P(w\|.)$	w	$P(w\|.)$
STOP	.2	toad	.02
the	.15	said	.03
a	.08	likes	.02
frog	.01	that	.05
		...	...

query: frog said that toad likes frog STOP

$P(\text{query}|M_{d1}) = 0.01 \cdot 0.03 \cdot 0.04 \cdot 0.01 \cdot 0.02 \cdot 0.01 \cdot 0.2$
$= 0.0000000000048 = 4.8 \cdot 10^{-12}$

$P(\text{query}|M_{d2}) = 0.01 \cdot 0.03 \cdot 0.05 \cdot 0.02 \cdot 0.02 \cdot 0.01 \cdot 0.2$
$= 0.0000000000120 = 12 \cdot 10^{-12}$

$P(\text{query}|M_{d1}) < P(\text{query}|M_{d2})$ Thus, document d_2 is "more relevant" to the query "frog said that toad likes frog STOP" than d_1 is.

Outline

Using language models in IR

- Each document is treated as (the basis for) a language model.
- Given a query q
- Rank documents based on $P(d|q)$
- $$P(d|q) = \frac{P(q|d)P(d)}{P(q)}$$
- $P(q)$ is the same for all documents, so ignore
- $P(d)$ is the prior – often treated as the same for all d
  - But we can give a higher prior to "high-quality" documents, e.g., those with high PageRank.
- $P(q|d)$ is the probability of q given d .
- Under the assumptions we made, ranking documents according to $P(q|d)P(d)$ and $P(d|q)$ is equivalent. □

How to compute $P(q|d)$

- We will make the same conditional independence assumption as in BIM.
- $$P(q|M_d) = P(\langle t_1, \ldots, t_{|q|} \rangle | M_d) = \prod_{1 \leq k \leq |q|} P(t_k|M_d)$$

  ($|q|$: length of q; t_k: the token occurring at position k in q)
- This is equivalent to:

$$P(q|M_d) = \prod_{\text{distinct term } t \text{ in } q} P(t|M_d)^{\text{tf}_{t,q}}$$

- $\text{tf}_{t,q}$: term frequency (# occurrences) of t in q
- Multinomial model (omitting constant factor)

Parameter estimation

- Missing piece: Where do the parameters $P(t|M_d)$ come from?
- Start with maximum likelihood estimates
- $$\hat{P}(t|M_d) = \frac{\mathrm{tf}_{t,d}}{|d|}$$

  ($|d|$: length of d; $\mathrm{tf}_{t,d}$: # occurrences of t in d)
- We have a problem with zeros.
- A single t in the query with $P(t|M_d) = 0$ will make $P(q|M_d) = \prod P(t|M_d)$ zero.
- We would give a single term in the query "veto power".
- For example, for query [Michael Jackson top hits] a document about "Michael Jackson top songs" (but not using the word "hits") would have $P(q|M_d) = 0$. – That's bad.
- We need to smooth the estimates to avoid zeros.

Smoothing

- Key intuition: A nonoccurring term is possible (even though it didn't occur), ...
- ...but no more likely than would be expected by chance in the collection.
- Notation: M_c: the collection model; cf_t: the number of occurrences of t in the collection; $T = \sum_t \mathrm{cf}_t$: the total number of tokens in the collection.
- $$\hat{P}(t|M_c) = \frac{\mathrm{cf}_t}{T}$$
- We will use $\hat{P}(t|M_c)$ to "smooth" $P(t|d)$ away from zero. □

Jelinek-Mercer smoothing

- $P(t|d) = \lambda P(t|M_d) + (1 - \lambda)P(t|M_c)$
- Mixes the probability from the document with the general collection frequency of the word.
- High value of λ: "conjunctive-like" search – tends to retrieve documents containing all query words.
- Low value of λ: more disjunctive, suitable for long queries
- Tuning λ is important for good performance.

Jelinek-Mercer smoothing: Summary

- $$P(q|d) \propto \prod_{1 \leq k \leq |q|} (\lambda P(t_k|M_d) + (1-\lambda) P(t_k|M_c))$$
- What we model: The user has a document in mind and generates the query from this document.
- $P(q|d)$ is the probability that the document that the user had in mind was in fact this one.

Example

- Collection: d_1 and d_2
- d_1: Jackson was one of the most talented entertainers of all time
- d_2: Michael Jackson anointed himself King of Pop
- Query q: Michael Jackson
- Use mixture model with $\lambda = 1/2$
- $P(q|d_1) = [(0/11 + 1/18)/2] \cdot [(1/11 + 2/18)/2] \approx 0.003$
- $P(q|d_2) = [(1/7 + 1/18)/2] \cdot [(1/7 + 2/18)/2] \approx 0.013$
- Ranking: $d_2 > d_1$

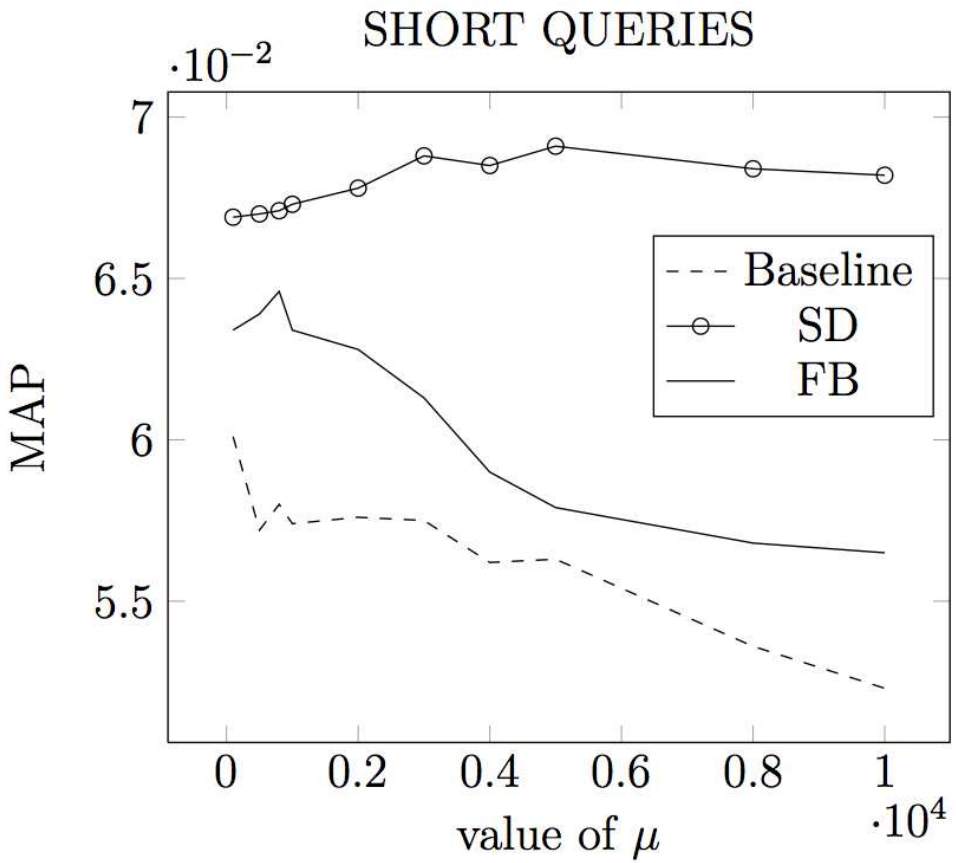
Dirichlet smoothing

- $$P(t|d) = \frac{\text{tf}_{t,d} + \alpha P(t|M_c)}{L_d + \alpha}$$
- The background distribution $P(t|M_c)$ is the prior for $P(t|d)$.
- Intuition: Before having seen any part of the document we start with the background distribution as our estimate.
- As we read the document and count terms we update the background distribution.
- The weighting factor α determines how strong an effect the prior has.

Jelinek-Mercer or Dirichlet?

- Dirichlet performs better for keyword queries, Jelinek-Mercer performs better for verbose queries.
- Both models are sensitive to the smoothing parameters – you shouldn't use these models without parameter tuning. □

Sensitivity of Dirichlet to smoothing parameter



μ is the Dirichlet smoothing parameter (called α on the previous slides)

Vector space (tf-idf) vs. LM

	precision			significant
Rec.	tf-idf	LM	%chg	
0.0	0.7439	0.7590	+2.0	
0.1	0.4521	0.4910	+8.6	
0.2	0.3514	0.4045	+15.1	*
0.4	0.2093	0.2572	+22.9	*
0.6	0.1024	0.1405	+37.1	*
0.8	0.0160	0.0432	+169.6	*
1.0	0.0028	0.0050	+76.9	
11-point average	0.1868	0.2233	+19.6	*

The language modeling approach always does better in these experiments . . .

. . . but note that where the approach shows significant gains is at higher levels of recall.

Summary: IR language models

1. View the document as a generative model that generates the query
2. Define the precise generative model we want to use
3. Estimate parameters (different parameters for each document's model)
4. Smooth to avoid zeros
5. Apply to query and find document most likely to have generated the query
6. Present most likely document(s) to user

Outline

Naive Bayes generative model

- We want to classify document d.
  - Human-defined classes: e.g., politics, economics, sports.
- Assume that d was generated by the generative model.
- Key question: Which of the classes (= class models) is most likely to have generated the document?
  - Or: for which class do we have the most evidence?

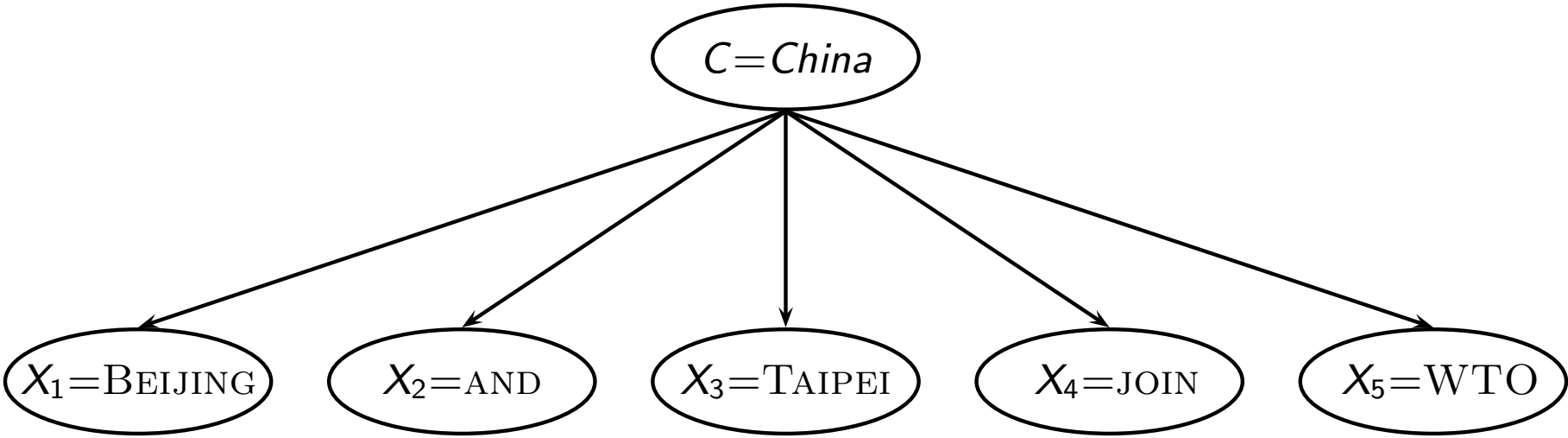
Naive Bayes and LM generative models

- We want to classify document d.
  - Human-defined classes: e.g., politics, economics, sports.
- Assume that d was generated by the generative model.
- Key question: Which of the classes (= class models) is most likely to have generated the document?
  - Or: for which class do we have the most evidence?

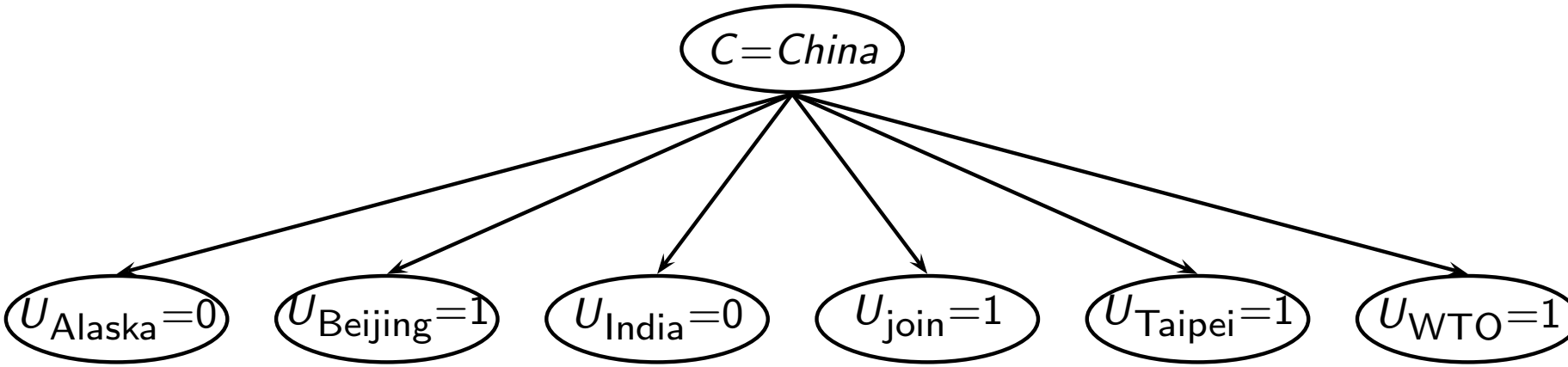
Naive Bayes and LM generative models

- We want to classify document d.
  We want to classify a query q.
  - Human-defined classes: e.g., politics, economics, sports.
    Each document in the collection is a different class.
- Assume that d was generated by the generative model.
  Assume that q was generated by a generative model
- Key question: Which of the classes (= class models) is most likely to have generated the document? Which document (=class) is most likely to have generated the query q?
  - Or: for which class do we have the most evidence? For which document (as the source of the query) do we have the most evidence?

Naive Bayes Multinomial model / IR language models



Naive Bayes Bernoulli model / Binary independence model



Comparison of the two models

	multinomial model / IR LM	Bernoulli model / BIM
event model	generation of (multi)set of tokens	generation of subset of vocabular
random variable(s)	$X = t$ iff t occurs at given pos	$U_t = 1$ iff t occurs in doc
doc. representation	$d = \langle t_1, \ldots, t_k, \ldots, t_{n_d} \rangle, t_k \in V$	$d = \langle e_1, \ldots, e_i, \ldots, e_M \rangle,$ $e_i \in \{0, 1\}$
parameter estimation	$\hat{P}(X = t\|c)$	$\hat{P}(U_i = e\|c)$
dec. rule: maximize	$\hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(X = t_k\|c)$	$\hat{P}(c) \prod_{t_i \in V} \hat{P}(U_i = e_i\|c)$
multiple occurrences	taken into account	ignored
length of docs	can handle longer docs	works best for short docs
# features	can handle more	works best with fewer
estimate for THE	$\hat{P}(X = \text{the}\|c) \approx 0.05$	$\hat{P}(U_{\text{the}} = 1\|c) \approx 1.0$

Vector space vs BM25 vs LM

- BM25/LM: based on probability theory
- Vector space: based on similarity, a geometric/linear algebra notion
- Term frequency is directly used in all three models.
  - LMs: raw term frequency, BM25/Vector space: more complex
- Length normalization
  - Vector space: Cosine or pivot normalization
  - LMs: probabilities are inherently length normalized
  - BM25: tuning parameters for optimizing length normalization
- idf: BM25/vector space use it directly.
- LMs: Mixing term and collection frequencies has an effect similar to idf.
  - Terms rare in the general collection, but common in some documents will have a greater influence on the ranking.
- Collection frequency (LMs) vs. document frequency (BM25, vector space)

Take-away

- Statistical language models: Introduction
- Statistical language models in IR
- Discussion: Properties of different probabilistic models in use in IR

Resources

- Chapter 12 of Introduction to Information Retrieval
- Resources at http://informationretrieval.org/essir2011
  - Ponte and Croft's 1998 SIGIR paper (one of the first on LMs in IR)
  - Zhai and Lafferty: A study of smoothing methods for language models applied to information retrieval. ACM Trans. Inf. Syst. (2004).
  - Lemur toolkit (good support for LMs in IR)
  - Bernoulli vs multinomial models

Exercise: Compute ranking

- Collection: d_1 and d_2
- d_1: Xerox reports a profit but revenue is down
- d_2: Lucene narrows quarter loss but revenue decreases further
- Query q: revenue down
- Use mixture model with $\lambda = 1/2$
- $P(q|d_1) = [(1/8 + 2/16)/2] \cdot [(1/8 + 1/16)/2] = 1/8 \cdot 3/32 = 3/256$
- $P(q|d_2) = [(1/8 + 2/16)/2] \cdot [(0/8 + 1/16)/2] = 1/8 \cdot 1/32 = 1/256$
- Ranking: $d_1 > d_2$

Introduction to Information Retrieval

http://informationretrieval.org

IIR 18: Latent Semantic Indexing

Hinrich Schütze

Institute for Natural Language Processing, Universität Stuttgart

2011-08-29

Models and Methods

1. Boolean model and its limitations (30)
2. Vector space model (30)
3. Probabilistic models (30)
4. Language model-based retrieval (30)
5. Latent semantic indexing (30)
6. Learning to rank (30)

Take-away

- Singular Value Decomposition (SVD): The math behind LSI
- SVD used for dimensionality reduction
- Latent Semantic Indexing (LSI): SVD used in information retrieval

Outline

Recall: Term-document matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
anthony	5.25	3.18	0.0	0.0	0.0	0.35
brutus	1.21	6.10	0.0	1.0	0.0	0.0
caesar	8.59	2.54	0.0	1.51	0.25	0.0
calpurnia	0.0	1.54	0.0	0.0	0.0	0.0
cleopatra	2.85	0.0	0.0	0.0	0.0	0.0
mercy	1.51	0.0	1.90	0.12	5.25	0.88
worser	1.37	0.0	0.11	4.15	0.25	1.95

...

This matrix is the basis for computing the similarity between documents and queries.

This lecture: Can we transform this matrix, so that we get a better measure of similarity between documents and queries? □

Latent semantic indexing: Overview

- We will decompose the term-document matrix into a product of matrices.
- The particular decomposition we'll use: singular value decomposition (SVD).
- SVD: $C = U\Sigma V^T$ (where C = term-document matrix)
- We will then use the SVD to compute a new, improved term-document matrix C'.
- We'll get better similarity values out of C' (compared to C).
- Using SVD for this purpose is called latent semantic indexing or LSI.

Example of $C = U\Sigma V^T$: The matrix C

C	d_1	d_2	d_3	d_4	d_5	d_6
ship	1	0	1	0	0	0
boat	0	1	0	0	0	0
ocean	1	1	0	0	0	0
wood	1	0	0	1	1	0
tree	0	0	0	1	0	1

This is a standard term-document matrix.

Actually, we use a non-weighted matrix here to simplify the example.

Example of $C = U\Sigma V^T$: The matrix U

U	1	2	3	4	5
ship	−0.44	−0.30	0.57	0.58	0.25
boat	−0.13	−0.33	−0.59	0.00	0.73
ocean	−0.48	−0.51	−0.37	0.00	−0.61
wood	−0.70	0.35	0.15	−0.58	0.16
tree	−0.26	0.65	−0.41	0.58	−0.09

Square matrix, $M \times M$

This is an orthonormal matrix: (i) Row vectors have unit length. (ii) Any two distinct row vectors are orthogonal to each other.

Think of the dimensions as "semantic" dimensions that capture distinct topics like politics, sports, economics. 2 = water/land

Each number u_{ij} in the matrix indicates how strongly related term i is to the topic represented by semantic dimension j. □

Example of $C = U\Sigma V^T$: The matrix Σ

Σ	1	2	3	4	5
1	2.16	0.00	0.00	0.00	0.00
2	0.00	1.59	0.00	0.00	0.00
3	0.00	0.00	1.28	0.00	0.00
4	0.00	0.00	0.00	1.00	0.00
5	0.00	0.00	0.00	0.00	0.39

This is a square, diagonal matrix of dimensionality $\min(M, N) \times \min(M, N)$.

The diagonal consists of the singular values of C.

The magnitude of the singular value measures the importance of the corresponding semantic dimension.

We'll make use of this by omitting unimportant dimensions. □

Example of $C = U\Sigma V^T$: The matrix V^T

V^T	d_1	d_2	d_3	d_4	d_5	d_6
1	−0.75	−0.28	−0.20	−0.45	−0.33	−0.12
2	−0.29	−0.53	−0.19	0.63	0.22	0.41
3	0.28	−0.75	0.45	−0.20	0.12	−0.33
4	0.00	0.00	0.58	0.00	−0.58	0.58
5	−0.53	0.29	0.63	0.19	0.41	−0.22
6	0.00	0.00	0.00	-0.58	0.58	0.58

$N \times N$ square matrix. Drop row 6 – only want $\min(M, N)$ LSI dims.

Again: This is an orthonormal matrix: (i) Column vectors have unit length. (ii) Any two distinct column vectors are orthogonal to each other.

These are again the semantic dimensions from matrices U and Σ that capture distinct topics like politics, sports, economics.

Each v_{ij} in the matrix indicates how strongly related document i is to the topic represented by semantic dimension j. □

Example of $C = U\Sigma V^T$: All four matrices

C	d_1	d_2	d_3	d_4	d_5	d_6
ship	1.00	0.00	1.00	0.00	0.00	0.00
boat	0.00	1.00	0.00	0.00	0.00	0.00
ocean	1.00	1.00	0.00	0.00	0.00	0.00
wood	1.00	0.00	0.00	1.00	1.00	0.00
tree	0.00	0.00	0.00	1.00	0.00	1.00

$=$

U	1	2	3	4	5
ship	−0.44	−0.30	0.57	0.58	0.25
boat	−0.13	−0.33	−0.59	0.00	0.73
ocean	−0.48	−0.51	−0.37	0.00	−0.61
wood	−0.70	0.35	0.15	−0.58	0.16
tree	−0.26	0.65	−0.41	0.58	−0.09

$\times$

Σ	1	2	3	4	5
1	2.16	0.00	0.00	0.00	0.00
2	0.00	1.59	0.00	0.00	0.00
3	0.00	0.00	1.28	0.00	0.00
4	0.00	0.00	0.00	1.00	0.00
5	0.00	0.00	0.00	0.00	0.39

$\times$

V^T	d_1	d_2	d_3	d_4	d_5	d_6
1	−0.75	−0.28	−0.20	−0.45	−0.33	−0.12
2	−0.29	−0.53	−0.19	0.63	0.22	0.41
3	0.28	−0.75	0.45	−0.20	0.12	−0.33
4	0.00	0.00	0.58	0.00	−0.58	0.58
5	−0.53	0.29	0.63	0.19	0.41	−0.22

LSI is decomposition of C into a representation of the terms, a representation of the documents and a representation of the importance of the "semantic" dimensions.

LSI: Summary

- We've decomposed the term-document matrix C into a product of three matrices: $U\Sigma V^T$.
- The term matrix U – consists of one (row) vector for each term
- The document matrix V^T – consists of one (column) vector for each document
- The singular value matrix Σ – diagonal matrix with singular values, reflecting importance of each dimension
- Next: Why are we doing this?

Outline

How we use the SVD in LSI

- Key property: Each singular value tells us how important its dimension is.
- By setting less important dimensions to zero, we keep the important information, but get rid of the "details".
- These details may
  - be noise – in that case, reduced LSI is a better representation because it is less noisy.
  - make things dissimilar that should be similar – again, the reduced LSI representation is a better representation because it represents similarity better.
- Analogy for "fewer details is better"
  - Image of a blue flower
  - Image of a yellow flower
  - Omitting color makes is easier to see the similarity

Reducing the dimensionality to 2

U	1	2	3	4	5
ship	−0.44	−0.30	0.00	0.00	0.00
boat	−0.13	−0.33	0.00	0.00	0.00
ocean	−0.48	−0.51	0.00	0.00	0.00
wood	−0.70	0.35	0.00	0.00	0.00
tree	−0.26	0.65	0.00	0.00	0.00

Σ_2	1	2	3	4	5
1	2.16	0.00	0.00	0.00	0.00
2	0.00	1.59	0.00	0.00	0.00
3	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00

V^T	d_1	d_2	d_3	d_4	d_5	d_6
1	−0.75	−0.28	−0.20	−0.45	−0.33	−0.12
2	−0.29	−0.53	−0.19	0.63	0.22	0.41
3	0.00	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00	0.00

Reducing the dimensionality to 2

U	1	2	3	4	5
ship	−0.44	−0.30	0.00	0.00	0.00
boat	−0.13	−0.33	0.00	0.00	0.00
ocean	−0.48	−0.51	0.00	0.00	0.00
wood	−0.70	0.35	0.00	0.00	0.00
tree	−0.26	0.65	0.00	0.00	0.00

Σ_2	1	2	3	4	5
1	2.16	0.00	0.00	0.00	0.00
2	0.00	1.59	0.00	0.00	0.00
3	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00

V^T	d_1	d_2	d_3	d_4	d_5	d_6
1	−0.75	−0.28	−0.20	−0.45	−0.33	−0.12
2	−0.29	−0.53	−0.19	0.63	0.22	0.41
3	0.00	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00	0.00

Actually, we only zero out singular values in Σ. This has the effect of setting the corresponding dimensions in U and V^T to zero when computing the product $C = U\Sigma V^T$. □

Reducing the dimensionality to 2

C_2	d_1	d_2	d_3	d_4	d_5	d_6
ship	0.85	0.52	0.28	0.13	0.21	−0.08
boat	0.36	0.36	0.16	−0.20	−0.02	−0.18
ocean	1.01	0.72	0.36	−0.04	0.16	−0.21
wood	0.97	0.12	0.20	1.03	0.62	0.41
tree	0.12	−0.39	−0.08	0.90	0.41	0.49

=

U	1	2	3	4	5
ship	−0.44	−0.30	0.57	0.58	0.25
boat	−0.13	−0.33	−0.59	0.00	0.73
ocean	−0.48	−0.51	−0.37	0.00	−0.61
wood	−0.70	0.35	0.15	−0.58	0.16
tree	−0.26	0.65	−0.41	0.58	−0.09

×

Σ_2	1	2	3	4	5
1	2.16	0.00	0.00	0.00	0.00
2	0.00	1.59	0.00	0.00	0.00
3	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00

×

V^T	d_1	d_2	d_3	d_4	d_5	d_6
1	−0.75	−0.28	−0.20	−0.45	−0.33	−0.12
2	−0.29	−0.53	−0.19	0.63	0.22	0.41
3	0.28	−0.75	0.45	−0.20	0.12	−0.33
4	0.00	0.00	0.58	0.00	−0.58	0.58
5	−0.53	0.29	0.63	0.19	0.41	−0.22

Recall unreduced decomposition $C = U\Sigma V^T$

C	d_1	d_2	d_3	d_4	d_5	d_6
ship	1.00	0.00	1.00	0.00	0.00	0.00
boat	0.00	1.00	0.00	0.00	0.00	0.00
ocean	1.00	1.00	0.00	0.00	0.00	0.00
wood	1.00	0.00	0.00	1.00	1.00	0.00
tree	0.00	0.00	0.00	1.00	0.00	1.00

$=$

U	1	2	3	4	5
ship	−0.44	−0.30	0.57	0.58	0.25
boat	−0.13	−0.33	−0.59	0.00	0.73
ocean	−0.48	−0.51	−0.37	0.00	−0.61
wood	−0.70	0.35	0.15	−0.58	0.16
tree	−0.26	0.65	−0.41	0.58	−0.09

$\times$

Σ	1	2	3	4	5
1	2.16	0.00	0.00	0.00	0.00
2	0.00	1.59	0.00	0.00	0.00
3	0.00	0.00	1.28	0.00	0.00
4	0.00	0.00	0.00	1.00	0.00
5	0.00	0.00	0.00	0.00	0.39

$\times$

V^T	d_1	d_2	d_3	d_4	d_5	d_6
1	−0.75	−0.28	−0.20	−0.45	−0.33	−0.12
2	−0.29	−0.53	−0.19	0.63	0.22	0.41
3	0.28	−0.75	0.45	−0.20	0.12	−0.33
4	0.00	0.00	0.58	0.00	−0.58	0.58
5	−0.53	0.29	0.63	0.19	0.41	−0.22

Original matrix C vs. reduced $C_2 = U\Sigma_2 V^T$

C	d_1	d_2	d_3	d_4	d_5	d_6
ship	1	0	1	0	0	0
boat	0	1	0	0	0	0
ocean	1	1	0	0	0	0
wood	1	0	0	1	1	0
tree	0	0	0	1	0	1

C_2	d_1	d_2	d_3	d_4	d_5	d_6
ship	0.85	0.52	0.28	0.13	0.21	−0.08
boat	0.36	0.36	0.16	−0.20	−0.02	−0.18
ocean	1.01	0.72	0.36	−0.04	0.16	−0.21
wood	0.97	0.12	0.20	1.03	0.62	0.41
tree	0.12	−0.39	−0.08	0.90	0.41	0.49

Original matrix C vs. reduced $C_2 = U\Sigma_2 V^T$

C	d_1	d_2	d_3	d_4	d_5	d_6
ship	1	0	1	0	0	0
boat	0	1	0	0	0	0
ocean	1	1	0	0	0	0
wood	1	0	0	1	1	0
tree	0	0	0	1	0	1

C_2	d_1	d_2	d_3	d_4	d_5	d_6
ship	0.85	0.52	0.28	0.13	0.21	−0.08
boat	0.36	0.36	0.16	−0.20	−0.02	−0.18
ocean	1.01	0.72	0.36	−0.04	0.16	−0.21
wood	0.97	0.12	0.20	1.03	0.62	0.41
tree	0.12	−0.39	−0.08	0.90	0.41	0.49

We can view C_2 as a two-dimensional representation of the matrix C. We have performed a dimensionality reduction to two dimensions.

Why the reduced matrix C_2 is better than C

C	d_1	d_2	d_3	d_4	d_5	d_6
ship	1	0	1	0	0	0
boat	0	1	0	0	0	0
ocean	1	1	0	0	0	0
wood	1	0	0	1	1	0
tree	0	0	0	1	0	1

C_2	d_1	d_2	d_3	d_4	d_5	d_6
ship	0.85	0.52	0.28	0.13	0.21	−0.08
boat	0.36	0.36	0.16	−0.20	−0.02	−0.18
ocean	1.01	0.72	0.36	−0.04	0.16	−0.21
wood	0.97	0.12	0.20	1.03	0.62	0.41
tree	0.12	−0.39	−0.08	0.90	0.41	0.49

Why the reduced matrix C_2 is better than C

C	d_1	d_2	d_3	d_4	d_5	d_6
ship	1	0	1	0	0	0
boat	0	1	0	0	0	0
ocean	1	1	0	0	0	0
wood	1	0	0	1	1	0
tree	0	0	0	1	0	1

C_2	d_1	d_2	d_3	d_4	d_5	d_6
ship	0.85	0.52	0.28	0.13	0.21	−0.08
boat	0.36	0.36	0.16	−0.20	−0.02	−0.18
ocean	1.01	0.72	0.36	−0.04	0.16	−0.21
wood	0.97	0.12	0.20	1.03	0.62	0.41
tree	0.12	−0.39	−0.08	0.90	0.41	0.49

Similarity of d_2 and d_3 in the original space: 0.

Why the reduced matrix C_2 is better than C

C	d_1	d_2	d_3	d_4	d_5	d_6
ship	1	0	1	0	0	0
boat	0	1	0	0	0	0
ocean	1	1	0	0	0	0
wood	1	0	0	1	1	0
tree	0	0	0	1	0	1

C_2	d_1	d_2	d_3	d_4	d_5	d_6
ship	0.85	0.52	0.28	0.13	0.21	−0.08
boat	0.36	0.36	0.16	−0.20	−0.02	−0.18
ocean	1.01	0.72	0.36	−0.04	0.16	−0.21
wood	0.97	0.12	0.20	1.03	0.62	0.41
tree	0.12	−0.39	−0.08	0.90	0.41	0.49

Similarity of d_2 and d_3 in the original space: 0.

Similarity of d_2 and d_3 in the reduced space: $0.52 * 0.28 + 0.36 * 0.16 + 0.72 * 0.36 + 0.12 * 0.20 + -0.39 * -0.08 \approx 0.52$

Why the reduced matrix C_2 is better than C

C	d_1	d_2	d_3	d_4	d_5	d_6
ship	1	0	1	0	0	0
boat	0	1	0	0	0	0
ocean	1	1	0	0	0	0
wood	1	0	0	1	1	0
tree	0	0	0	1	0	1

C_2	d_1	d_2	d_3	d_4	d_5	d_6
ship	0.85	0.52	0.28	0.13	0.21	−0.08
boat	0.36	0.36	0.16	−0.20	−0.02	−0.18
ocean	1.01	0.72	0.36	−0.04	0.16	−0.21
wood	0.97	0.12	0.20	1.03	0.62	0.41
tree	0.12	−0.39	−0.08	0.90	0.41	0.49

"boat" and "ship" are semantically similar. The "reduced" similarity measure reflects this. □

Outline

Why we use LSI in information retrieval

- LSI takes documents that are semantically similar (= talk about the same topics), ...
- ... but are not similar in the vector space (because they use different words) ...
- ... and re-represents them in a reduced vector space ...
- ... in which they have higher similarity.
- Thus, LSI addresses the problems of synonymy and semantic relatedness.
- Standard vector space: Synonyms contribute nothing to document similarity.
- Desired effect of LSI: Synonyms contribute strongly to document similarity.

How LSI addresses synonymy and semantic relatedness

- The dimensionality reduction forces us to omit a lot of "detail".
- We have to map differents words (= different dimensions of the full space) to the same dimension in the reduced space.
- The "cost" of mapping synonyms to the same dimension is much less than the cost of collapsing unrelated words.
- SVD selects the "least costly" mapping (see below).
- Thus, it will map synonyms to the same dimension.
- But it will avoid doing that for unrelated words.

LSI: Comparison to other approaches

- Relevance feedback and query expansion are used to increase recall in information retrieval – if query and documents have no terms in common.
- LSI increases recall and hurts precision.
- Thus, it addresses the same problems as (pseudo) relevance feedback and query expansion ...
- ...and it has the same problems. □

Implementation

- Compute SVD of term-document matrix
- Reduce the space and compute reduced document representations
- Map the query into the reduced space $\vec{q}_k = \Sigma_k^{-1} U_k^T \vec{q}$.
- This follows from: $C_k = U\Sigma_k V^T \Rightarrow \Sigma_k^{-1} U^T C = V_k^T$
- Compute similarity of q_k with all reduced documents in V_k.
- Output ranked list of documents as usual
- Exercise: What is the fundamental problem with this approach?

Optimality

- SVD is optimal in the following sense.
- Keeping the k largest singular values and setting all others to zero gives you the optimal approximation of the original matrix C. Eckart-Young theorem
- Optimal: no other matrix of the same rank (= with the same underlying dimensionality) approximates C better.
- Measure of approximation is Frobenius norm: $||C||_F = \sqrt{\sum_i \sum_j c_{ij}^2}$
- So LSI uses the "best possible" matrix.
- There is only one best possible matrix – unique solution (modulo signs).
- Caveat: There is only a tenuous relationship between the Frobenius norm and cosine similarity between documents. □

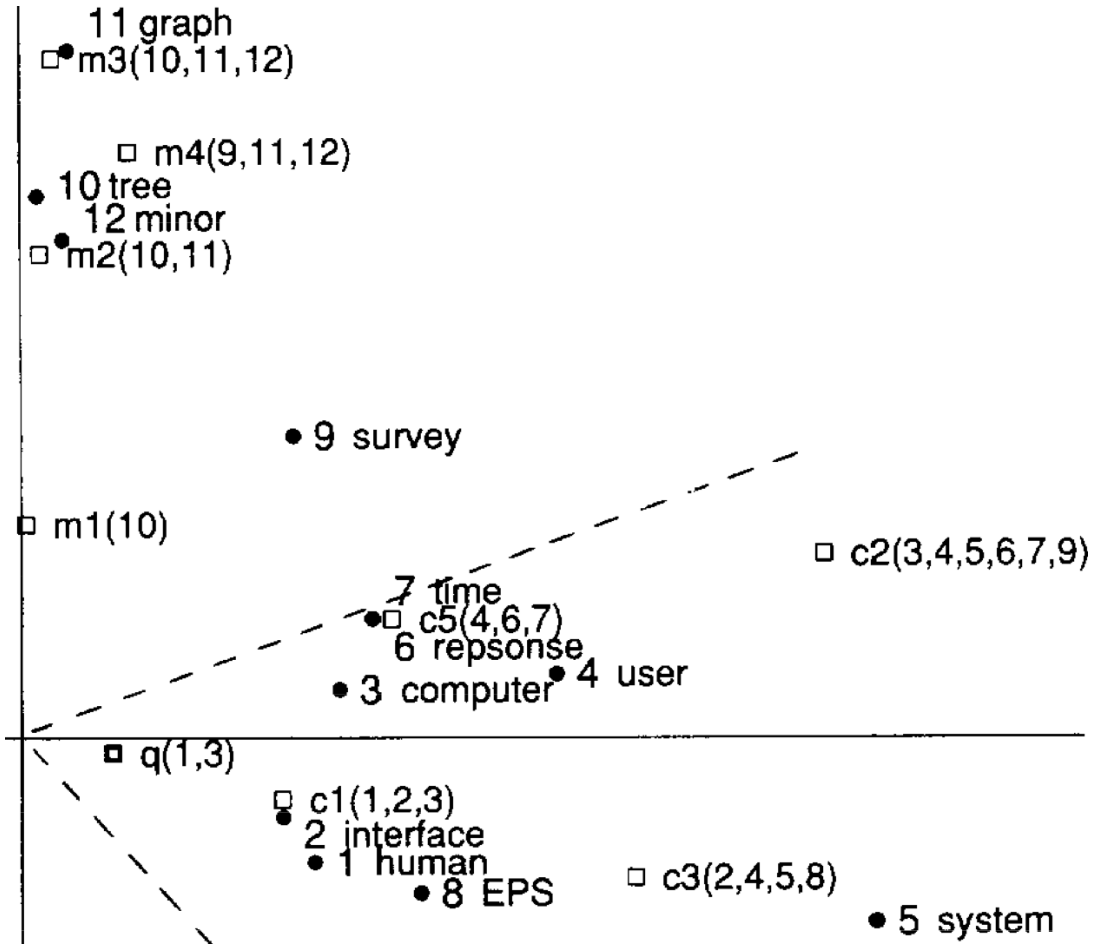
Data for graphical illustration of LSI

c_1 Human machine interface for lab abc computer applications
c_2 A survey of user opinion of computer system response time
c_3 The EPS user interface management system
c_4 System and human system engineering testing of EPS
c_5 Relation of user perceived response time to error measurement
m_1 The generation of random binary unordered trees
m_2 The intersection graph of paths in trees
m_3 Graph minors IV Widths of trees and well quasi ordering
m_4 Graph minors A survey

The matrix C

	c1	c2	c3	c4	c5	m1	m2	m3	m4
human	1	0	0	1	0	0	0	0	0
interface	1	0	1	0	0	0	0	0	0
computer	1	1	0	0	0	0	0	0	0
user	0	1	1	0	1	0	0	0	0
system	0	1	1	2	0	0	0	0	0
response	0	1	0	0	1	0	0	0	0
time	0	1	0	0	1	0	0	0	0
EPS	0	0	1	1	0	0	0	0	0
survey	0	1	0	0	0	0	0	0	1
trees	0	0	0	0	0	1	1	1	0
graph	0	0	0	0	0	0	1	1	1
minors	0	0	0	0	0	0	0	1	1

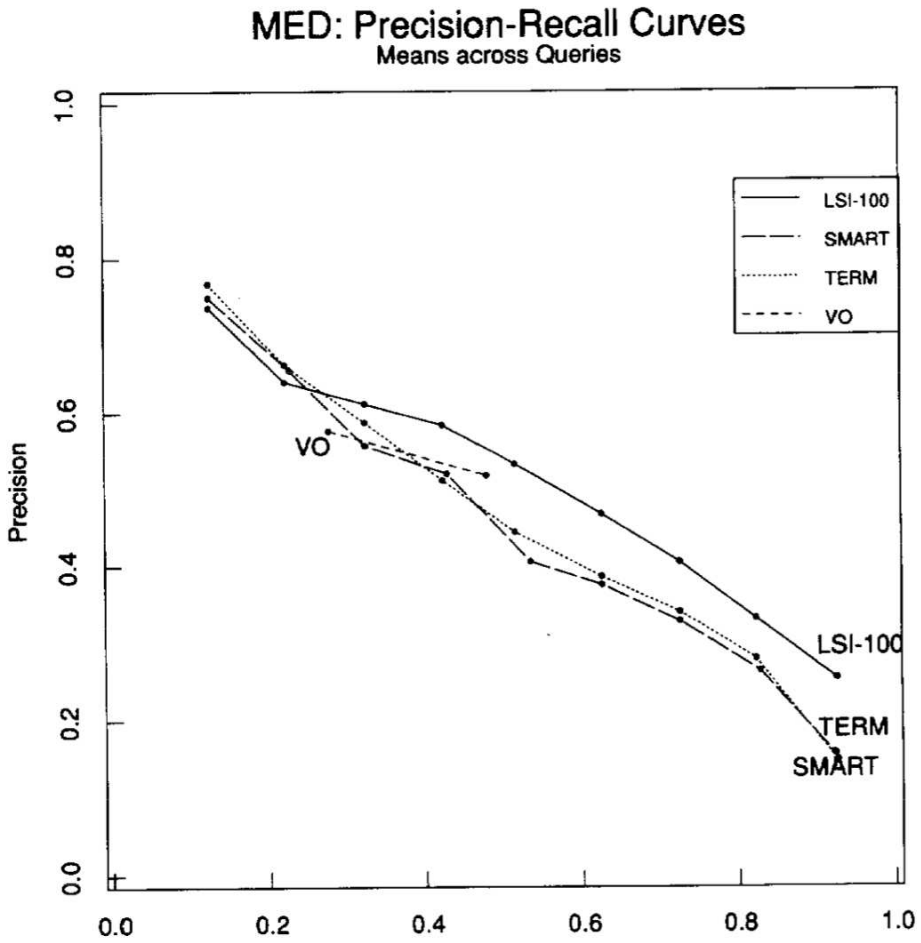
Graphical illustration of LSI: Plot of C_2



2-dimensional plot of C_2 (scaled dimensions). Circles = terms. Open squares = documents (component terms in parentheses). q = query "human computer interaction".

The dotted cone represents the region whose points are within a cosine of .9 from q . All documents about human-computer documents (c1-c5) are near q, even c3/c5 although they share no terms. None of the graph theory documents (m1-m4) are near q. □

LSI performs better than vector space on MED collection



LSI-100 = LSI reduced to 100 dimensions; SMART = SMART implementation of vector space model

Take-away

- Singular Value Decomposition (SVD): The math behind LSI
- SVD used for dimensionality reduction
- Latent Semantic Indexing (LSI): SVD used in information retrieval

Resources

- Chapter 18 of Introduction to Information Retrieval
- Resources at http://informationretrieval.org/essir2011
  - Latent semantic indexing by Deerwester et al. (original paper)
  - Probabilistic LSI by Hofmann
  - Word space: LSI for words

Introduction to Information Retrieval

http://informationretrieval.org

IIR 15-2: Learning to Rank

Hinrich Schütze

Institute for Natural Language Processing, Universität Stuttgart

2011-08-29

Models and Methods

1. Boolean model and its limitations (30)
2. Vector space model (30)
3. Probabilistic models (30)
4. Language model-based retrieval (30)
5. Latent semantic indexing (30)
6. Learning to rank (30)

Take-away

- Machine-learned relevance: We use machine learning to learn the relevance score (retrieval status value) of a document with respect to a query.
- Learning to rank: A machine-learning method that directly optimizes the ranking (as opposed to classification or regression accuracy)

Outline

Machine-learned relevance: Basic idea

- Given: A training set of examples, each of which is a tuple of: a query q, a document d, a relevance judgment for d on q
- Learn weights from this training set, so that the learned scores approximate the relevance judgments in the training set

Machine-learned relevance vs. Text classification

- Both are machine learning approaches
- Text classification (if used for information retrieval, e.g., in relevance feedback) is query-specific.
  - We need a query-specific training set to learn the ranker.
  - We need to learn a new ranker for each query.
- Machine-learned relevance and learning to rank usually refer to query-independent ranking.
- We learn a single classifier or ranker.
- We can then rank documents for a query that we don't have any relevance judgments for. □

Two typical features used in machine-learned relevance

- The vector space cosine similarity between query and document (denoted α)
- The minimum window width within which the query terms lie (denoted ω)
- Thus, we have
  - one feature (α) that captures overall query-document similarity
  - one feature (ω) that captures query term proximity (often indicative of topical relevance)

Machine-learned relevance: Setup for these two features

Training set

Example	DocID	Query	α	ω	Judgment
Φ_1	37	linux ...	0.032	3	relevant
Φ_2	37	penguin ...	0.02	4	nonrelevant
Φ_3	238	operating system	0.043	2	relevant
Φ_4	238	runtime ...	0.004	2	nonrelevant
Φ_5	1741	kernel layer	0.022	3	relevant
Φ_6	2094	device driver	0.03	2	relevant
Φ_7	3191	device driver	0.027	5	nonrelevant

α is the cosine score. ω is the window width.

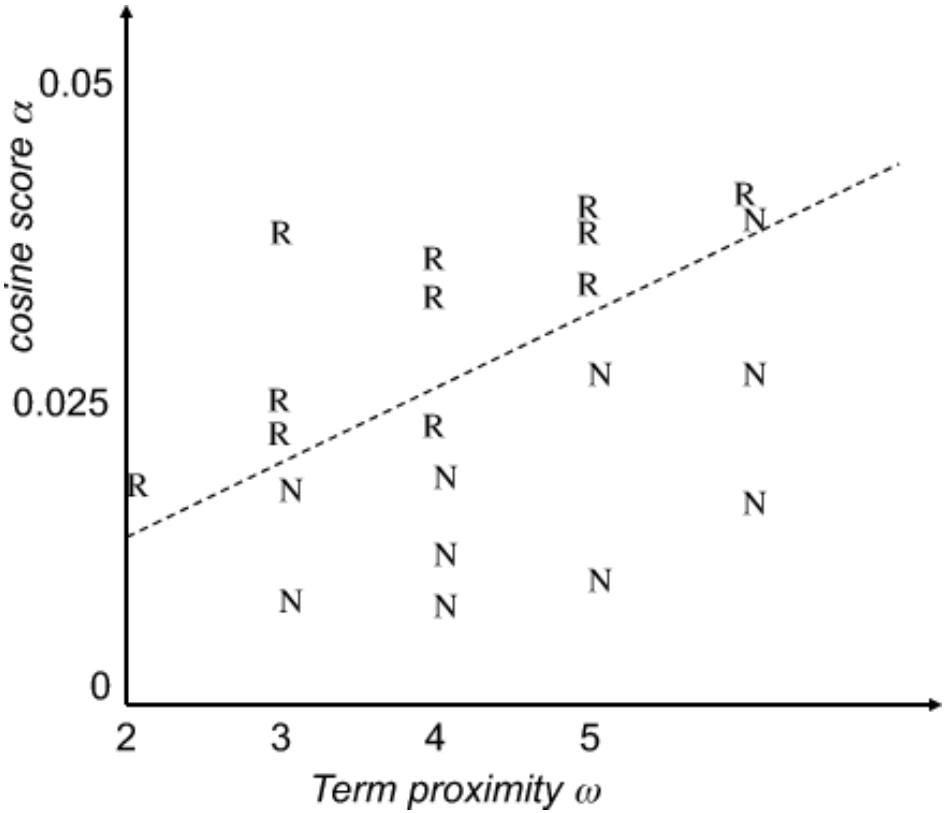
Machine-learned relevance: Setup (2)

- Two classes: relevant = 1 and nonrelevant = 0
- We now seek a scoring function that combines the values of the features to generate a value that is (close to) 0 or 1.
- We wish this function to be in agreement with our set of training examples as much as possible.
- The simplest classifier is a linear classifier, defined by an equation of the form:

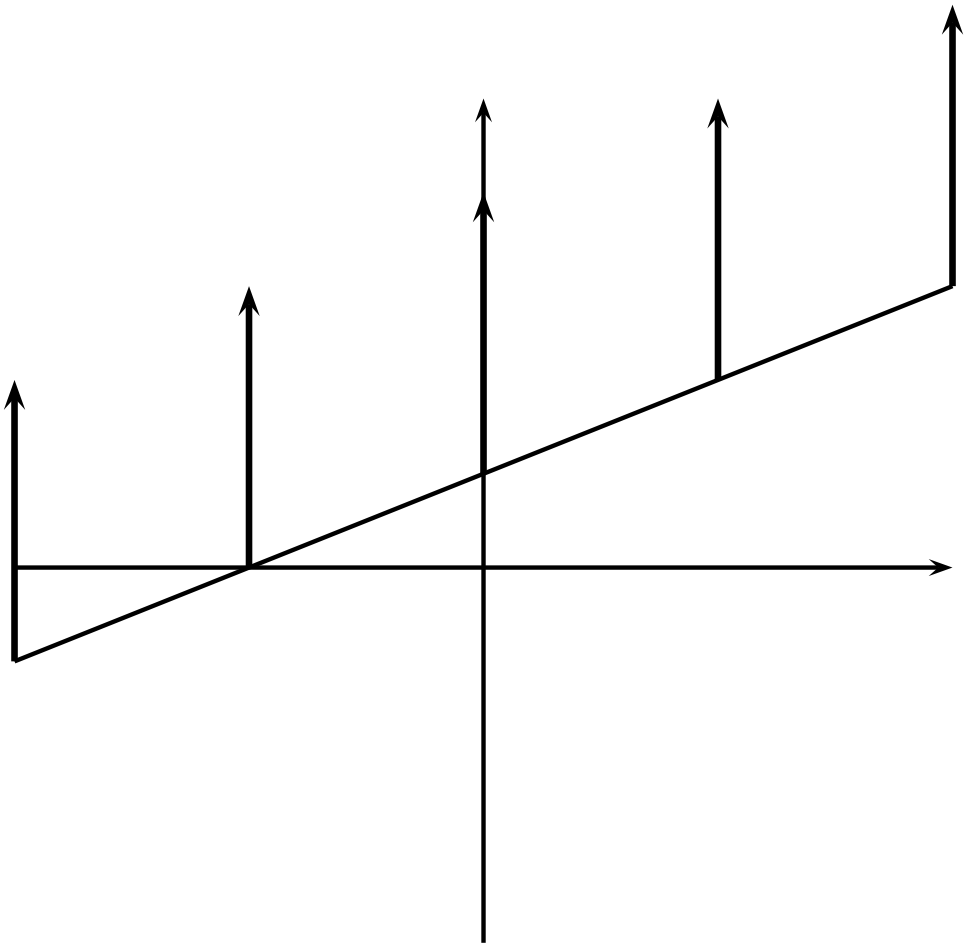
$$Score(d, q) = Score(\alpha, \omega) = a\alpha + b\omega + c,$$

where we learn the coefficients a, b, c from training data.

Graphic representation of the training set

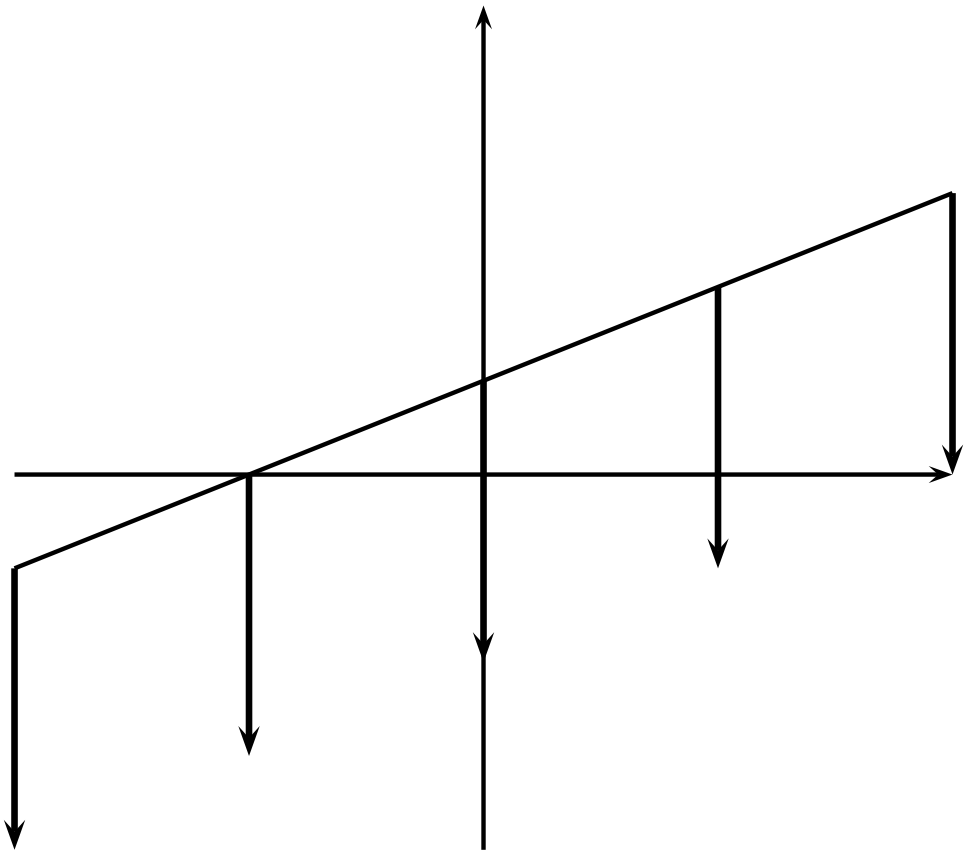


In this case, we learn a linear classifier in 2D



- A linear classifier in 2D is a line described by the equation $w_1 d_1 + w_2 d_2 = \theta$
- Example for a 2D linear classifier
- Points $(d_1\ d_2)$ with $w_1 d_1 + w_2 d_2 \geq \theta$ are in the class c.
- Points $(d_1\ d_2)$ with $w_1 d_1 + w_2 d_2 < \theta$ are in the complement class $\overline{c}$.

In this case, we learn a linear classifier in 2D



- A linear classifier in 2D is a line described by the equation $w_1 d_1 + w_2 d_2 = \theta$
- Example for a 2D linear classifier
- Points $(d_1\ d_2)$ with $w_1 d_1 + w_2 d_2 \geq \theta$ are in the class c.
- Points $(d_1\ d_2)$ with $w_1 d_1 + w_2 d_2 < \theta$ are in the complement class $\overline{c}$.

Summary

- Machine-learned relevance
  - Assemble a training set of query-document-judgment triples
  - Train classification or regression model on training set
  - For a new query, apply model to all documents (actually: a subset)
  - Rank documents according to model's decisions
  - Return the top K (e.g., $K = 10$) to the user
- In principle, any classification/regression method can be used.
- Big advantage: we avoid hand-tuning scoring functions and simply learn them from training data.
- Bottleneck: we need to maintain a representative set of training examples whose relevance assessments must be made by humans.

Machine-learned relevance for more than two features

- The approach can be readily generalized to a large number of features.
- Any measure that can be calculated for a query-document pair is fair game for this approach.

LTR features used by Microsoft Research (1)

- Features derived from standard IR models: query term number, query term ratio, length, idf, sum/min/max/mean/variance of term frequency, sum/min/max/mean/variance of length normalized term frequency, sum/min/max/mean/variance of tf-idf weight, boolean model, BM25, LM-absolute-discounting, LM-dirichlet, LM-jelinek-mercer
- Most of these features can be computed for different zones: body, anchor, title, url, whole document

LTR features used by Microsoft Research (2)

- Web-specific features: number of slashes in url, length of url, inlink number, outlink number, PageRank, SiteRank
- Spam features: QualityScore
- Usage-based features: query-url click count, url click count, url dwell time
- All of these features can be assembled into a big feature vector and then fed into the machine learning algorithm. □

Shortcoming of what we've presented so far

- Approaching IR ranking like we have done so far is not necessarily the right way to think about the problem.
- Statisticians normally first divide problems into classification problems (where a categorical variable is predicted) versus regression problems (where a real number is predicted).
- In between: specialized field of ordinal regression
- Machine learning for ad hoc retrieval is most properly thought of as an ordinal regression problem.
- Next up: ranking SVMs, a machine learning method that learns an ordering directly.

Outline

Basic setup for ranking SVMs

- As before we begin with a set of judged query-document pairs.
- But we do not represent them as query-document-judgment triples.
- Instead, we ask judges, for each training query q, to order the documents that were returned by the search engine with respect to relevance to the query.
- We again construct a vector of features $\psi_j = \psi(d_j, q)$ for each document-query pair – exactly as we did before.
- For two documents d_i and d_j, we then form the vector of feature differences:

$$\Phi(d_i, d_j, q) = \psi(d_i, q) - \psi(d_j, q)$$

Training a ranking SVM

- Vector of feature differences: $\Phi(d_i, d_j, q) = \psi(d_i, q) - \psi(d_j, q)$
- By hypothesis, one of d_i and d_j has been judged more relevant.
- Notation: We write $d_i \prec d_j$ for "d_i precedes d_j in the results ordering".
- If d_i is judged more relevant than d_j, then we will assign the vector $\Phi(d_i, d_j, q)$ the class $y_{ijq} = +1$; otherwise -1.
- This gives us a training set of pairs of vectors and "precedence indicators".
- We can then train an SVM on this training set with the goal of obtaining a classifier that returns

$$\vec{w}^{\mathsf{T}}\Phi(d_i, d_j, q) > 0 \quad \text{iff} \quad d_i \prec d_j$$

Advantages of Ranking SVMs vs. Classification/regression

- Documents can be evaluated relative to other candidate documents for the same query ...
- ... rather than having to be mapped to a global scale of goodness.
- This often is an easier problem to solve since just a ranking is required rather than an absolute measure of relevance. □

Why simple ranking SVMs don't work that well

- Ranking SVMs treat all ranking violations alike.
  - But some violations are minor problems, e.g., getting the order of two relevant documents wrong.
  - Other violations are big problems, e.g., ranking a nonrelevant document ahead of a relevant document.
- In most IR settings, getting the order of the top documents right is key.
  - In the simple setting we have described, top and bottom ranks will not be treated differently.
- $\rightarrow$ Learning-to-rank frameworks actually used in IR are more complicated than what we have presented here. □

Example for superior performance of LTR

SVM algorithm that directly optimizes MAP (as opposed to ranking).
Proposed by: Yue, Finley, Radlinski, Joachims, ACM SIGIR 2002.
Performance compared to state-of-the-art models: cosine, tf-idf, BM25, language models (Dirichlet and Jelinek-Mercer)

	TREC 9		TREC 10	
Model	MAP	W/L	MAP	W/L
SVM^{Δ}_{map}	0.242	–	0.236	–
Best Func.	0.204	39/11 **	0.181	37/13 **
2nd Best	0.199	38/12 **	0.174	43/7 **
3rd Best	0.188	34/16 **	0.174	38/12 **

Learning-to-rank clearly better than non-machine-learning approaches

Assessment of learning to rank

- The idea of learning to rank is old.
  - Early work by Norbert Fuhr and William S. Cooper
- Renewed recent interest due to:
  - Better machine learning methods becoming available
  - More computational power
  - Willingness to pay for large annotated training sets
- Strengths of learning-to-rank
  - Humans are bad at fine-tuning a ranking function with dozens of parameters.
  - Machine-learning methods are good at it.
  - Web search engines use a large number of features $\rightarrow$ web search engines need some form of learning to rank. □

Information retrieval models: Pros and Cons

- Least effort: Boolean system
  - In general, low user satisfaction
- A little bit more effort: Vector space model
  - Acceptable performance in many cases
- State-of-the-art performance: BM25, LMs
  - You need to tune parameters.
- Best performance: learning to rank
  - But you need an expensive training set
- Noisy data or vocabulary mismatch queries/documents & no time to custom-build a solution & collection is not too large
  - Use Latent Semantic Indexing

Take-away

- Machine-learned relevance: We use machine learning to learn the relevance score (retrieval status value) of a document with respect to a query.
- Learning to rank: A machine-learning method that directly optimizes the ranking (as opposed to classification or regression accuracy)

Resources

- Chapter 15 of Introduction to Information Retrieval
- Resources at http://informationretrieval.org/essir2011
  - References to learning to rank literature
  - Microsoft learning to rank datasets
  - How Google tweaks ranking

Exercise

Write down the training set from the last exercise as a training set for a ranking SVM. □