NPFL103: Information Retrieval (4)

Ranked retrieval, Term weighting, Vector space model

Pavel Pecina

pecina@ufal.mff.cuni.cz

Institute of Formal and Applied Linguistics
Faculty of Mathematics and Physics
Charles University

Original slides are courtesy of Hinrich Schütze, University of Stuttgart.

Contents

Ranked retrieval

Ranked retrieval

- So far, our queries have been boolean - document is a match or not.
- Good for experts: precise understanding of the needs and collection.
- Good for applications: can easily consume thousands of results.
- Not good for the majority of users.
- Most users are not capable or lazy to write Boolean queries.
- 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.
- This doesn't overwhelm the user.
- Premise: the ranking algorithm works.
- ...More relevant results are ranked higher than less relevant results.

Scoring as the basis of ranked retrieval

- We wish to rank documents that are more relevant higher than documents that are less relevant.
- How can we accomplish such a ranking of the documents in the collection with respect to a query?
- Assign a score to each query-document pair, say in $[0, 1]$.
- This score measures how well document and query "match".

Query-document matching scores

- How do we compute the score of a query-document pair?
- Let's start with a one-term query.
- If the query term does not occur in the document: score should be 0.
- The more frequent the query term in the document, the higher the score
- We will look at a number of alternatives for doing this.

Take 1: Jaccard coefficient

- A commonly used measure of overlap of two sets
- Let A and B be two sets
- Jaccard coefficient:

$$\textsc{jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}, \quad \text{where}(A \neq \emptyset \text{ or } B \neq \emptyset)$$

- $\textsc{jaccard}(A, A) = 1$
- $\textsc{jaccard}(A, B) = 0$ if $A \cap B = 0$
- A and B don't have to be the same size.
- Always assigns a number between 0 and 1.

Jaccard coefficient: Example

What is the query-document score the Jaccard coefficient computes for:

- Query: “ides of March”
- Document: “Caesar died in March”
- $\textsc{jaccard}(q, d) = 1/6$

What's wrong with Jaccard?

- It ignores term frequency (how many occurrences a term has).
- Rare terms are more informative than frequent terms. Jaccard does not consider this information.

→ We need a more sophisticated way of normalizing for the length of a document.

Term weighting

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

Bag of words model

- We do not consider the order of words in a document.
- *John is quicker than Mary* and *Mary is quicker than John* are represented the same way.
- This is called a bag of words model.
- In a sense, this is a step back: The positional index was able to distinguish these two documents.
- We will look at "recovering" positional information later in this course.
- For now: bag of words model

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 use tf when computing query-document match 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.

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.

- Score for a document-query pair: sum over terms t in both q and d:

$$\text{tf-matching-score}(q, d) = \sum_{t \in q \cap d} (1 + \log \mathrm{tf}_{t,d})$$

- The score is 0 if none of the query terms is present in the document.

Frequency in document vs. frequency in collection

- In addition, to 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.

Desired weight for rare terms

- Rare terms are more informative than frequent terms.
- Consider a term in the query that is rare in the collection (e.g., ARACHNOCENTRIC).
- A document containing this term is very likely to be relevant.

→ we want high weights for rare terms like ARACHNOCENTRIC.

Desired weight for frequent terms

- Frequent terms are less informative than rare terms.
- Consider a term in the query that is frequent in the collection (e.g., GOOD, INCREASE, LINE).
- A document containing this term is more likely to be relevant than a document that doesn't but words like GOOD, INCREASE and LINE are not sure indicators of relevance.

→ For frequent terms like GOOD, INCREASE, and LINE, we want positive weights but lower weights than for rare terms.

Document frequency

- We want high weights for rare terms like ARACHNOCENTRIC.
- We want low (positive) weights forfrequent words like GOOD, INCREASE, and LINE.
- We will use document frequency to factor this into computing the matching score.
- The document frequency is the number of documents in the collection that the term occurs in.

idf weight

- df_t is document frequency, the number of documents t occurs in.
- df_t is an inverse measure of the informativeness of term t.
- We define the idf weight of term t in a collection of N documents as:

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

- idf_t is a measure of the informativeness of the term.
- $\log N/\text{df}_t$ instead of $[N/\text{df}_t]$ to "dampen" the effect of idf
- Note that we use the log transformation for both term frequency and document frequency.

Examples for idf

Compute idf_t using the formula: $\text{idf}_t = \log_{10} \frac{1{,}000{,}000}{\text{df}_t}$

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 affects the documents ranking 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.

Collection frequency vs. Document frequency

word	collection frequency	document frequency
INSURANCE	10440	3997
TRY	10422	8760

- Collection frequency of t: number of tokens of t in the collection
- Document frequency of t: number of documents t occurs in
- Why these numbers?
- Which word is a better search term (should get a higher weight)?
- This example suggests that df/idf is better for weighting than cf.

tf-idf weighting

- tf-idf weight of a term is product of its tf weight and its idf weight.

$$w_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \cdot \log \frac{N}{\mathrm{df}_t}$$

- tf-weight
- idf-weight
- Best known weighting scheme in information retrieval.
- Increases with the number of occurrences within a document (tf).
- Increases with the rarity of the term in the collection (idf).
- Note: the "-" in tf-idf is a hyphen, not a minus (altso tf.idf, tf x idf).

Vector space model

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

Weight matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	5.25	3.18	0.00	0.00	0.00	0.35	
BRUTUS	1.21	6.10	0.00	1.00	0.00	0.00	
CAESAR	8.59	2.54	0.00	1.51	0.25	0.00	
CALPURNIA	0.00	1.54	0.00	0.00	0.00	0.00	
CLEOPATRA	2.85	0.00	0.00	0.00	0.00	0.00	
MERCY	1.51	0.00	1.90	0.12	5.25	0.88	
WORSER	1.37	0.00	0.11	4.15	0.25	1.95	
...							

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

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/hundreds of millions of dimensions when you apply this to web search engines
- Each vector is very sparse - most entries are zero.

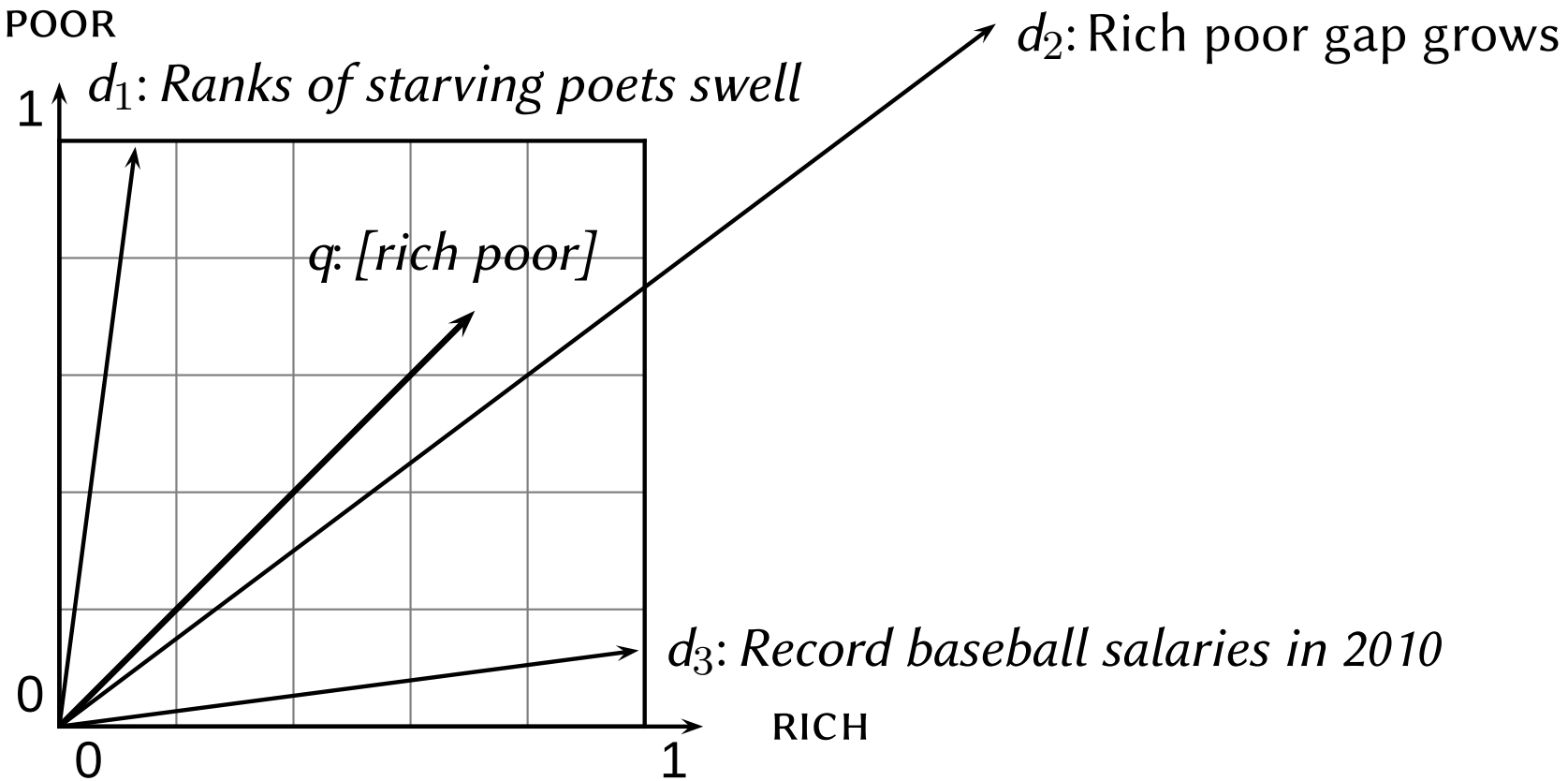
Queries as vectors

- Key idea 1: Do the same for queries: represent them as vectors
- Key idea 2: Rank documents according to their proximity to 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 ones

How do we formalize vector space similarity?

- Negative distance between two points/end points of the two vectors?
- Euclidean distance?
- Bad idea – 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 query q and the distribution of terms in document d_2 are similar.

Use angle instead of distance

- Rank documents according to angle with query
- Thought experiment: take a document d and append it to itself. Call this document d' (d' is twice as long as d).
- "Semantically" d and d' have the same content.
- The angle between the two documents is 0, corresponding to maximal similarity ...

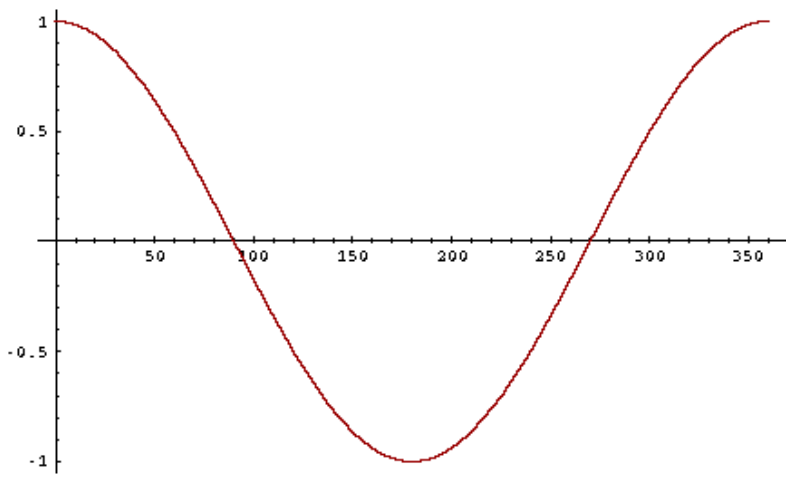
  ...even though the Euclidean distance between the two documents can be quite large.

From angles to cosines

- Ranking documents according to the angle between query and document in decreasing order

is equivalent to

- Ranking documents according to cosine(query,document) in increasing order.
- Cosine is a monotonically decreasing function of the angle for the interval $[0°, 180°]$



Length normalization

- How do we compute the cosine?
- A vector can be (length-) normalized by dividing each of its components by its length – e.g. by the L_2 norm: $||x||_2 = \sqrt{\sum_i x_i^2}$
- This maps vectors onto the unit sphere since after normalization: $||x||_2 = \sqrt{\sum_i x_i^2} = 1.0$
- As a result, longer documents and shorter documents have weights of the same order of magnitude.
- Effect on the two documents d and d' (d appended to itself) from earlier slide: they have identical vectors after length-normalization.

Cosine similarity between query and document

$$\cos(\vec{q}, \vec{d}) = \text{SIM}(\vec{q}, \vec{d}) = \frac{\vec{q} \cdot \vec{d}}{|\vec{q}||\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2} \sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

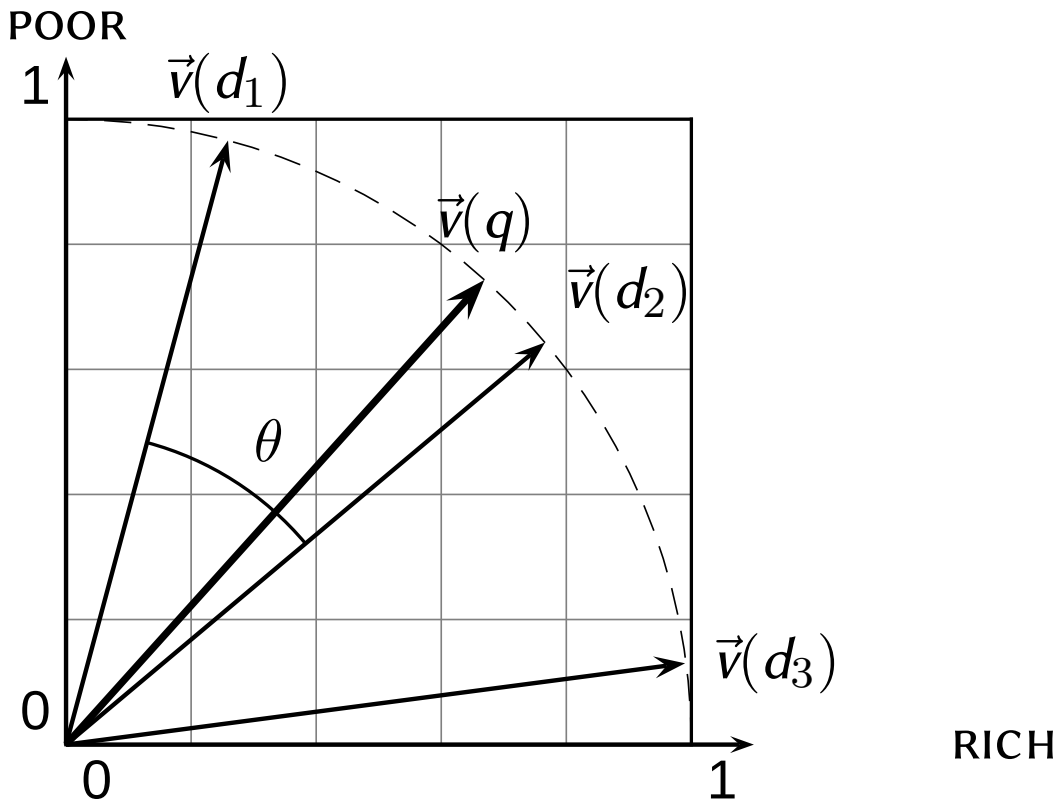
- q_i is the tf-idf weight of term i in the query.
- d_i is the tf-idf weight of term i in the document.
- $|\vec{q}|$ and $|\vec{d}|$ are the lengths of $\vec{q}$ and $\vec{d}$.

- This is the cosine similarity of $\vec{q}$ and $\vec{d}$ or, equivalently: the cosine of the angle between $\vec{q}$ and $\vec{d}$.

- For normalized vectors, the cosine is equivalent to the dot product:

$$\cos(\vec{q}, \vec{d}) = \vec{q} \cdot \vec{d} = \sum_i q_i \cdot d_i$$

Cosine similarity illustrated



Cosine: Example

How similar are these novels?

SaS: Sense and Sensibility

PaP: Pride and Prejudice

WH: Wuthering Heights

term frequencies (counts)

term	SaS	PaP	WH
AFFECTION	115	58	20
JEALOUS	10	7	11
GOSSIP	2	0	6
WUTHERING	0	0	38

Cosine: Example

term frequencies (counts)

term	SaS	PaP	WH
AFFECTION	115	58	20
JEALOUS	10	7	11
GOSSIP	2	0	6
WUTHERING	0	0	38

log frequency weighting

term	SaS	PaP	WH
AFFECTION	3.06	2.76	2.30
JEALOUS	2.0	1.85	2.04
GOSSIP	1.30	0	1.78
WUTHERING	0	0	2.58

(To simplify this example, we don't do idf weighting.)

Cosine: Example

log frequency weighting

term	SaS	PaP	WH
AFFECTION	3.06	2.76	2.30
JEALOUS	2.0	1.85	2.04
GOSSIP	1.30	0	1.78
WUTHERING	0	0	2.58

log frequency weighting & cosine normalization

term	SaS	PaP	WH
AFFECTION	0.79	0.83	0.52
JEALOUS	0.52	0.56	0.47
GOSSIP	0.34	0.0	0.41
WUTHERING	0.0	0.0	0.59

- $\cos(\text{SaS,PaP}) \approx 0.79 * 0.83 + 0.52 * 0.56 + 0.34 * 0.0 + 0.0 * 0.0 \approx 0.94$
- $\cos(\text{SaS,WH}) \approx 0.79$
- $\cos(\text{PaP,WH}) \approx 0.69$
- Why do we have $\cos(\text{SaS,PaP}) > \cos(\text{SAS,WH})$?

Computing the cosine score

CosineScore(q)

```
1  float Scores[N] = 0
2  float Length[N]
3  for each query term t
4  do calculate w_{t,q} and fetch postings list for t
5      for each pair(d, tf_{t,d}) in postings list
6      do Scores[d] += w_{t,d} × w_{t,q}
7  Read the array Length
8  for each d
9  do Scores[d] = Scores[d]/Length[d]
10 return Top K components of Scores[]
```

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

Default: no weighting

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
- 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	tf-w	df	idf	weight	tf	tf-w	weight	n'lized	
auto	0	0.0	5000	2.3	0.0	1	1.0	1.0	0.52	0.00
best	1	1.0	50000	1.3	1.3	0	0.0	0.0	0.00	0.00
car	1	1.0	10000	2.0	2.0	1	1.0	1.0	0.52	1.04
insurance	1	1.0	1000	3.0	3.0	2	1.3	1.3	0.68	2.04

Key to columns: tf: raw term frequency, tf-w: 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

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

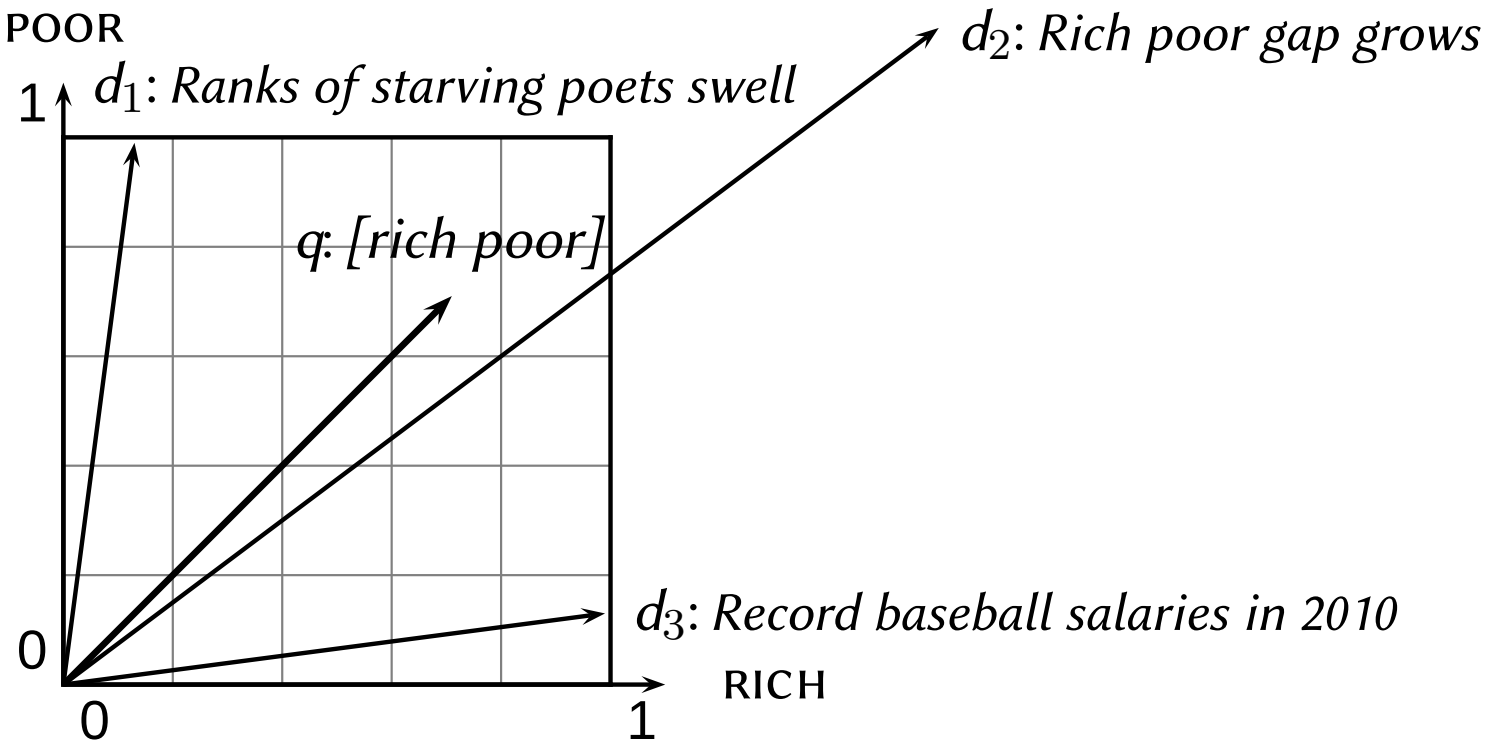
Similarity score between query and document: $\sum_i w_{qi} \cdot w_{di} = 0 + 0 + 1.04 + 2.04 = 3.08$

Summary: Ranked retrieval in the vector space model

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

Length normalization

Why distance is a bad idea



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

That's why we do length normalization or, equivalently, use cosine to compute query-document matching scores.

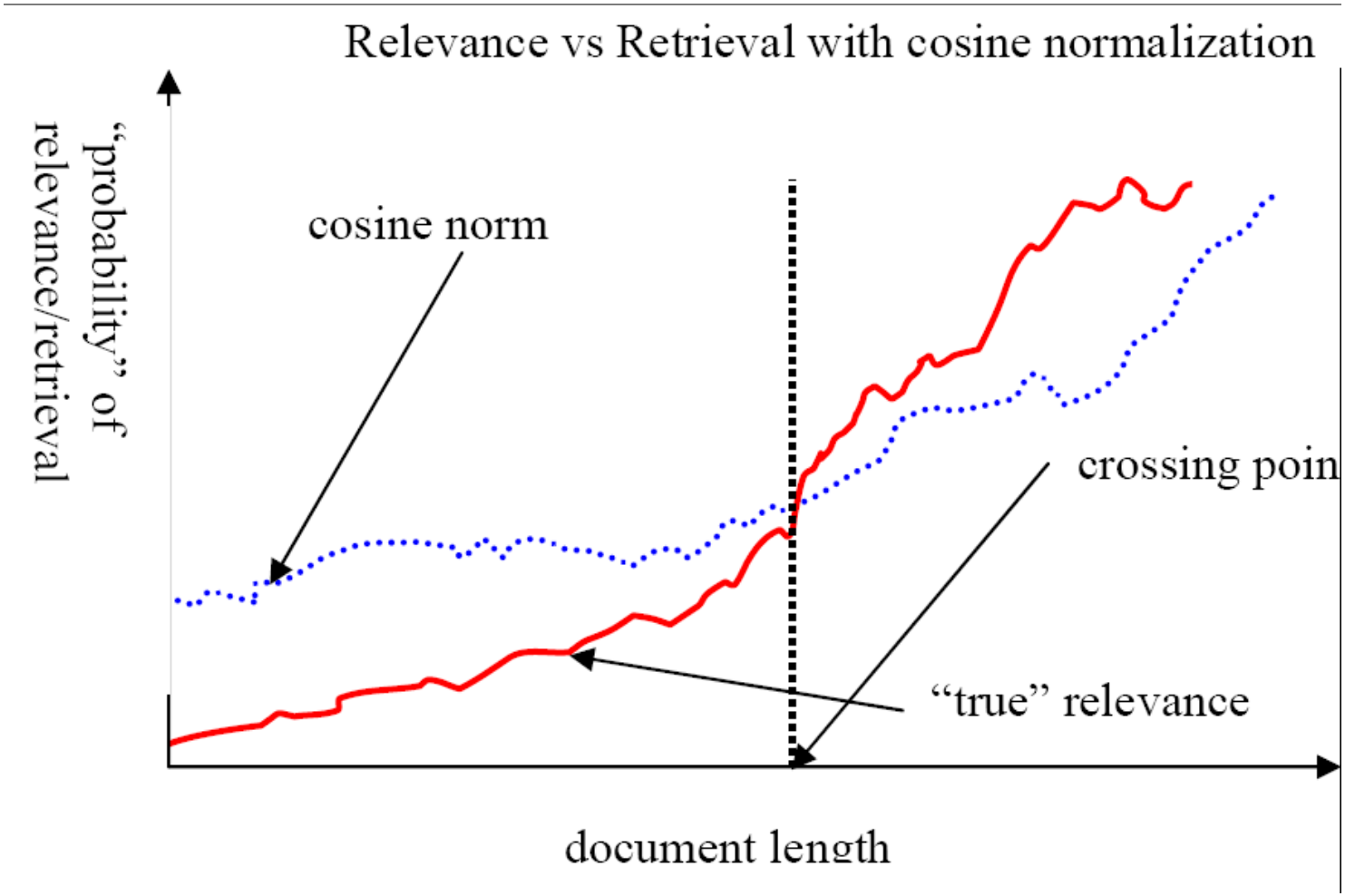
Exercise: A problem for cosine normalization

- 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?
  - d_2 is likely to be ranked below d_3 ...
  - ...but d_2 is more relevant than d_3.
- What can we do about this?

Pivot normalization

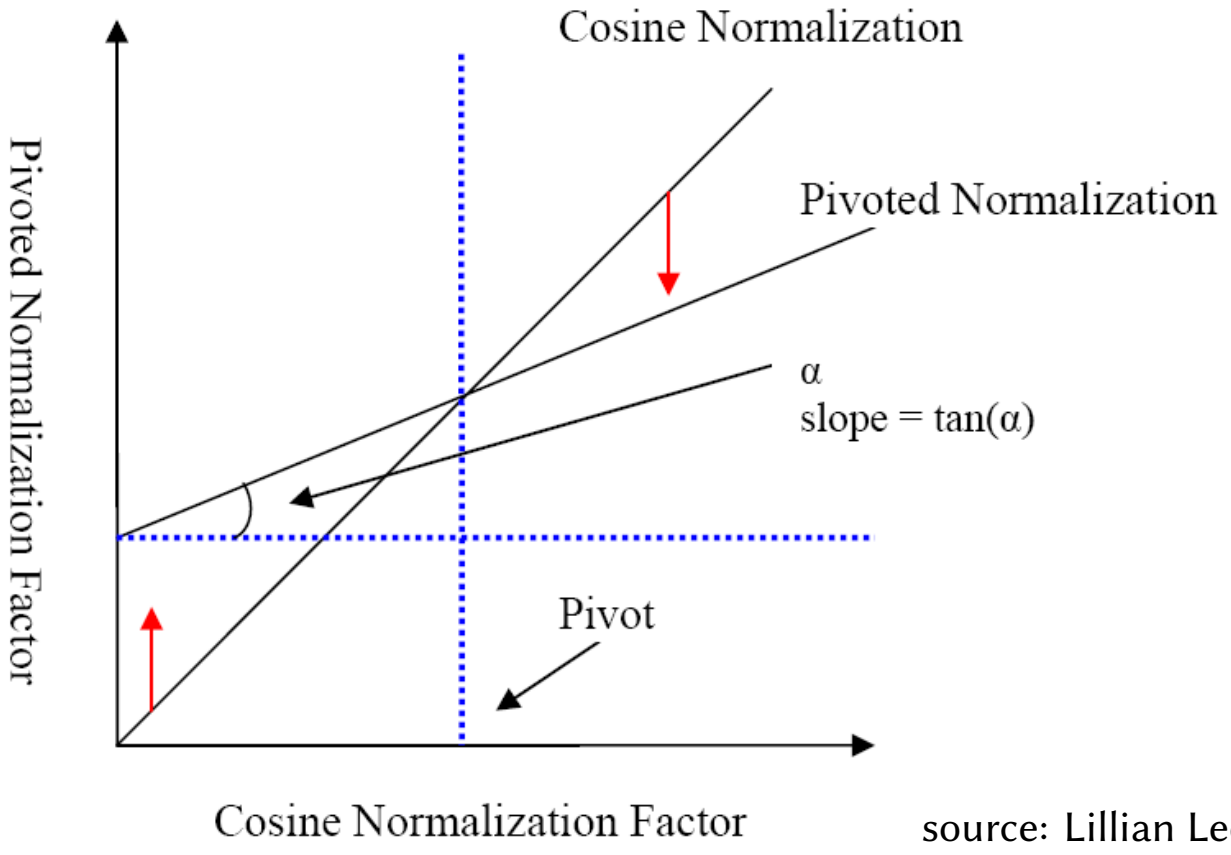
- Cosine normalization produces weights that are too large for short documents and too small for long documents (on average).
- Adjust cosine normalization by linear adjustment: "turning" the average normalization on the pivot
- Effect: Similarities of short documents with query decrease; similarities of long documents with query increase.
- This removes the unfair advantage that short documents have.
- Note that "pivoted" scores are no longer bounded by 1.

Predicted and true probability of relevance



source: Lillian Lee

Pivot normalization



source: Lillian Lee

- Normalizing factor: $\alpha|\vec{d}| + (1-\alpha)piv$, where $|\vec{d}| = \sqrt{\sum_{i=1}^{|V|} d_i^2}$
- The slope is $\alpha < 1$
- It crosses the $y = x$ line at *piv*

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)