Lecture 4: Term Weighting and the Vector Space Model

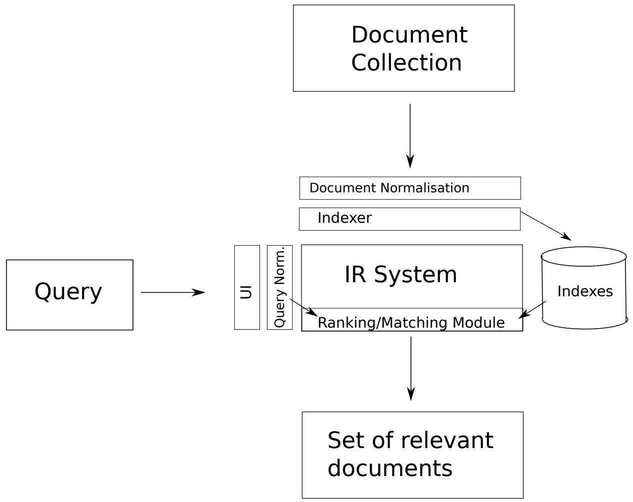
Information Retrieval
Computer Science Tripos Part II

Ronan Cummins[1]

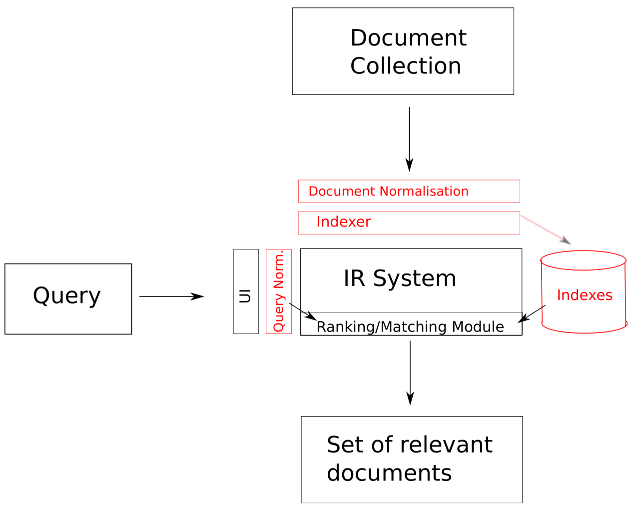
Natural Language and Information Processing (NLIP) Group
UNIVERSITY OF CAMBRIDGE
ronan.cummins@cl.cam.ac.uk

[1] Adapted from Simone Teufel's original slides

IR System Components

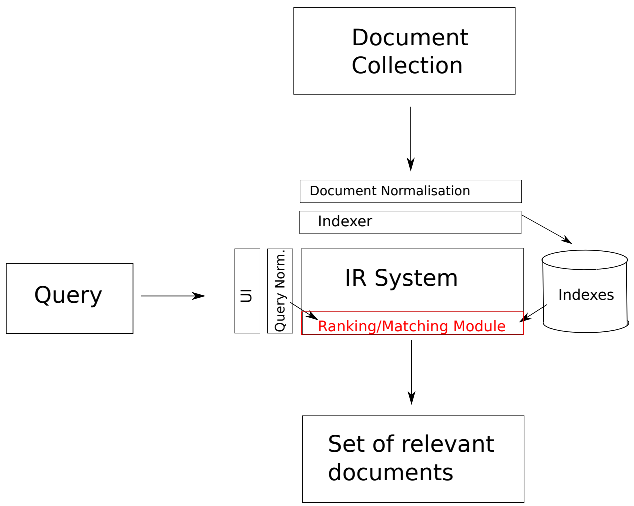


IR System Components



Finished with indexing, query normalisation

IR System Components



Today: the matcher

Overview

1. Recap
2. Why ranked retrieval?
3. Term frequency
4. Zipf's Law and tf-idf weighting
5. The vector space model

Overview

1. Recap
2. Why ranked retrieval?
3. Term frequency
4. Zipf's Law and tf-idf weighting
5. The vector space model

Recap: Tolerant Retrieval

- What to do when there is no exact match between query term and document term?
- Dictionary as hash, B-tree, trie
- Wildcards via permuterm
- and k-gram index
- k-gram index and edit-distance for spelling correction

Upcoming

- Ranking search results: why it is important (as opposed to just presenting a set of unordered Boolean results)
- Term frequency: This is a key ingredient for ranking.
- Tf-idf ranking: best known traditional ranking scheme
- And one explanation for why it works: Zipf's Law
- Vector space model: One of the most important formal models for information retrieval (along with Boolean and probabilistic models)

Overview

1. Recap
2. Why ranked retrieval?
3. Term frequency
4. Zipf's Law and tf-idf weighting
5. The vector space model

Ranked retrieval

- Thus far, our queries have been Boolean.
  - 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
- Don't want to write Boolean queries or wade through 1000s of results.
- This is particularly true of web search.

Problem with Boolean search: Feast or famine

- Boolean queries often have either too few or too many results.

Query 1

standard AND user AND dlink AND 650
→ 200,000 hits Feast!

Query 2

standard AND user AND dlink AND 650 AND no AND card AND 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.
- In ranked retrieval, "feast or famine" is less of a problem.
- Condition: Results that are more relevant are ranked higher than results that are less relevant. (i.e., the ranking algorithm works.)

Scoring as the basis of ranked retrieval

- Rank documents in the collection according to how relevant they are to a query
- Assign a score to each query-document pair, say in $[0, 1]$.
- This score measures how well document and query "match".
- If the query consists of just one term . . .

lioness

  - Score should be 0 if the query term does not occur in the document.
  - 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:

$$\textsc{jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

$(A \neq \emptyset$ 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 match score that 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 doesn't consider term frequency (how many occurrences a term has).
- It also does not consider that that some terms are inherently more informative than frequent terms.
- We need a more sophisticated way of normalizing for the length of a document.
  - Later in this lecture, we'll use $|A \cap B|/\sqrt{|A \cup B|}$ (cosine) ...
  - ... instead of $|A \cap B|/|A \cup B|$ (Jaccard) for length normalization.

Overview

1. Recap
2. Why ranked retrieval?
3. Term frequency
4. Zipf's Law and tf-idf weighting
5. The 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 now represented as a count vector $\in \mathbb{N}^{|V|}$.

Bag of words model

- We do not consider the order of words in a document.
- Represented the same way:

> John is quicker than Mary
> Mary is quicker than John

- 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.

Term frequency tf

- The term frequency $\mathrm{tf}_{t,d}$ of term t in document d is defined as the number of times that t occurs in d.
- We could just use *tf* as is ("raw term frequency").
- A document with $\mathrm{tf} = 10$ occurrences of the term is more relevant than a document with $\mathrm{tf} = 1$ 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

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

$\mathrm{tf}_{t,d}$	0	1	2	10	1000
$w_{t,d}$	0	1	1.3	2	4

- 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.

Overview

1. Recap
2. Why ranked retrieval?
3. Term frequency
4. Zipf's Law and tf-idf weighting
5. The vector space model

Frequency in document vs. frequency in collection

- In addition, to term frequency (the frequency of the term in the document) . . .
- . . . we also want to reward terms which are rare in the document collection overall.
- Now: excursion to an important statistical observation about language.

Zipf's law

- How many frequent vs. infrequent terms should we expect in a collection?
- In natural language, there are a few very frequent terms and very many very rare terms.

Zipf's law

The i^{th} most frequent term has frequency cf_i proportional to $1/i$:

$$cf_i \propto \frac{1}{i}$$

- cf_i is collection frequency: the number of occurrences of the term t_i in the collection.

Zipf's law

Zipf's law

The i^{th} most frequent term has frequency cf_i proportional to $1/i$:

$$cf_i \propto \frac{1}{i}$$

- So if the most frequent term (*the*) occurs cf_1 times, then the second most frequent term (*of*) has half as many occurrences $cf_2 = \frac{1}{2} cf_1$ ...
- ...and the third most frequent term (*and*) has a third as many occurrences $cf_3 = \frac{1}{3} cf_1$ etc.
- Equivalent: $cf_i = p \cdot i^k$ and $\log cf_i = \log p + k \log i$ (for $k = -1$)
- Example of a power law

Zipf's Law: Examples from 5 Languages

Top 10 most frequent words in some large language samples:

	English			German			Spanish			Italian			Dutch	
1	the	61,847	1	der	7,377,879	1	que	32,894	1	non	25,757	1	de	4,770
2	of	29,391	2	die	7,036,092	2	de	32,116	2	di	22,868	2	en	2,709
3	and	26,817	3	und	4,813,169	3	no	29,897	3	che	22,738	3	het/'t	2,469
4	a	21,626	4	in	3,768,565	4	a	22,313	4	è	18,624	4	van	2,259
5	in	18,214	5	den	2,717,150	5	la	21,127	5	e	17,600	5	ik	1,999
6	to	16,284	6	von	2,250,642	6	el	18,112	6	la	16,404	6	te	1,935
7	it	10,875	7	zu	1,992,268	7	es	16,620	7	il	14,765	7	dat	1,875
8	is	9,982	8	das	1,983,589	8	y	15,743	8	un	14,460	8	die	1,807
9	to	9,343	9	mit	1,878,243	9	en	15,303	9	a	13,915	9	in	1,639
10	was	9,236	10	sich	1,680,106	10	lo	14,010	10	per	10,501	10	een	1,637
	BNC, 100Mw			"Deutscher Wortschatz", 500Mw			subtitles, 27.4Mw			subtitles, 5.6Mw			subtitles, 800Kw	

Zipf's law: Rank $\times$ Frequency $\sim$ Constant

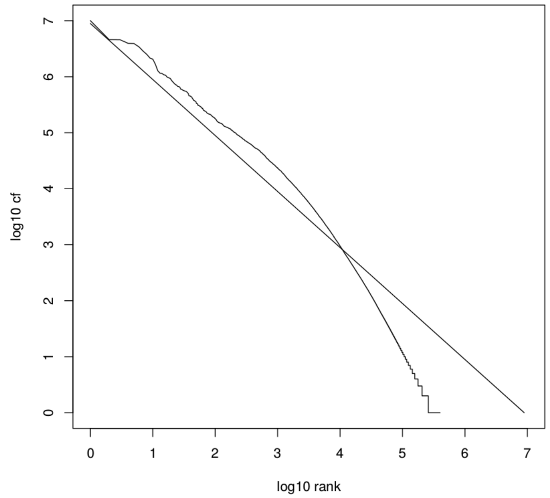
English:	Rank R	Word	Frequency f	$R \times f$
	10	he	877	8770
	20	but	410	8200
	30	be	294	8820
	800	friends	10	8000
	1000	family	8	8000

German:	Rank R	Word	Frequency f	$R \times f$
	10	sich	1,680,106	16,801,060
	100	immer	197,502	19,750,200
	500	Mio	36,116	18,059,500
	1,000	Medien	19,041	19,041,000
	5,000	Miete	3,755	19,041,000
	10,000	vorläufige	1.664	16,640,000

Other collections (allegedly) obeying power laws

- Sizes of settlements
- Frequency of access to web pages
- Income distributions amongst top earning 3% individuals
- Korean family names
- Size of earth quakes
- Word senses per word
- Notes in musical performances
- ...

Zipf's law for Reuters



Fit is not great.

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 for frequent 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 the document frequency, the number of documents that t occurs in.
- df_t is an inverse measure of the informativeness of term t.
- We define the idf weight of term t as follows:

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

  (N is the number of documents in the collection.)
- idf_t is a measure of the informativeness of the term.
- $\log \frac{N}{\mathrm{df}_t}$ instead of $\frac{N}{\mathrm{df}_t}$ 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: $\mathrm{idf}_t = \log_{10} \frac{1{,}000{,}000}{\mathrm{df}_t}$

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 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.

Collection frequency vs. Document frequency

Term	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
- Clearly, INSURANCE is a more discriminating search term and should get a higher weight.
- This example suggests that df (and idf) is better for weighting than cf (and "icf").

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

tf-idf weight

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

- tf-weight
- idf-weight
- Best known weighting scheme in information retrieval
- Alternative names: tf.idf, tf x idf

1. Recap
2. Why ranked retrieval?
3. Term frequency
4. Zipf's Law and tf-idf weighting
5. The 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 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 $\in \mathbb{R}^{|V|}$.

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.

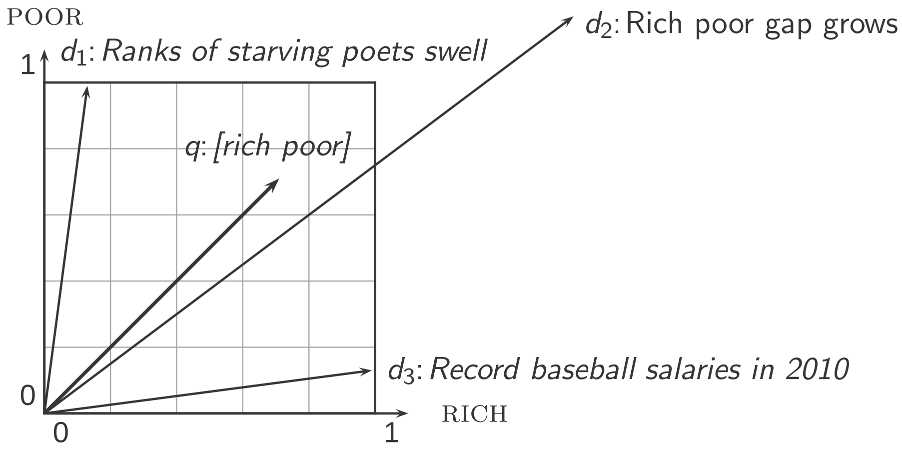
Queries as vectors

- 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 $\approx$ negative distance
- This allows us to 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
- 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

- 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 $[0^\circ, 180^\circ]$

Length normalization

- How do we compute the cosine?
- A vector can be (length-) normalized by dividing each of its components by its length – here we use 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

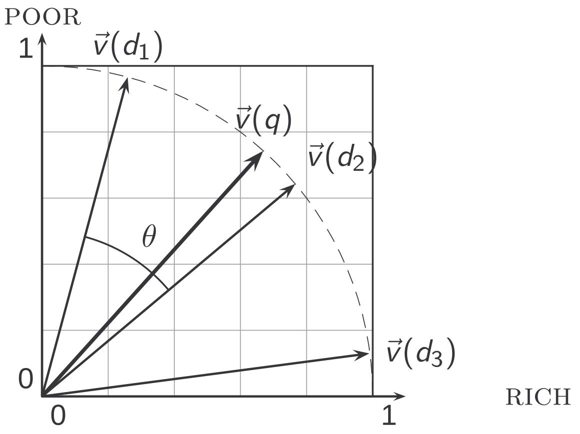
$$\cos(\vec{q},\vec{d}) = \textsc{sim}(\vec{q},\vec{d}) = \frac{\vec{q}\cdot\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

- 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 $\vec{d}$.

Cosine for normalized vectors

- For normalized vectors, the cosine is equivalent to the dot product or scalar product.
- $\cos(\vec{q},\vec{d}) = \vec{q}\cdot\vec{d} = \sum_i q_i \cdot d_i$
  - (if $\vec{q}$ and $\vec{d}$ are length-normalized).

Cosine similarity illustrated



Cosine: Example

How similar are the following novels?

SaS: Sense and Sensibility

PaP: Pride and Prejudice

WH: Wuthering Heights

Term frequencies (raw counts)

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

Cosine: Example

term	SaS	PaP	WH	SaS	PaP	WH	SaS	PaP	WH
	Term frequencies (raw counts)			Log frequency weighting			Log frequency weighting and cosine normalisation		
AFFECTION	115	58	20	3.06	2.76	2.30	0.789	0.832	0.524
JEALOUS	10	7	11	2.0	1.85	2.04	0.515	0.555	0.465
GOSSIP	2	0	6	1.30	0.00	1.78	0.335	0.000	0.405
WUTHERING	0	0	38	0.00	0.00	2.58	0.000	0.000	0.588

- (To simplify this example, we don't do idf weighting.)
- cos(SaS,PaP) $\approx$ $0.789 * 0.832 + 0.515 * 0.555 + 0.335 * 0.0 + 0.0 * 0.0 \approx 0.94$.
- cos(SaS,WH) ≈ 0.79
- cos(PaP,WH) ≈ 0.69

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)	$\max\{0, \log \frac{N-df_t}{df_t}\}$	u (pivoted unique)	$1/u$
b (boolean)	$\begin{cases} 1 & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/CharLength^\alpha$, $\alpha < 1$
L (log ave)	$\frac{1+\log(tf_{t,d})}{1+\log(ave_{t\in d}(tf_{t,d}))}$				

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:
- l ogarithmic tf
- n o df weighting
- c osine normalization

Query:
- l ogarithmic tf
- t – means idf
- n o normalization

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

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

Final 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

Reading

- MRS, Chapter 5.1.2 (Zipf's Law)
- MRS, Chapter 6 (Term Weighting)