Vector Space Models

Chenghua Lin

Department of Computer Science

University of Aberdeen

The Basic Question

Given a query, how do we know if document A is more relevant than B?

One Possible Answer

If document A uses more query words than document B

(Word usage in document A is more similar to that in query)

Relevance = Similarity

- Assumptions
  - Query and document are represented similarly
  - A query can be regarded as a “document”
  - Relevance(d,q) $\propto$ similarity(d,q)
- Key issues
  - How to represent query/document?
  - How to define the similarity measure?

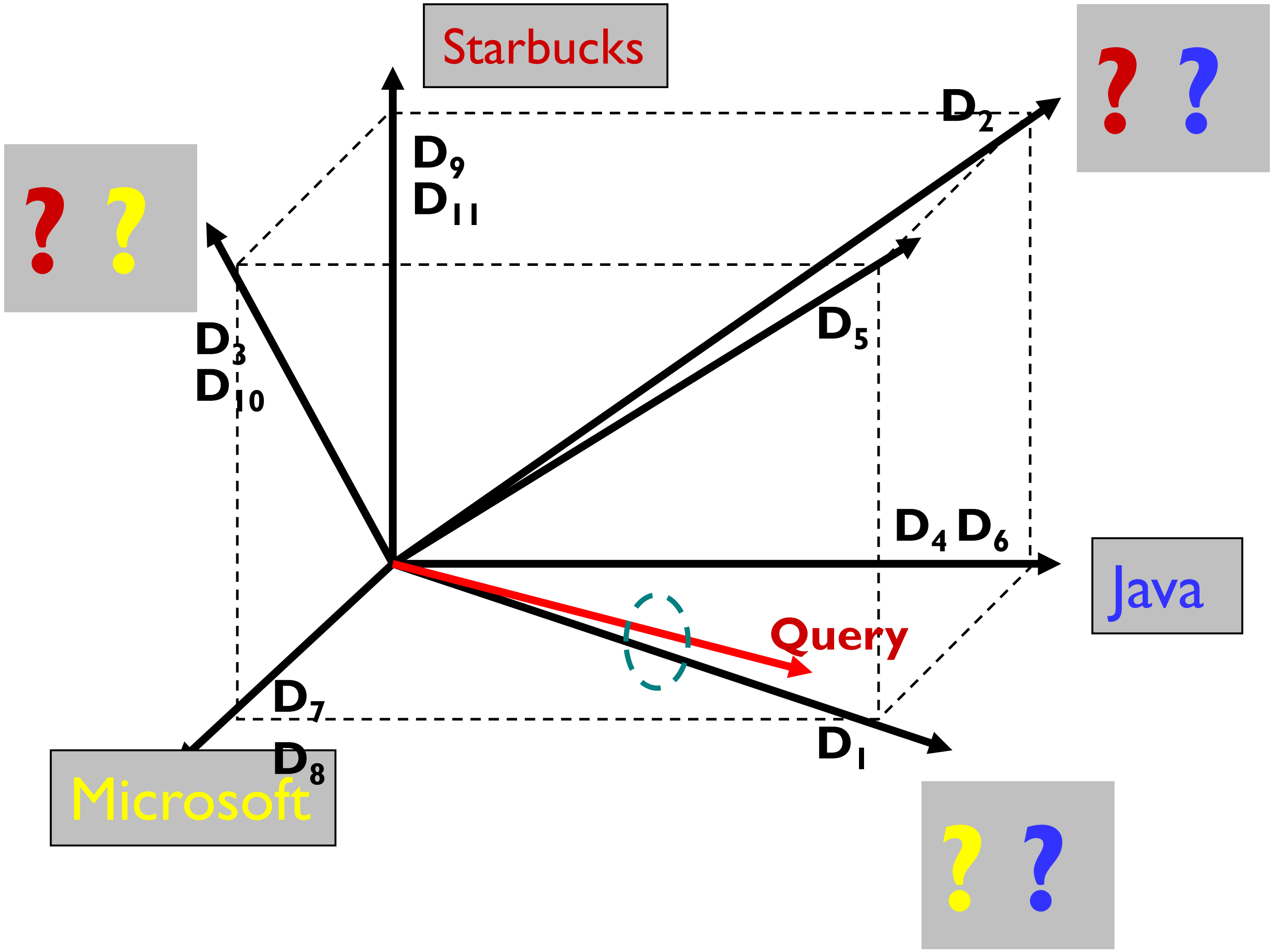
Bag of words **model**

- Vector representation doesn't consider the ordering of words in a document
- *John is quicker than Mary* and *Mary is quicker than John* have the same vectors
- This is called the <u>bag of words</u> model.
- For now: bag of words model

Vector Space Model

- Represent a doc/query by a term vector
  - Term: basic concept, e.g., word or phrase
  - Each term defines one dimension
  - N terms define a high-dimensional space
  - Element of vector corresponds to term weight
  - E.g., $d=(x_1,\ldots,x_N)$, x_i is “importance” of term i
- Measure relevance by the distance between the query vector and document vector in the vector space

VS Model: illustration



How to Assign Weights?

- Very very important!
- Why weighting
  - Query side: Not all terms are equally important
  - Doc side: Some terms carry more information about contents
- How?
  - Two basic heuristics
    - TF (Term Frequency) = Within-doc-frequency
    - IDF (Inverse Document Frequency)
  - Document length normalization

Term Frequency

- 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 want to use tf when computing query-document match scores. But how?
- Raw term frequency is not what we want:
  - A document with 10 occurrences of the term is more relevant than a document with 1 occurrence of the term.
  - But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

Log-frequency weighting

- The log frequency weight of term t in d is

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

- 0→0, 1→1, 2→1.3, 10→2, 1000→4, etc.
- Score for a document-query pair: sum over terms *t* in both *q* and *d*:
- Score = $\sum_{t \in q \cap d} (1 + \log_{10} tf_{t,d})$
- The score is 0 if none of the query terms is present in the document.

Empirical distribution of words

- There are stable language-independent patterns in how people use natural languages
- A few words occur very frequently; most occur rarely. E.g., in news articles,
  - Top 4 words: 10~15% word occurrences
  - Top 50 words: 35~40% word occurrences
- The most frequent word in one corpus may be rare in another

Top 10 most frequent words in a large language sample:

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

Document frequency

- Rare terms are more informative than frequent terms
  - Recall stop words
- Consider a term in the query that is rare in the collection (e.g., *cardio*)
- A document containing this term is very likely to be relevant to the query *cardio*
- → We want a high weight for rare terms like *cardio*.

Document frequency (cont.)

- Frequent terms are less informative than rare terms
- Consider a query term that is frequent in the collection (e.g., *high, increase, line*)
- A document containing such a term is more likely to be relevant than a document that doesn't
- But it's not a sure indicator of relevance.
- → For frequent terms, we want high positive weights for words like *high, increase,* and *line*
- But lower weights than for rare terms.
- We will use document frequency (df) to capture this.

idf weight

- df_t is the <u>document</u> frequency of t: the number of documents that contain *t*
  - df_t is an inverse measure of the informativeness of *t*
  - $\mathrm{df}_t \leq N$
- We define the idf (inverse document frequency) of *t* by

$$idf_t = \log_{10}(N/df_t)$$

  - We use log (N/df_t) instead of N/df_t to "dampen" the effect of idf.

idf example, suppose N = 1 million

term	df_t	idf_t
calpurnia	1	
animal	100	
sunday	1,000	
fly	10,000	
under	100,000	
the	1,000,000	

$$\mathrm{idf}_t = \log_{10}\left(N/\mathrm{df}_t\right)$$

There is one idf value for each term t in a collection.

tf-idf weighting

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

$$\mathrm{w}_{t,d} = \log(1+\mathrm{tf}_{t,d}) \times \log_{10}(N/\mathrm{df}_t)$$

- Best known weighting scheme in information retrieval
  - **Note: the "-" in tf-idf is a hyphen, not a minus sign!**
  - Alternative names: tf.idf, tf x idf
- Increases with the number of occurrences within a document
- Increases with the rarity of the term in the collection

Documents as vectors

- So we have a |V|-dimensional 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 a web search engine
- These are very sparse vectors - most entries are zero.

count → weight matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5.25	3.18	0	0	0	0.35
Brutus	1.21	6.1	0	1	0	0
Caesar	8.59	2.54	0	1.51	0.25	0
Calpurnia	0	1.54	0	0	0	0
Cleopatra	2.85	0	0	0	0	0
mercy	1.51	0	1.9	0.12	5.25	0.88
worser	1.37	0	0.11	4.15	0.25	1.95

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

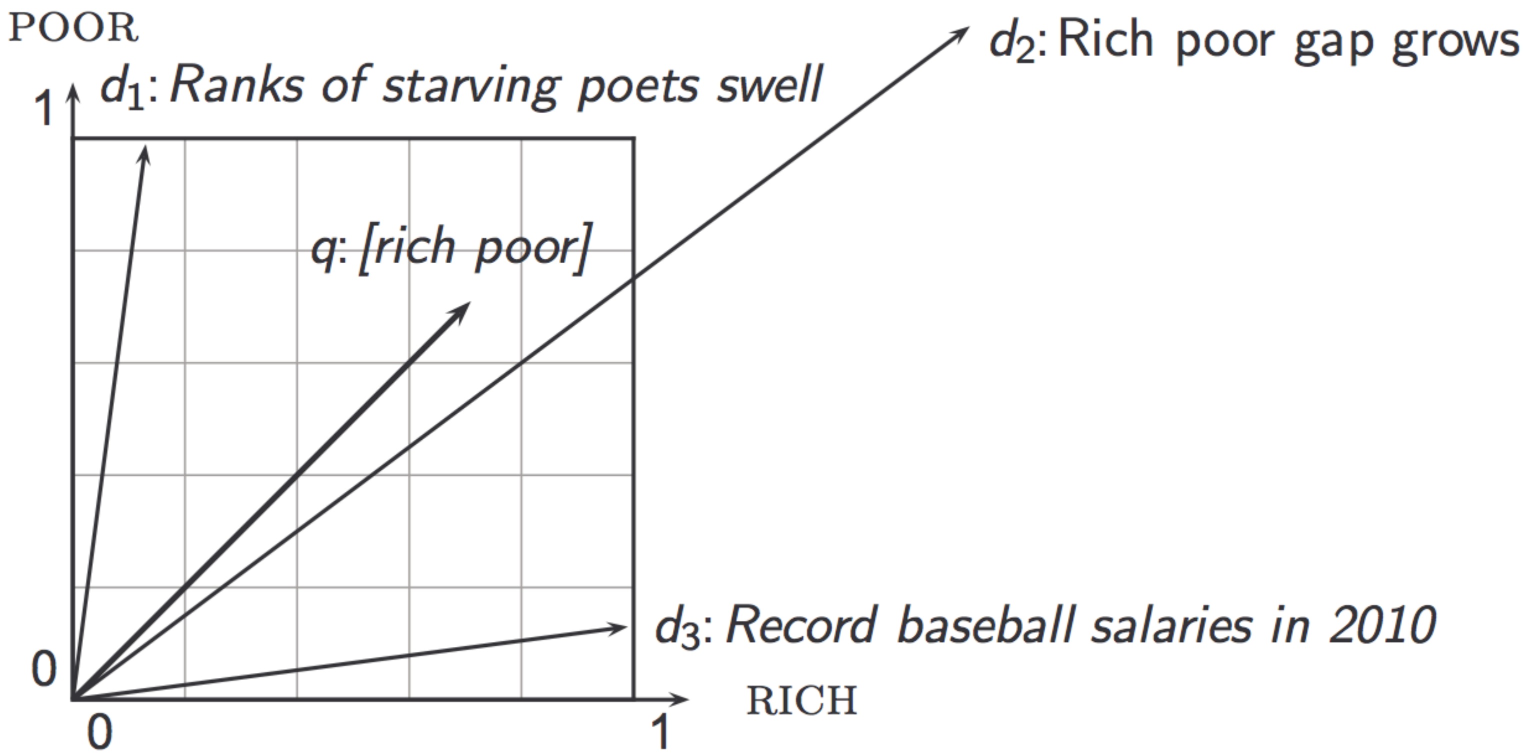
Queries as vectors

- Key idea 1: Do the same for queries: represent them as vectors in the space
- Key idea 2: Rank documents according to their proximity to the query in this space
- proximity = similarity of vectors
- proximity ≈ inverse of distance
- Instead: rank more relevant documents higher than less relevant documents

Formalising vector space proximity

- First cut: 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 between $\vec{q}$ and $\vec{d_2}$ is large even though the distribution of terms in the query q and the distribution of terms in the document d_2 are very similar.

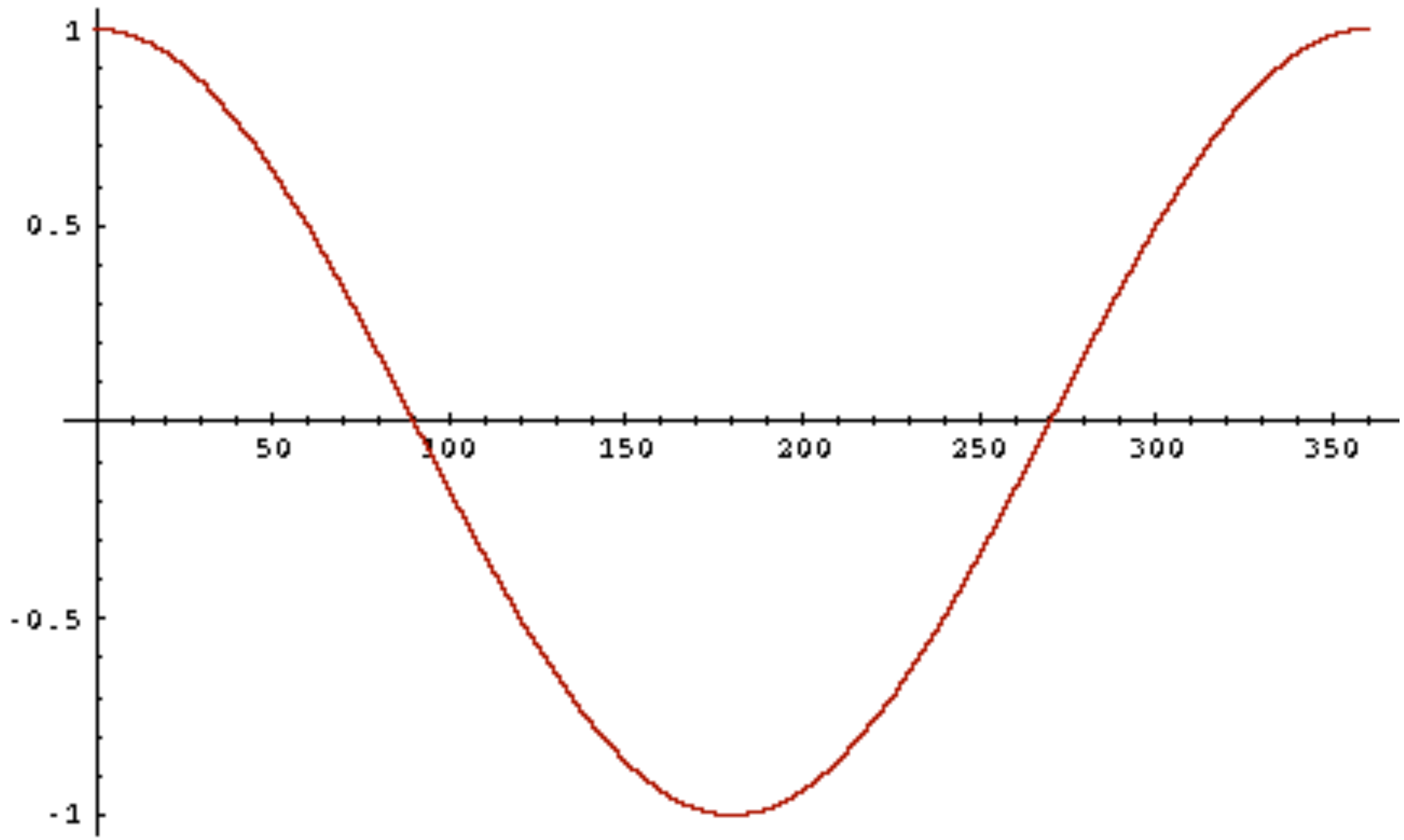
Use angle instead of distance

- Thought experiment: take a document *d* and append it to itself. Call this document *d*′.
- "Semantically" d and d′ have the same content
- The Euclidean distance between the two documents can be quite large
- The angle between the two documents is 0, corresponding to maximal similarity.
- Key idea: Rank documents according to angle with query.

From angles to cosines

- The following two notions are equivalent.
  - Rank documents in decreasing order of the angle between query and document
  - Rank documents in increasing order of cosine(query, document)
- Cosine is a monotonically decreasing function for the interval [0^o, 180^o]

From angles to cosines



- But how – *and why* – should we be computing cosines?

Length normalization

- A vector can be (length-) normalized by dividing each of its components by its length – for this we use the L_2 norm:

$$\left\|\vec{x}\right\|_2 = \sqrt{\sum_i x_i^2}$$

- Dividing a vector by its L_2 norm makes it a unit (length) vector (on surface of unit hypersphere)

- Effect on the two documents d and d′ (d appended to itself) from earlier slide: they have identical vectors after length-normalization.
  - Long and short documents now have comparable weights

cosine(query,document)

Dot product

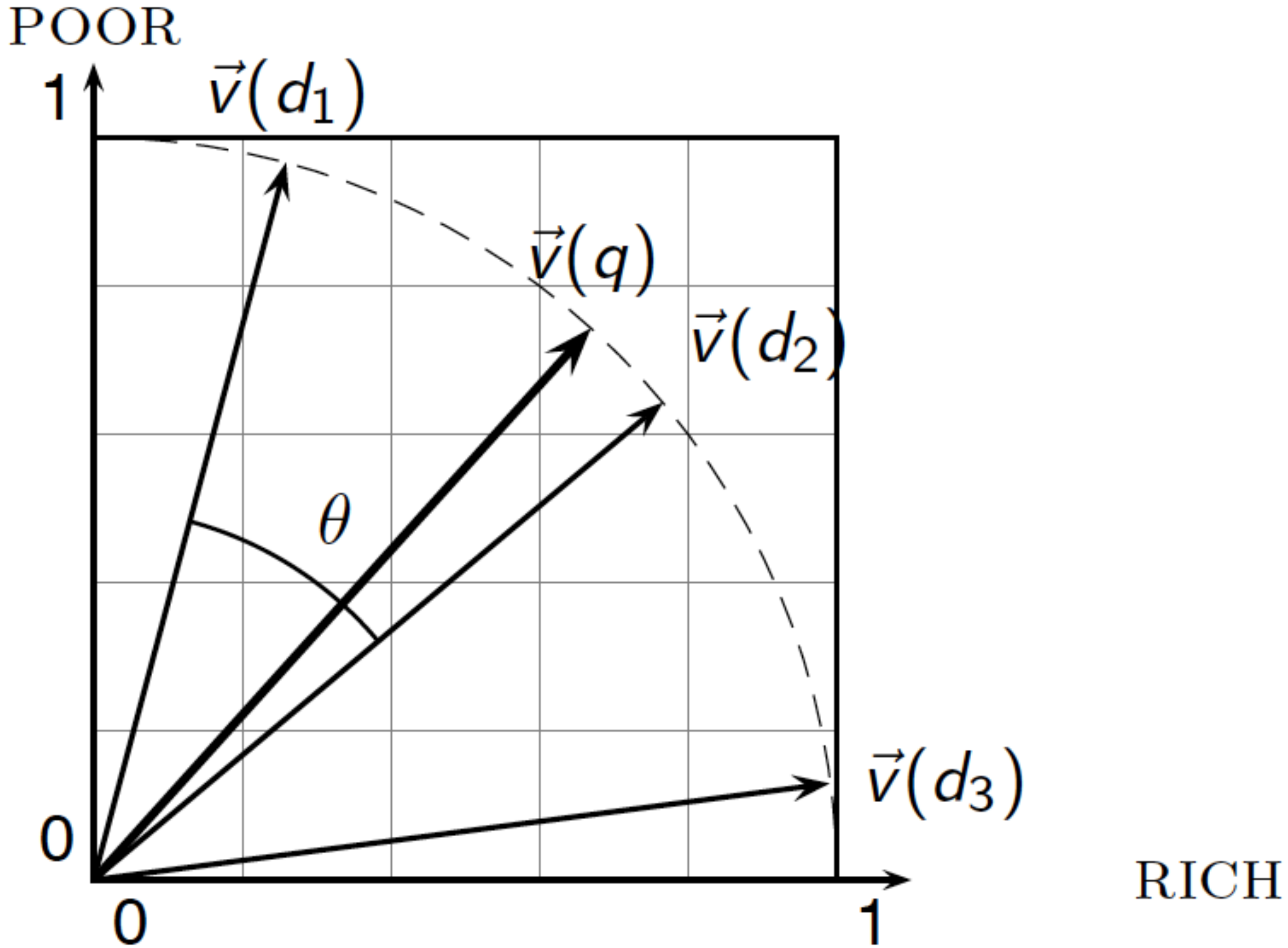
Unit vectors

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

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

$\cos(\vec{q},\vec{d})$ 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 illustrated



Cosine similarity amongst 3 documents

How similar are the novels

SaS: *Sense and Sensibility*

PaP: *Pride and Prejudice*, and

WH: *Wuthering Heights*?

term	SaS	PaP	WH
affection	115	58	20
jealous	10	7	11
gossip	2	0	6
wuthering	0	0	38

Term frequencies (counts)

Note: To simplify this example, we don't do idf weighting.

3 documents example contd.

Log frequency weighting

term	SaS	PaP	WH
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58

After length normalization

term	SaS	PaP	WH
affection	0.789	0.832	0.524
jealous	0.515	0.555	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

cos(SaS,PaP) ≈
$0.789 \times 0.832 + 0.515 \times 0.555 + 0.335 \times 0.0 + 0.0 \times 0.0$
≈ 0.94

cos(SaS,WH) ≈ 0.79

cos(PaP,WH) ≈ 0.69

Note: To simplify this example, we don't do idf weighting.

Summary – vector space ranking

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