Introduction to Information Retrieval

(Manning, Raghavan, Schutze)

Chapter 6
Scoring term weighting and the vector space model

Ranked retrieval

- Thus far, our queries have all been Boolean.
  - Documents either match or don't
- Good for expert users with precise understanding of their needs and the collection.
  - Also good for applications, which can easily consume 1000s of results
- Not good for the majority of users.
  - Most users incapable 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: "*standard user dlink 650*"
  - 200,000 hits
- Query 2: "*standard user dlink 650 no card found*"
  - 0 hits
- It takes skill to come up with a query that produces a manageable number of hits.
  - AND gives too few; OR gives too many

Ranked retrieval models

- Rather than a set of documents satisfying a query expression, in ranked retrieval, the system returns an ordering over the (top) documents in the collection for a query
- Free text queries: rather than a query language of operators and expressions, the user's query is just one or more words in natural language
  - Ranked retrieval has normally been associated with free text queries and vice versa
- With a ranked list of documents it does not matter how large the retrieved set is.
  - Just show top k results, don't overwhelm the user

Scoring as the basis of ranked retrieval

- We wish to return in order the documents most likely to be useful to the searcher
- How can we rank-order the documents in the collection with respect to a query?
- Assign a score – say in [0, 1] – to each document
- This score measures how well document and query “match”.

Query-document matching scores

- We need a way of assigning a score to 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 (should be)
- We will look at a number of alternatives for this.

Take 1: Jaccard coefficient

- A commonly used measure of overlap of two sets *A* and *B*
- jaccard$(A,B) = |A \cap B| / |A \cup B|$
- jaccard$(A,A) = 1$
- jaccard$(A,B) = 0$ if $A \cap B = 0$

- Always assigns a number between *0* and *1*.

Jaccard coefficient: Scoring example

- What is the query-document match score that the Jaccard coefficient computes for each of the two documents below?
- Query: *ides of march*
- Document 1: *caesar died in march*
- Document 2: *the long march*

Issues with Jaccard for scoring

- It doesn't consider term frequency (how many times a term occurs in a document)
  - tf weight
- Rare terms in a collection are more informative than frequent terms. Jaccard doesn't consider this information
  - idf weight
- We need a more sophisticated way of normalizing for length
  - cosine

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 bag of words model.

Term frequency

- The term frequency $tf_{t,d}$ of term t in document d is defined as the number of times that t occurs in d.
- We want to use term frequency when computing query-document match scores. But how?
- Raw term frequency may not be 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

term frequency (tf) weight

- many variants for tf weight, where log-frequency weighting is a common one, dampening the effect of raw tf (raw count)

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

- $0 \rightarrow 0$, $1 \rightarrow 1$, $2 \rightarrow 1.3$, $10 \rightarrow 2$, $1000 \rightarrow 4$, etc.
- The score is 0 if none of the query terms is present in the document.

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., *arachnocentric*)
- A document containing this term is very likely to be relevant to the query *arachnocentric*
- → We want a high weight for rare terms like *arachnocentric*.

Document frequency, continued

- 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 positive weights for words like *high, increase, and line*, but lower weights than for rare terms.
- We will use <u>document frequency</u> (df) to capture this in the score.
- df ($\leq N$) is the number of documents that contain the term

Inverse document frequency (idf) weight

- df_t is the document frequency of t: the number of documents that contain t
  - df_t is an inverse measure of the informativeness of t
  - Inverse document frequency is a direct measure of the informativeness of t
- We define the idf (inverse document frequency) of t by

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

  - use log to dampen the effect of N/df_t
- Most common variant of idf weight

idf example, suppose N= 1 million

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

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

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

Effect of idf on ranking

- Does idf have an effect on ranking for one-term queries, like
  - iPhone
- idf has no effect on ranking one term queries
  - idf affects the ranking of documents for queries with at least two terms
  - For the query capricious person, idf weighting makes occurrences of capricious count for much more in the final document ranking than occurrences of person.

Collection vs. Document frequency

- The collection frequency of *t* is the number of occurrences of *t* in the collection, counting multiple occurrences.
- Example: which word is a better search term (and should get a higher weight)?

Word	Collection frequency	Document frequency
insurance	10440	3997
try	10422	8760

- The example suggests that df is better for weighting than cf

tf-idf weighting

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

$$tf\text{-}idf_{t,d} = \text{tf weight (t,d) x idf weight (t)}$$

- Increases with the number of occurrences within a document
- Increases with the rarity of the term in the collection
- Best known instantiation of TF-IDF weighting

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

Note on terminology

- terminology is not standardized in the textbook/literature. $tf_{t,d}$ sometimes refers to the raw count, sometimes the weight derived from the raw count.

- We use $tf_{t,d}$ to mean the raw count only. So tf (raw count) and tf weight (weight derived from the raw count) are different. tf can be used as tf weight, but log tf is a more common variant.

Recall: Binary term-document incidence matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	0
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 by a binary vector $\in \{0,1\}^{|V|}$

Term-document count matrices

- Consider the number of occurrences of a term in a document:
  - Each document is a count vector in $\mathbb{N}^{v}$: a column below

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

Binary → count → weight matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5.25	2.44	0	0	0	0
Brutus	0.16	6.10	0	0.04	0	0
Caesar	8.59	8.40	0	0.07	0.04	0.04
Calpurnia	0	1.54	0	0	0	0
Cleopatra	2.85	0	0	0	0	0
mercy	1.51	0	2.27	3.78	3.78	0.76
worser	1.37	0	0.69	0.69	0.69	0

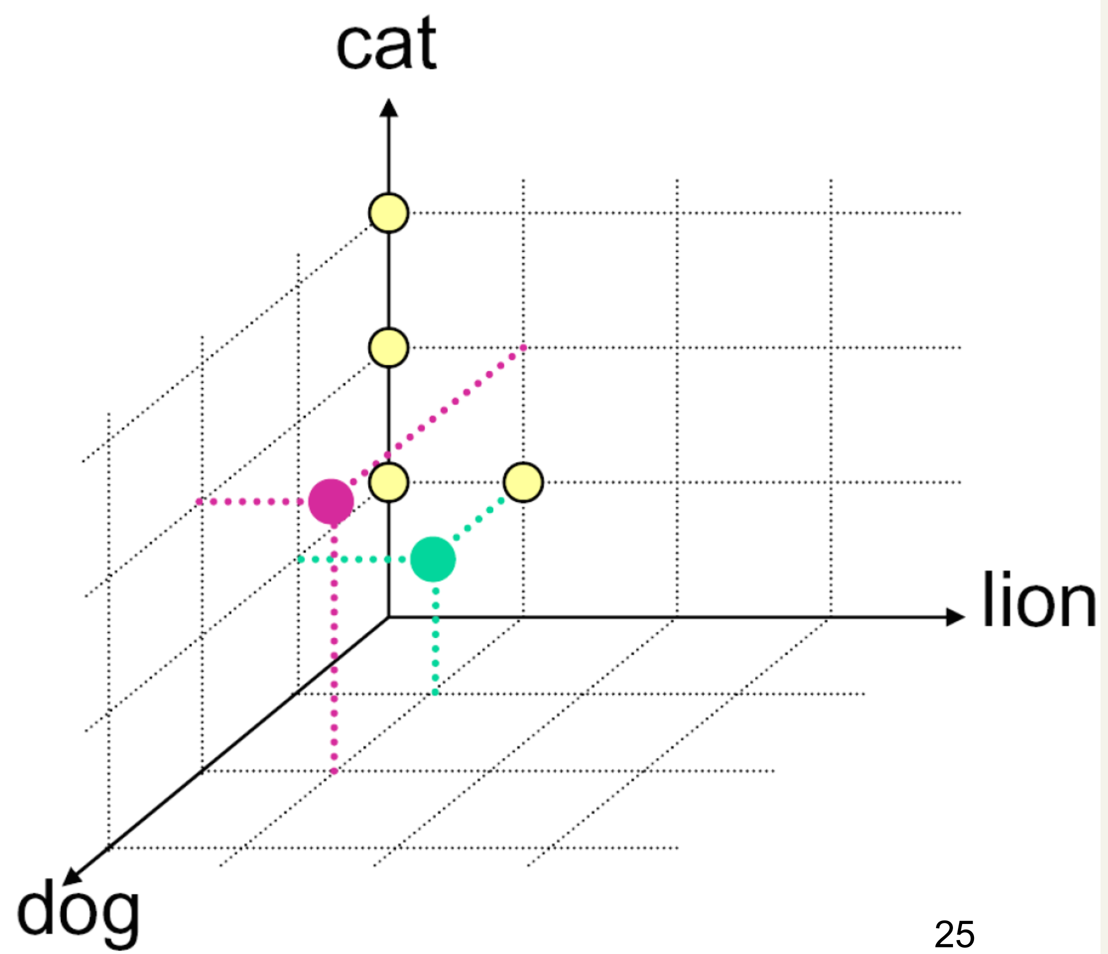
Each document is now represented by a real-valued vector of TF-IDF weights $\in \mathbb{R}^{|V|}$

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
  - hundreds of millions of dimensions when you apply this to a web search engine
- This is a very sparse vector
  - most entries are zero

Example: raw counts as weights

- cat
- cat cat
- cat cat cat
- cat lion
- lion cat
- cat lion dog
- cat cat lion dog dog



Queries as vectors

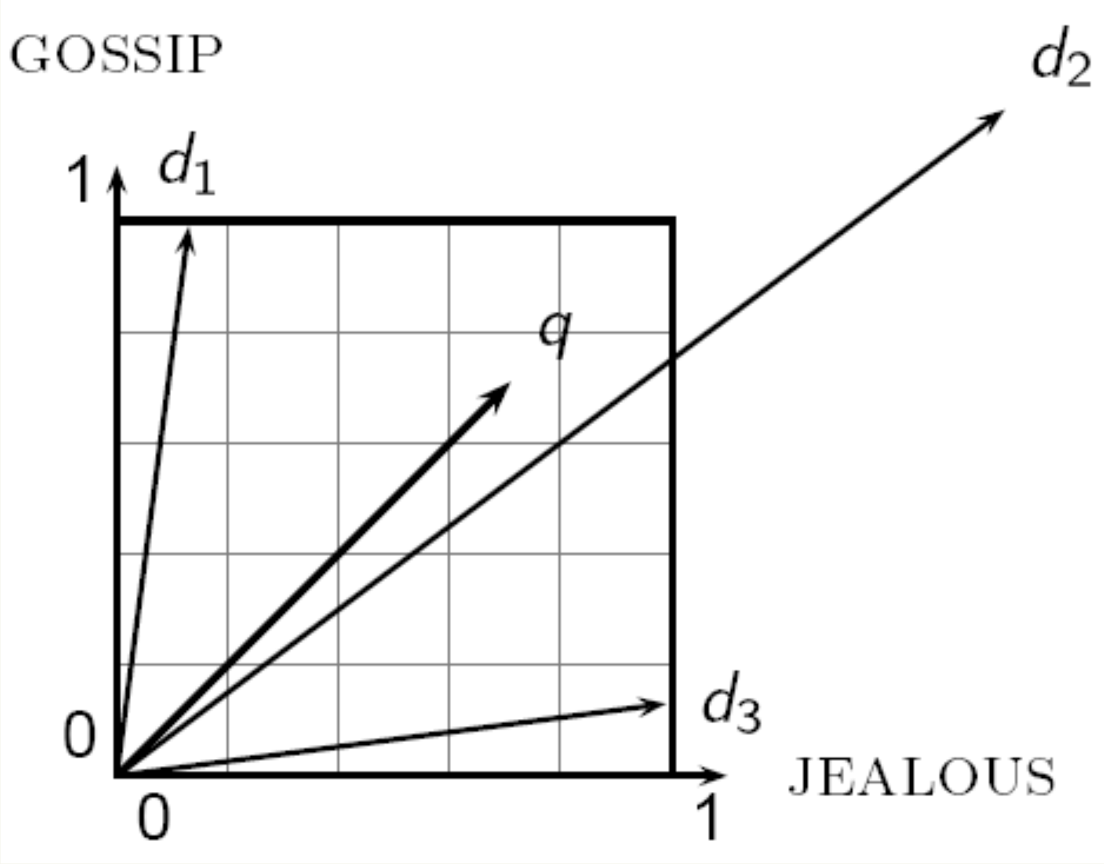
- Key idea 1: Do the same for queries: represent them as vectors in the space
- Key idea 2: Rank documents according to their proximity to the query in this space
- proximity = similarity of vectors
- proximity ≈ inverse of distance
- Recall: We do this because we want to get away from the either-in-or-out Boolean model.
- Instead: rank more relevant documents higher than less relevant documents

Formalizing vector space proximity

- Distance between two vectors
  - between two end points of the two vectors
  - Euclidean distance?
  - a bad idea. It’s large for vectors of different lengths.

Why Euclidean distance is bad

The Euclidean distance between $\vec{q}$ and $\vec{d_2}$ is large even though the distribution of terms in the query $\vec{q}$ and the distribution of terms in the document $\vec{d_2}$ are very similar.

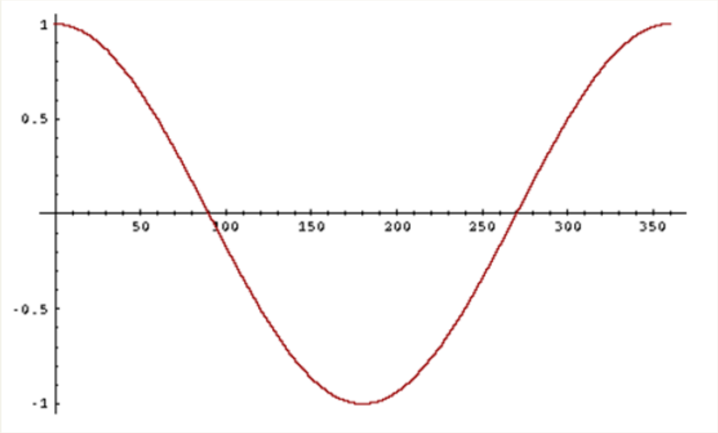


Use angle instead of distance

- 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.

From angles to cosines

- The following two notions are equivalent.
  - Rank documents in <u>decreasing</u> order of the angle between query and document
  - Rank documents in <u>increasing</u> order of cosine(query, document)



- Cosine is a monotonically decreasing function for the interval [0^o, 180^o]
- In general, cosine similarity ranges [-1, 1]
- In the case of information retrieval, the cosine similarity of two documents will range from 0 to 1
  - term frequencies (tf-idf weights) cannot be negative
  - The angle between two term frequency vectors cannot be greater than 90°
  - cosine (90) = 0, (completely unrelated)
  - cosine (0) = 1, (completely related)

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: $\|\vec{x}\|_2 = \sqrt{\sum_i x_i^2}$
- Dividing a vector by its L_2 norm makes it a unit (length) vector
- Effect on the two documents d and d′ (d appended to itself) from earlier slide: they have identical vectors after length-normalization.
- The cosine of the angle between two normalized vectors is the dot product of the two

cosine(query,document)

Dot product

Unit vectors

$$\cos(\vec{q},\vec{d}) = \frac{\vec{q}\bullet\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\vec{q}}{|\vec{q}|}\bullet\frac{\vec{d}}{|\vec{d}|} = \frac{\sum_{t=1}^{|V|} q_t d_t}{\sqrt{\sum_{t=1}^{|V|} q_t^2}\sqrt{\sum_{t=1}^{|V|} d_t^2}}$$

q_t is the tf-idf weight of term t in the query
d_t is the tf-idf weight of term t 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 $\vec{d}$.

- The cosine similarity can be seen as a method of normalizing document length during comparison

Cosine similarity example

	d	q	normalized d	normalized q
t1	1.4	0.7	0.84	0.83
t2	0.8	0.47	0.48	0.56
t3	0.4	0	0.24	0

$$sim(d, q) = \frac{1.4 \times 0.7 + 0.8 \times 0.47 + 0.4 \times 0}{sqrt(1.4^2+0.8^2+0.4^2) \times sqrt(0.7^2+0.47^2+0^2)}$$

$$= \frac{1.36}{1.66 \times 0.84}$$

$$= 0.97$$

$$sim(d,q) = 0.84 \times 0.83 + 0.48 \times 0.56 + 0.24 \times 0 = 0.97$$

More on the cosine formula

$$\cos(\vec{q},\vec{d}) = \frac{\vec{q}\bullet\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\vec{q}}{|\vec{q}|}\bullet\frac{\vec{d}}{|\vec{d}|} = \frac{\sum_{t=1}^{|V|} q_t d_t}{\sqrt{\sum_{t=1}^{|V|} q_t^2}\sqrt{\sum_{t=1}^{|V|} d_t^2}} = \frac{\sum_{t\in T} q_t d_t}{\|q\| * \|d\|}$$

- T is the set of terms q and d share in common. If T is empty, then cosine similarity = 0
- q_t is the tf-idf weight of term t in the query q, i.e,
  tf weight(t,q) x idf weight(t)
- d_t is the tf-idf weight of term t in the document d, i.e,
  tf weight(t,d) x idf weight(t)
- In actual implementation, do we need to represent q and d as vectors of size |V| ?

More variants 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)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t\in d}(\mathrm{tf}_{t,d}))}$				

SMART notation: columns headed ‘n’ are acronyms for weight schemes.

Weighting may differ in queries vs documents

- Many search engines allow for different weightings for queries vs documents
- SMART notation: denotes the combination in use in an engine, with the notation ddd.qqq, using acronyms from the previous table
- A very standard weighting scheme: lnc.ltc
- Document: logarithmic tf, no idf, cosine normalization
  - no idf: for both effectiveness and efficiency reasons
- Query: logarithmic tf, idf, cosine normalization

lnc.ltc example

- document 1 = "good good news"   document 2 = "awful awful news"
- query = "good awful"
- In the table, log tf is the tf weight based on log-frequency weighting. d is the document vector. d' is the length-normalized d. q is the query vector. q' is the length-normalized q. Assume N=10,000,000

			d1="good good news"				d2="awful awful news"				query="good awful"			
terms	df	idf	tf	logtf	d	d'	tf	logtf	d	d'	tf	logtf	q	q'
awful	1000	4	0	0	0	0	2	1.3	1.3	0.793	1	1	4	0.894
good	100000	2	2	1.3	1.3	0.793	0	0	0	0	1	1	2	0.447
news	10000	3	1	1	1	0.61	1	1	1	0.61	0	0	0	0

- The cosine similarity between d and q is the dot product of d' and q'.
  - Cosine(d1,q) = 0x0.894 + 0.793x0.447 + 0.61x0 = 0.354
  - Cosine(d2,q) = 0.793x0.894 + 0x0.447 + 0.61x0 = 0.709
  - If idf is not used for the weighting of q?
- In implementation: representing vectors and computing cosine

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

Gerard Salton

- 1927-1995. Born in Germany, Professor at Cornell (co-founded the CS department), Ph.D from Harvard in Applied Mathematics
- Father of information retrieval
  - Vector space model
  - SMART information retrieval system
- First recipient of SIGIR outstanding contribution award, now called the Gerard Salton Award