Term Weighting and Vector Space Model

Reference: Introduction to Information Retrieval
by C. Manning, P. Raghavan, H. Schutze

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: Applications 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.
- With a ranked list of documents, it does not matter how large the retrieved set is.

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)

Term frequency *tf*

- 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 can use *tf* when computing query-document match scores.

Term frequency *tf*

- Sometimes, we may refine the raw term frequency:
  - A document with 10 occurrences of the term is more relevant than a document with one occurrence of the term.
  - But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

Log-frequency weighting

- Sometimes, the log frequency weight of term *t* in *d* is used:
- $$w_{t,d} = \begin{cases} 1 + \log tf_{t,d} & , \quad \text{if } tf_{t,d} > 0 \\ 0 & , \quad \text{otherwise} \end{cases}$$

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.
  - Just term frequency is not sufficient
  - We wish to capture the notion of rare terms
- We will use document frequency (df) to capture this in the score.
- df ($\leq N$) is the number of documents that contain the term

idf weight

- df_t is the document frequency of term *t*: the number of documents that contain *t*
  - df is a measure of the informativeness of *t*
- We define the idf (inverse document frequency) of *t* by

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

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

Will turn out the base of the log is unimportant

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

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.

$$w_{t,d} = \left(1 + \log(\mathrm{tf}_{t,d})\right) \times \log\left({}^{N}/_{\mathrm{df}_t}\right)$$

- A very common 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

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 $\in \mathbb{R}^{|V|}$

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.
- Does not consider the word ordering
  - Bag-of-word model

Queries as vectors

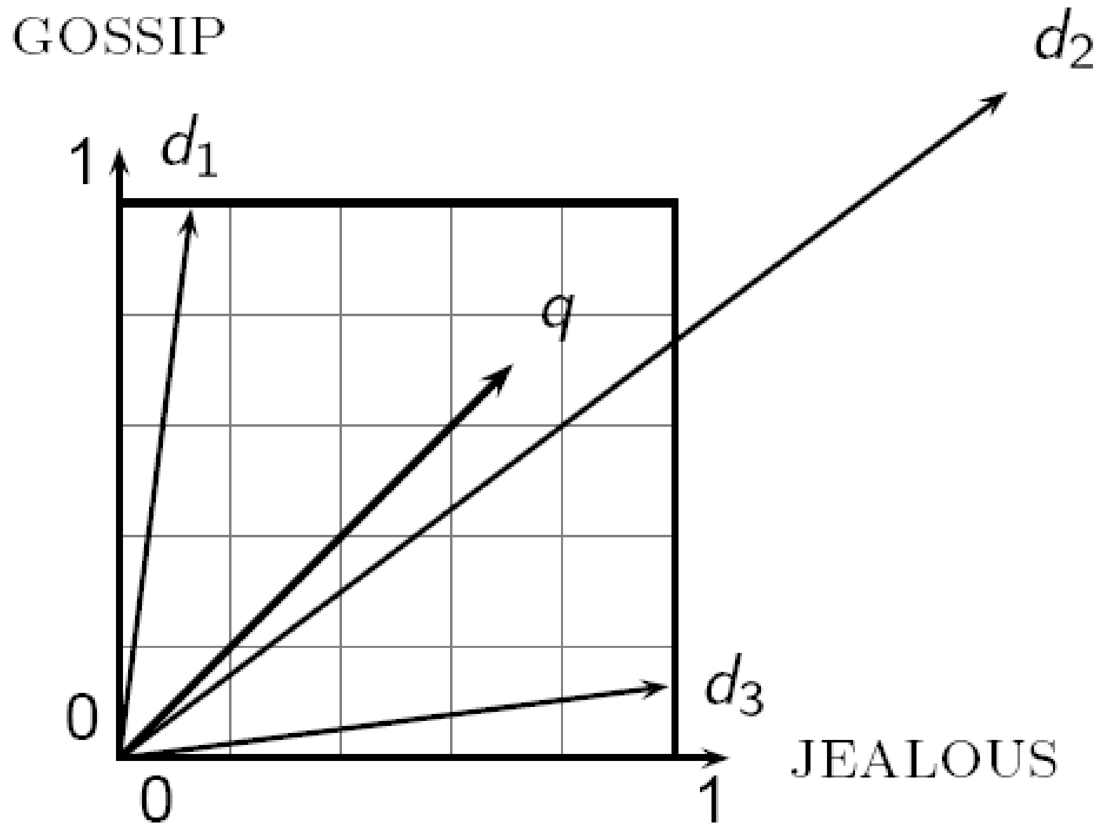
- Key idea 1: Do the same for queries: represent them as vectors in the space
- Key idea 2: Rank documents according to their proximity to the query in this space
- proximity = similarity of vectors
- proximity ≈ inverse of distance
- Recall: We do this because we want to get away from the you're-either-in-or-out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

Formalizing 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 $\vec{q}$ 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.

- Key idea: Rank documents according to angle with query.

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]

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

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 d … or, equivalently, the cosine of the angle between $\vec{q}$ and $\vec{d}$.

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)

- Assume that idf is 1. We only make use of term frequency.

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 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) ≈
$0.789 * 0.832 + 0.515 * 0.555 + 0.335 * 0.0 + 0.0 * 0.0$
≈ 0.94

cos(SaS,WH) ≈ 0.79

cos(PaP,WH) ≈ 0.69

General 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 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

Basic Algorithm

- Let *Length[N]* hold the lengths (normalization factors) for each of the *N* documents.
- Let *Scores[N]* hold the scores for each of the documents.
- Store ${}^{N}/_{\mathrm{df}_t}$ at the head of the posting for *t*.
- Store the term frequency $\mathrm{tf}_{t,d}$ for each posting entry.

Basic Algorithm

```
CosineScore(q)
float Scores[N] = 0
float Length[N]
for each query term t
do calculate w_{t,q} and fetch postings list for t
    for each pair(d, tf_{t,d}) in postings list
    do Scores[d] += w_{t,d} × w_{t,q}
Read the array Length
for each d
do Scores[d] = Scores[d]/Length[d]
return Top K components of Scores[]
```

tf-idf weighting has many variants

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

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
- To denote the combination in use, we use the notation qqq.ddd with the acronyms from the previous table
- Example: ltn.lnc means:
  - Query: logarithmic tf (l in leftmost column), idf (t in the second column), no normalization
  - Document logarithmic tf, no idf, and cosine normalization

Another Example

Document: *car insurance auto insurance*
Query: *best car insurance*

Weighting Scheme:
Query: ltn Document: lnc

Term	Query					Document			Product
	tf-raw	tf-wt	df	idf	wt	tf-raw	tf-wt	n'lized	
auto	0	0	5000	2.3	0	1	1	0.41	0
best	1	1	50000	1.3	1.3	0	0	0	0
car	1	1	10000	2.0	2.0	1	1	0.41	0.82
insurance	1	1	1000	3.0	3.0	2	2	0.82	2.46

The number of docs (N) is 1,000,000

Score = 0+0+0.41*2+0.82*3 = 3.28