Information Retrieval

Term Weighting and Vector Space

Boolean queries

- Not good for the majority of users
  - Incapable of writing Boolean queries (or not willing)
  - Don't want to wade through 1000s of results.
    - This is particularly true of web search.

- Often too many (1000s) or too few (=0) or results
  - Query 1: "standard user dlink 650" → 200,000 hits
  - Query 2: "standard user dlink 650 no card found" → 0 hits

- It takes a lot of skill to come up with a query that produces a manageable number of hits.
  - AND gives too few; OR gives too many

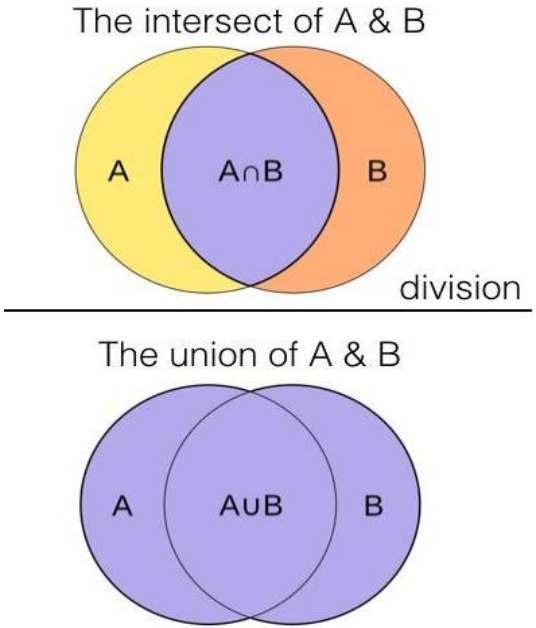
Solution: Ranked retrieval models

- Free text queries
  - Rather than a query language of operators and expressions, the user's query is just one or more words in a human language
  - No expertise needed
- Ranked retrieval model
  - the system returns an ordering over the (top) documents in the collection with respect to a query
  - Queries with large number of results are not a problem

- Two separate choices, but in practice, ranked retrieval models are normally been associated with free text queries and vice versa

Jaccard Coefficient – a naïve approach

- A commonly used measure of overlap of two sets
- Let A and B be two sets
  - $J(A,B) = |A \cap B| \ / \ |A \cup B|$
  - $J(A, A) = 1$
  - $J(A, B) = 0$ if $A \cap B = 0$
  - A and B don't have to be the same size
  - Always assigns a number between 0 and 1
- What's wrong with Jaccard?
  - It doesn't consider term frequency
    - how many occurrences a term has
  - Rare terms are more informative than frequent terms
    - Jaccard does not consider this information
    - We need to consider both



Term-document incidence matrix – binary

- Consider the **yes/no** occurrence of a term ***t*** in a document ***d*** ($tf_{t,d}$):
  - Each document is represented by a binary vector $\in \{0,1\}^{|V|}$

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	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

Term-document incidence matrix – counts

- Consider the **total** of occurrences of a term ***t*** in a document ***d*** ($tf_{t,d}$):
  - 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

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

- But, 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

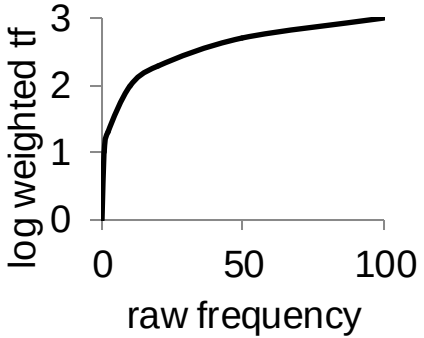
- Relevance does not increase proportionally with term frequency

Log-frequency weighting

- The log frequency weight of term t in d is

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

  - 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*:

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

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

Example – counts

	AaC	JC	TT	H	O	M
antony	157	73	0	0	0	0
brutus	4	157	0	1	0	0
caesar	232	127	0	2	1	1
calpurnia	0	10	0	0	0	0
cleopatra	57	0	0	0	0	0
mercy	2	0	0	3	0	0

Example – $w_{t,d}$ *(log frequency weight)*

	AaC	JC	TT	H	O	M
antony	3,20	2,86	0,00	0,00	0,00	0,00
brutus	1,60	3,20	0,00	1,00	0,00	0,00
caesar	3,37	3,10	0,00	1,30	1,00	1,00
calpurnia	0,00	2,00	0,00	0,00	0,00	0,00
cleopatra	2,76	0,00	0,00	0,00	0,00	0,00
mercy	1,30	0,00	0,00	1,48	0,00	0,00

Example – query

	AaC	JC	TT	H	O	M
antony	3,20	2,86	0,00	0,00	0,00	0,00
brutus	1,60	3,20	0,00	1,00	0,00	0,00
caesar	3,37	3,10	0,00	1,30	1,00	1,00
calpurnia	0,00	2,00	0,00	0,00	0,00	0,00
cleopatra	2,76	0,00	0,00	0,00	0,00	0,00
mercy	1,30	0,00	0,00	1,48	0,00	0,00

Query: Caesar mercy

caesar	3,37	3,10	0,00	1,30	1,00	1,00
mercy	1,30	0,00	0,00	1,48	0,00	0,00
score	**4,67**	**3,10**	**0,00**	**2,78**	**1,00**	**1,00**

Term Frequency issues

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

    → We want higher weights for rare terms
- **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

    → We want lower weights for frequent terms

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*
  - $df_t \leq N$

- We define the **idf_t** (inverse document frequency) of *t* by

$$\text{idf}_t = \log_{10}(N/\text{df}_t)$$

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

idf example, suppose N = 1 million

$$\text{idf}_t = \log_{10}(N/\text{df}_t)$$

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_t value for each term t in a collection.

Effect of idf on ranking

- Does idf have an effect on ranking for one-term queries?

tf-idf weighting

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

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

- Best known weighting scheme in information retrieval
- Increases with the number of occurrences within a document
- Increases with the rarity of the term in the collection

Example – $w_{t,d}$ (log frequency weight tf)

	AaC	JC	TT	H	O	M	idf
antony	3,20	2,86	0,00	0,00	0,00	0,00	
brutus	1,60	3,20	0,00	1,00	0,00	0,00	
caesar	3,37	3,10	0,00	1,30	1,00	1,00	
calpurnia	0,00	2,00	0,00	0,00	0,00	0,00	
cleopatra	2,76	0,00	0,00	0,00	0,00	0,00	
mercy	1,30	0,00	0,00	1,48	0,00	0,00	

Example – *idf*

	AaC	JC	TT	H	O	M	idf
antony	3,20	2,86	0,00	0,00	0,00	0,00	**0,48**
brutus	1,60	3,20	0,00	1,00	0,00	0,00	**0,30**
caesar	3,37	3,10	0,00	1,30	1,00	1,00	**0,08**
calpurnia	0,00	2,00	0,00	0,00	0,00	0,00	**0,78**
cleopatra	2,76	0,00	0,00	0,00	0,00	0,00	**0,78**
mercy	1,30	0,00	0,00	1,48	0,00	0,00	**0,48**

$$\text{idf}_t = \log_{10}(N/\text{df}_t)$$

Example – *tf-idf*

	AaC	JC	TT	H	O	M	idf
antony	3,20	2,86	0,00	0,00	0,00	0,00	**0,48**
brutus	1,60	3,20	0,00	1,00	0,00	0,00	**0,30**
caesar	3,37	3,10	0,00	1,30	1,00	1,00	**0,08**
calpurnia	0,00	2,00	0,00	0,00	0,00	0,00	**0,78**
cleopatra	2,76	0,00	0,00	0,00	0,00	0,00	**0,78**
mercy	1,30	0,00	0,00	1,48	0,00	0,00	**0,48**

Query: Caesar mercy

caesar	0,27	0,25	0,00	0,10	0,08	0,08
mercy	0,62	0,00	0,00	0,71	0,00	0,00
score	**0,89**	**0,25**	**0,00**	**0,81**	**0,08**	**0,08**
score	**4,67**	**3,10**	**0,00**	**2,78**	**1,00**	**1,00**

tf only

Binary → count → weight matrix

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

	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

- The ranking of each document for a query *q* can be now:

$$\text{Score}(q,d) = \sum_{t \in q \cap d} \text{tf.idf}_{t,d}$$

Documents as vectors

- So we have a multi-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.

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

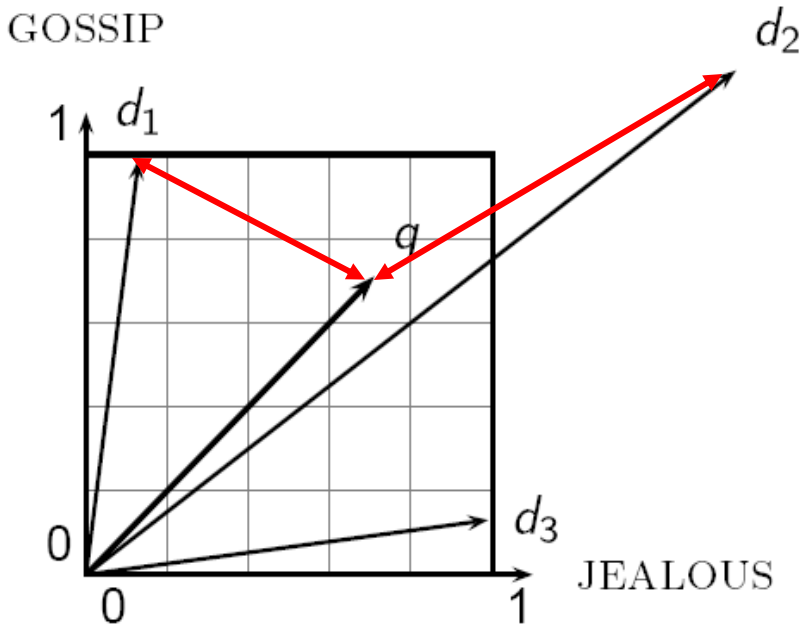
- proximity = similarity of vectors

- Rank more relevant documents higher than less relevant documents

Euclidean distance

- Consider query q and documents d1, d2 and d3.
  - Which is the closed document to the query q?

- The Euclidean distance between q and 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.



Euclidean distance

- Take a document d and append it to itself. Call this document d′.

Information retrieval (IR) is the activity of obtaining information system resources relevant to an information need from a collection of information resources.

Information retrieval (IR) is the activity of obtaining information system resources relevant to an information need from a collection of information resources.
Information retrieval (IR) is the activity of obtaining information system resources relevant to an information need from a collection of information resources.

Euclidean distance

- Take a document d and append it to itself. Call this document d′.

Information retrieval (IR) is the activity of obtaining information system resources relevant to an information need from a collection of information resources.

activity: 1
collection: 1
information: 4
need: 1
obtaining: 1
relevant: 1
resources: 2
retrieval: 1
system: 1

Information retrieval (IR) is the activity of obtaining information system resources relevant to an information need from a collection of information resources.
Information retrieval (IR) is the activity of obtaining information system resources relevant to an information need from a collection of information resources.

activity: 2
collection: 2
information: 8
need: 2
obtaining: 2
relevant: 2
resources: 4
retrieval: 2
system: 2

Use angle instead of distance

- Take a document d and append it to itself. Call this document d′.

Information retrieval (IR) is the activity of obtaining information system resources relevant to an information need from a collection of information resources.

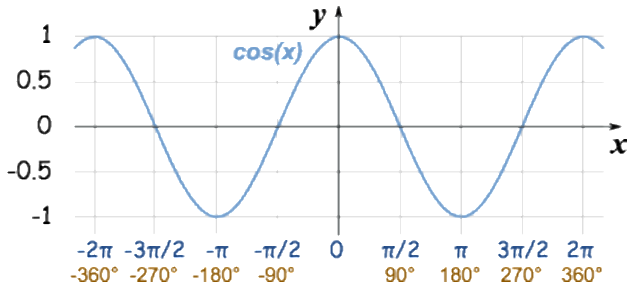
Information retrieval (IR) is the activity of obtaining information system resources relevant to an information need from a collection of information resources.
Information retrieval (IR) is the activity of obtaining information system resources relevant to an information need from a collection of information resources.

- "Semantically" d and d′ have the same content
  - But 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 increasing order of the angle between query and document
  - Rank documents in decreasing order of cosine(query,document)

- Cosine is a monotonically decreasing function for the interval [0°, 180°]



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(q, d) is the cosine similarity of q and d
  - or, equivalently, the cosine of the angle between q and d.

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:

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

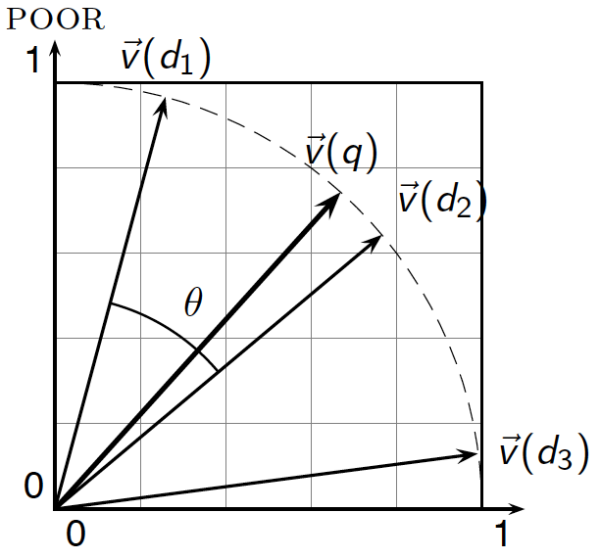
- Dividing a vector by its L_2 norm makes it a unit (length) vector (on surface of unit hypersphere)
- Effect on the two documents d and d′ (d appended to itself) from earlier slide: they have identical vectors after length-normalization.
  - Long and short documents now have comparable weights

Cosine for length-normalized vectors

- For length-normalized vectors, cosine similarity is simply the dot product (or scalar product):

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

  - for q, d length-normalized.



tf-idf weighting has many variants

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)	$1/CharLength^{\alpha}$, $\alpha < 1$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t \in d}(\mathrm{tf}_{t,d}))}$				

Columns headed ‘n’ are acronyms for weight schemes.

Why is the base of the log in idf immaterial?

Weighting may differ in queries vs documents

- Many search engines allow for different weightings for queries and for documents
- SMART Notation: denotes the combination in use in an engine, with the notation **ddd.qqq**, using the acronyms from the previous table
- A standard weighting scheme is: **lnc.ltc**
  - Document (lnc)
    - logarithmic tf (l as first character)
    - no idf
    - cosine normalization
  - Query (ltc):
    - logarithmic tf (l in leftmost column)
    - idf (t in second column)
    - no normalization …

why?

Computing cosine scores

```
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[]
```

tf-idf example – inverted index

- Document: *car insurance auto insurance*

Term			Document				Query			
	df	idf	tf-raw	tf-wt	wt	n'lize	tf-raw	tf-wt	wt	n'lize
auto	5000	2.3	1	1	1	0.52				
best	50000	1.3	0	0	0	0				
car	10000	2.0	1	1	1	0.52				
insurance	1000	3.0	2	1.3	1.3	0.68				

1+log10(tf-raw)

lnc -> no idf

$$wt_i \Big/ \sqrt{\sum wt_i^2}$$

- Exercise:
  - what is N, the number of docs?

tf-idf example – query

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

Term			Document				Query			
	df	idf	tf-raw	tf-wt	wt	n'lize	tf-raw	tf-wt	wt	n'lize
auto	5000	2.3	1	1	1	0.52	0	0	0	0
best	50000	1.3	0	0	0	0	1	1	1.3	0.34
car	10000	2.0	1	1	1	0.52	1	1	2.0	0.52
insurance	1000	3.0	2	1.3	1.3	0.68	1	1	3.0	0.78

tf-wt*idf

tf-idf example – SMART scores

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

Term			Document				Query			
	df	idf	tf-raw	tf-wt	wt	n'lize	tf-raw	tf-wt	wt	n'lize
auto	5000	2.3	1	1	1	0.52	0	0	0	0
best	50000	1.3	0	0	0	0	1	1	1.3	0.34
car	10000	2.0	1	1	1	0.52	1	1	2.0	0.52
insurance	1000	3.0	2	1.3	1.3	0.68	1	1	3.0	0.78

- *Score(lnc.ltc)* = 0.52*0 + 0*0.34 + 0.52*0.52 + 0.68*0.78 = 0.8
- ***Score(lnc.ltn)*** = 0.52*0 + 0*1.3 + 0.52*2 + 0.68*3 = 3.08

Summary – vector space ranking

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