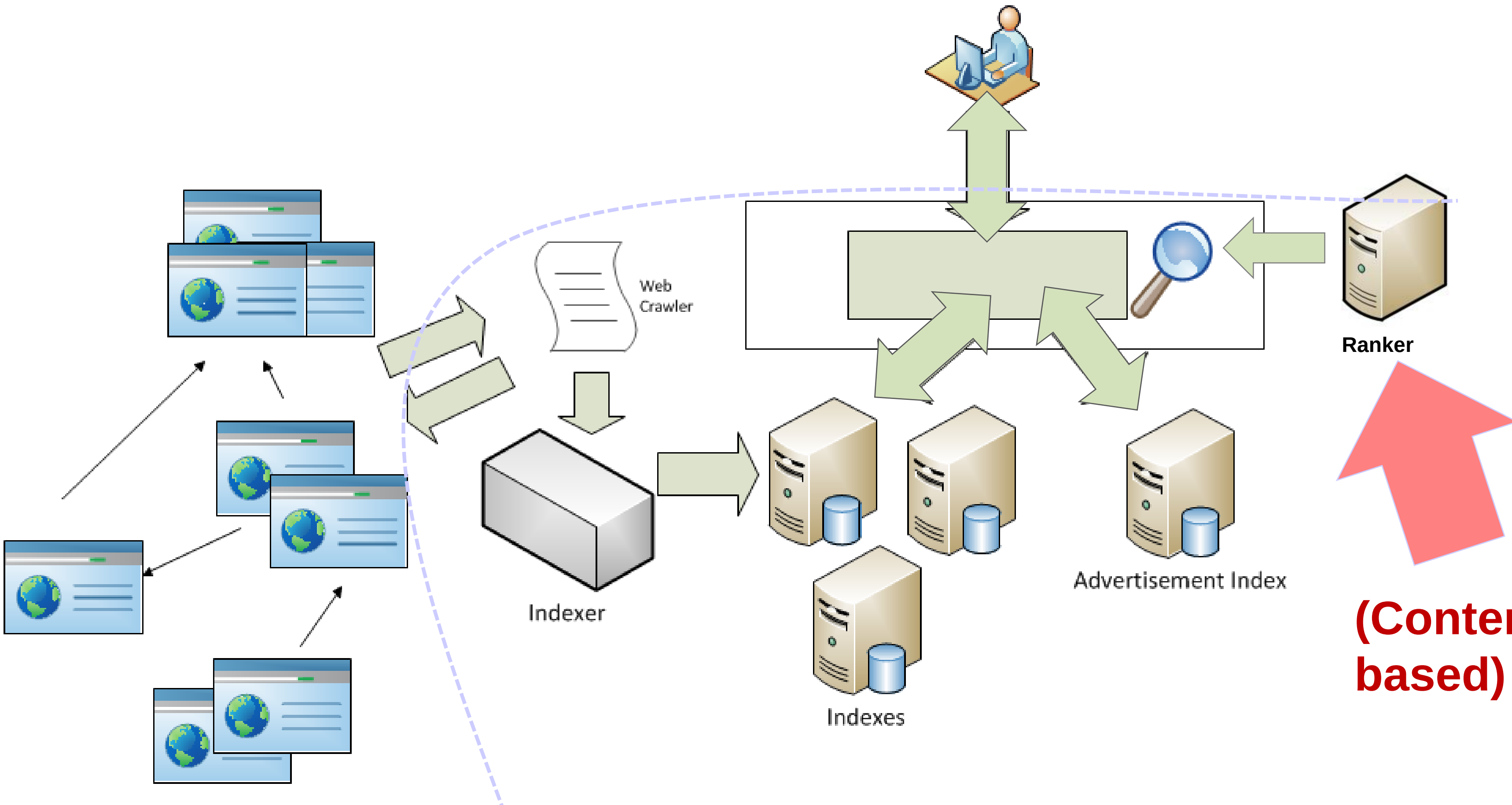
AALBORG UNIVERSITY

Ranking

Peter Dolog
dolog@cs.aau.dk
http://people.cs.aau.dk/~dolog

Based (heavily) on Standford slides by Pandu Nayak & Prabhakar Raghavan and on the 'Introduction to Information Retrieval' book (Chaps. 6 & 7) by Christopher Manning, Prabhakar Raghavan & Hinrich Schütze as well as Bo Thiesson.

Search Engine Architecture



(Content-based)

Outline

- Recap: Boolean Search
- Content-based ranking techniques
  - Variants of the vector space model
  - Efficient implementation
  - Approximate scores – variants
- (Zone indexes)

Term-Document Binary Matrix

Term	Term ID	Doc1	Doc2	Doc3
as	t1	1	1	0
activ	t2	0	1	0
play	t3	0	0	0
and	t4	1	1	1
area	t5	1	0	0
depart	t6	1	0	1
comput	t7	0	1	0

- **A 0/1 term vector for each document (*typically very large and sparse*)**
- **A 0/1 incidence vector for each term (*typically very large and sparse*)**

The Boolean Query Model: Example

Simple model based on **set theory** and **Boolean algebra**

Queries specified as Boolean expressions

To answer query "$as \wedge and \wedge \neg comput$" perform bitwise AND on incidence vectors

d1	d2	d3
1	1	0
1	1	1
1	0	1
1	0	0

Term	Term ID	Doc1	Doc2	Doc3
as	t1	1	1	0
activ	t2	0	1	0
play	t3	0	0	0
and	t4	1	1	1
area	t5	1	0	0
depart	t6	1	0	1
comput	t7	0	1	0

Boolean Retrieval: Drawbacks

- Documents either match a query or not!
- Often difficult to define a query that will match a reasonable number of documents
  - Either too few (=0) or too many (1000s)
  - AND restricts, OR expands!
- Not good for the majority of users.
  - Most users incapable of writing Boolean queries (or too much work)
  - Most users don't want to wade through 1000s of results.
- Can be good for few expert users with precise understanding of their needs and the collection.
  - Also good for applications: Applications can easily consume 1000s of results.

Ranking

Goal: order the answers to a query in decreasing order of value

- Just show top k (≈10) results (at a time)
- User is not overwhelmed

Some Ranking Criteria

- Content-based techniques (vector space model or probabilistic model) – query-dependent
- Ad-hoc factors (anti-porn heuristics, location on page, publication/location data, length …) – mostly query-independent
- Human annotations
- Structure-based techniques (next lecture)
  - PageRank – query-independent
  - HITS – query-dependent

Ranking criteria defines a **ranking score** that measures how well a document and query "match".

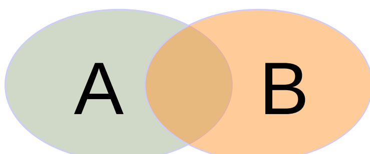
- E.g., ranking score $\in [0,1]$

Content-based ranking techniques

Jaccard Similarity (from first lecture)

Similarity measure between documents A and B

$$Jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|} \quad \left(\frac{\text{Overlap}}{\text{Union}}\right)$$



	A	B	A∩B	A∪B
t1	1	1	1	1
t2	1	0	0	1
t3	0	0	0	0
t4	0	0	0	0
t5	1	1	1	1
t6	0	1	0	1

$$\boldsymbol{Jaccard} = 2\ /\ 4$$

Can we do better?

- **What if a term appears multiple times?**
- **Are all terms equally important?**

What if a term appears multiple times in a document?

Should that affect the terms importance?

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*.
- Frequent terms in a document are (often) more important than infrequent terms
  - E.g., if a document frequently mentions the term *iPhone*, it is likely about the iPhone
- However, relevance does not increase proportionally with term frequency
  - 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.

Log-frequency weighting

- The log frequency weight of term t in d is

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

- $0 \to 0, \quad 1 \to 1, \quad 2 \to 1.3, \quad 10 \to 2, \quad 1000 \to 4$, etc.
- Score for a document-query pair - Take 1:

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

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

Term Frequency (TF) Vector

- Vector representation doesn't consider the ordering of words in a document – aka. bag of words
  - software engineers are smarter than computer scientists and computer scientists are smarter than software engineers have the same vectors

Doc

the research in the department has computers, programming, as well as software and computer systems as its field.

→

TF Vector

Term	Freq.
⋮	0
and	1
⋮	0
as	3
⋮	0
computer(s)	2
⋮	0
department	1
⋮	0
field	1
⋮	

Term-Document Frequency Matrix

Each cell represents a **term-frequency**

$tf_{t,d}$

t=term, d=document

$tf_{7,2}$

Term	Term ID	Doc1	Doc2	Doc3
as	t1	5	2	0
activ	t2	0	3	0
play	t3	0	0	0
and	t4	6	11	9
area	t5	10	0	0
depart	t6	16	0	1
comput	t7	0	3	0

Are all terms equally important?

Should that affect the terms importance?

Document Frequency -- *df*

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

Ultracrepidarian: noting or pertaining to a person who criticizes, judges, or gives advice outside the area of his or her expertise

Inverse Document Frequency -- *idf*

- 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
  - $df_t \leq N$
- The idf (inverse document frequency) of t is defined 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

N = number of documents in corpus
Suppose *N* = 1 million

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

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

only
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 ultracrepidarian person, idf weighting makes occurrences of ultracrepidarian count for much more in the final document ranking than occurrences of person.

Collection vs. Document frequency

- The document frequency (df) of t is the number of documents that contain t
- The collection frequency (cf) of t is the number of occurrences of t in the collection, counting multiple occurrences.
- Example:

Word	Collection frequency	Document frequency
insurance	10440	3997
try	10422	8760

Q: Which word is a better search term (and should get a higher weight)?

A: Word with *low document frequency*

- We are trying to discriminate between documents!
- df is document-level statistic; cf is collection (corpus) wide statistic

tf-idf weighting

- The tf-idf weight of a term is the product of its tf weight and its idf weight. Examples:
  - $tf\text{-}idf_{t,d} = tf_{t,d} \times \log_{10}(N/df_t)$
  - $tf^{*}\text{-}idf_{t,d} = \left(1 + \log_{10} tf_{t,d}\right) \times \log_{10}(N/df_t)$
- Best known weighting schemes 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 or more precisely the rarity of the

Example: tf weighting

$$tf^*_{t,d} = 1 + log_{10}(tf_{t,d})$$

doc1
To do is to be.
To be is to do.

doc2
To be or not to be.
I am what I am.

doc3
I think therefore I am.
Do be do be do.

doc4
Do do do, da da da.
Let it be, let it be.

	terms	$tf_{t,1}$	$tf_{t,2}$	$tf_{t,3}$	$tf_{t,4}$	$tf^*_{t,1}$	$tf^*_{t,2}$	$tf^*_{t,3}$	$tf^*_{t,4}$
1	to	4	2	-	-	1.60	1.30	-	-
2	do	2	-	3	3	1.30	-	1.48	1.48
3	is	2	-	-	-	1.30	-	-	-
4	be	2	2	2	2	1.30	1.30	1.30	1.30
5	or	-	1	-	-	-	1	-	-
6	not	-	1	-	-	-	1	-	-
7	i	-	2	2	-	-	1.30	1.30	-
8	am	-	2	1	-	-	1.30	1	-
9	what	-	1	-	-	-	1	-	-
10	think	-	-	1	-	-	-	1	-
11	therefore	-	-	1	-	-	-	1	-
12	da	-	-	-	3	-	-	-	1.48
13	let	-	-	-	1	-	-	-	1
14	it	-	-	-	2	-	-	-	1.30

Example: idf weighting

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

doc1
To do is to be.
To be is to do.

doc2
To be or not to be.
I am what I am.

doc3
I think therefore I am.
Do be do be do.

doc4
Do do do, da da da.
Let it be, let it be.

	terms	df_t	idf_t
1	to	2	0.3
2	do	3	0.12
3	is	1	0.6
4	be	4	0
5	or	1	0.6
6	not	1	0.6
7	i	2	0.3
8	am	2	0.3
9	what	1	0.6
10	think	1	0.6
11	therefore	1	0.6
12	da	1	0.6
13	let	1	0.6
14	it	1	0.6

Example: tf-idf weighting

$$tf\text{−}idf_{t,d} = tf^{*}_{t,d} \times idf_t$$

doc1
To do is to be.
To be is to do.

doc2
To be or not to be.
I am what I am.

doc3
I think therefore I am.
Do be do be do.

doc4
Do do do, da da da.
Let it be, let it be.

	terms	idf	tf*t,1	tf*t,2	tf*t,3	tf*t,3	tfidft,1	tfidft,2	tfidft,3	tfidft,4
1	To	0,3	1,6	1,3	-	-	0,48	0,39	-	-
2	Do	0,12	1,3	-	1,48	1,48	0,156	-	0,178	0,178
3	Is	0,6	1,3	-	-	-	0,78	-	-	-
4	be	0	1,3	1,3	1,3	1,3	0	0	0	0
5	or	0,6	-	1	-	-	-	0,6	-	-
6	not	0,6	-	1	-	-	-	0,6	-	-
7	i	0,3	-	1,3	1,3	-	-	0,39	0,39	-
8	am	0,3	-	1,3	1	-	-	0,39	0,3	-
9	what	0,6	-	1	-	-	-	0,6	-	-
10	think	0,6	-	-	1	-	-	-	0,6	-
11	therefore	0,6	-	-	1	-	-	-	0,6	-
12	da	0,6	-	-	-	1,48	-	-	-	0,888
13	let	0,6	-	-	-	1	-	-	-	0,6
14	it	0,6	-	-	-	1,3	-	-	-	0,78

tf-idf weighting (cont.)

Score for a document-query pair - Take 2:

$$score(q,d) = \sum_{t \in q \cap d} tf\text{-}idf_{t,d}$$

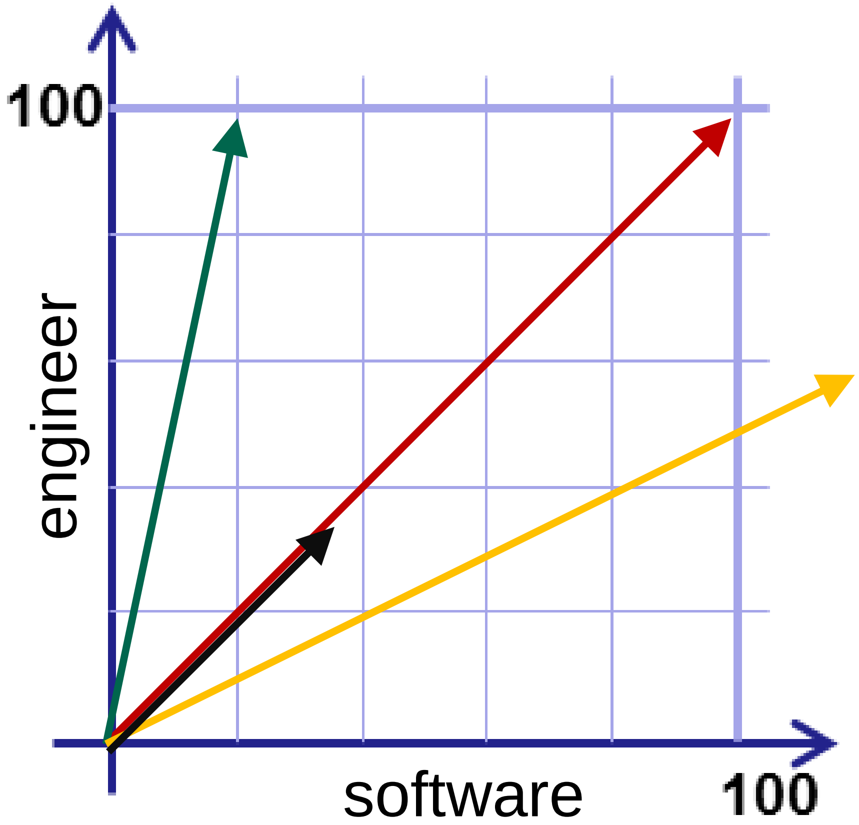
…but, still not good enough!

Example: Two terms (software, engineer)

Number of appearances of the two terms in documents are represented by vector end-points

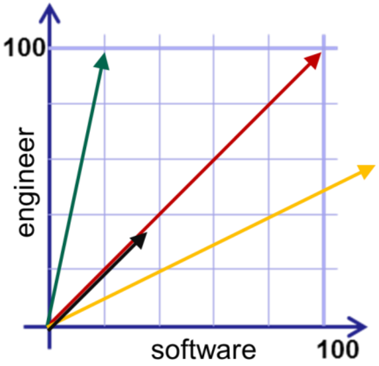
Q: Which document is the most similar to the red document?

A: Black, the *distribution* of terms in the two documents is the most similar



Documents as vectors

- Terms are the axes in a N dimensional vector space
- Documents are represented as points (or vectors) in this vector space
- Very **high-dimensional** vectors; dimensionality = N = number of *different* terms in corpus
- Very **sparse** vectors



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

- … but, what proximity/distance measure should we use?
We saw before that Euclidian distance wasn't such a good idea!

Math…

Has some resemblance to the "take 2" score

$$score(q,d) = \sum_{t \in q \cap d} tf\text{-}idf_{t,d}$$

Inner (dot) product

$$A \cdot B = \sum_{i=1}^{N} a_i b_i$$

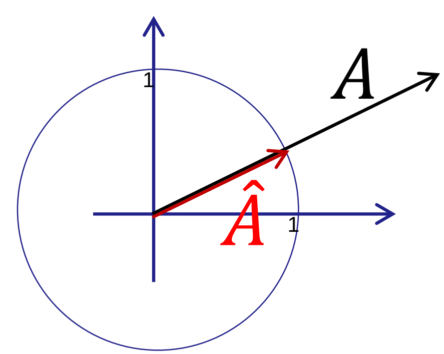
	A		B		A · B
t1	a1		b1		a1b1
t2	a2		b2		+ a2b2
t3	a3	·	b3	=	+ a3b3
t4	a4		b4		+ a4b4
t5	a5		b5		+ a5b5

Vector length (L_2-norm)

$$|A| = \sqrt{\sum_{i=1}^{N} a_i^2}$$

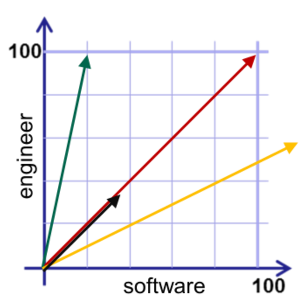
Normalized vector (unit vector)

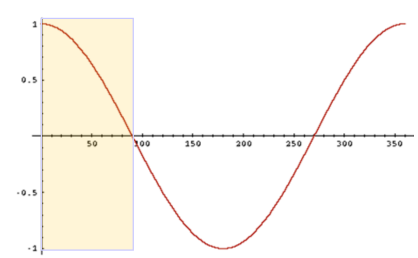
$$\hat{A} = \frac{A}{|A|}$$



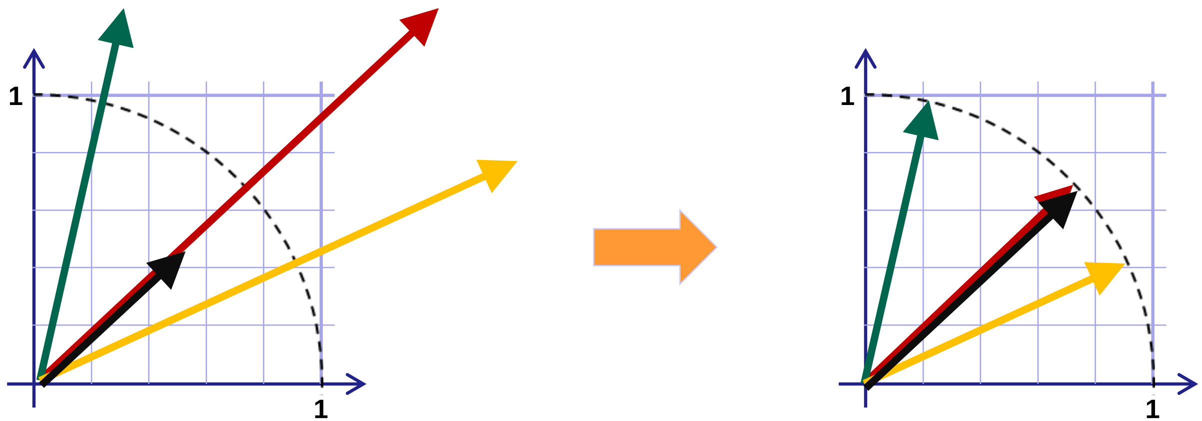
Cosine similarity score – ”the final fix”

- **Key idea**: Use angle between *tf-idf* vectors to measure the similarity between (query,document)
- Notice
  - All term counts are positive ⇒ vectors in first quadrant
  - Cosine (of angle) is a monotonically decreasing function in first quadrant (angles from 0 to 90 dregrees)
- The following two notions are equivalent.
  - Rank documents in decreasing order of the angle between query and document
  - Rank documents in increasing order cosine(query,document)





Cosine similarity
– as normalized vector distance



$$\cos(q,d) = \frac{q \cdot d}{|q||d|} = \frac{q}{|q|} \cdot \frac{d}{|d|} = \hat{q} \cdot \hat{d}$$

$$= \frac{\sum_{i=1}^{N} q_i\, d_i}{\sqrt{\sum_{i=1}^{N} q_i^2}\,\sqrt{\sum_{i=1}^{N} d_i^2}} = \sum_{i=1}^{N} \hat{q}_i \hat{d}_i = \cos(\hat{q}, \hat{d})$$

$\hat{q}$		$\hat{d}$
-		-
-		$\hat{d}_2$
-		-
-		-
$\hat{q}_5$		-
-		-
-		$\hat{d}_7$
-		-
-	$\cdot$	-
-		-
-		$\hat{d}_{11}$
-		$\hat{d}_{12}$
$\hat{q}_{13}$		$\hat{d}_{13}$
-		-
		-
$\hat{q}_{16}$		$\hat{d}_{16}$
-		$\hat{d}_{17}$
-		-

Remember the **Boolean** index!

Term	DocID
a	1
ambition	1
and	2
as	1
as	2
as	2
as	2
be	1
computer	1
computer	2
computer	2
department	1
department	2
environment	1
field	2
…	…

Term	Doc. Freq.	Posting lists
a	1	→ 1
ambition	1	→ 1
and	1	→ 2
as	2	→ 1 → 2
be	1	→ 1
computer	2	→ 1 → 2
department	2	→ 1 → 2
environment	1	→ 1
field	1	→ 2

…

The "tf-idf index" for cosine ranking

Term	DocID
a	1
ambition	1
and	2
as	1
as	2
as	2
as	2
be	1
computer	1
computer	2
computer	2
department	1
department	2
environment	1
field	2
...	...

docID:term-weight

Term	Doc. Freq.	Posting lists	
a	1	→ 1:1	
ambition	1	→ 1:1	
and	1	→ 2:1	
as	2	→ 1:1	→ 2:3
be	1	→ 1:1	
computer	2	→ 1:1	→ 2:2
department	2	→ 1:1	→ 2:1
environment	1	→ 1:1	
field	1	→ 2:1	

...

Cosine score -- Implementation

$\textsc{CosineScore}(q)$

$\mathit{float}\ \mathit{Scores}[N] = 0$
$\mathit{float}\ \mathit{Length}[N]$
for each query term t
do calculate $w_{t,q}$ and fetch postings list for t
    for each pair$(d, \mathrm{tf}_{t,d})$ in postings list
    do $\mathit{Scores}[d] += w_{t,d} \times \cancel{w_{t,q}}$
Read the array Length
for each d
do $\mathit{Scores}[d] = \mathit{Scores}[d] / \mathit{Length}[d]$
return Top K components of $\mathit{Scores}[]$

The vector-length normalization for each document

For uniformly weighted terms in query

Q: Why do we not normalize wrt the vector-length of the query?
A: Constant, so doesn't affect the ranking!

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

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 the acronyms from the previous table

A very standard weighting scheme is: lnc.ltc

Document: logarithmic tf (l as first character), no idf and cosine normalization

Query: logarithmic tf (l in leftmost column), idf (t in second column), cosine normalization …

tf-idf example: lnc.ltc

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

Term	Query						Document				Prod
	tf-raw	tf*-wt	df	idf	wt	norm wt	tf-raw	tf*-wt	wt	norm wt	
auto	0	0	5000	2.3	0	0	1	1	1	0.52	0
best	1	1	50000	1.3	1.3	0.34	0	0	0	0	0
car	1	1	10000	2.0	2.0	0.52	1	1	1	0.52	0.27
insurance	1	1	1000	3.0	3.0	0.78	2	1.3	1.3	0.68	0.53

Doc length = $\sqrt{1^2 + 0^2 + 1^2 + 1.3^2} \approx 1.92$

Score = 0+0+0.27+0.53 = 0.8

Summary – vector space ranking

- Represent the query as a tf-idf vector
- Represent each document as a 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

The cosine score computation can be a bottleneck

Can we alleviate this computational burden?

Yes, but may sometimes get it wrong. A doc *not* in the top *K* may creep into the list of *K* output docs

Q: Is this such a bad thing?

A: No, cosine similarity is just proxy for user happiness anyway!

Generic approach

Find a set A of contenders, with $K < |A| << N$

- A does not necessarily contain the top K, but has many docs from among the top K
- Return the top K docs in A

Think of A as pruning non-contenders

Several schemes following this approach…

Index elimination

Basic cosine computation algorithm only considers docs containing at least one query term

- Otherwise, score is 0

Take this further:

- Only consider high-idf query terms
- Only consider docs containing many query terms

High-idf query terms only

For a query such as

the biggest city in Denmark

Only accumulate scores from

(biggest, city, *Denmark)*

Intuition:

- *the* and *in* contribute little to the scores and therefore don't alter rank-ordering much
- Similar in principle to stop-words

Benefit:

- Postings of low-idf terms have many docs $\rightarrow$ these (many) docs get eliminated from set *A* of contenders

Docs containing many query terms

Any doc with at least one query term is a candidate for the top *K* output list

For multi-term queries, only compute scores for docs containing several of the query terms

- Say, at least 3 out of 4
- Imposes a “soft conjunction” on queries seen on web search engines (early Google)

Easy to implement in postings traversal

Champion lists

Precompute for each dictionary term *t*, the *r* docs of highest weight in *t*'s postings

- Call this the <u>champion list</u> for *t* (aka. <u>fancy list</u> or <u>top docs</u> for *t*)

Note that *r* has to be chosen at index build time

- Thus, it's possible that $r < K$

At query time, only compute scores for docs in the champion list of some query term

- Pick the *K* top-scoring docs from amongst these

Cluster pruning: preprocessing

Pick √N *docs* at random: call these *leaders*

For every other doc, pre-compute nearest leader

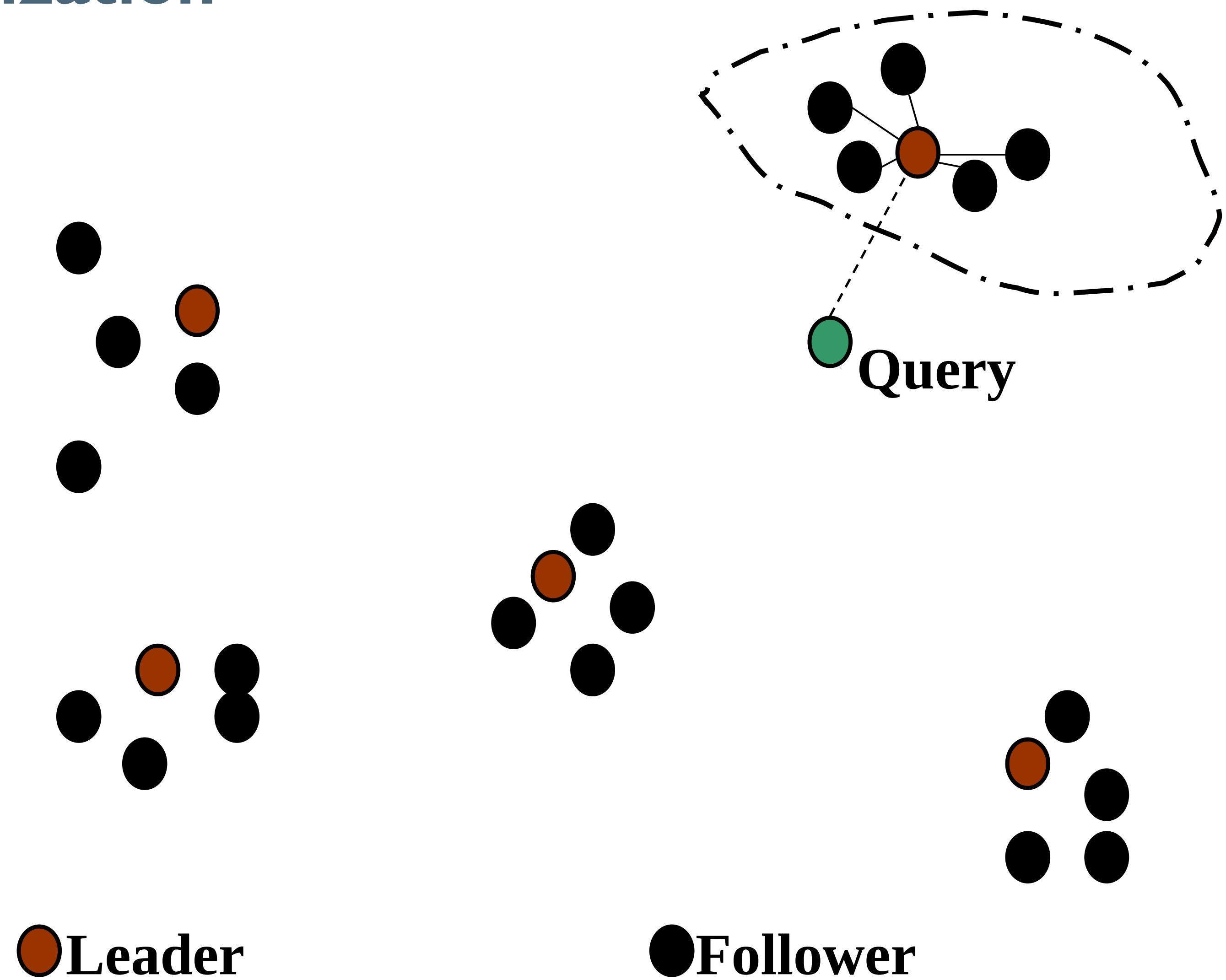
- Docs attached to a leader: its *followers;*
- Likely: each leader has ~ √*N* followers.

Cluster pruning: query processing

Given query *Q*, find its nearest *leader L.*

Seek *K* nearest docs from among *L*'s followers.

Visualization



General variants

Have each follower attached to $b1$=3 (say) nearest leaders.

From query, find $b2$=4 (say) nearest leaders and their followers.

Can recurse on leader/follower construction.

More examples of contender-pruning methods in the IIR-book, Chap.7

Zone indexes

- Web documents are structured mostly in HTML
- Different tags can inform about an importance of the content they embed
- For example title can be more important than lists
- Example from IRIS system (http://trec.nist.gov/pubs/trec8/papers/unc_tr8final.pdf):
  - Rank: title, header1, header 2, header 3, link, emphasized, lists, emphasized 2, header 4, header 5, header 6, delimiters (P, DT,…)
- Example from Webor system: (http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=809831)
  - Rank: Anchor, H1-H2, H3-H6, Strong, Title, Plain Text

Implementation:

- Index for each zone/importance
- Linear combination of scores

Outline

- Recap: Boolean Search
- Content-based ranking techniques
  - Variants of the vector space model
  - Efficient implementation
  - Approximate scores – variants
- (Zone indexes)