Research Group
Databases and Information Systems

Ranked Retrieval and Retrieval Models

Günther Specht
Eva Zangerle

Summer Term 2019

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.

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 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
  - Even with tolerant retrieval not feasible

The Boolean Model

$d_1 = [1, \quad 1,1]^T$
$d_2 = [1, \quad 0,0]^T$
$d_3 = [0, \quad 1,0]^T$

$R_{t1} = \{d_1, d_2\} \quad R_{t2} = \{d_1, d_3\} \quad R_{t3} = \{d_1\}$

$q = t_1$
$q = t_1 \text{ AND } t_2$
$q = t_1 \text{ OR } t_2$
$q = \text{NOT } t_1$

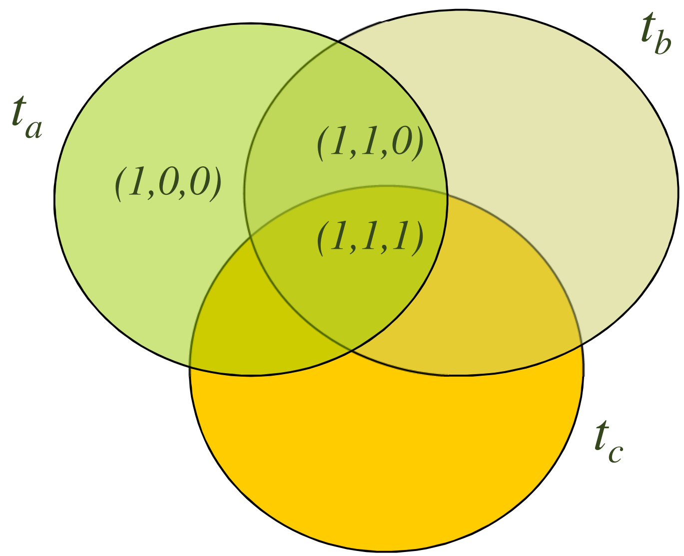
$R_{t1} = \{d_1, d_2\}$
$R_{t1} \cap R_{t2} = d_1$
$R_{t1} \cup R_{t2} = \{d_1, d_2, d_3\}$
$\neg R_{t1} = d_3$

The Boolean Model

- Each Boolean query can be rewritten in a Disjunctive Normal Form
  - $q = t_a \wedge (t_b \vee \neg t_c)$

    $q_{dnf} = (t_a \wedge t_b \wedge t_c) \vee (t_a \wedge t_b \wedge \neg t_c) \vee (t_a \wedge \neg t_b \wedge \neg t_c)$

    $q_{dnf} = \quad (1,1,1) \quad \vee \quad (1,1,0) \quad \vee \quad (1,0,0)$



- Each disjunction represents an ideal set of documents
- The query is satisfied by a document if such document is contained in a disjunction term

Considerations on the Boolean Model

- Strategy is based on a **binary decision criterion** (i.e., a document is predicted to be either relevant or non-relevant) without any notion of a grading scale
  - The Boolean model is in reality much more a data (instead of information) retrieval model

- Pros:
  - Boolean expressions have **precise** semantics
  - **Structured** queries
  - For expert users, **intuitivity**
  - Simple and neat formalism → great attention in past years and was adopted by many of the early commercial bibliographic systems

- Cons:
  - Frequently hard to translate an information need into a Boolean expression.
  - Most users find it difficult and awkward to express their query requests in terms of Boolean expressions.
  - **No ranking**

Recall: A Formal Characterization

- An IR model *IRM* can be defined as:

$$IRM =< D, Q, F, R(q_i, d_j) >$$

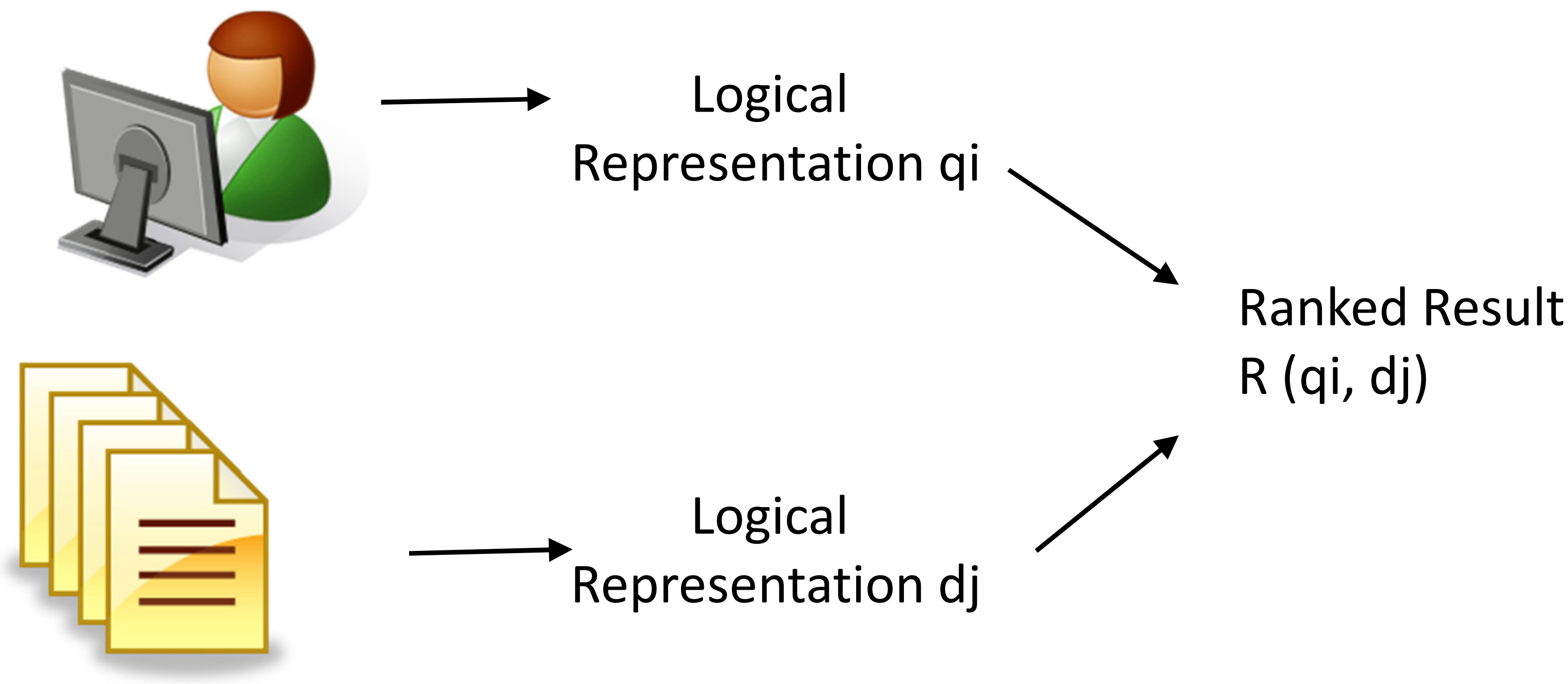
where

  - ***D*** – set of **logical views** (or representations) for the documents in the collection
  - ***Q*** – set of logical views (or representations) for the user's needs. Such representations are called **queries**
  - ***F*** – **framework** (or strategy) for modeling the document and query representation, and their relationship
  - ***R(qi,dj)*** – **ranking function**, associates a real number to a document representation d_j with a query q_i. Such ranking defines an ordering among the documents with regard to the query q_i

Typical tasks covered in IR

- **Search** (‘ad hoc’ retrieval)
  - Static document collection
  - Dynamic queries
  - Changed dramatically with the rise of the web

Ad-Hoc query



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 a human language
- In principle, there are two separate choices here, but in practice, ranked retrieval has normally been associated with free text queries and vice versa

Feast or Famine: Ranked Retrieval?

- When a system produces a ranked result set, large result sets are not an issue
  - Indeed, the size of the result set is not an issue
  - We just show the top *k* (≈ 10) results
  - We don’t overwhelm the user

  - Premise: the ranking algorithm works

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

- 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 $A \cap B = 0$
- A and B don’t have to be the same size.
- 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)
- Rare terms in a collection are more informative than frequent terms.
- Jaccard doesn't consider this information.
- We need a more sophisticated way of normalizing for length
- Longer documents are more likely to contain desired information.

- Later in this lecture, we'll use

$$|A \cap B| / \sqrt{|A \cup B|}$$

- . . . instead of $|A \cap B|/|A \cup B|$ (Jaccard) for length normalization.

Binary Term-Document Incidence Matrix

	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

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

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.
- In a sense, this is a step back: The positional index was able to distinguish these two documents.
- For now: bag of words model

Term Frequency tf

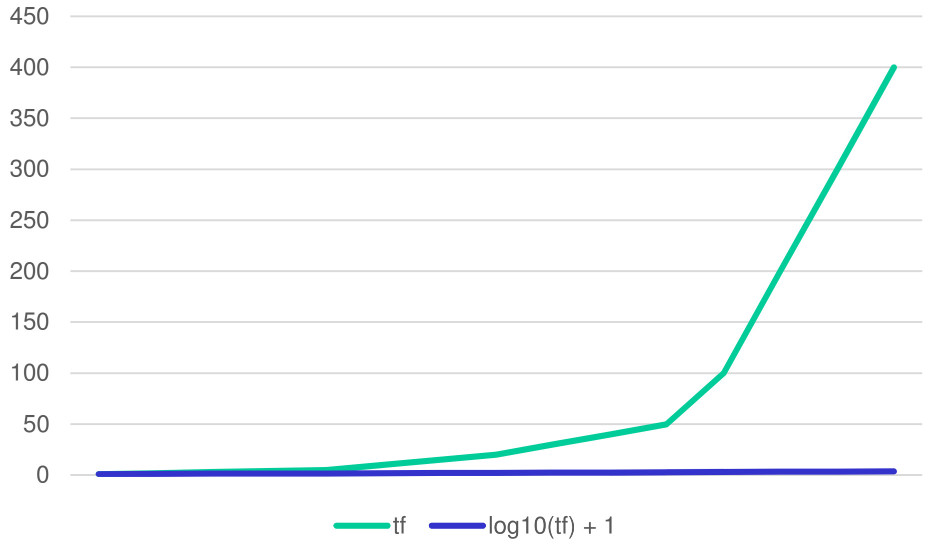
- Def.: The term frequency $\mathrm{tf}_{t,d}$ of term *t* in document *d* is defined as the number of times that *t* occurs in *d*.

- 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

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



tf	log10(tf) + 1
1	1,0000
2	1,3010
3	1,4771
4	1,6021
5	1,6990
10	2,0000
15	2,1761
20	2,3010
30	2,4771
40	2,6021
50	2,6990
100	3,0000
200	3,3010
300	3,4771
400	3,6021

Log-frequency Weighting

- Score for a document-query pair: sum over terms *t* in both *q* and *d*:

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

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

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., *anguilliform ('resembling an eel', taken from http://www.oxforddictionaries.com/words/weird-and-wonderful-words*)
- A document containing this term is very likely to be relevant to the query *anguilliform*
- → We want a high weight for rare terms like *anguilliform*.

Document Frequency

- Frequent terms are less informative than rare terms
- Consider a query term that is frequent in the collection
- 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
- But lower weights than for rare terms.
- We will use document frequency (df) to capture this.

idf Weight

- Def.: The document frequency df_t of t is
  the number of documents that contain t

  - df_t is an inverse measure of the informativeness of t
  - $\mathrm{df}_t \leq N$

- Def.: 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.

idf Example

Suppose N = 1,000,000

term	df	N/df	log10(N/df)
calpurnia	1	1,000,000	6
animal	100	10,000	4
sunday	1,000	1,000	3
fly	10,000	100	2
under	100,000	10	1
the	1,000,000	1	0

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

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.

  - (capricious = often changing suddenly in mood or behavior
    dt.: kapriziös, aber auch: launenhaft)

Collection vs. Document Frequency

- Def.: The collection frequency of *t* is
  the number of occurrences of *t* in the collection, counting multiple occurrences.

- Example from Reuters Collection:

Word	Collection frequency	Document frequency
insurance	10440	3997
try	10422	8760

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

tf-idf Weighting

- Def.: 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

Score for a Document given a Query

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

- There are many variants
  - How “tf” is computed (with/without logs)
  - Whether the terms in the query are also weighted
  - …

Tf-Idf Variants

Table 5 Evaluated variants to calculate the term frequency $r_{d,t}$

Abbr.	Description	Formulation
TF_A	Formulation used for binary match SB = b	$r_{d,t} = \begin{cases} 1 & \text{if } t \in T_d \\ 0 & \text{otherwise} \end{cases}$
TF_B	Standard formulation SB = t	$r_{d,t} = f_{d,t}$
TF_C	Logarithmic formulation	$r_{d,t} = 1 + \log_e f_{d,t}$
TF_C2	Alternative logarithmic formulation suited for $f_{d,t} < 1$	$r_{d,t} = \log_e (1 + f_{d,t})$
TF_C3	Alternative logarithmic formulation as used in *ltc* variant	$r_{d,t} = 1 + \log_2 f_{d,t}$
TF_D	Normalized formulation	$r_{d,t} = \frac{f_{d,t}}{f_d^m}$
TF_E	Alternative normalized formulation. Similar to Zobel and Moffat (1998) we use $K = 0.5$. SB = n	$r_{d,t} = K + (1 - K) \cdot \frac{f_{d,t}}{f_d^m}$
TF_F	Okapi formulation, according to Robertson et al. (1995), Zobel and Moffat (1998). For W we use the vector space formulation, i.e., the Euclidean length	$r_{d,t} = \frac{f_{d,t}}{f_{d,t} + W_d / av_{d \in D}(W_d)}$
TF_G	Okapi BM25 formulation, according to Robertson et al. (1999)	$r_{d,t} = \frac{(k_1 + 1) \cdot f_{d,t}}{f_{d,t} + k_1 \cdot \left[(1 - b) + b \frac{W_d}{av_{d \in D}(W_d)}\right]}$ $k_1 = 1.2,\ b = 0.75$

Taken from Schedl, Markus. "# nowplaying Madonna: a large-scale evaluation on estimating similarities between music artists and between movies from microblogs." *Information retrieval* 15.3-4 (2012): 183-217.

Tf-Idf Variants

Table 6 Evaluated variants to calculate the inverse document frequency w_t

Abbr.	Description	Formulation
IDF_A	Formulation used for binary match SB $= x$	$w_t = 1$
IDF_B	Logarithmic formulation SB $= f$	$w_t = \log_e\left(1 + \frac{N}{f_t}\right)$
IDF_B2	Logarithmic formulation used in *ltc* variant	$w_t = \log_e\left(\frac{N}{f_t}\right)$
IDF_C	Hyperbolic formulation	$w_t = \frac{1}{f_t}$
IDF_D	Normalized formulation	$w_t = \log_e\left(1 + \frac{f_m}{f_t}\right)$
IDF_E	Another normalized formulation SB $= p$	$w_t = \log_e \frac{N - f_t}{f_t}$
	The following definitions are based on the term's noise n_t and signal s_t.	$n_t = \sum_{d \in \mathcal{D}_t} \left(-\frac{f_{d,t}}{F_t} \log_2 \frac{f_{d,t}}{F_t}\right)$
		$s_t = \log_2 (F_t - n_t)$
IDF_F	Signal	$w_t = s_t$
IDF_G	Signal-to-noise ratio	$w_t = \frac{s_t}{n_t}$
IDF_H		$w_t = \left(\max_{t' \in \mathcal{T}} n_{t'}\right) - n_t$
IDF_I	Entropy measure	$w_t = 1 - \frac{n_t}{\log_2 N}$
IDF_J	Okapi BM25 IDF formulation, according to Pérez-Iglesias et al. (2009), Robertson et al. (1999)	$w_t = \log \frac{N - f_t + 0.5}{f_t + 0.5}$

Taken from Schedl, Markus. "# nowplaying Madonna: a large-scale evaluation on estimating similarities between music artists and between movies from microblogs." *Information retrieval* 15.3-4 (2012): 183-217.

Binary → 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 $\in \mathbb{R}^{|V|}$

Documents as Vectors

- Now 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.

Queries as Vectors

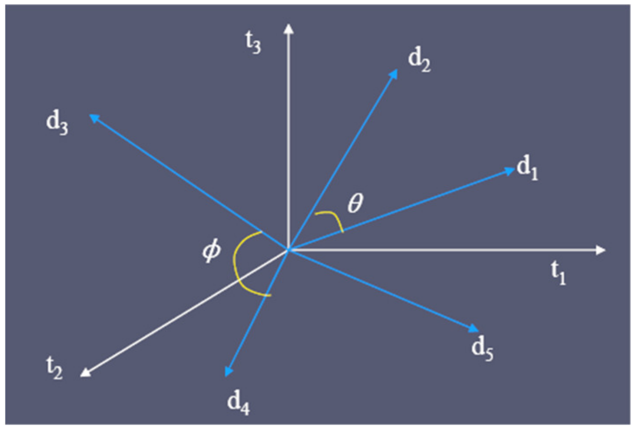
- Key idea 1: Do the same for queries: represent them as vectors in the space
- Key idea 2: Rank documents according to their proximity to the query in this space
- proximity = similarity of vectors
- proximity ≈ inverse of distance
- Recall: We do this because we want to get away from the you're-either-in-or-out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

Intuition

- A document d_j and a user query q are represented as *t-dimensional* vectors
  - t is the total number of index terms in the system
  - Each term is identified by a base in the n-dimensional space

$$\vec{t}_i = [0,0,0,....,1,....,0]$$

  - The query vector q is defined as $\underline{q} = (w_{1,q}, w_{2,q}, . . . , w_{t,q})$
  - The vector for a document d_j is represented by $\underline{d_j} = (d_{1,j}, d_{2,j}, . . . , d_{t,j})$
  - $w_{t,q}$ *and* $d_{t,j}$ can assume positive values {0,1}
    1 if the term is present, 0 otherwise
- A document d_j is represented as a document vector $\vec{d}_j = \sum_{i=1}^{N} w_{ij} \vec{t}_i$



- The degree of similarity of the document d_j with regard to the query q is the *correlation* between the vectors $\underline{q}$ *and* $\underline{d_j}$

Formalizing Vector Space Proximity

- First take: distance between two points
  - (= distance between the end points of the two vectors)

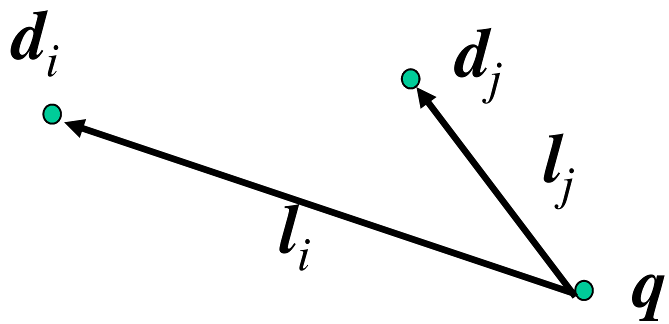
- Euclidean distance?
- Euclidean distance is a bad idea . . .

  . . . because Euclidean distance is large for vectors of different lengths.

Similarity 1st Take: Euclidean Distance

- The euclidean distance increases with decreasing similarity
  - Euclidean *distance*:

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$



Example: IR for Text Documents

Documents:

Dimension	$d1$	$d2$	$d3$
Corsica	0,1	0,6	1
Beach	0,3	0,2	0,8

3 documents (d1, d2, d3) and 2 dimensions (Corsica and Beach):

The weights in the table indicate the importance of the dimension in the document and affect the length of the vector.

Example: IR for Text Documents

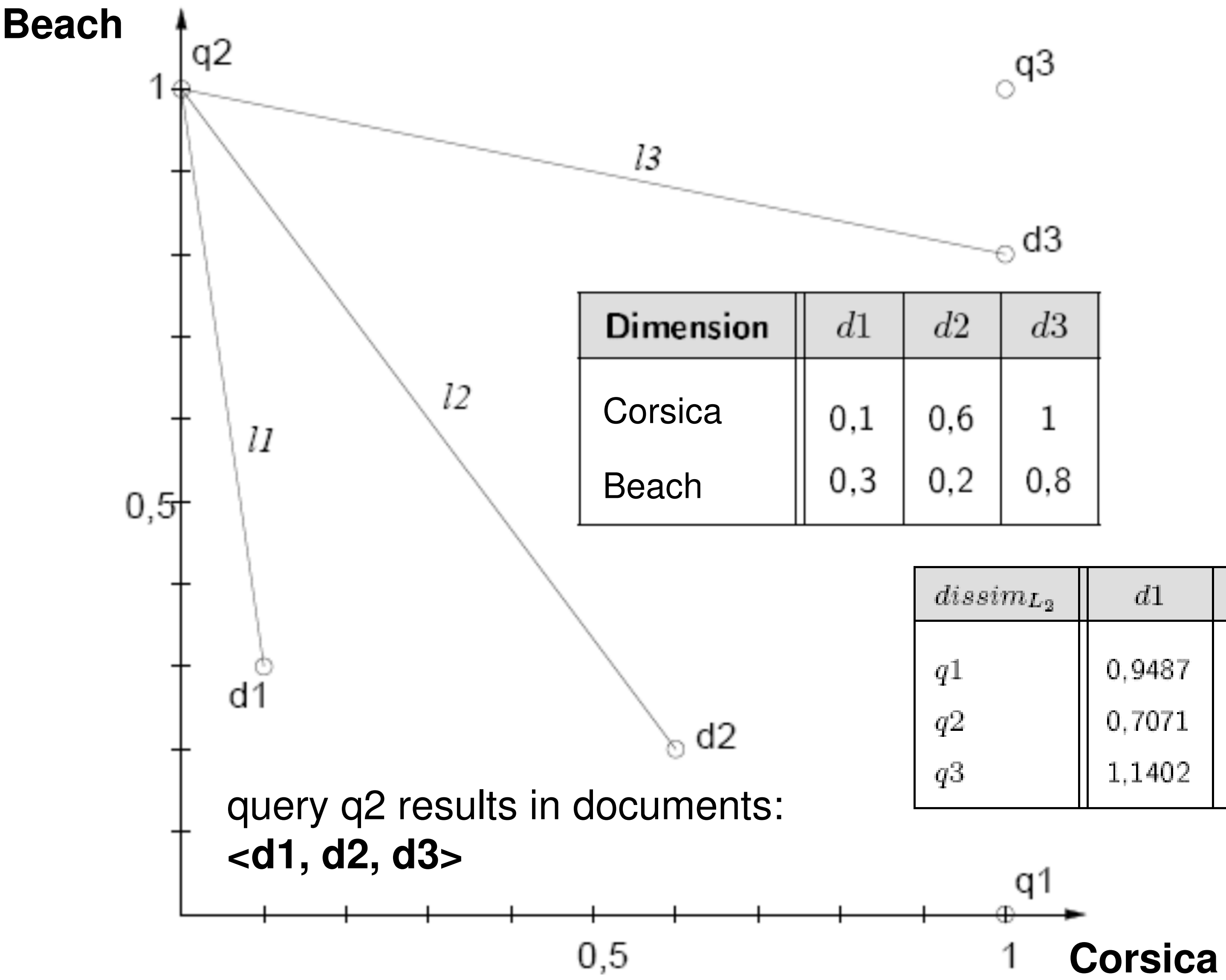
Queries:

Dimension	$q1$	$q2$	$q3$
Corsica	1	0	1
Beach	0	1	1

3 queries (d1, d2, d3) and 2 dimensions (Corsica and Beach):

q1 is only interested in documents with Corsica as content,
q2 is interrested in documents with Beach as content and
q3 requests all documents which cover both contents.

Example: Euclidean Distance



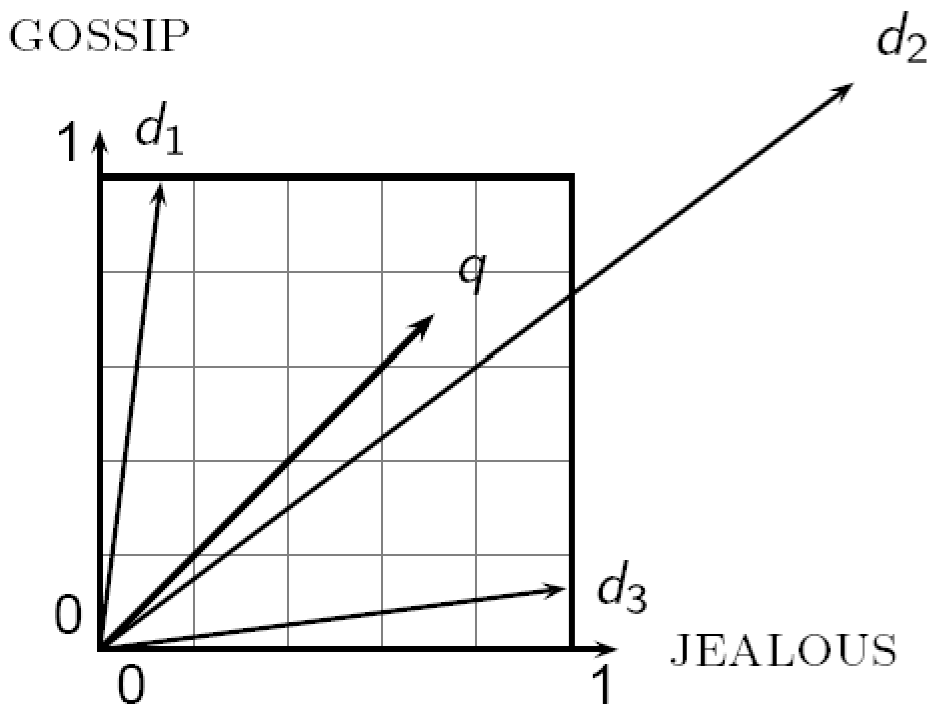
Dimension	$d1$	$d2$	$d3$
Corsica	0,1	0,6	1
Beach	0,3	0,2	0,8

$dissim_{L_2}$	$d1$	$d2$	$d3$
$q1$	0,9487	0,4472	0,8
$q2$	0,7071	1	1,0198
$q3$	1,1402	0,8944	0,2

query q2 results in documents:
<d1, d2, d3>

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.

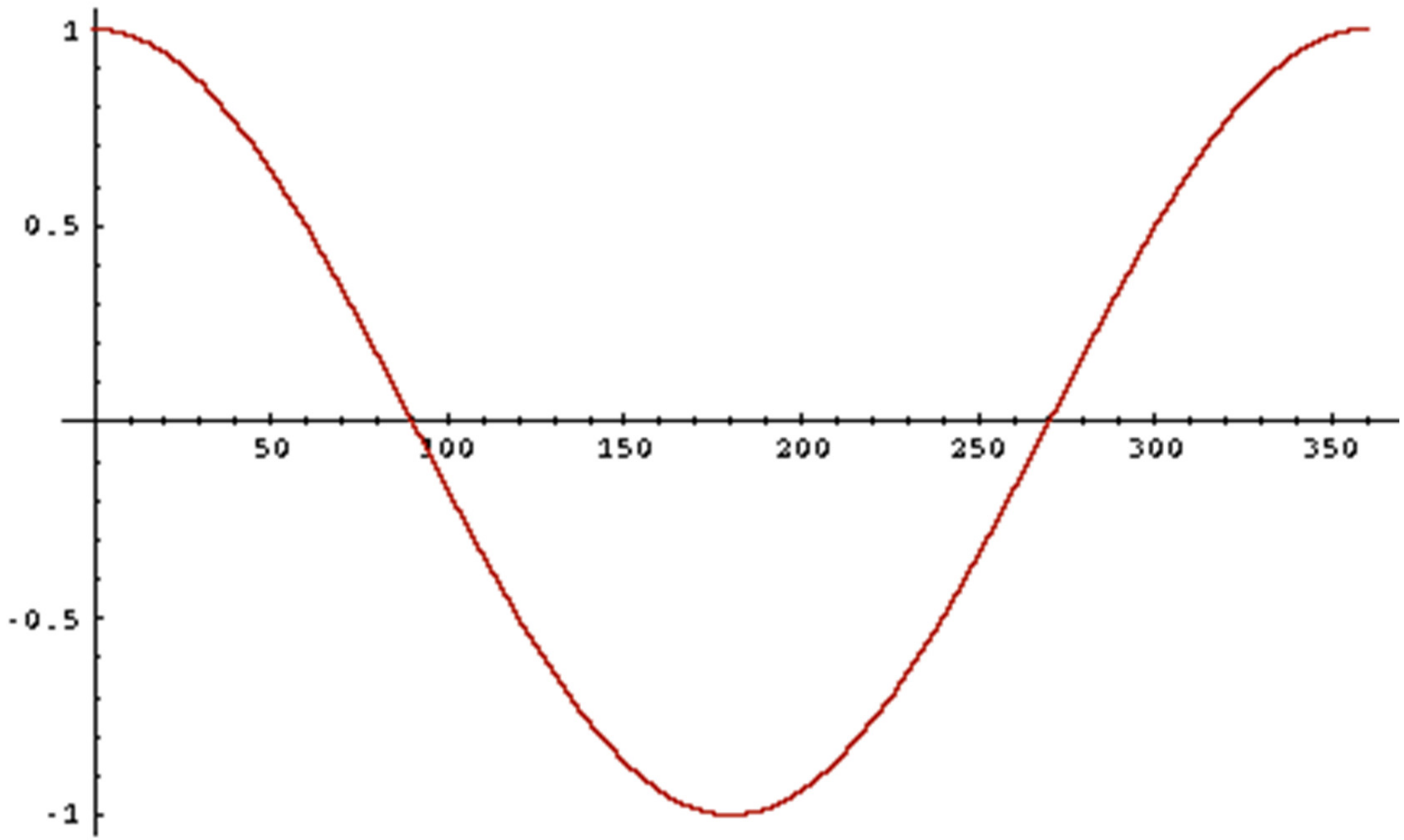
- Key idea: Rank documents according to angle with query.

From Angles to Cosines

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

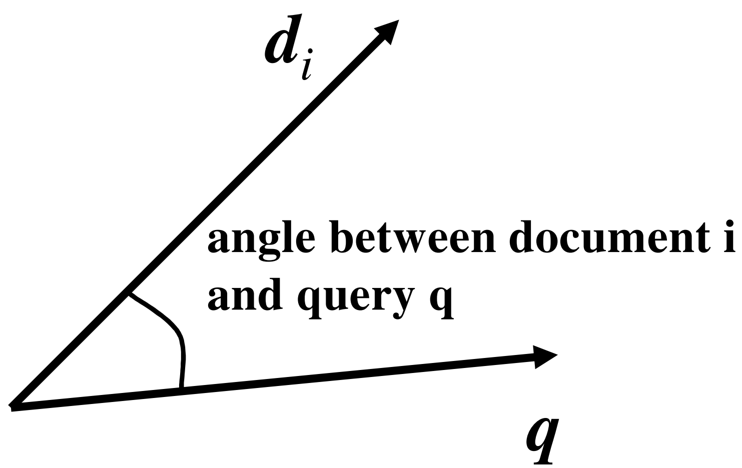
From Angles to Cosines

- But how should we be computing cosines?



Similarity 2nd Take: Cosine of Angle

- The cosine measure increases with decreasing angle. The largest cosine measures indicate the best results.
  - cosine of angle: $sim_{cos}(d,q) = \dfrac{\langle d,q\rangle}{|d| * |q|}$



Cosine(Query,Document)

Dot (inner) product

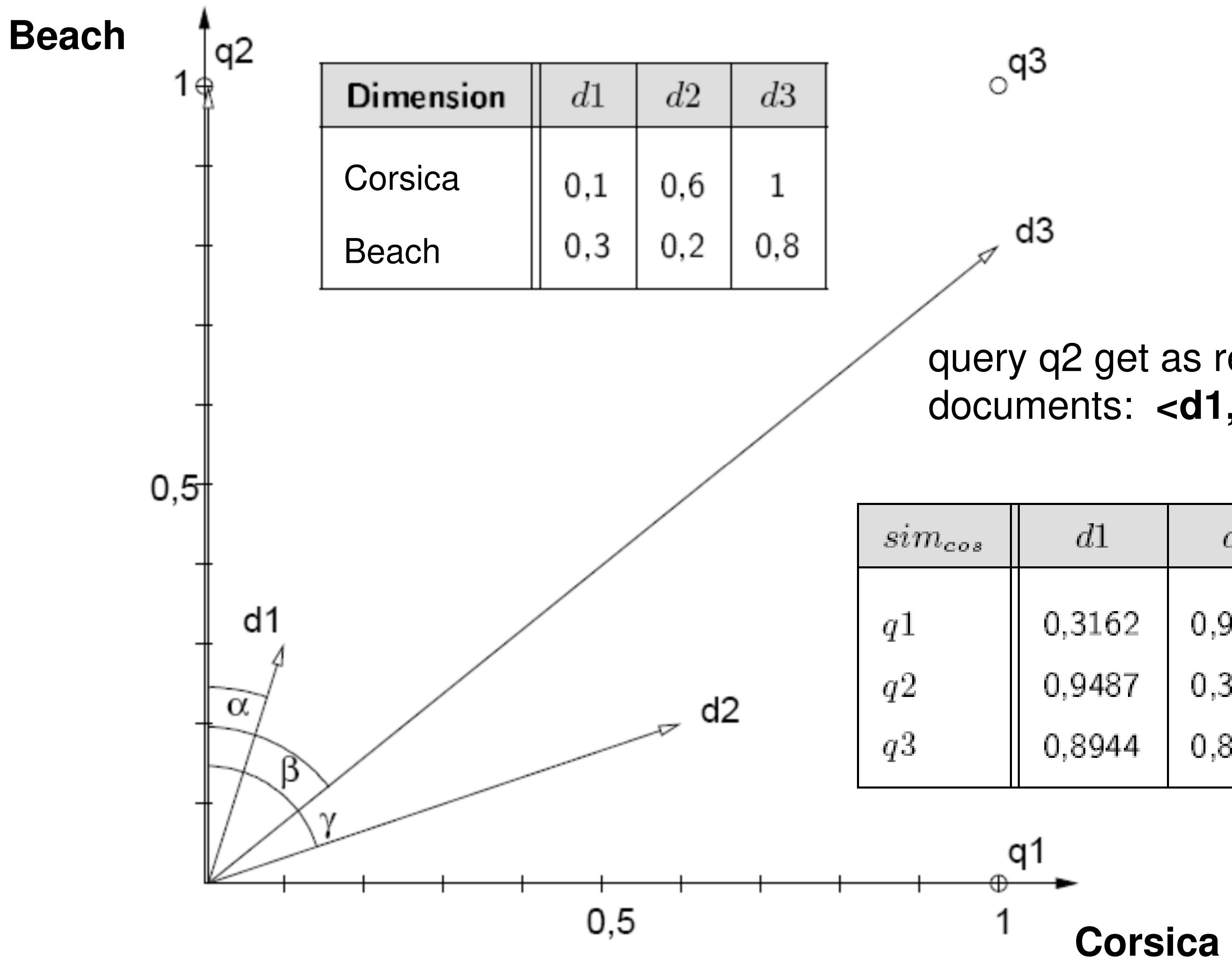
$$\cos(\vec{q},\vec{d}) = \frac{\vec{q}\bullet\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\vec{q}}{|\vec{q}|}\bullet\frac{\vec{d}}{|\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

length

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 $\vec{d}$.

Example: Cosine Measure



Dimension	$d1$	$d2$	$d3$
Corsica	0,1	0,6	1
Beach	0,3	0,2	0,8

query q2 get as result the documents: **<d1, d3, d2>**

sim_{cos}	$d1$	$d2$	$d3$
$q1$	0,3162	0,9487	0,7809
$q2$	0,9487	0,3162	0,6247
$q3$	0,8944	0,8944	0,9939

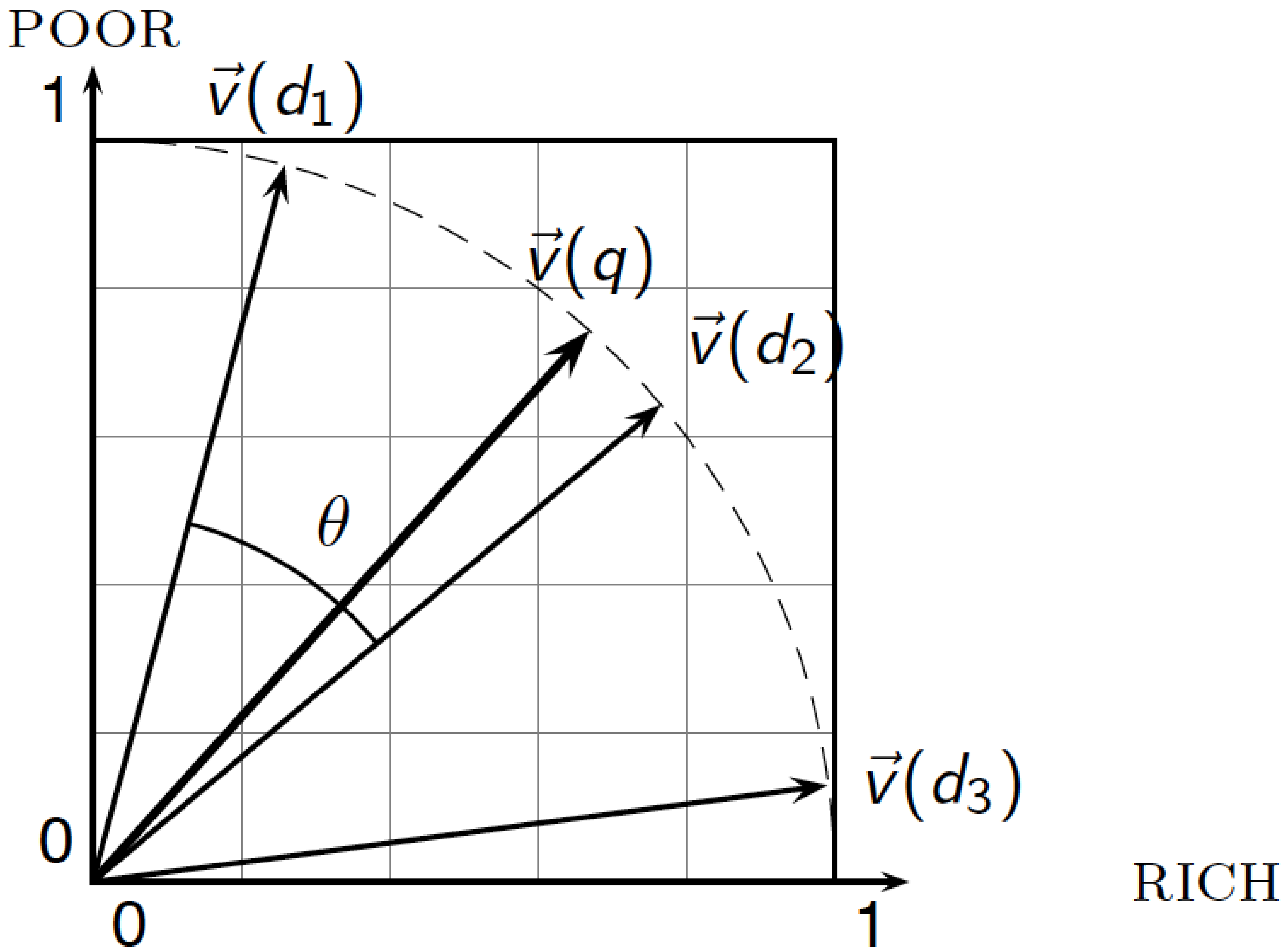
Length Normalization

- 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 Similarity Illustrated



Cosine Similarity amongst 3 Documents

term	SaS	PaP	WH
Affection (dt.: Zuneigung)	115	58	20
Jealous (dt.: Eifersüchtig)	10	7	11
Gossip (dt.: Klatschtante)	2	0	6
Wuthering (dt.: brausend)	0	0	38

Term frequencies (counts)

- How similar are the novels
- SaS: *Sense and Sensibility*
- PaP: *Pride and Prejudice*, and
- WH: *Wuthering Heights*?

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) $\approx$ 0.94
cos(SaS,WH) $\approx$ 0.79
cos(PaP,WH) $\approx$ 0.69

Computing Cosine Scores

CosineScore(q)

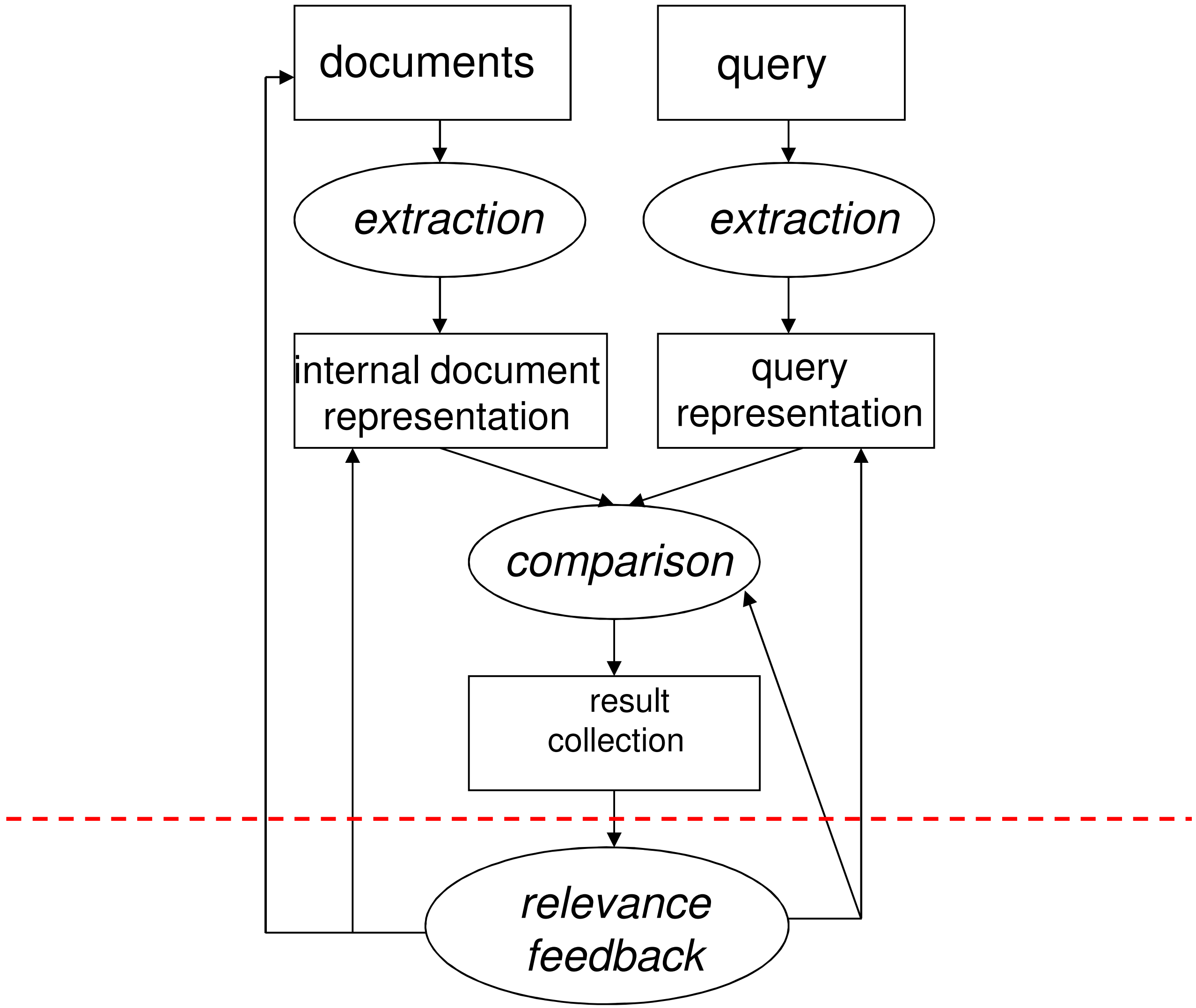
$float\ Scores[N] = 0$
$float\ Length[N]$
for each query term t
do calculate $w_{t,q}$ and fetch postings list for t
    for each pair(d, $tf_{t,d}$) in postings list
    do $Scores[d] += w_{t,d} \times w_{t,q}$
Read the array $Length$
for each d
do $Scores[d] = Scores[d]/Length[d]$
return Top K components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

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

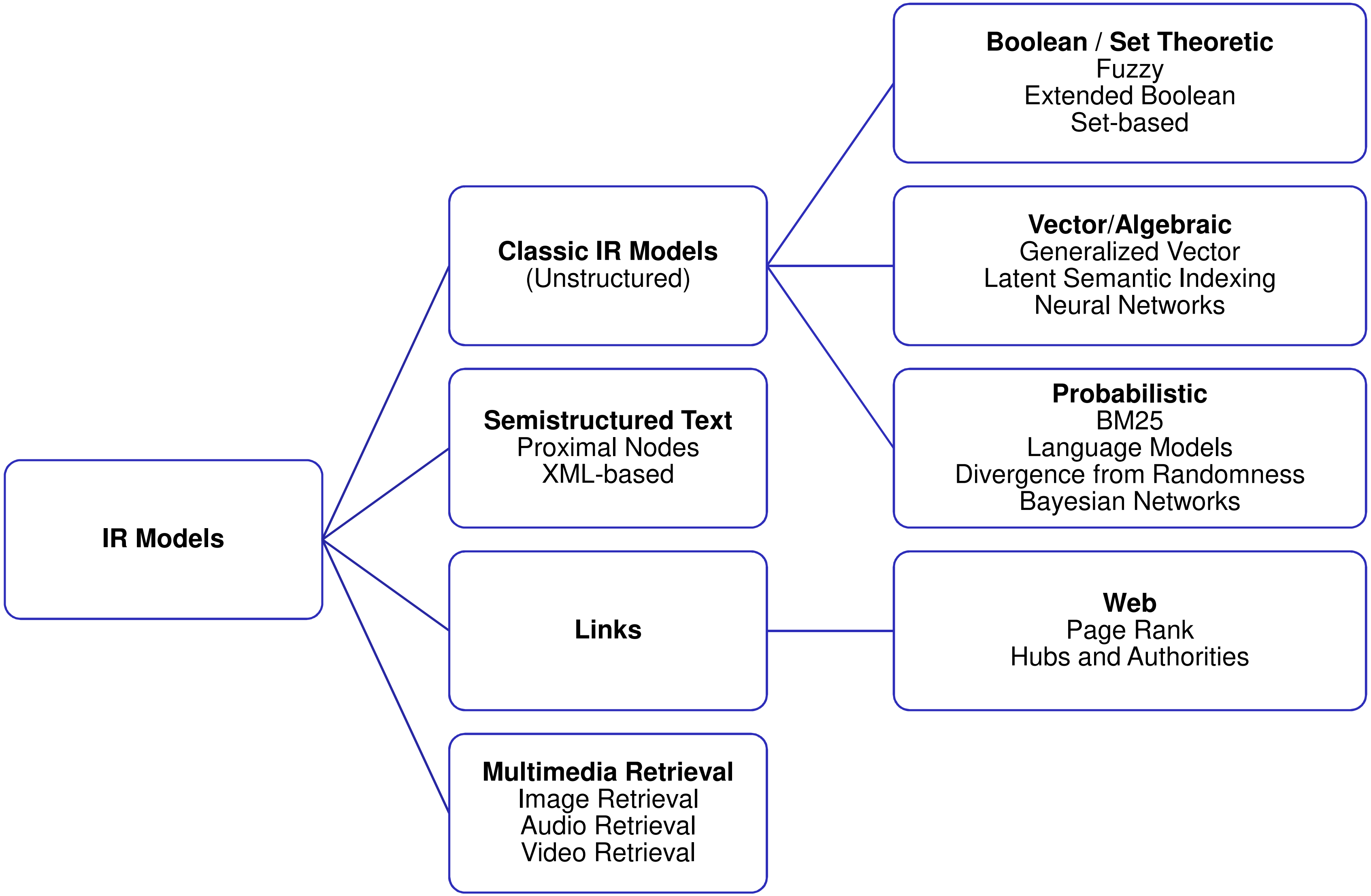
Recap: Retrieval Models

Overview


documents
query
extraction
extraction
internal document representation
query representation
comparison
result collection
relevance feedback

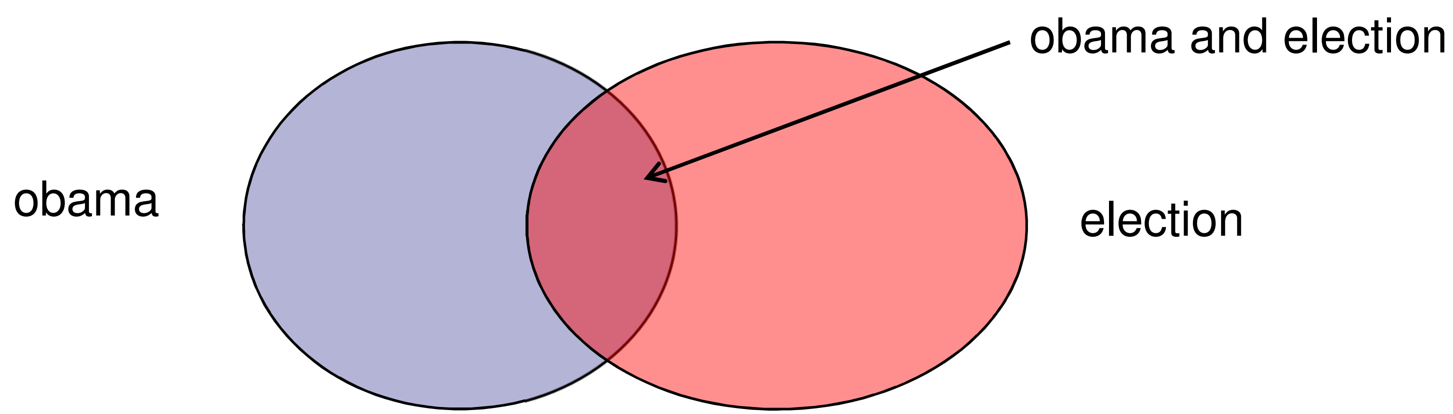
IR Models

Figure taken from R. Baeza-Yates, B. Ribeiro-Neto: Information Retrieval, 2nd edition, Addison-Wesley



Boolean Model

- Combination of set theory and boolean algebra
- Query term describes a set of documents
- Retrieve all documents matching the query
- Terms can be combined by boolean operators NOT, AND, OR
  - obama AND election, romney OR ryan
- Simple model, reason why a certain document was retrieved is inherently clear
- Ranking/weighting not possible, no similarity given, just 0 and 1



Vector Space Model

- Represent documents and queries as vectors of (weighted) terms

  $\vec{d} = (t_1, t_2, \dots, t_n)$

  $\vec{q} = (t_1, t_2, \dots, t_n)$
- Retrieve those documents most similar to the query
- Similarity defined by cosine angle between two vectors

$$sim(\vec{q}, \vec{d}) = \frac{\sum_{k=1}^{n} d_k \cdot q_k}{\sqrt{\sum_{k=1}^{n} (d_k)^2 \cdot \sum_{k=1}^{n} (q_k)^2}}$$

- Ranking possible
- Used for multiple purposes

Set-based Model

- Enhance boolean model with ranking
- Adapted association rule mining
- Boolean model assumes that terms are not dependent
- Set-based model exploits related (dependent) index terms
- Main idea: reduce boolean model to comparing sets of terms which frequently co-occur and hence are somewhat related (Possas et al. 2002)

- *Related* index terms form a termset
- n-termset

  $$S_i = (t_1, t_2, \ldots, t_n)$$
- Document d contains a termset if all terms of termset are contained in d

Set-based Model

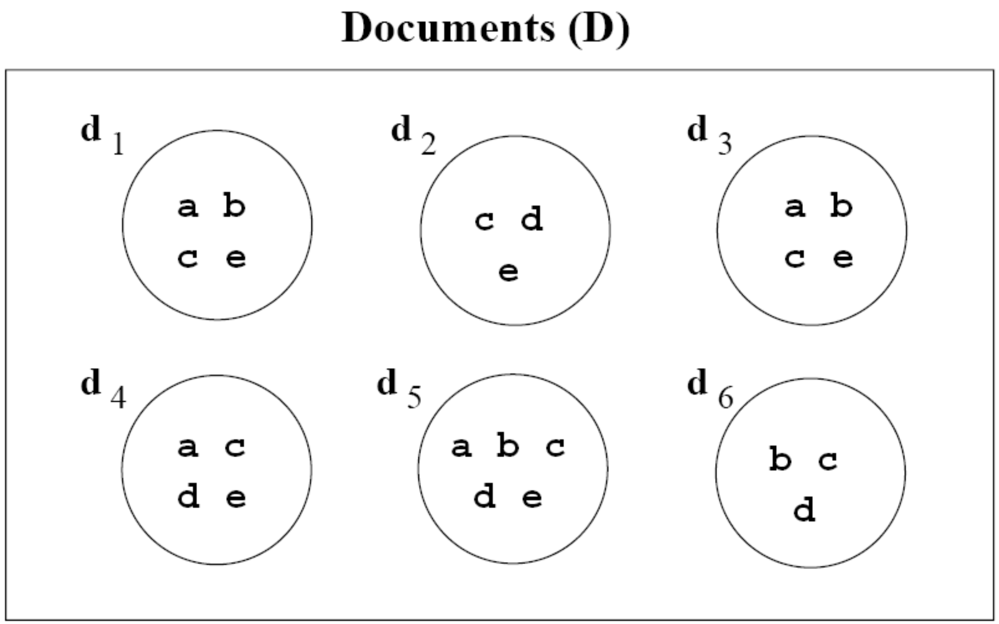
- Frequency = support in the context of association rules (cf. Apriori algorithm by Agrawal et al.)
- Termset S is called frequent if the number of occurrences in documents is higher than a given treshold (support)
- Termset is only frequent if all its subsets are frequent
- Closed termset = frequent termset which is the largest termset for which all subsets occur in the same documents
- Maximal termset = frequent termset which is not subset of any other frequent termset

Set-based Model

- Algorithm
  - check all 1-element sets for support > treshold (n = 1)
  - for all termsets of size n+1
    - determine all pairs of termsets with first n-1 items
    - $(s_i, s_j), 1 \leq i, j \leq 2^t$
    - create new termset by $(s_i \cup s_j)$
    - new list of documents where new termset appears $(l_i \cap l_j)$
    - check for all closed termsets if new termset f is closure of these sets
      - if f is a closure, discard all these sets, set f as closed termset
      - if f is not a closure, discard f

Set-based Model

•



- S_i = {∅, a, b, c, d, e, ab, ac, ad, ae, bc, cd, bd, be, ce, de, abc, abd, abe, acd, ace, ade, bcd, bce, bde, cde, abce, acde, bcde, abcde}

Figure taken from Possas et al. : *Set-based model: a new approach for information retrieval*, SIGIR '02 Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval, pages 230 – 237.

Set-based Model

- Support treshold = 3
- Determine frequency of itemsets

Frequency	Itemsets
6	c
5	e, ce
4	a, b, d, ac, ae, bc, cd, ace
3	ab, be, de, abc, abe, bce, cde, abce
2	ad, bd, acd, ade, bcd, abce
1	bde, abd, bcde, abcde

frequent itemsets

Set-based Model

Frequency	Frequent Termsets	Documents
6	c	{d1, d2, d3, d4, d5, d6}
5	e, ce	{d1, d2, d3, d4, d5}
4	a, ac, ae, ace	{d1, d3, d4, d5}
4	b, bc	{d1, d3, d5, d6}
4	d, cd	{d2, d4, d5, d6}
4	cd	{d2, d4, d5, d6}
3	ab, be, abc, abe, bce, abce	{d1, d3, d5}
3	de, cde	{d2, d4, d5}

Set-based Model

Frequency	Frequent Termsets	Closed Termset
6	c	c
5	e, ce	ce
4	a, ac, ae, ace	ace
4	b, bc	bc
4	d, cd	cd
4	cd	cd
3	ab, be, abc, abe, bce, abce	abce
3	de, cde	cde

Ranking Computation

- Vector-based
- Vectors based on (weighted) termsets (maximum number of 2^t entries)

  → document described by contained (weighted) termsets

- Termsets have to be weighted
  - How relevant is the termset for the description of a single document
  - How relevant is the termset for the whole document collection

Summary

- Ranking of result documents required
- Different retrieval models
  - Boolean model
  - Set-based model
  - Vector-Space model