universität koblenz
weiter:denken

Web Information Retrieval
Vector Space Model & Relevance Feedback
(SOSE 2023, 10.05.2023)

Frank Hopfgartner, Stefania Zourlidou
Institute for Web Science and Technologies

Credit for these slides

universität koblenz
weiter:denken

These slides have been adapted from

- Web IR (Zeyd Boukhers-WeST, SOSE 2020)

Recapitulation

- What is a Boolean model?
- What is inverted index?
- What are phrase queries?
- What are proximity queries?

Objectives of this lecture

- Boolean Model: Pros and Cons
- Ranked retrieval model
- Documents scoring
  - TF-IDF
- Query-document matching
  - Jaccard
  - Cosine
- Vector Space Model
- Relevance feedback

1. Boolean vs. Ranked Retrieval Models

universität koblenz
weiter:denken

Boolean Model: pros and cons

- Advantages
  - Good for expert users with precise understanding of their needs and the collection
  - Good for applications which can easily consume (process) 1000s of results
- Disadvantages
  - Not good for the majority of users: expressing information needs as Boolean expressions is unintuitive
  - Not practical for users: no ranking + too many results

Boolean Model

- 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

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

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 the 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 is, the higher the score (should be)
      - We will consider a number of alternatives to this

Jaccard coefficient

- A commonly used measure of overlap of two sets
- Let *A* and *B* be two sets
  - Jaccard coefficient

$$\text{JACCARD}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

  - 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: Example

- What is the query-document match score that the Jaccard coefficient computes for the
  - Query: “ides of March”, and
  - Document “Caesar died in March”
    - JACCARD(q, d) = 1/6

Issues with Jaccard for scoring

$$\textsc{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

- It doesn't consider *term frequency* (how many times a term occurs in a document)
- Are all words in a document equally important? *(stop words)*
  - Rare terms in a collection are more informative than frequent terms
    - Jaccard does not consider this aspect
- We need a more sophisticated way of normalizing for length

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
- We will look at "recovering" positional information later in this course
- For now: bag of words model

Term frequency tf

- The term frequency $\mathbf{tf}_{t,d}$ 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 defined as follows

$$w_{t,d} = \begin{cases} 1 + \log_{10} tf_{t,d} & if\ tf_{t,d} > 0 \\ 0 & Otherwise \end{cases}$$

- $tf_{t,d} \rightarrow w_{t,d}$
  - $0 \rightarrow 0, 1 \rightarrow 1, 2 \rightarrow 1.3, 10 \rightarrow 2, 1000 \rightarrow 4$, etc.
- Score for a document-query pair: sum over terms t in both q and d:
  - tf-matching-score$(q, d) = \sum_{t \in q \cap d} (1 + \log tf_{t,d})$
- 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., *archaeology*)
- A document containing this term is very likely to be relevant to the query *archaeology*
  - We want a high weight for rare terms like *archaeology*

Document frequency, continued

- 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
- But it’s not a sure indicator of relevance
- For frequent terms, we want high positive weights for words like *high, increase, and line*
- But lower weights than for rare terms
- We will use document frequency (df) to capture this

Inverse document frequency (idf) weight

- **df$_t$** is the document frequency of term t: the number of documents in the collection that contain a term t
  - df$_t$ is an inverse measure of the informativeness of t
  - df$_t$ $\leq N$
- We define the **idf** (inverse document frequency) of t by

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

  - We use log (N/df$_t$) instead of N/df$_t$ to "dampen" the effect of idf

idf example, suppose N = 1 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 weighing 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

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

- The $\mathbf{tf\text{-}idf_{t,d}}$ weight of a term *t* in a document *d* is the product of its tf weight and its idf weight

$$\mathrm{tf\text{-}idf_{t,d}} = \mathrm{tf}_{t,d} \times \mathrm{idf_{t,d}}$$

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

- 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

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

- There are many variants based on
  - how “tf” is computed (with/without logs)
  - whether the terms in the query are weighted too
  - and more

2. Vector Space Model

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 $\in \{0,1\}^{|V|}$

Count matrices

	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	3,18	0	0	0	0
Brutus	1,21	6,1	0	1	0	0
Caesar	8,59	2,54	0	1,51	0,25	1.95
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	0

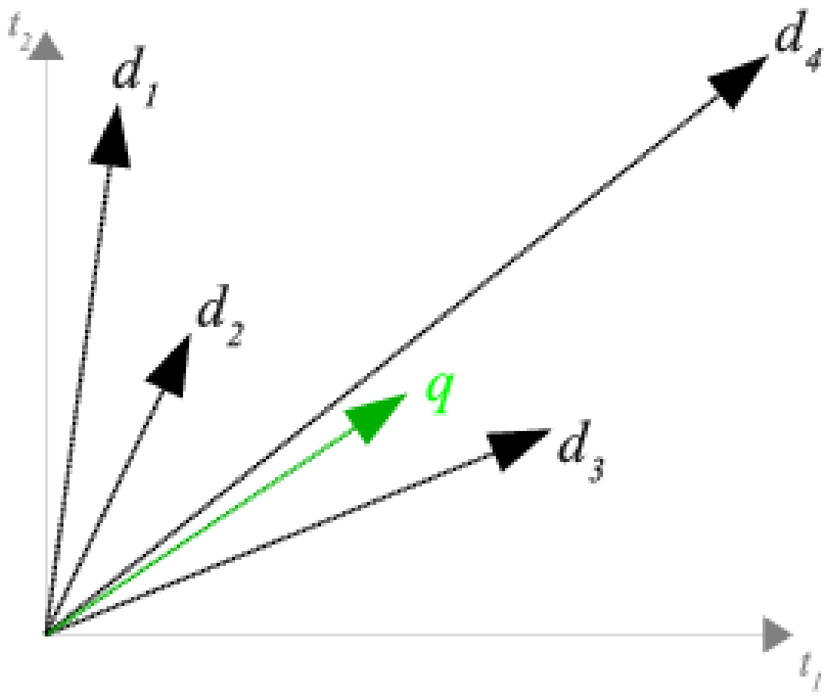
- Each document is now represented by a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$

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: tens of millions of dimensions when you apply this to a web search engine
- These are very sparse vectors - most entries are zero

Idea

- Documents as vectors in a high dimensional space
  - Queries are documents
- Terms define dimensions (base vectors)
- Assumptions
  - Base vectors orthogonal
  - Similar documents have similar vectors
  - Vector similarity indicates document similarity
    - Distance
    - Angle



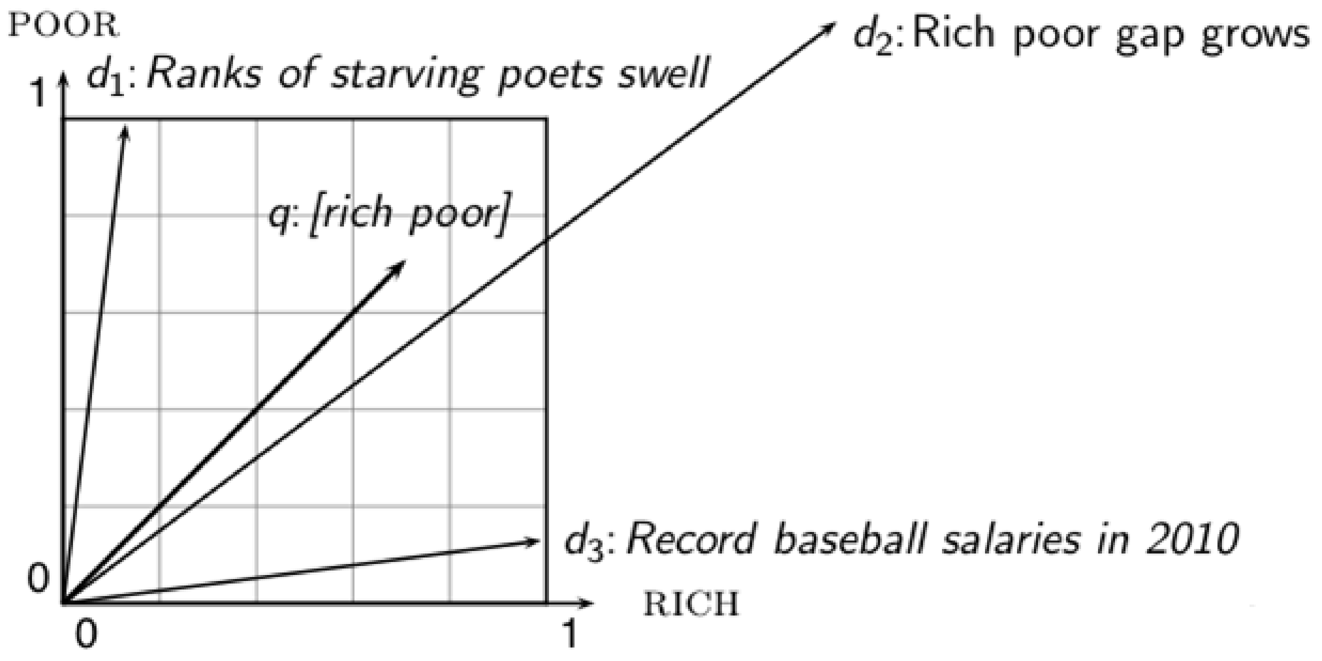
Similarity ranking for *q*?
- With distance
- With angle

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 <u>large</u> for vectors of <u>different lengths</u>

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



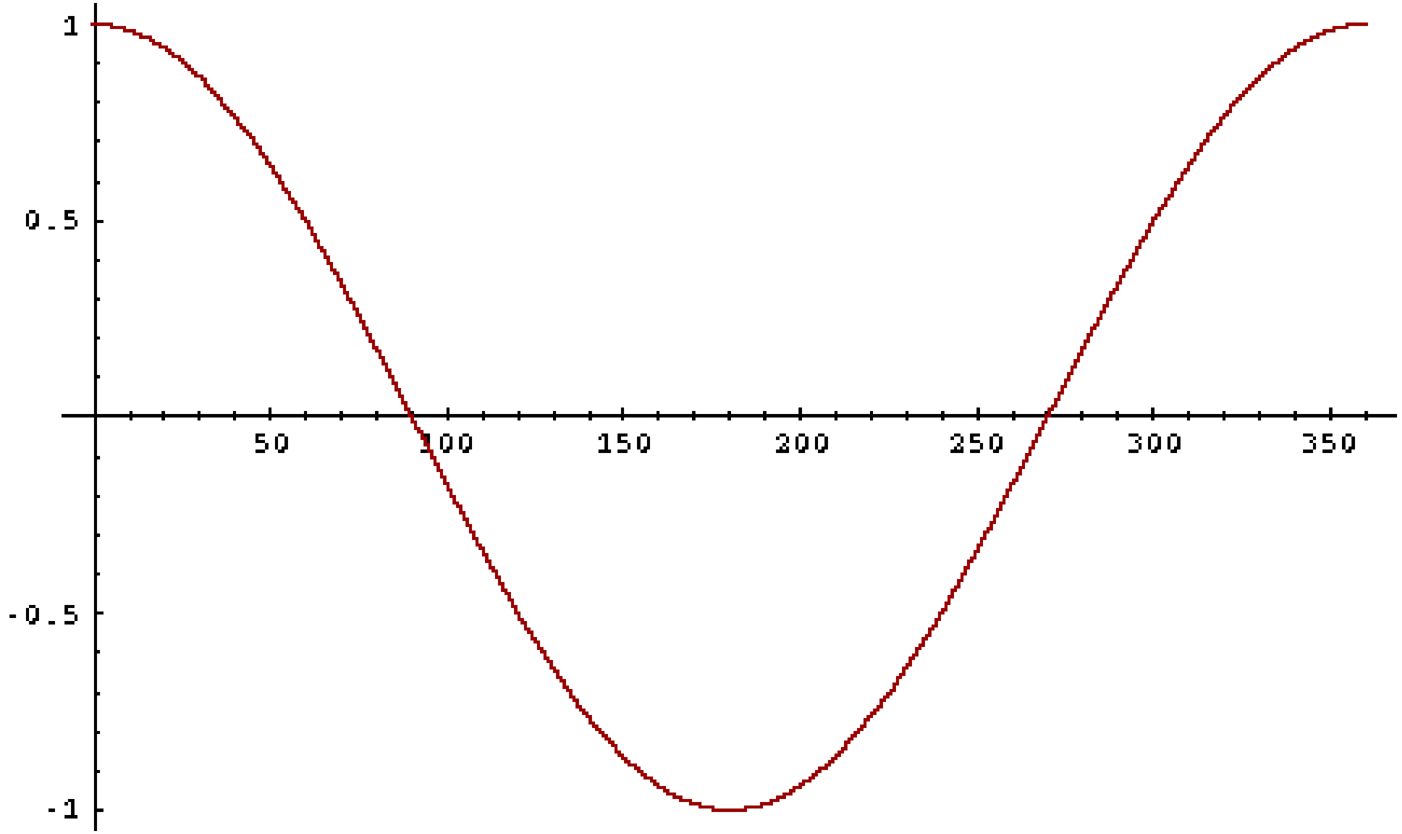
Use angle instead of distance

- 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

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



The following two notions are equivalent

- Rank documents in decreasing order of the angle between query and document
- Rank documents in increasing order of cosine(query,document)

- But how – *and why* – should we compute cosines?

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 *q* and *d* … or, equivalently, the cosine of the angle between *q* and *d*

Vector Space Model (VSM) Framework

universität koblenz
weiter:denken

- Corpus: $D = \{d_1, \dots, d_N\}$
- Vocabulary: $V = \{t_1, \dots, t_M\}$
- Documents as vectors in R^M

$$\vec{d}_i = \left(w_i^{[1]}, w_i^{[2]}, \cdots, w_i^{[M]}\right)$$

  – Where $w_i^{[j]}$ weight of term t_j in document d_i

- Queries:

$$\vec{q} = \left(w_q^{[1]}, w_q^{[2]}, \cdots, w_q^{[M]}\right)$$

VSM Framework

weiter:denken

- Retrieval function

$$\rho\left(\vec{d}_i,\vec{q}\right) = \mathrm{sim}\left(\vec{d}_i,\vec{q}\right)$$

- Similarity measure

$$\mathrm{sim}: \mathrm{R}^M \times \mathrm{R}^M \to [0,1]$$

  - With
    - Value 1: same vector
    - Value 0: „completely different“ vector
- Result list
  - All documents with ρ>0, sorted by descending score

Combined Weights

- Combined weight: TF-IDF

$$w_{\mathrm{TF.IDF}}\left(t_j, d_i\right) = w_{\mathrm{local}}\left(t_j, d_i\right) \times w_{\mathrm{global}}\left(t_j\right)$$
$$= \mathrm{tf}\left(t_j, d_i\right) \times \log\left({}^{N}\!\big/_{\mathrm{df}\left(t_j\right)}\right)$$

  – In particular: weight zero for tf(t_j,d_i)=0 or df(t_j)=N

- Similarity: Cosine measure

$$\mathrm{sim}\left(d_i, q\right) = \cos\left(\vec{d}_i, \vec{q}\right) = \frac{\vec{d}_i \cdot \vec{q}}{\left|\vec{d}_i\right| \cdot \left|\vec{q}\right|}$$

  – Includes length normalization

Example

- Global weights
  - E.g. for „cup“

$$w_{\text{global}}(\text{cup}) = \log\left({N}/{\text{df(cup)}}\right)$$

$$\log\left({5}/{3}\right) \approx 0.22$$

t_j	$df(t_j)$	$w_{global}(t_j)$
coffee	3	0.22
cup	3	0.22
jar	4	0.10
tea	2	0.40
water	1	0.70

1. coffee, coffee
2. cup, jar, jar, tea, tea
3. coffee, cup, cup, jar
4. coffee, coffee, coffee, cup, cup, cup, jar, jar, jar, tea
5. jar, jar, water, water

Example

- Combined with local weights

t_j	d_1	d_2	d_3	d_4	d_5
coffee	2		1	3	
cup		1	2	3	
jar		2	1	3	2
tea		2		1	
water					2

$$\vec{d}_2 = (0\cdot 0.22, 1\cdot 0.22, 2\cdot 0.10, 2\cdot 0.4, 0\cdot 0.7)$$

t_j	d_1	d_2	d_3	d_4	d_5
coffee	0.44	0	0.22	0.66	0
cup	0	0.22	0.44	0.66	0
jar	0	0.20	0.10	0.30	0.20
tea	0	0.80	0	0.40	0
water	0	0	0	0	1.40

1. coffee, coffee
2. cup, jar, jar, tea, tea
3. coffee, cup, cup, jar
4. coffee, coffee, coffee, cup, cup, cup, jar, jar, jar, tea
5. jar, jar, water, water

t_j	$df(t_j)$	$w_{global}(t_j)$
coffee	3	0.22
cup	3	0.22
jar	4	0.10
tea	2	0.40
water	1	0.70

Query vector

- Construction of query vector
  - Just like a document
  - Using global weights from corpus

$$w_{\mathrm{TF.IDF}}(t_j, q) = w_{\mathrm{local}}(t_j, q) \times w_{\mathrm{global}}(t_j)$$
$$= \mathrm{tf}(t_j, q) \times \log\left({}^{N}\!/_{\mathrm{df}(t_j)}\right)$$

- Note
  - Frequency of query terms matters
  - Sequence does not matter (bag of words)

Example

- Query: „cup jar“
- Query vector

$$\vec{q} = (0, 0.22, 0.1, 0, 0)$$

- Vector lengths:

$$|\vec{q}| = \sqrt{0.22^2 + 0.1^2} = 0.24$$

$$|\vec{d}_1| = 0.44$$

$$|\vec{d}_2| = 0.85$$

$$|\vec{d}_3| = 0.50$$

$$|\vec{d}_4| = 1.06$$

$$|\vec{d}_5| = 1.41$$

1. coffee, coffee
2. cup, jar, jar, tea, tea
3. coffee, cup, cup, jar
4. coffee, coffee, coffee, cup, cup, cup, jar, jar, jar, tea
5. jar, jar, water, water

t_j	d_1	d_2	d_3	d_4	d_5
coffee	0.44	0	0.22	0.66	0
cup	0	0.22	0.44	0.66	0
jar	0	0.20	0.10	0.30	0.20
tea	0	0.80	0	0.40	0
water	0	0	0	0	1.40

Example

- Query vector:

$$\vec{q} = (0, 0.22, 0.1, 0, 0)$$

- Relevance (e.g. d_5):

$$\rho(\vec{d}_5, \vec{q}) = \frac{0 \cdot 0 + 0.22 \cdot 0 + 0.1 \cdot 0.2 + 0 \cdot 0 + 0 \cdot 1.4}{0.24 \cdot 1.41}$$

- Ranking

Rank	Document	ρ
1	d_3	0.89
2	d_4	0.69
3	d_2	0.33
4	d_5	0.05

1. coffee, coffee
2. cup, jar, jar, tea, tea
3. coffee, cup, cup, jar
4. coffee, coffee, coffee, cup, cup, cup, jar, jar, jar, tea
5. jar, jar, water, water

t_j	d_1	d_2	d_3	d_4	d_5
coffee	0.44	0	0.22	0.66	0
cup	0	0.22	0.44	0.66	0
jar	0	0.20	0.10	0.30	0.20
tea	0	0.80	0	0.40	0
water	0	0	0	0	1.40
Length	***0.44***	***0.85***	***0.50***	***1.06***	***1.41***

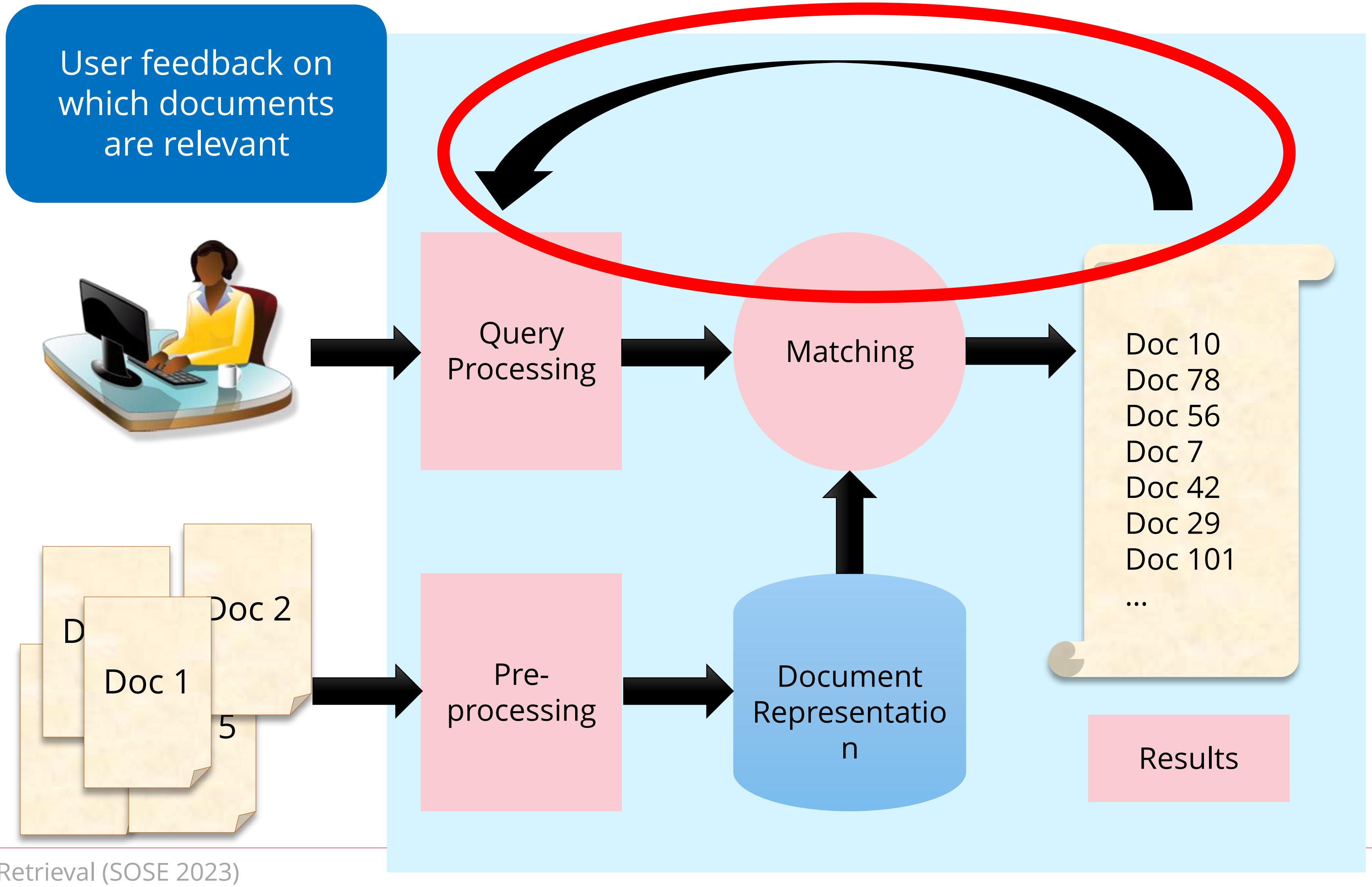
d_1 not in result list as $\rho(d_1, q) = 0$

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

3. Relevance Feedback (Rocchio)

Relevance Feedback for VSM



Incorporate Feedback

- User provides for some documents if they are relevant or irrelvant:
  - D^p : set of documents with positive feedback (relevant)
  - D^n : set of documents with negative feedback (irrelevant)
- Adjust query vector

$$\vec{q}_{FB} = \alpha \cdot \vec{q} + \beta \frac{1}{|D^p|} \sum_{d_i \in D^p} \vec{d}_i - \gamma \frac{1}{|D^n|} \sum_{d_i \in D^n} \vec{d}_i$$

  - Parameters: typically $\alpha > \beta > \gamma$
  - Adjust negative term weights to 0

Example

- Query:
  - paris hilton
- Documents

	paris	hilton	hotel	france	eiffel	blonde	heiress	actress
d_1	3	1	1	2	1			
d_2	1	3	4	1	3			
d_3	2	1		1				
d_4	3			2			1	
d_5		1	3					
d_6	3	3				2		
d_7	2	2					2	
d_8	2	1	1			1	1	
d_9	3	2				1		4
d_{10}	3	2		1			2	3

Example

- Initial result list based on VSM
- User provides relevance feedback
  - Positive 
  - Negative 
- Parameter:
  - $\alpha = 1$
  - $\beta = 0.75$
  - $\gamma = 0.15$

Rank	Doc	ρ	
1	d_3	0.395	
2	d_6	0.183	
3	d_7	0.161	
4	d_4	0.132	
5	d_1	0.128	
6	d_8	0.125	
7	d_{10}	0.071	
8	d_9	0.057	
9	d_2	0.049	
10	d_5	0.027	

Example

$$\vec{q}_{FB} = \alpha \cdot \vec{q} + \beta \frac{1}{|D^p|} \sum_{d_i \in D^p} \vec{d}_i - \gamma \frac{1}{|D^n|} \sum_{d_i \in D^n} \vec{d}_i$$

- Adjust query

$\alpha = 1$
$\beta = 0.75$
$\gamma = 0.15$

$$\vec{q}_{FB} = 1 \cdot \begin{pmatrix} 0.046 \\ 0.046 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + 0.75 \cdot \frac{1}{2} \cdot \left[\begin{pmatrix} 0.137 \\ 0.046 \\ 0.398 \\ 0.602 \\ 0.699 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0.092 \\ 0.092 \\ 0 \\ 0.301 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \right]$$

$$-0.15 \cdot \frac{1}{2} \cdot \left[\begin{pmatrix} 0.092 \\ 0.092 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0.796 \\ 0 \end{pmatrix} + \begin{pmatrix} 0.092 \\ 0.046 \\ 0 \\ 0 \\ 0 \\ 0.523 \\ 0.398 \\ 0 \end{pmatrix} \right] \mapsto \begin{pmatrix} 0.118 \\ 0.087 \\ 0.119 \\ 0.339 \\ 0.262 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Rank	Doc	ρ
1	d_1	0.957
2	d_3	0.787
3	d_2	0.691
4	d_4	0.640
5	d_5	0.262
6	d_8	0.172
7	d_{10}	0.119
8	d_6	0.056
9	d_7	0.050
10	d_9	0.018

- Retrieve new results

Relevance Feedback in Practice

- Users unwilling to give feedback
  - Query reformulation is easier
- Pseudo relevance feedback
  - Retrieve result list (do not show to user)
  - Use top-k results as positive feedback
    - Rarely low-ranking documents as negative feedback
  - Adjust query vector
  - Retrieve final result list
- Works good, when initial results are good

4. Summary

Summary

- At the end of this lecture you should understand the following concepts
  - Boolean Model
  - Ranked retrieval model
  - Scoring
  - Term frequency
  - Document frequency
  - TF-IDF
  - Vector Space Model
  - Relevance feedback

References

[1] https://olat.vcrp.de/auth/RepositoryEntry/4071063853

[2] https://nlp.stanford.edu/IR-book/information-retrieval-book.html Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Introduction to Information Retrieval, Cambridge University Press. 2008.

- Chapter 5 (VSM) and Chapter 9 (Relevance Feedback)