
Web-Mining Agents

Prof. Dr. Ralf Möller

Universität zu Lübeck

Institut für Informationssysteme

Tanya Braun (Übungen)

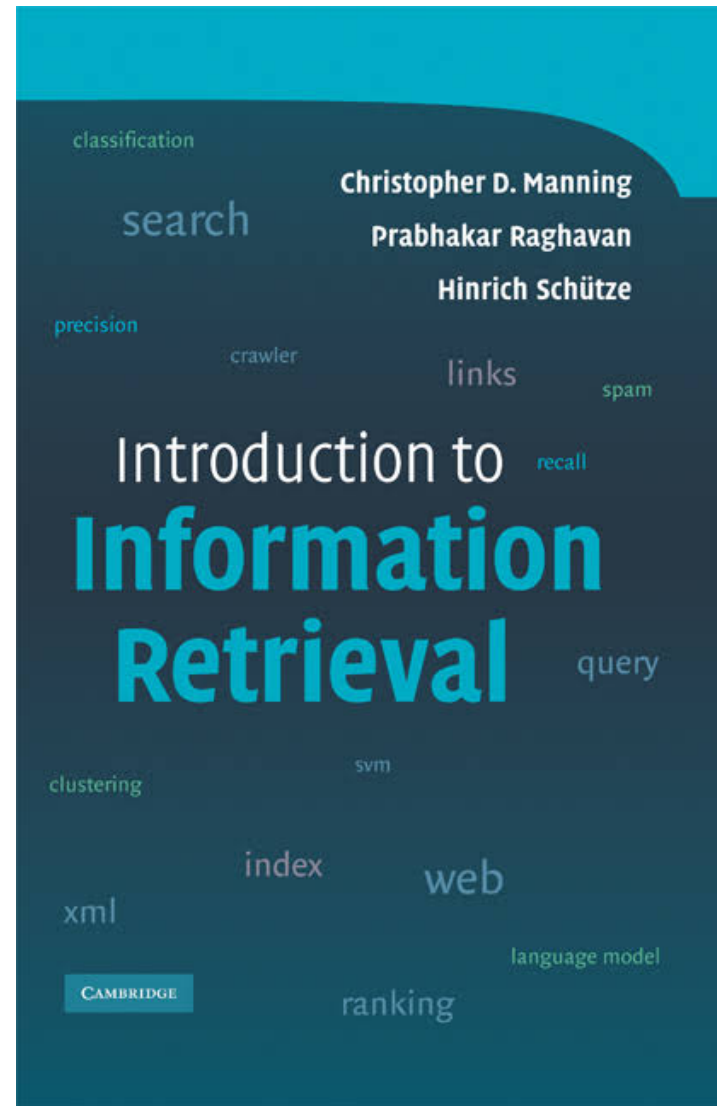


Agents for Information Retrieval (on the web)



Information Retrieval

- Slides taken from presentation material for the following book:

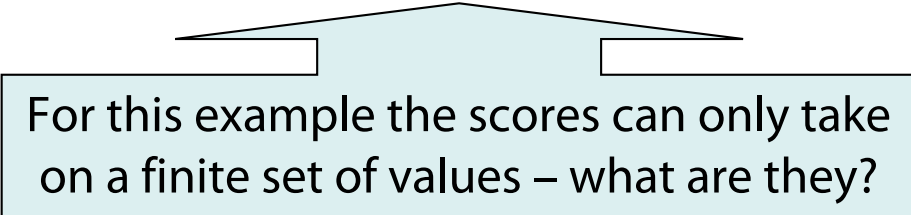


Text-based queries

- Agents are asked to return documents most likely to be useful to the searcher (w/ ordering)
- How can we rank order the docs in a corpus with respect to a query?
- Assign a score – say in $[0,1]$
 - for each doc on each query

Linear zone combinations

- First generation of scoring methods: use a linear combination of occurrence queries (boolean):
 - E.g., Score =
 $0.6 * \langle \text{"sorting"} \text{ in Title} \rangle + 0.3 * \langle \text{"sorting"} \text{ in Abstract} \rangle + 0.05 * \langle \text{"sorting"} \text{ in Body} \rangle + 0.05 * \langle \text{"sorting"} \text{ in Boldface} \rangle$
 - Each expression such as $\langle \text{"sorting"} \text{ in Title} \rangle$ takes on a value in $\{0,1\}$
 - Then the overall score is in $[0,1]$



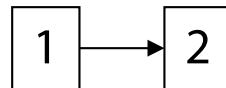
For this example the scores can only take on a finite set of values – what are they?

Inverted index – Postings lists

- On the query **bill OR rights** suppose that we retrieve the following docs from the various zone indexes:

Author

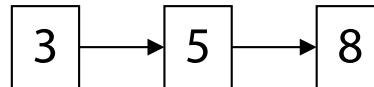
bill



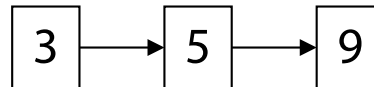
rights

Title

bill

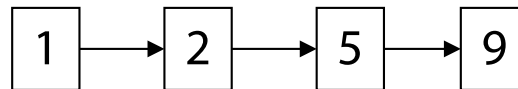


rights

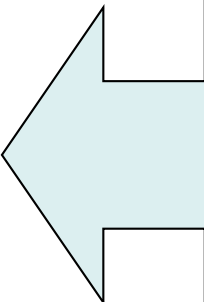
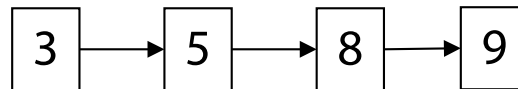


Body

bill



rights



Compute the score for each doc based on the weightings 0.6,0.3,0.1

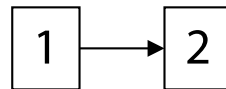
General idea

- We are given a weight vector whose components sum up to 1.
 - There is a weight for each zone/field.
- Given a Boolean query, we assign a score to each doc by adding up the weighted contributions of the zones/fields
- Typically users want to see **k** highest-scoring docs (top-k query)

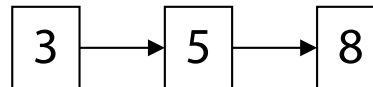
Index support for zone combinations

- In the simplest version we have a separate inverted index for each zone
- Variant: have a single index with a separate dictionary entry for each term and zone
- E.g.,

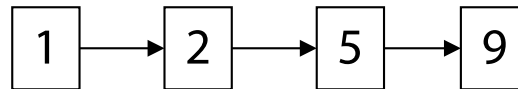
bill.author



bill.title



bill.body

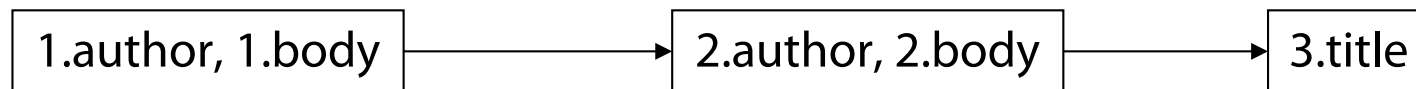


Of course, compress zone names like author/title/body.

Zone combinations index

- The above scheme is still wasteful: each term is potentially replicated for each zone
- In a slightly better scheme, we encode the zone in the postings:

bill

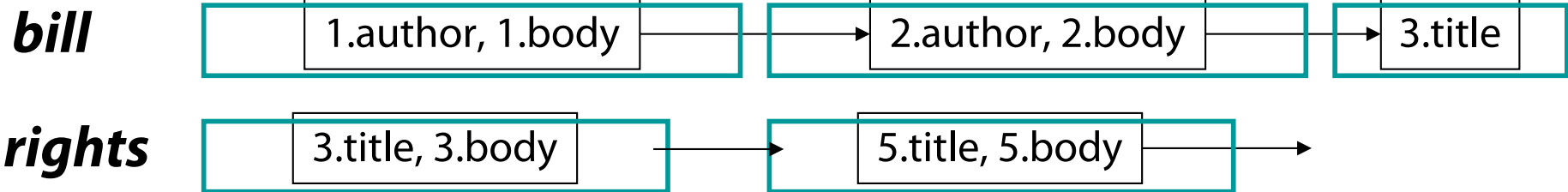


As before, the zone names get compressed.

- At query time, accumulate contributions to the total score of a document from the various postings, e.g.,

Score accumulation

1	0.7
2	0.7
3	0.4
5	0.4



- As we walk the postings for the query **bill OR rights**, we accumulate scores for each doc in a linear merge as before.
- Note: we get both **bill** and **rights** in the Title field of doc 3, but score it no higher.
- Should we give more weight to more hits?

Where do these weights come from?

- Machine learned relevance
- Given
 - *A test corpus*
 - *A suite of test queries*
 - *A set of relevance judgments*
- Learn a set of weights such that relevance judgments matched
- Can be formulated as **an optimization problem** (see lecture part on data mining/machine learning)

Full text queries

- We just scored the Boolean query **bill OR rights**
- Most users more likely to type **bill rights** or **bill of rights**
 - How do we interpret these *full text* queries?
 - No Boolean connectives
 - Of several query terms some may be missing in a doc
 - Only some query terms may occur in the title, etc.

Full text queries

- To use zone combinations for free text queries, we need
 - A way of assigning a score to a pair <free text query, zone>
 - Zero query terms in the zone should mean a zero score
 - More query terms in the zone should mean a higher score
 - Scores don't have to be Boolean
- Will look at some alternatives now

Incidence matrices

- Bag-of-words model
- Document (or a zone in it) is binary vector X in $\{0,1\}^V$
- Query is a vector Y
- Score: Overlap measure:

$$|X \cap Y|$$

	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
...

Example

- On the query ***ides of march***, Shakespeare's *Julius Caesar* has a score of 3
- All other Shakespeare plays have a score of 2 (because they contain ***march***) or 1
- Thus in a rank order, *Julius Caesar* would come out tops

Overlap matching

- What's wrong with the overlap measure?
- It doesn't consider:
 - Term **frequency** in document
 - Term **scarcity** in collection
(document mention frequency)
 - **of** is more common than **ides** or **march**
 - **Length of documents**
 - (and queries: score not normalized)

Overlap matching

- One can normalize in various ways:

- Jaccard coefficient:

$$|X \cap Y| / |X \cup Y|$$

- Cosine measure:

$$(X \cdot Y) / \sqrt{|X| \times |Y|}$$

- What documents would score best using Jaccard against a typical query?
- Does the cosine measure fix this problem?

Scoring: density-based

- Thus far: position and overlap of terms in a doc – title, author etc.
- Obvious next idea: If a document talks *more* about a topic, then it is a better match
- This applies even when we only have a single query term.
- Document is relevant if it has a lot of the terms
- This leads to the idea of term weighting.


Term-document count matrices

- Consider the number of occurrences of a term in a document:
 - Bag of words model
 - Document is a 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 view of a doc

- Thus the doc
 - **John is quicker than Mary.**
- is indistinguishable from the doc
 - **Mary is quicker than John.**



Which of the indexes discussed so far distinguish these two docs?

Counts vs. frequencies

- Consider again the **ides of march** query.
 - Julius Caesar has 5 occurrences of **ides**
 - No other play has **ides**
 - **march** occurs in over a dozen
 - All the plays contain **of**
- By this scoring measure, the top-scoring play is likely to be the one with the most **ofs**

Term frequency **tf**

- Long docs are favored because they're more likely to contain query terms
- Can fix this to some extent by normalizing for document length (**term frequency, tf**)
- But is raw **tf** the right measure?

Digression: terminology

- WARNING: In a lot of IR literature, “frequency” is used to mean “count”
 - Thus term **frequency** in IR literature is used to mean **number of occurrences** in a doc
 - Not divided by document length (which would actually make it a frequency)

Weighting term frequency: tf

- What is the relative importance of
 - 0 vs. 1 occurrence of a term in a doc
 - 1 vs. 2 occurrences
 - 2 vs. 3 occurrences ...
- Unclear: While it seems that more is better, a lot isn't proportionally better than a few
 - Can just use raw tf
 - Another option commonly used in practice:

$$wf_{t,d} = 0 \text{ if } tf_{t,d} = 0, 1 + \log tf_{t,d} \text{ otherwise}$$

Score computation

- Score for a query q = sum over terms t in q :

$$= \sum_{t \in q} tf_{t,d}$$

- [Note: 0 if no query terms in document]
- This score can be zone-combined
- Can use wf instead of tf in the above
- Still doesn't consider term scarcity in collection (***ides*** is rarer than ***of***)

Weighting should depend on the term overall

- Which of these tells you more about a doc?
 - 10 occurrences of *hernia*?
 - 10 occurrences of *the*?
- Would like to attenuate the weight of a common term
 - But what is “common”?
- Suggest looking at collection frequency (*cf*)
 - The total number of occurrences of the term in the entire collection of documents

Document frequency

- But document frequency (*df*) may be better:
- *df* = number of docs in the corpus containing the term

Word	<i>cf</i>	<i>df</i>
<i>ferrari</i>	10422	17
<i>insurance</i>	10440	3997

- Document/collection frequency weighting is only possible in known (static) collection.
- So how do we make use of *df*?

tf x idf term weights

- tf x idf measure combines:
 - term frequency (*tf*)
 - or *wf*, some measure of term density in a doc
 - inverse document frequency (*idf*)
 - measure of informativeness of a term: its rarity across the whole corpus
 - could just be raw count of number of documents the term occurs in (*idf_i* = n/df_i)
 - but by far the most commonly used version is:

$$idf_i = \log\left(\frac{n}{df_i}\right)$$

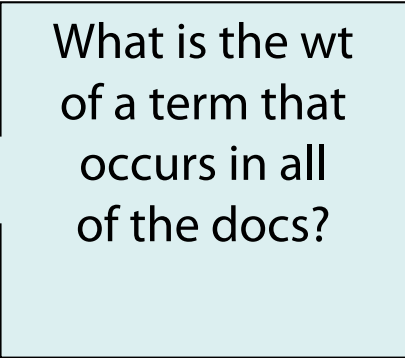
- See Kishore Papineni, NAACL 2, 2002 for theoretical justification

K. Spärck Jones. A Statistical Interpretation of Term Specificity and Its Application in Retrieval. *Journal of Documentation* 28: 11–21, 1972

Summary: tf x idf (or tf.idf)

- Assign a tf.idf weight to each term i in each document d

$$w_{i,d} = tf_{i,d} \times \log(n / df_i)$$



What is the wt of a term that occurs in all of the docs?

$tf_{i,d}$ = frequency of term i in document d

n = total number of documents

df_i = the number of documents that contain term i

- Increases with the number of occurrences *within* a doc
- Increases with the rarity of the term *across* the whole corpus

Real-valued term-document matrices

- Function (scaling) of count of a word in a document:
 - Bag of words model
 - Each is a vector in \mathbb{R}^v
 - Here log-scaled *tf.idf*

Note: can be >1!

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	13.1	11.4	0.0	0.0	0.0	0.0
Brutus	3.0	8.3	0.0	1.0	0.0	0.0
Caesar	2.3	2.3	0.0	0.5	0.3	0.3
Calpurnia	0.0	11.2	0.0	0.0	0.0	0.0
Cleopatra	17.7	0.0	0.0	0.0	0.0	0.0
mercy	0.5	0.0	0.7	0.9	0.9	0.3
worser	1.2	0.0	0.6	0.6	0.6	0.0

Documents as vectors

- Each doc d can now be viewed as a vector of $wf \times idf$ values, one component for each term
- So we have a vector space
 - terms are axes
 - docs live in this space
 - even with stemming, may have 20,000+ dimensions
- (The corpus of documents gives us a matrix, which we could also view as a vector space in which words live – transposable data)

Recap: tf x idf (or tf.idf)

- Assign a tf.idf weight to each term i in each document d

$$w_{i,d} = tf_{i,d} * \log(n / df_i)$$

$tf_{i,d}$ = frequency of term i in document d

n = total number of documents

df_i = the number of documents that contain term i

- Instead of tf, sometimes wf is used:

$$wf_{t,d} = 0 \text{ if } tf_{t,d} = 0, \quad 1 + \log tf_{t,d} \text{ otherwise}$$

Web-Mining Agents

Prof. Dr. Ralf Möller

Universität zu Lübeck

Institut für Informationssysteme

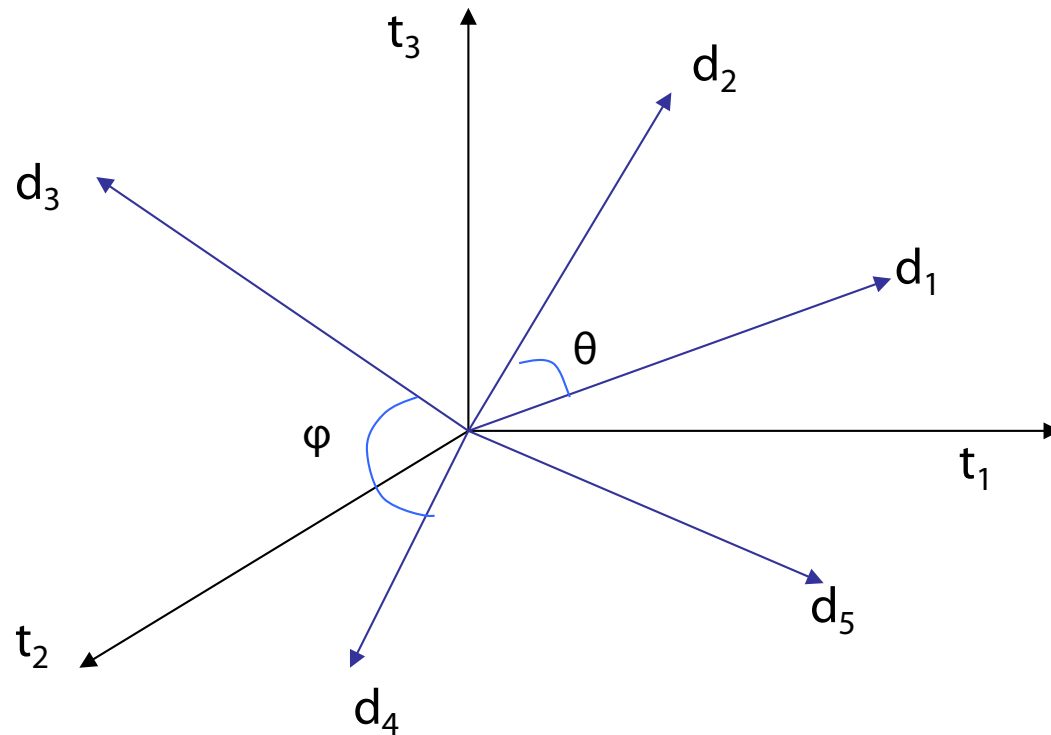
Tanya Braun (Übungen)



Documents as vectors

- At the end of the last lecture we said:
- Each doc d can now be viewed as a vector of $tf*idf$ values, one component for each term
- So we have a **vector space**
 - terms are axes
 - docs live in this space
 - even with stemming, may have 50,000+ dimensions
- First application: **Query-by-example**
 - Given a doc d , find others “like” it.
- Now that d is a vector, find vectors (docs) “near” it.

Intuition



Postulate: Documents that are “close together” in the vector space talk about the same things.

Desiderata for proximity/distance

- If d_1 is near d_2 , then d_2 is near d_1 .
- If d_1 near d_2 , and d_2 near d_3 , then d_1 is not far from d_3 .
- No doc is closer to d than d itself.
- Triangle inequality

First cut

- Idea: Distance between d_1 and d_2 is the length of the vector $d_1 - d_2$.

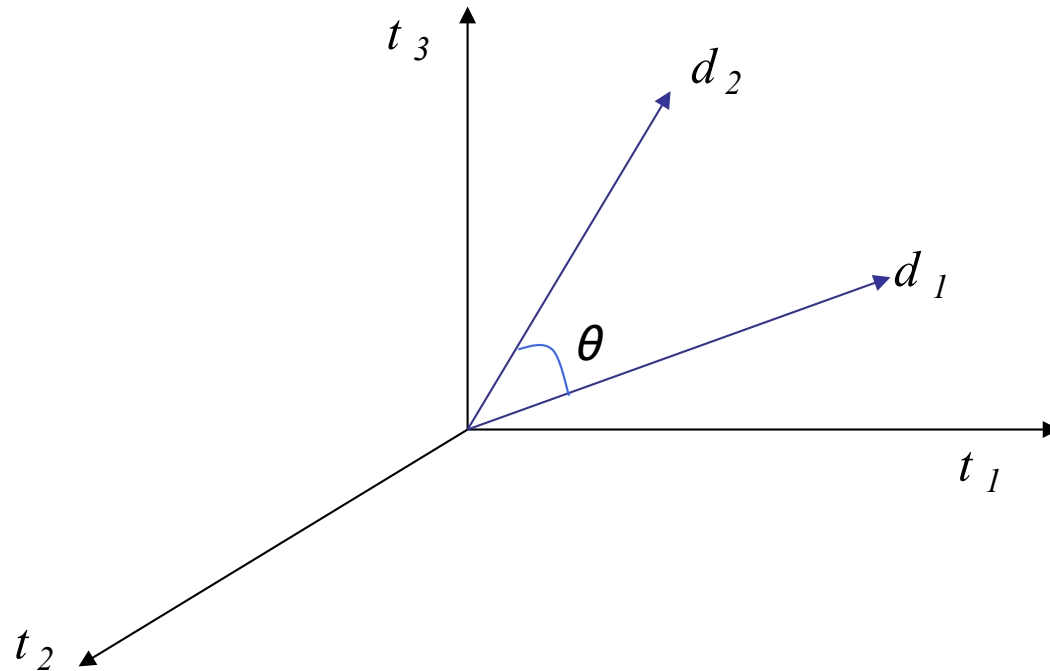
- Euclidean distance:

$$|d_j - d_k| = \sqrt{\sum_{i=1}^n (d_{i,j} - d_{i,k})^2}$$

- Why is this not a great idea?
- We still haven't dealt with the issue of **length normalization**
 - Short documents would be more similar to each other by virtue of length, not topic
- However, we can implicitly normalize by looking at angles instead

Cosine similarity

- Distance between vectors d_1 and d_2 captured by the cosine of the angle θ between them.
- Note – this is **similarity**, not distance
 - No triangle inequality for similarity.



Cosine similarity

- A vector can be *normalized* (given a length of 1) by dividing each of its components by its length – here we use the L_2 norm

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

- This maps vectors onto the unit sphere:

- Then,
$$\left| \vec{d}_j \right| = \sqrt{\sum_{i=1}^n w_{i,j}^2} = 1$$

- Longer documents don't get more weight

Cosine similarity

$$\text{sim}(d_j, d_k) = \frac{\vec{d}_j \cdot \vec{d}_k}{|\vec{d}_j| |\vec{d}_k|} = \frac{\sum_{i=1}^n w_{i,j} w_{i,k}}{\sqrt{\sum_{i=1}^n w_{i,j}^2} \sqrt{\sum_{i=1}^n w_{i,k}^2}}$$

- Cosine of angle between two vectors
- The denominator involves the lengths of the vectors.



- For normalized vectors, the cosine is simply the dot product:

$$\cos(\vec{d}_j, \vec{d}_k) = \vec{d}_j \cdot \vec{d}_k$$

Queries in the vector space model

Central idea: the **query as a vector**:

- We regard the query as short document
- We return the documents ranked by the closeness of their vectors to the query, also represented as a vector.

$$\text{sim}(d_j, d_q) = \frac{\vec{d}_j \cdot \vec{d}_q}{|\vec{d}_j| |\vec{d}_q|} = \frac{\sum_{i=1}^n w_{i,j} w_{i,q}}{\sqrt{\sum_{i=1}^n w_{i,j}^2} \sqrt{\sum_{i=1}^n w_{i,q}^2}}$$

- Note that d_q is very sparse!

What's the point of using vector spaces?

- A well-formed algebraic space for retrieval
- Key: A user's query can be viewed as a (very) short document.
- Query becomes a vector in the same space as the docs.
- Can measure each doc's proximity to it.
- Natural measure of scores/ranking – no longer Boolean.
 - Queries are expressed as **bags of words**
 - Clean metaphor for similar-document queries
- Not a good combination with Boolean, wild-card, positional query operators
- But ...

Efficient cosine ranking

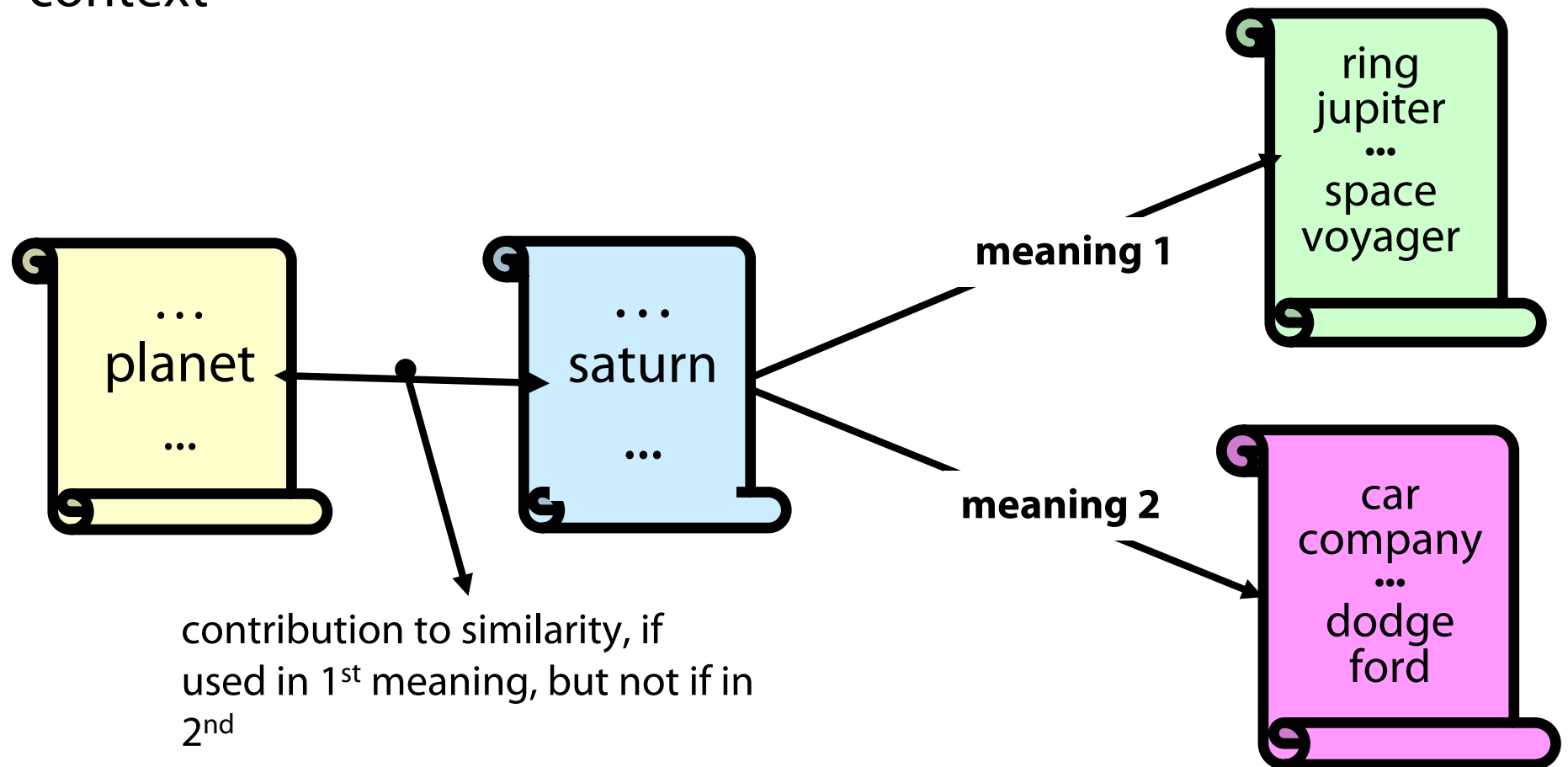
- Find the k docs in the corpus “nearest” to the query
⇒ Compute k best query-doc cosines
 - Nearest neighbor problem for a query vector
 - Multidimensional Index-Structures (see Non-Standard DBs lecture)
 - For a “reasonable” number of dimensions (say 10-100)
 - Otherwise space almost empty (curse of dimensionality)
 - What about zoning? Keep vectors, no linear combination!
 - Compute k best solutions with different zone-specific vectors for each doc
 - Can we do this without testing all combinations w.r.t. all zones?
 - Fagin’s algorithm (see Non-Standard Databases lecture)
 - What about multiple repositories?

Ronald Fagin. Combining Fuzzy Information from Multiple Systems. PODS-96, 216-226., 1996

Ronald Fagin: Fuzzy Queries in Multimedia Database Systems.
Proc. PODS-98, 1-10, 1998

Polysemy and Context

- Document similarity on single word level: polysemy and context



Problems with Lexical Semantics

- Ambiguity and association in natural language
 - **Polysemy**: Words often have a **multitude of meanings** and different types of usage (more severe in very heterogeneous collections).
 - The vector space model is unable to discriminate between different meanings of the same word.

$$\text{sim}_{\text{true}}(d, q) < \cos(\angle(\vec{d}, \vec{q}))$$

Problems with Lexical Semantics

- **Synonymy**: Different terms may have an **identical or a similar meaning** (weaker: words indicating the same topic).
- No associations between words are made in the simple vector space representation.

$$\text{sim}_{\text{true}}(d, q) > \cos(\angle(\vec{d}, \vec{q}))$$

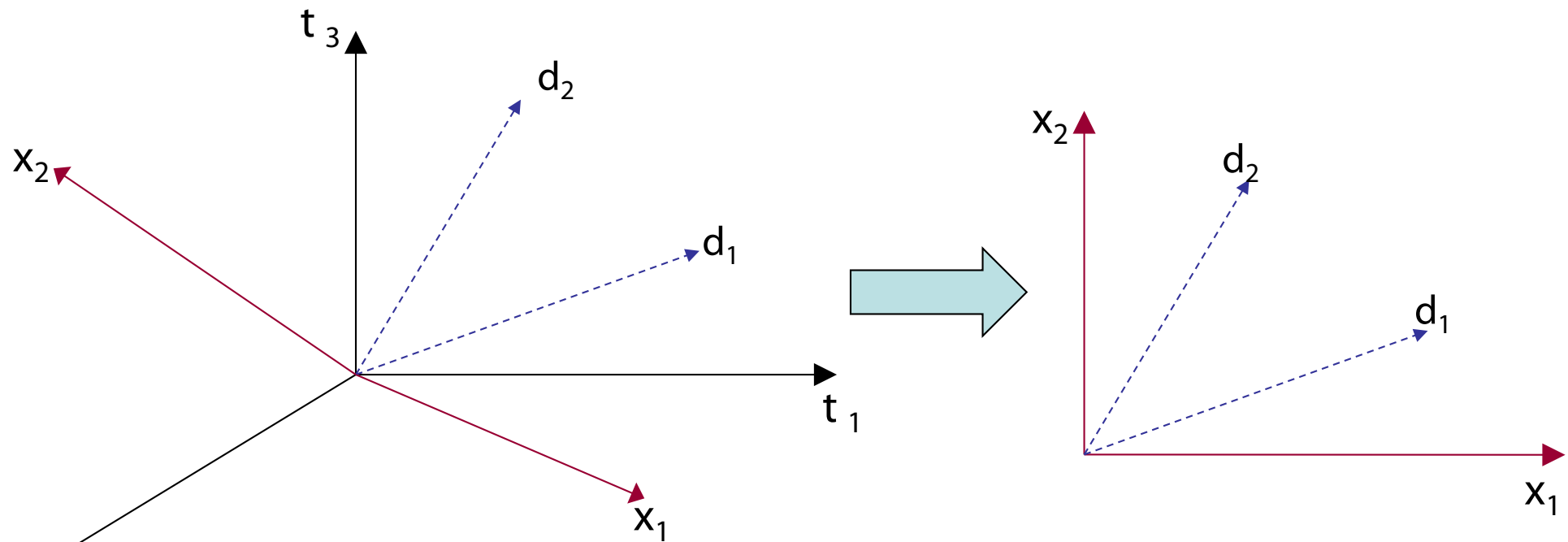
Dimensionality reduction

- What if we could take our vectors and “pack” them into fewer dimensions (say 50,000→100) while preserving distances?
- Two methods:
 - Random projection.
 - “Latent semantic indexing”.

Random projection onto $k \ll m$ axes

- Choose a random direction x_1 in the vector space.
- For $i = 2$ to k
 - choose a random direction x_i
that is orthogonal to x_1, x_2, \dots, x_{i-1} .
- Project each document vector into the subspace spanned by $\{x_1, x_2, \dots, x_k\}$.

E.g., from 3 to 2 dimensions



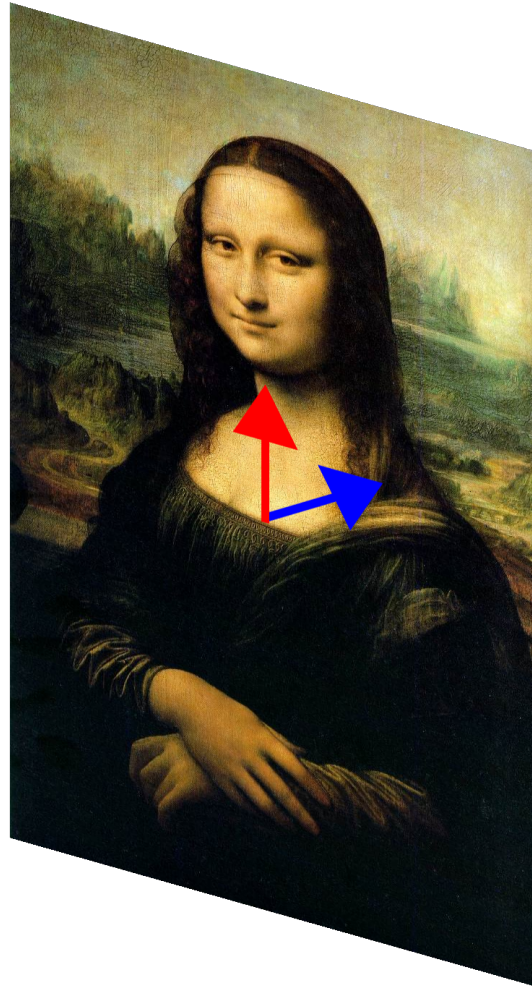
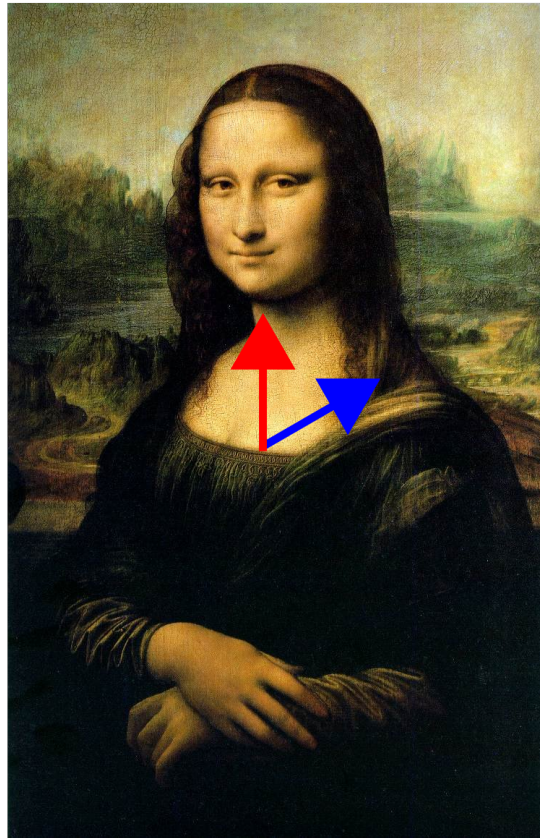
x_1 is a random direction in (t_1, t_2, t_3) space.
 x_2 is chosen randomly but orthogonal to x_1 .

Dot product of x_1 and x_2 is zero.

Guarantee

- With high probability, relative distances are (approximately) preserved by projection
- But: expensive computations

Mapping Data



- Red arrow not changed in shear mapping
- Eigenvector

$$Sv = \lambda v$$

Eigenvalues & Eigenvectors

- **Eigenvectors** (for a square $m \times m$ matrix \mathbf{S})

$$\mathbf{S}\mathbf{v} = \lambda\mathbf{v}$$

(right) eigenvector eigenvalue
 $\mathbf{v} \in \mathbb{R}^m \neq \mathbf{0}$ $\lambda \in \mathbb{R}$

Example

$$\begin{pmatrix} 6 & -2 \\ 4 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \end{pmatrix} = 2 \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

- **How many eigenvalues** are there at most?

$$\mathbf{S}\mathbf{v} = \lambda\mathbf{v} \iff (\mathbf{S} - \lambda\mathbf{I})\mathbf{v} = \mathbf{0}$$

only has a non-zero solution if $|\mathbf{S} - \lambda\mathbf{I}| = 0$

determinant

This is a m -th order equation in λ which can have **at most m distinct solutions** (roots of the characteristic polynomial) – can be complex even though \mathbf{S} is real.

Singular Value Decomposition

For an $m \times n$ matrix \mathbf{A} of rank r there exists a factorization (Singular Value Decomposition = **SVD**) as follows:

$$A = U \Sigma V^T$$

$m \times m$ $m \times n$ V is $n \times n$

The columns of \mathbf{U} are left-singular eigenvectors of $\mathbf{A}\mathbf{A}^T$.

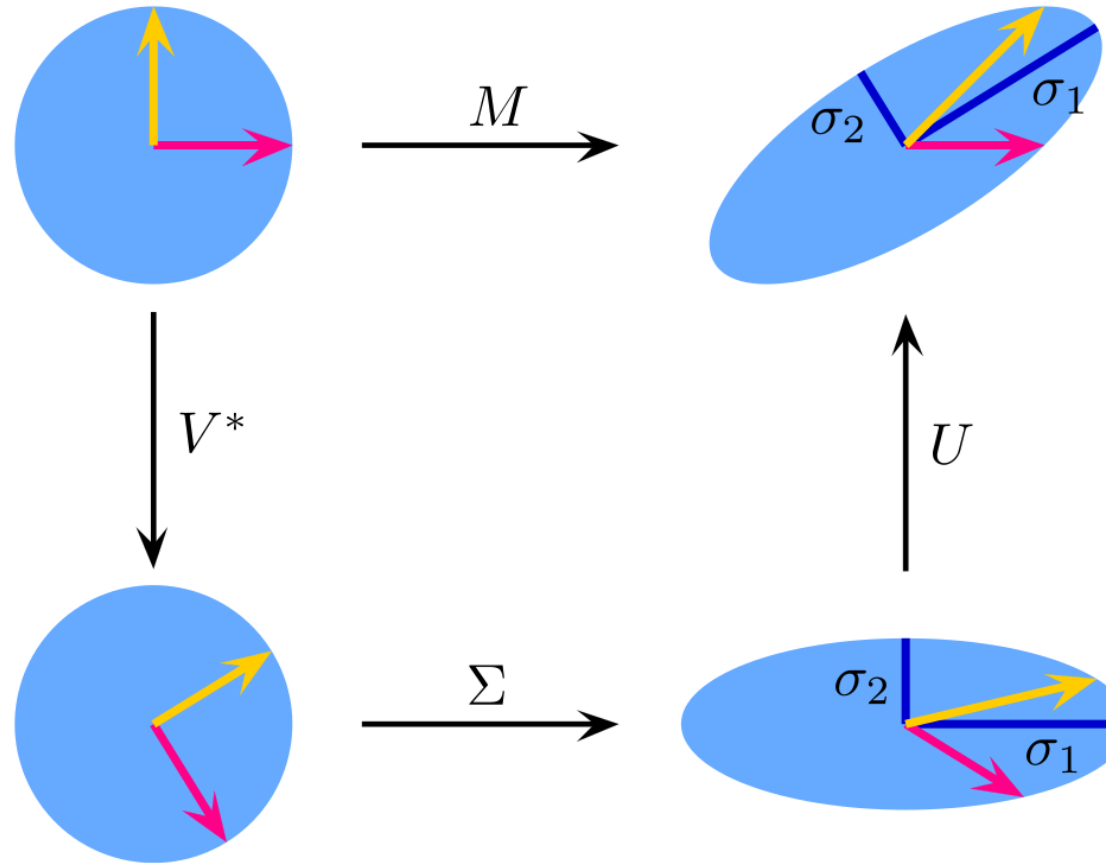
The columns of \mathbf{V} are right-singular eigenvectors of $\mathbf{A}^T\mathbf{A}$.

Eigenvalues $\lambda_1 \dots \lambda_r$ of $\mathbf{A}\mathbf{A}^T$ are the eigenvalues of $\mathbf{A}^T\mathbf{A}$.

$$\sigma_i = \sqrt{\lambda_i}$$
$$\Sigma = \text{diag}(\sigma_1 \dots \sigma_r)$$

Singular values.

Shear Mapping Unit Vectors



$$M = U \cdot \Sigma \cdot V^*$$

Low-rank Approximation

- SVD can be used to compute optimal **low-rank approximations** for a Matrix A of rank r
- Approximation problem: Find A_k of rank k such that

$$A_k = \arg \min_{X: \text{rank}(X)=k} \|A - X\|_F \longleftarrow \text{Frobenius norm}$$

$$\|A\|_F \equiv \sqrt{\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2}.$$

A_k and X are both $m \times n$ matrices

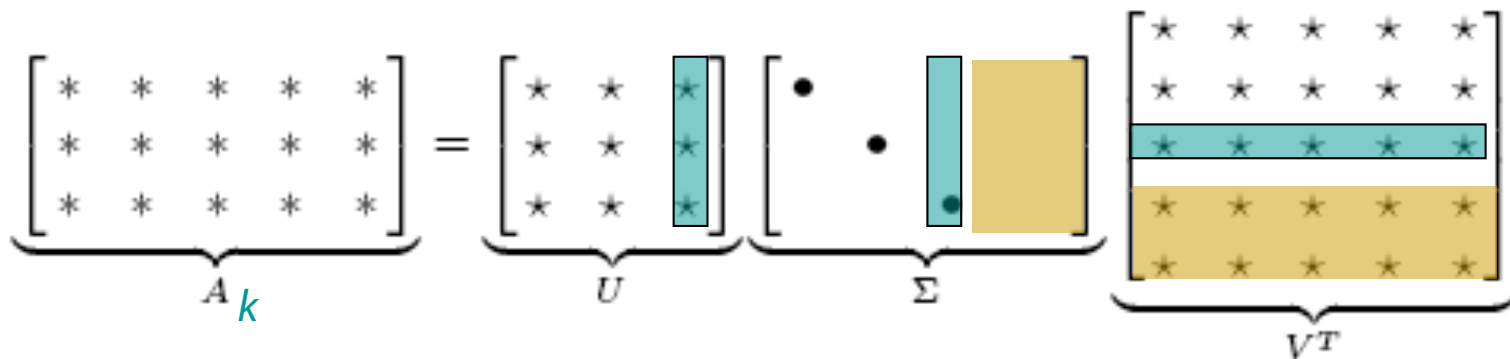
Typically, want $k \ll r$

Low-rank Approximation

- Solution via SVD

$$A_k = U \operatorname{diag}(\sigma_1, \dots, \sigma_k, \underbrace{0, \dots, 0}) V^T$$

set smallest $r-k$
singular values to zero



SVD Low-rank approximation

- A term-doc matrix A may have $m=50000$, $n=10$ million (and rank close to 50000)
- We can construct an approximation A_{100} with rank 100.
 - Of all rank 100 matrices, it would have the lowest Frobenius error.
- Great ... but why would we??
- Answer: **Latent Semantic Indexing**
aka Principle Component Analysis
- **E.g.,** Text to Matrix Generator (TMG) as a **MATLAB[®]** toolbox

How to deal with queries?

- A query q need to be mapped into this space, by
 - Query NOT a sparse vector.

$$q_k = q^T U_k \Sigma_k^{-1} T$$

LSI: Summary

- ▶ Significant reduction of the vector space's dimension
 - ▶ reduced storage requirements
 - ▶ faster processing
 - ▶ noise reduction
- ▶ Semantic clustering
 - ▶ similar topics are mapped to the same dimension
 - ▶ synonymies and polysemies can be handled

Scott Deerwester, Susan Dumais, George Furnas, Thomas Landauer, Richard Harshman: [*Indexing by Latent Semantic Analysis*](#). In: *Journal of the American society for information science*. 1990.

Landauer, Thomas; Foltz, Peter W.; Laham, Darrell. "Introduction to Latent Semantic Analysis". *Discourse Processes*. 25 (2–3): 259–284, 1998.



Application in Computer Vision

Applications in computer vision-

PCA to find patterns-

- 20 face images: NxN size
- One image represented as follows-

$$X = (x_1 \quad x_2 \quad x_3 \quad . \quad . \quad x_{N^2})$$

- Putting all 20 images in 1 big matrix as follows-

$$ImagesMatrix = \begin{pmatrix} ImageVec1 \\ ImageVec2 \\ \cdot \\ \cdot \\ ImageVec20 \end{pmatrix}^T$$

- Performing PCA to find patterns in the face images
- Identifying faces by measuring differences along the new axes (PCs)

Back to IR agents

- Agents make decisions about which documents to select and report to the agents' creators
 - Recommend the k top-ranked documents
- How to evaluate an agent's performance
 - Externally (creator satisfaction)
 - Internally (relevance feedback, reinforcement)

External evaluation of query results

Precision/Recall

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

F-measure

The weighted **harmonic mean** of precision and recall, the traditional F-measure or balanced F-score is:

$$F = 2 \cdot (\text{precision} \cdot \text{recall}) / (\text{precision} + \text{recall}).$$

Unranked retrieval evaluation

- Precision: fraction of retrieved docs that are relevant = $P(\text{retr}\&\text{rel}|\text{retrieved})$
- Recall: fraction of relevant docs that are retrieved = $P(\text{retr}\&\text{rel}|\text{relevant in repos})$

	Relevant	Not Relevant
Retrieved	true positives (tp)	false positives (fp)
Not Retrieved	false negatives (fn)	true negatives (tn)

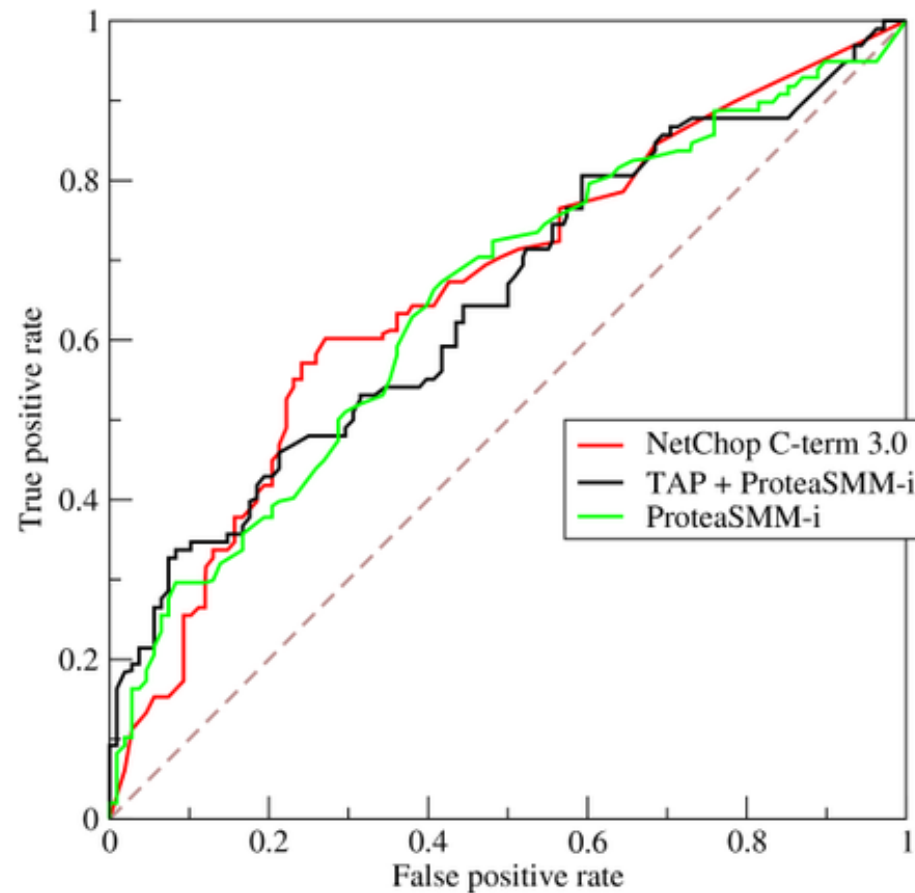
- Precision: $P = \text{tp}/(\text{tp} + \text{fp})$
- Recall: $R = \text{tp}/(\text{tp} + \text{fn})$

Overview on evaluation measures

		True condition			
		Condition positive	Condition negative		
Total population				Prevalence = $\frac{\Sigma \text{Condition positive}}{\Sigma \text{Total population}}$	
Predicted condition	Predicted condition positive	True positive	False positive (Type I error)	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{True positive}}{\Sigma \text{Test outcome positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{False positive}}{\Sigma \text{Test outcome positive}}$
	Predicted condition negative	False negative (Type II error)	True negative	False omission rate (FOR) = $\frac{\Sigma \text{False negative}}{\Sigma \text{Test outcome negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{True negative}}{\Sigma \text{Test outcome negative}}$
Accuracy (ACC) = $\frac{\Sigma \text{True positive} + \Sigma \text{True negative}}{\Sigma \text{Total population}}$		True positive rate (TPR), Sensitivity, Recall = $\frac{\Sigma \text{True positive}}{\Sigma \text{Condition positive}}$	False positive rate (FPR), Fall-out = $\frac{\Sigma \text{False positive}}{\Sigma \text{Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR-}}$
		False negative rate (FNR), Miss rate = $\frac{\Sigma \text{False negative}}{\Sigma \text{Condition positive}}$	True negative rate (TNR), Specificity (SPC) = $\frac{\Sigma \text{True negative}}{\Sigma \text{Condition negative}}$	Negative likelihood ratio (LR-) = $\frac{\text{FNR}}{\text{TNR}}$	

Relative operating characteristic (ROC)

- Investigate effects of parameter adjustments
- Compare TP rate and FP rate
- Example w/ three classifiers
- Measure:
Area under curve
(AUC)



Back to IR agents

- Still need
 - Test queries
 - Relevance assessments
- Test queries
 - Must be adequate to docs available
 - Best designed by domain experts
 - Random query terms generally not a good idea
- Relevance assessments?
 - Consider classification results of other agents
 - Need a measure to compare different „judges“

Web Mining Agents

- Mining in complex networks requires the management of
 - Distributed work (problem decomposition)
 - Autonomous work (no central control, proactive agents)
 - Collaboration between agents (solution sharing, "collective intelligence in the small")
 - Feedback and adaptation (learning by reinforcement)
- Considered here: IR Agents ("Semantic Computation")

Weiss, G. "A Multiagent Perspective of Parallel and Distributed Machine Learning". *Agents 98*: 226–230, **1998**.

Klusch, M.; Lodi, S.; Moro, G.. "Agent-Based Distributed Data Mining". *LNCS 2586*: 104–122, **2003**.

Cao, Longbing; Gorodetsky, Vladimir; Mitkas, Pericles A. "Agent Mining: The Synergy of Agents and Data Mining,". *IEEE Intelligent Systems*. **24** (3): 64–72, **2009**.

Cao, Longbing; Weiss, Gerhard; Yu, Philip. "A Brief Introduction to Agent Mining". *Journal of Autonomous Agents and Multi-Agent Systems*. **25**: 419–424, **2012**.

Collaboratoin: Measure for inter-judge (dis)agreement

- Kappa measure
 - Agreement measure among judges
 - Designed for categorical judgments
 - Corrects for chance agreement
- $\kappa = [P(A) - P(E)] / [1 - P(E)]$
- $P(A)$ – proportion of time judges agree (observed)
- $P(E)$ – what agreement would be by chance (hypothetical)
- $\kappa = 0$ for chance agreement, 1 for total agreement
- In statistics many other measures are defined

Kappa Measure: Example

$P(A)? P(E)?$

Number of docs	Judge 1	Judge 2
300	Relevant	Relevant
70	Nonrelevant	Nonrelevant
20	Relevant	Nonrelevant
10	Nonrelevant	Relevant

Kappa Example

- $P(A) = 370/400 = 0.925$
- $P(\text{nonrelevant}) = (10+20+70+70)/800 = 0.2125$
- $P(\text{relevant}) = (10+20+300+300)/800 = 0.7878$
- $P(E) = 0.2125^2 + 0.7878^2 = 0.665$
- $\kappa = (0.925 - 0.665)/(1-0.665) = 0.776$

- $\kappa > 0.8 =$ good agreement
- $0.67 < \kappa < 0.8 \rightarrow$ “tentative conclusions”
- Depends on purpose of study
- For >2 judges: average pairwise κ s

Confusion Matrix

In the example confusion matrix below, of the 8 actual cats, the system predicted that three were dogs, and of the six dogs, it predicted that one was a rabbit and two were cats. We can see from the matrix that the system in question has trouble distinguishing between cats and dogs, but can make the distinction between rabbits and other types of animals pretty well.

Example confusion matrix

	Cat	Dog	Rabbit
Cat	5	3	0
Dog	2	3	1
Rabbit	0	2	11

Understanding where an agent has deficiencies

(Direct) feedback:

Present confusion matrix to an agent

Reinforcement:

[Relevance feedback](#) for retrieval results

(agent might build confusion matrix internally)

Relevance Feedback: Rocchio Algorithm

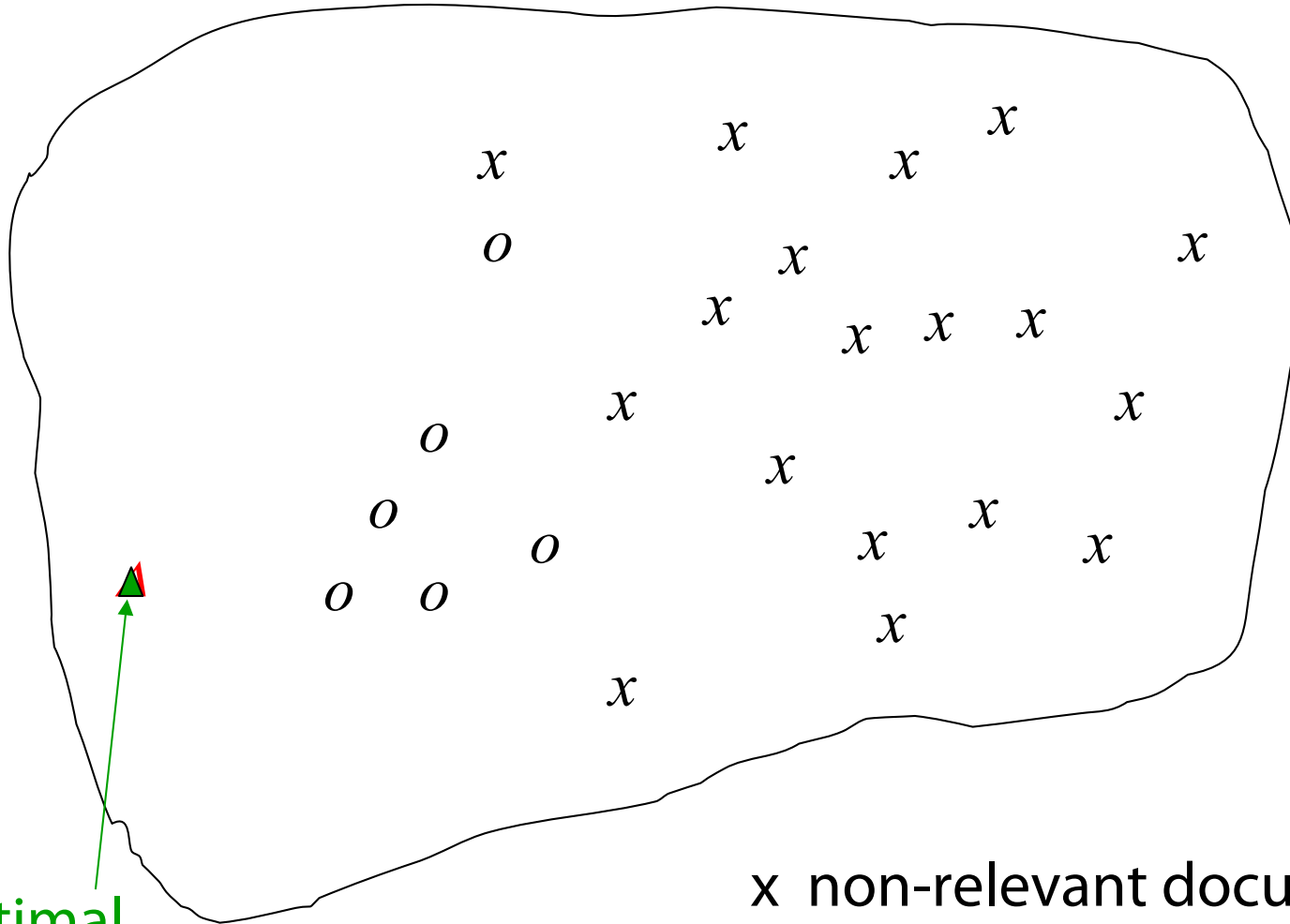
- The Rocchio algorithm incorporates relevance feedback information into the vector space model.
- Want to **maximize** $\text{sim}(Q, C_r) - \text{sim}(Q, C_{nr})$ where C_r and C_{nr} denote relevant and non-relevant doc vectors, respectively
- The **optimal** query vector for **separating** relevant and non-relevant documents (with cosine sim.):

$$\vec{Q}_{opt} = \frac{1}{|C_r|} \sum_{\vec{d}_j \in C_r} \vec{d}_j - \frac{1}{N - |C_r|} \sum_{\vec{d}_j \notin C_r} \vec{d}_j$$

Q_{opt} = optimal query; C_r = set of rel. doc vectors in corpus; N = collection size

- Unrealistic definition:
We don't know relevant documents in corpus

The Theoretically Best Query



Optimal query

x non-relevant documents
o relevant documents

Rocchio 1971 Algorithm (SMART System)

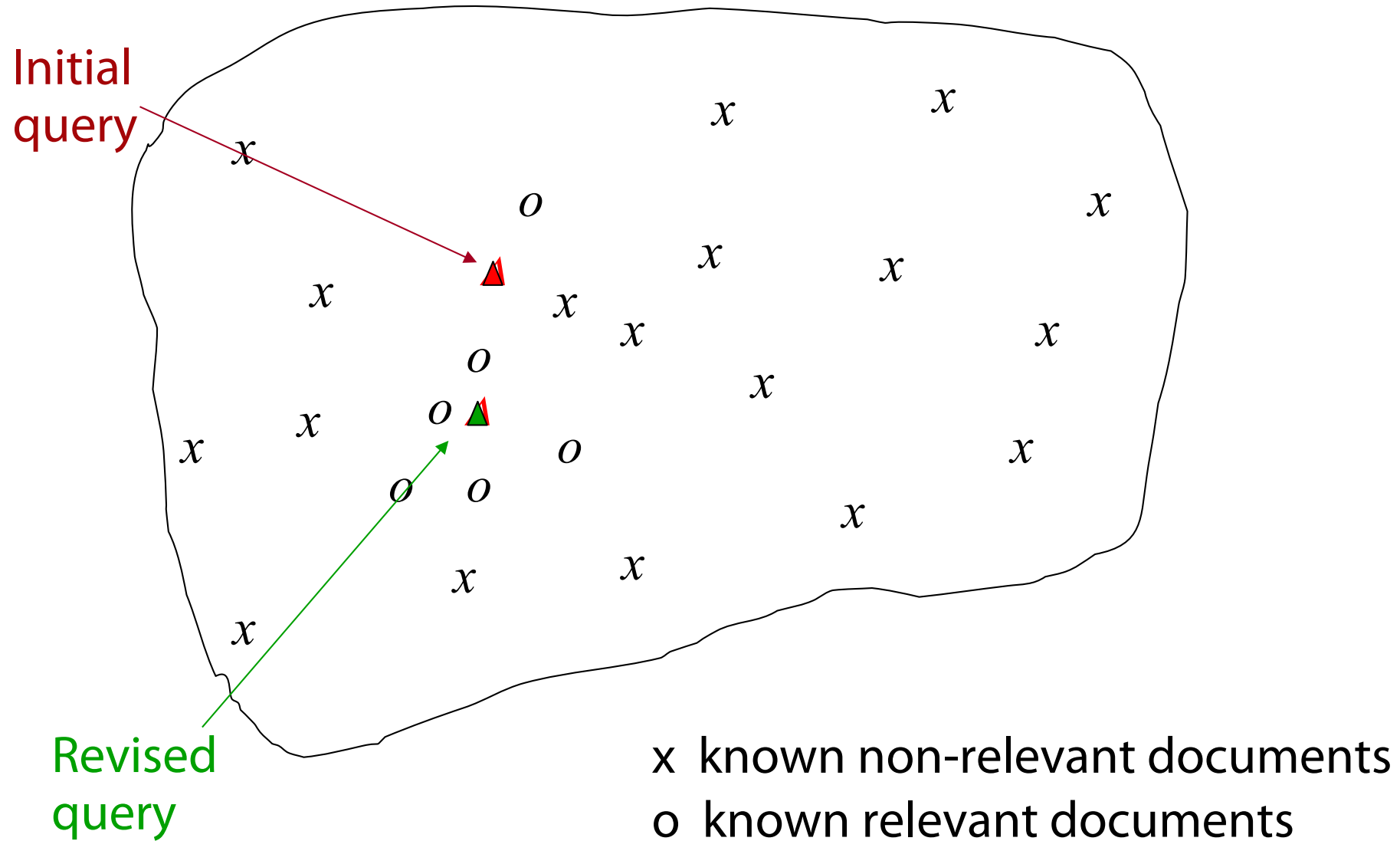
- Used in practice:

$$\vec{q}_m = \alpha \vec{q}_0 + \beta \frac{1}{|D_r|} \sum_{\vec{d}_j \in D_r} \vec{d}_j - \gamma \frac{1}{|D_{nr}|} \sum_{\vec{d}_j \in D_{nr}} \vec{d}_j$$

- q_m = modified query vector; q_0 = original query vector; α, β, γ : weights (hand-chosen or set empirically); D_r = set of known relevant doc vectors; D_{nr} = set of known irrelevant doc vectors
- New query moves toward relevant documents and away from irrelevant documents
- Tradeoff α vs. β/γ : If we have a lot of judged documents, we want a higher β/γ .
- Term weight can go negative
 - Negative term weights are ignored (set to 0)

Wikipedia: Gerard Salton, The **SMART** (System for the **M**echanical **A**nalysis and **R**etrieval of **T**ext or Salton's Magic Automatic Retriever of Text) Information Retrieval System is an information retrieval system, developed at Cornell University in the **1960s**. Many important concepts in information retrieval were developed as part of research on the SMART system, including the vector space model, relevance feedback, and Rocchio algorithm.

Relevance feedback on initial query

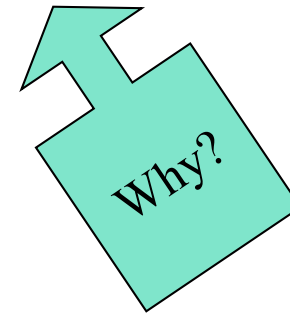


Relevance Feedback in vector spaces

- We can **modify the query** based on relevance feedback and apply standard vector space model.
- Use only the docs that were marked.
- Relevance feedback can **improve recall and precision**

Positive vs Negative Feedback

Positive feedback is more valuable than negative feedback (so, set $\gamma < \beta$; e.g. $\gamma = 0.25$, $\beta = 0.75$).



Many systems only allow positive feedback ($\gamma=0$).

Multimodal information retrieval

- What about images, videos, audio data?
- Compute feature vectors from data representations in a data-driven fashion
- Which features?
 - Example: MPEG-7 General information descriptors
- Define respective vector spaces
- Use vector space retrieval model with cosine similarity
- Texts, images, videos, audio data are called **documents**