Kapitel 2: Suchmaschinentechnologie für Intranets und das Web

2.1 Information-Retrieval-Systeme
2.2 Web-Crawling und Indexierung
2.3 Vektorraummodell für IR mit Ranking
2.4 Anfrageausführung mit Ranking
2.5 Grundlagen aus der Linearen Algebra
2.6 Latent Semantic Indexing

2.1 Information-Retrieval-Systeme

Information Retrieval (IR) ist die Technologie zum Suchen
in Kollektionen (Korpora, Intranets, Web)
schwach strukturierter Dokumente: Text, HTML, XML, ...

Darunter fällt auch:

- Text- und Strukturanalyse
- Inhaltserschließung und -repräsentation
- Gruppierung und Klassifikation
- Zusammenfassung
- Filtern und Personalisieren (z.B. von Nachrichten-“Feeds“)
- „Routing“ (Metasuche)

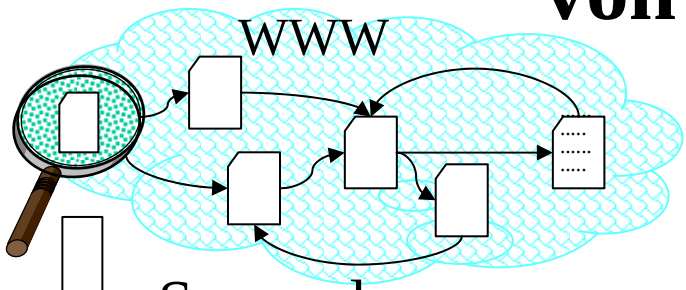
Globales Ziel:
Informationsbedürfnisse befriedigen - und dabei
Beseitigung des Engpasses (teurer) intellektueller Zeit !

Schnittstellen von IR-Systemen

- **Ausgabe:**
  - Menge von Dokumenten, die Suchstring(s) enthalten: **Freitextsuche**
  - Menge inhaltlich relevanter Dokumente: **Inhaltssuche**
    - ungeordnete Menge: **Boolesches Retrieval**
  - nach Relevanz absteigend sortierte Rangliste: **Ranked Retrieval** (Ähnlichkeitssuche)

- **Eingabe:**
  - Keywords (positiv/negativ) (plus Phrasen, ganze Sätze)
  - (Boolesche) Ausdrücke über Keyword-Bedingungen
  - Strukturbedingungen (z.B. Tags, Links)
  - ontologisch basierte Bedingungen
  - Suchsprache (z.B. SQL mit Oracle/Text, XPath Full-Text)

Sammeln, Analyse und Indexierung von Web-Dokumenten



Sammeln (Crawling)

Web Surfing:
In Saarbrücker Internet-Cafes mit oder ohne Web Suit ...

Dokumente

Surfing
Saarbrücker
Internet
Cafes
...

Extraktion relevanter Wörter

Surf
Saarbrücken
Internet
Cafe
...

Linguist. Methoden: Reduktion auf Stamm-formen

Surf
Wellenritt
Internet
WWW
eService
Cafe
Bistro
...

Bildung von Features (Deskriptoren, Terme) mit Gewichten

Index-aufbau

Thesaurus (Ontologie)

Synonyme, Hypernyme, Hyponyme

Index (B*-Baum)

Bistro | Cafe | ...

URLs

Problem: Inhaltserschließung

Umgang mit „unscharfen“ Daten (und „unscharfen“ Anfragen)

→ Dokumente werden typischerweise durch **Features** charakterisiert, z.B.:

- Wörter, Wortpaare oder Phrasen
- Worthäufigkeiten
- Anzahl eingehender Hyperlinks
- title, weitere Tags, Struktur von HTML- oder XML-Seiten
- Farbhäufigkeiten in Bildern (Bildmitte, oberer Rand, etc.)
- usw. usw.

→ Abbildung von natürlichsprachlichem Text auf Features:

- Behandlung von morphologischer Variation
- Behandlung von Synonymen, Hypernymen/Hyponymen und Polysemen (u.a. mittels Thesaurus)

Problem: Effektivität (Suchresultatsgüte)

query = „Chernoff theorem“

AltaVista: Fermat's last theorem. Previous topic. Next topic. ... URL: www-groups.dcs.st-and.ac.uk/~history/His...st_theorem.html

Northernlight: J. D. Biggins- Publications. Articles on the Branching Random Walk http:/ / www.shef.ac.uk/ ~st1jdb/ bibliog.html

Excite: The Official Web Site of Playboy Lingerie Model Mikki Chernoff http://www.mikkichernoff.com/

Google: ...strong convergence \cite{Chernoff}. \begin{theorem} \label{T1} Let... http://mpej.unige.ch/mp_arc/p/00-277

Yahoo: Moment-generating Functions; Chernoff's Theorem; http://www.siam.org/catalog/mcc10/bahadur.htm

Mathsearch: No matches found.

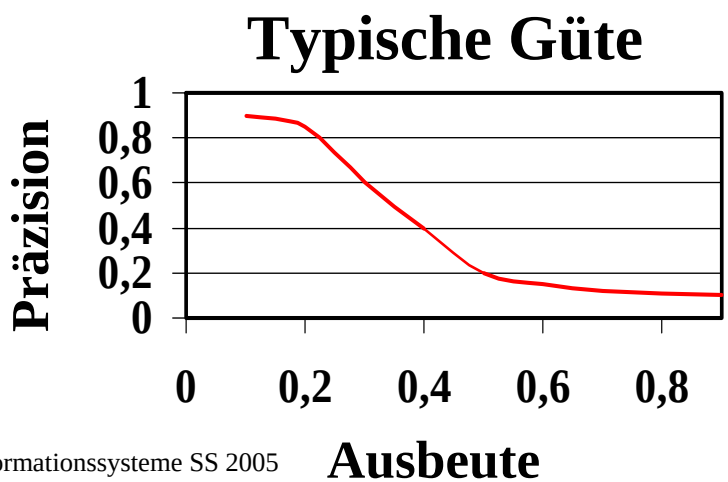
Bewertung der Retrieval-Güte (Effektivität)

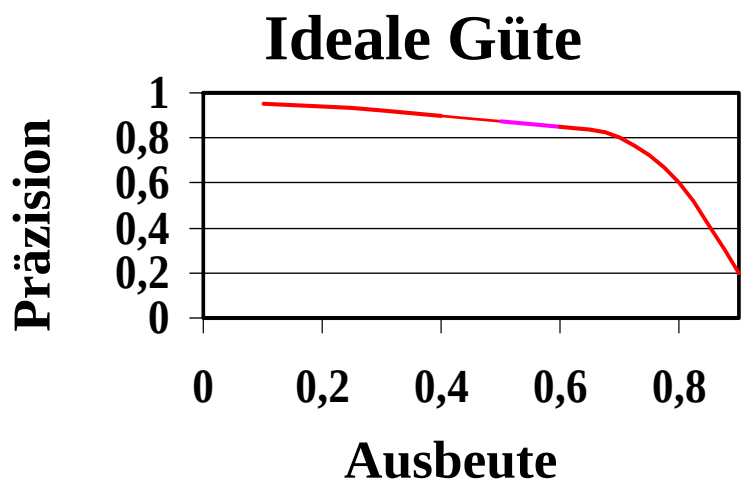
Fähigkeit, zu einer Anfrage **nur** relevante Dokumente zu liefern:

$$\textbf{\textit{Präzision (precision)}} = \frac{Anzahl\ relevanter\ Dokumente\ unter\ Top\ r}{r}$$

Fähigkeit, zu einer Anfrage **alle** relevante Dokumente zu liefern:

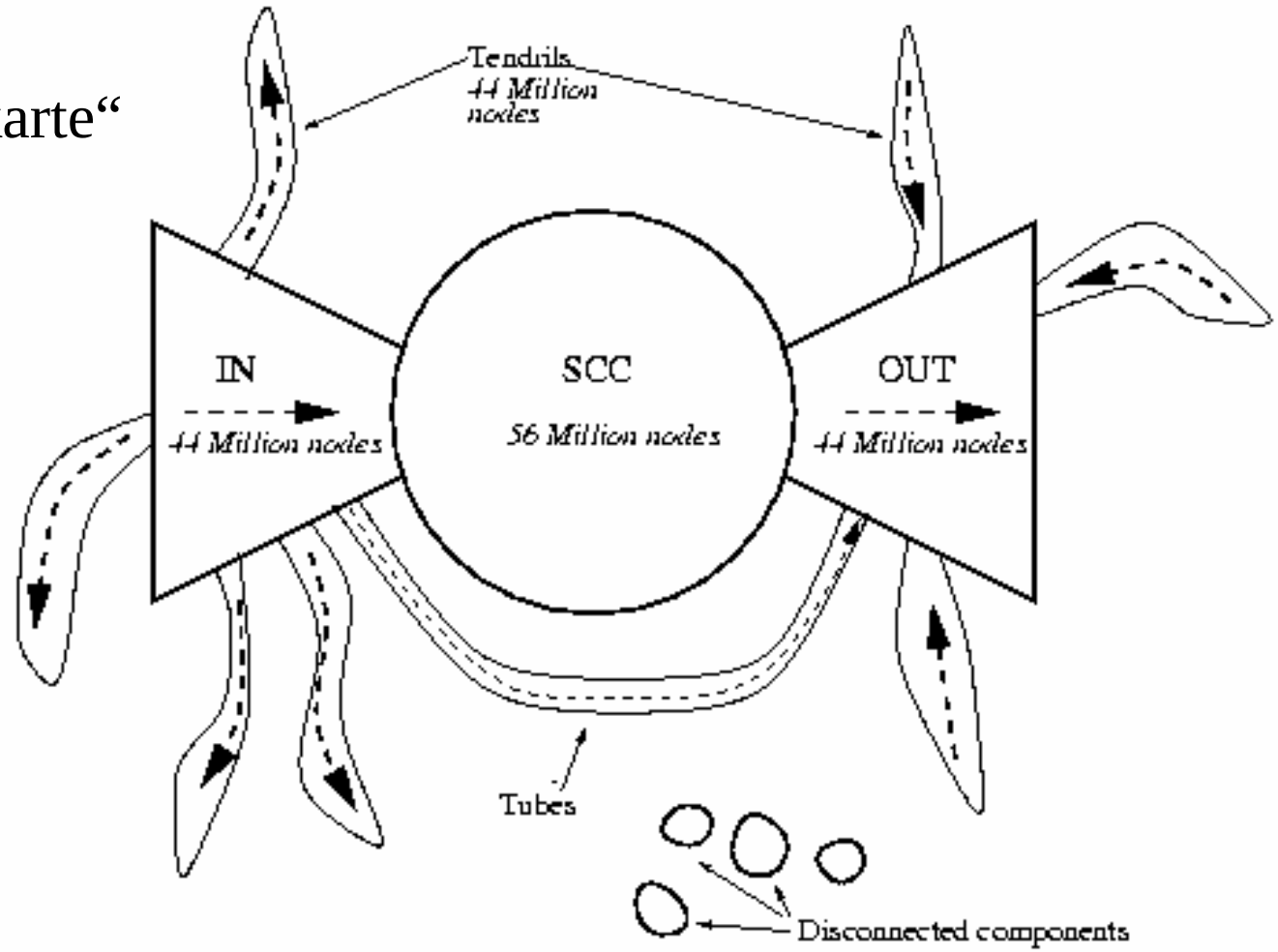
$$\textbf{\textit{Ausbeute (recall)}} = \frac{Anzahl\ relevanter\ Dokumente}{Anzahl\ aller\ relevanten\ Dokumente}$$





Problem: Effizienz und Skalierbarkeit

Aktuelle „Landkarte“ des Webs:



!!! Aber:

Suchmaschinen überdecken das „Surface Web“:
8 Mrd. Dokumente, 40 TBytes
Die meisten Daten sind im „Deep Web“ hinter Portalen:
> 500 Mrd. „Dokumente“, > 8 PBytes

Dimensionen sehr großer Web-Suchmaschinen

- > 8 Mrd. Web-Dokumente + 1 Mrd. News-Dokumente
  > 40 Terabytes Rohdaten
- > 10 Mio. Terme
  > 8 Terabytes Index
- > 150 Mio. Anfragen pro Werktag
  < 1 Sek. mittlere Antwortzeit
- < 30 Tage Indexaktualität
  > 1000 Webseiten pro Sek. Crawling

High-End-Server-Farm:
> 10 000 Intel-Server mit jeweils
> 1 GB Hauptspeicher, 2 Platten und
partitionierten, gespiegelten Daten, die über alle Server verteilt sind, sowie Lastbalancierung der Queries, Remote-Administration, usw.

2.2 Web-Crawling und Indexierung

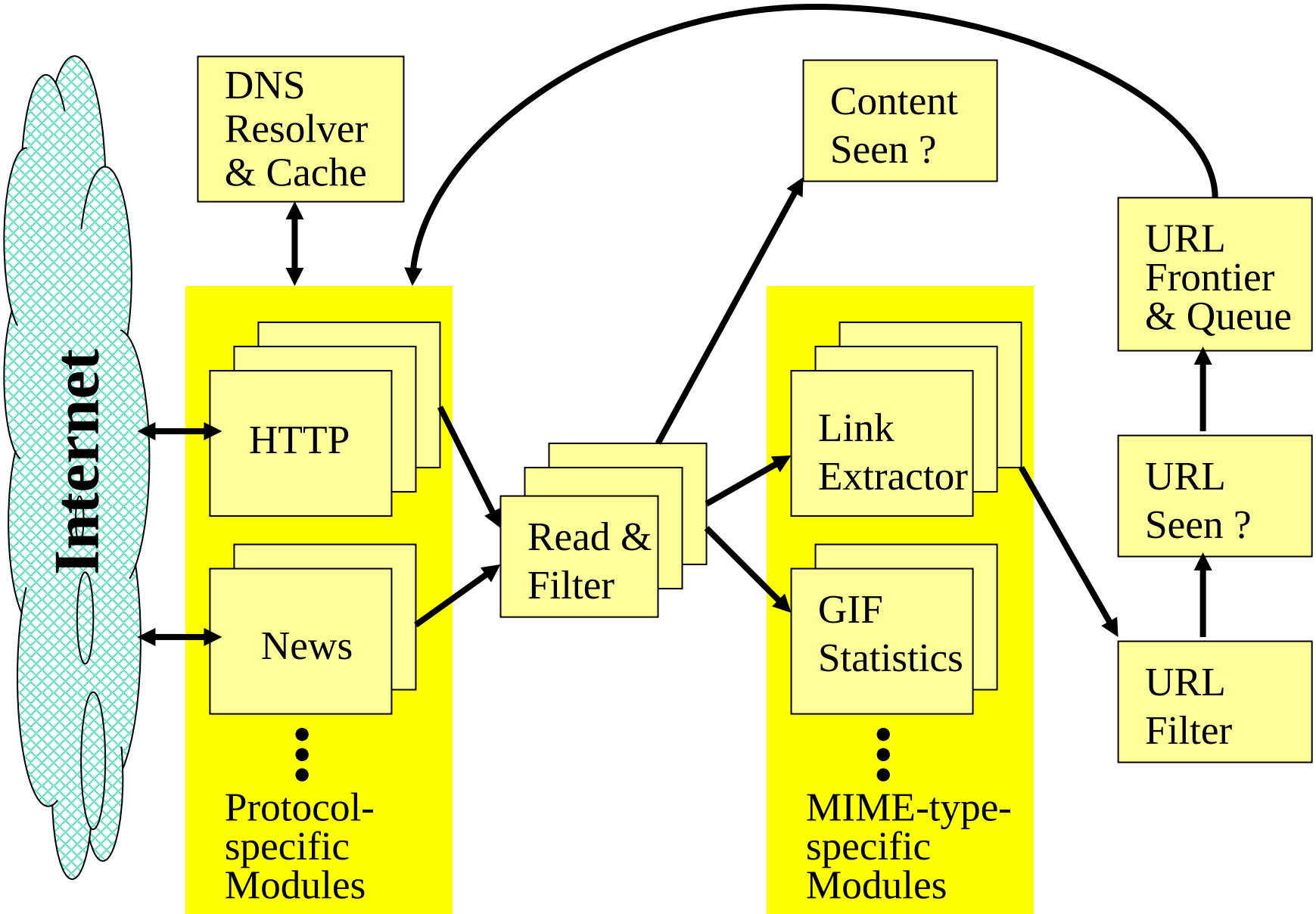
Komponenten einer Web-Suchmaschine:

- **URL Queue Server**: verwaltet Priority-Queue (noch) zu traversierender Links
- **Crawler (Robot, Spider):** holt Dokumente unter Beachtung von Nebenbedingungen (Filetyp, Robot Exclusion Protocol, usw.)
- **Repository Server:** verwaltet DocumentRepository
- **Indexer (inkl. Parser, Stemmer):** analysiert Dokumente und erzeugt Einträge in Lexicon, Anchors und DocumentIndex
- **URL Resolver:** übersetzt URLs in DocIds
- **Link Analyzer:** berechnet Autoritäts-Ranking aufgrund von Links
- **Query Processor:** wertet Anfragen durch Index-Lookups aus und berechnet Resultats-Ranking

Datenstrukturen einer Web-Suchmaschine

- **DocumentRepository *(DocId, DocContent):***
  alle (HTML-) Dokumente in komprimierter Form
- **Lexicon *(TermId, Term):***
  alle vorkommenden Stammformen jeweils mit TermId
- **DocumentIndex *(DocId, TermId, Weight, ...):***
  alle Vorkommen von Termen in Dokumenten,
  optimiert für Zugriff nach DocId
- **TermIndex *(TermId, DocId, Weight, ...):***
  alle Dokumente zu allen Termen,
  optimiert für Zugriff nach TermId
- **Anchors *(SourceDocId, TargetDocId, AnchorText):***
  alle Hyperlinks
- **URLIndex *(URL, DocId):***
  Umsetzung von URLs auf interne Ids

Architektur eines skalierbaren Crawlers



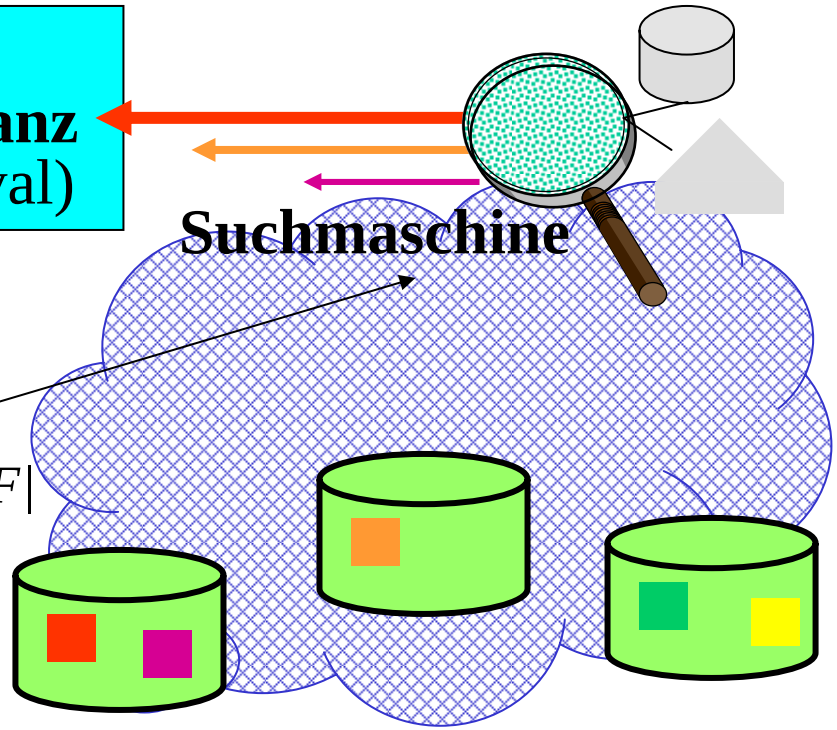
2.3 Vektorraummodell für IR mit Ranking

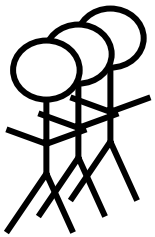
Grundprinzipien:

- **Featureraum**: Wörter in Dokumenten werden auf Terme reduziert.
- **Dokumentenmodell:** Jedes Dokument di wird als Vektor $\in [0,1]^{|F|}$ repräsentiert, wobei dij das Gewicht des j-ten Terms in di angibt.
- **Anfragemodell:** Anfragen sind Vektoren $q \in [0,1]^{|F|}$
- **Relevanz:** Suchresultatsranking basiert auf einer Ähnlichkeitsfunktion im Vektorraum $[0,1]^{|F|}$
- **Crawling:** Das Web wird entlang von Hyperlinks traversiert, um Dokumente zu analyiseren und zu indexieren.
- **Indexierung:** Zu jedem Term wird eine Liste von Dokumenten-Ids (z.B. URLs) mit dem jeweiligen Gewicht in einem „invertierten File“ (Suchbaum oder Hash-File) angelegt.
- **Anfrageverarbeitung:** Anfragen werden zerlegt in Index-Lookups für Einzelterme, um Trefferkandidatenlisten zu bestimmen.

Vektorraummodell für Relevanz-Ranking

Rangliste, nach fallender **Relevanz** (Ranked Retrieval)





Query $q \in [0,1]^{|F|}$ (Menge gewichteter Such-Features)

Ähnlichkeitsfunktion

$sim(d_i, q)$

→ Scores (RSVs = Retrieval Status Values)

Dokumente sind **Feature-Vektoren**

$d_i \in [0,1]^{|F|}$

Verwendete Ähnlichkeitsfunktionen sind z.B.:

$$sim(d_i, q) := \sum_{j=1}^{|F|} d_{ij}\, q_j$$

(Skalarprodukt)

$$sim(d_i, q) := \frac{\sum_{j=1}^{|F|} d_{ij}\, q_j}{\sqrt{\sum_{j=1}^{|F|} d_{ij}^2 \sum_{j=1}^{|F|} q_j^2}}$$

(Cosinus-Maß)

Termgewichtung in Dokumenten

Betrachtet werden die folgenden Werte
(für N Dokumente und M Terme):

- tf_{ij}: Häufigkeit (term frequency) des Terms ti in Dokument dj
- df_i: Anzahl der Dokumente mit Term ti (doc. frequency)
- idf_i: N / df_i (inverse document frequency)
- cf_i: Häufigkeit von ti in allen Dokumenten (corpus frequency)

(ggf. mit separater Berücksichtigung von Termen in title u.ä.)

Grundprinzip:

Das Gewicht w_{ij} von Term ti in Dokument dj sollte
mit tf_{ij} und mit idf_i monoton wachsen.

$\rightarrow$ erster Ansatz: wij = tf_{ij} * idf_i **(tf-idf-Formel)**

Ggf. sollten die Gewichte w_{ij} wie folgt zu ω_{ij} normiert werden:

$$\omega_{ij} := w_{ij} / \sqrt{\sum_k w_{kj}^2}$$

Variationen der Termgewichtung mit tf und idf

Empirische Resultate zeigen, daß in der Regel die
tf- und idf-Werte normalisiert und/oder gedämpft sein sollten.

Normalisierung tf-Werte: $tf_{ij} := \frac{tf_{ij}}{\max_k tf_{kj}}$

Gedämpfte tf-Werte: $tf_{ij} := (1 + \log tf_{ij})$

Gedämpfte idf-Werte: $idf_i := \log \frac{N}{df_i}$

Häufige Variante:

$$w_{ij} := \frac{tf_{ij}}{\max_k tf_{kj}} \log \frac{N}{df_i}$$

$$\omega_{ij} := w_{ij} / \sqrt{\sum_k w_{kj}^2}$$

tf*idf-Formel

Query-Verfeinerung mit Relevanz-Feedback nach Rocchio

Für Resultat D der Query q bestehe das Relevanz-Feedback des Benutzers aus einer Partitionierung von D in

- D^+: die Menge der relevanten Dokumente in D und
- D^-: die Menge der nicht relevanten Dokumente in D.

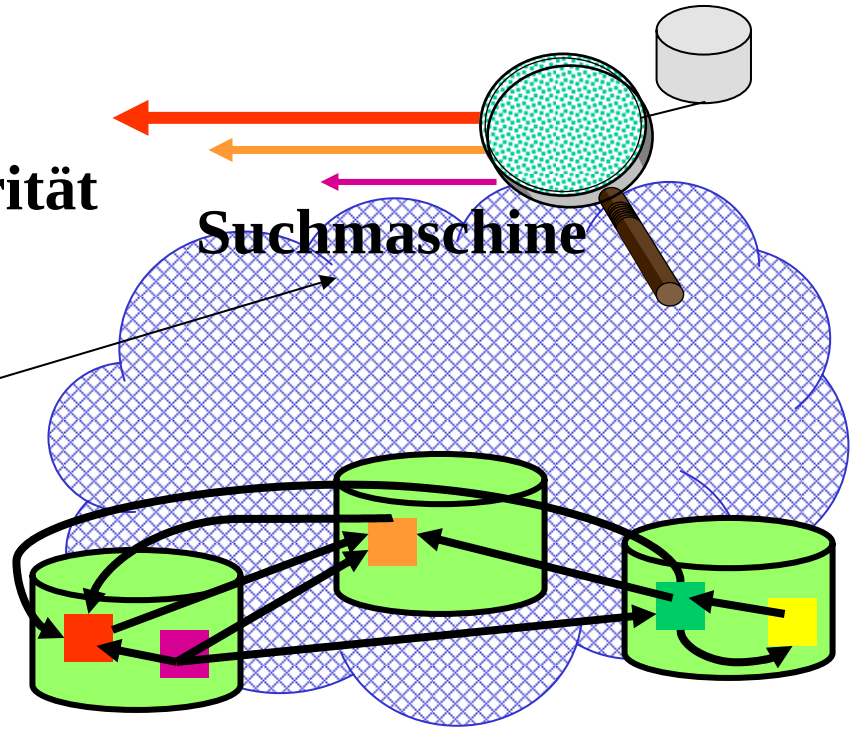
Generiere verfeinerte Query q‘:

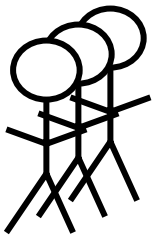
$$\vec{q}' := \alpha\vec{q} + \frac{\beta}{\left|D^+\right|} \sum_{d_j \in D^+} \vec{d}_j - \frac{\gamma}{\left|D^-\right|} \sum_{d_j \in D^-} \vec{d}_j$$

mit geeigneten Gewichten $\alpha, \beta, \gamma \in [0,1]$ (typischerweise $\alpha > \beta > \gamma$)

Linkanalyse für Autoritäts-Ranking

Rangliste nach fallender **Relevanz & Autorität**





Query $q \in [0,1]^{|F|}$ (Menge gewichteter Such-Features)

+ Berücksichtige Fanin und Fanout von Web-Seiten:

Autoritätsrang (d_i) :=

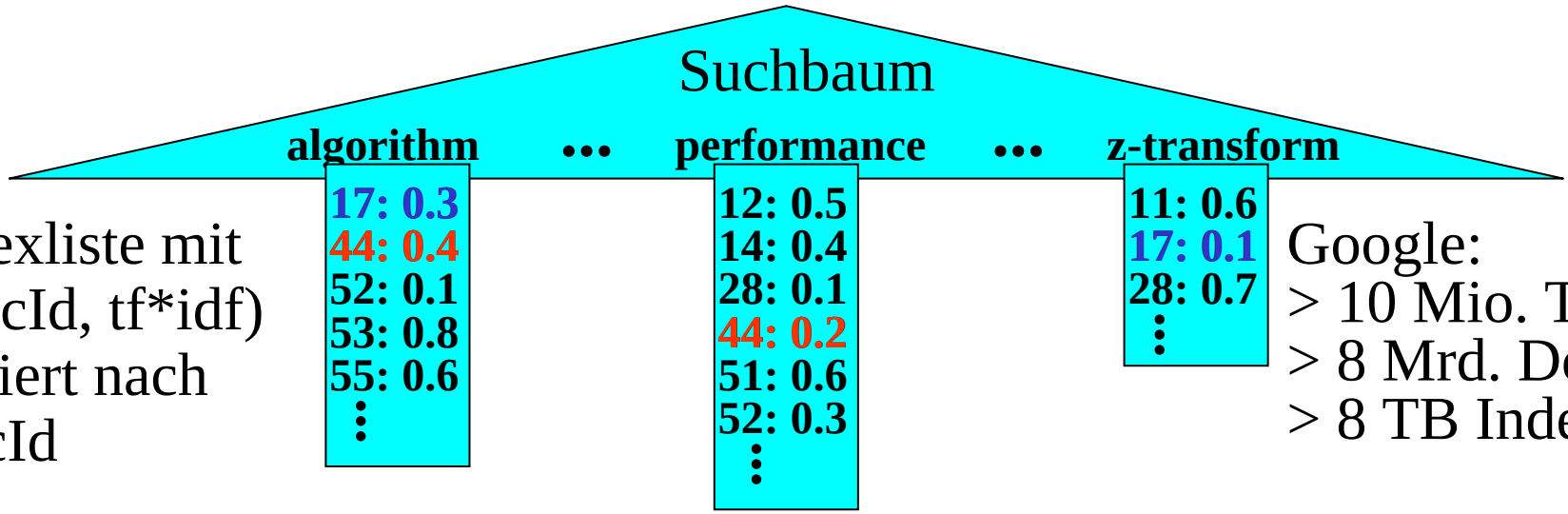
Stationäre Besuchswahrscheinlichkeit P [d_i] bei Random Walk auf dem Web

„Integration“ von Relevanz- und Autoritätsmaßen durch (ad hoc) gewichtete Linearkombination

Weitere IR-Modelle

- **Probabilistisches Retrieval & Statistische Sprachmodelle:**
  Ranking aufgrund von Relevanzwahrscheinlichkeiten,
  die aus - geschätzten - Basisparametern abgeleitet werden.
- **Fuzzy-Set-Modell:**
  Queries (inkl. einzelner Terme) beschreiben Fuzzy-Mengen
  mit Dokumenten als Elementen vom Grad $\mu \in [0,1]$.
  Mengenoperationen verwenden Funktionen max, min, $1-\mu$.
- **Latent Semantic Indexing:**
  Berücksichtigung von Termkorrelationen durch
  Transformation des Term-Vektorraums in einen
  Themen-Vektorraum niedrigerer Dimensionalität
- **Neuronale Netze** und andere Inferenznetze
  zum Lernen von Termgewichten

2.4 Anfrageausführung mit Ranking



Indexliste mit (DocId, tf*idf) sortiert nach DocId

Google:
> 10 Mio. Terme
> 8 Mrd. Dok.
> 8 TB Index

Gegeben: Query $q = t_1\ t_2\ ...\ t_z$ mit z Keywords
Ähnlichkeitsfunktion score(q,d) für Dok. $d \in D$, z.B.: $\vec{q} \cdot \vec{d}$

Finde: Top-k-Resultate bzgl. score(q,d) (z.B.: $\Sigma_{i \in q}\ s_i(d)$)

Naiver QP Algorithmus:

```
candidate-docs := ∅;
for i=1 to z do {
    candidate-docs := candidate-docs ∪ index-lookup(ti) };
for each dj ∈ candidate-docs do {compute score(q,dj)};
sort candidate-docs by score(q,dj) descending;
```

Top-k-Anfragen über m-dimensionalen Datenräumen

Für ***lokale Scores*** s_i für Query q, Datenobjekt d und Dimension i

berechne ***globale Scores s*** der Form $s(q,d) = aggr\{ s_i(q,d) | i = 1..m \}$

mit ***monotoner*** Aggregationsfunktion $aggr : [0,1]^m \rightarrow [0,1]$

Beispiele: $s(q,d) = \sum_{i=1}^{m} s_i(q,d)$ $s(q,d) = max\{ s_i(q,d) | i = 1..m \}$

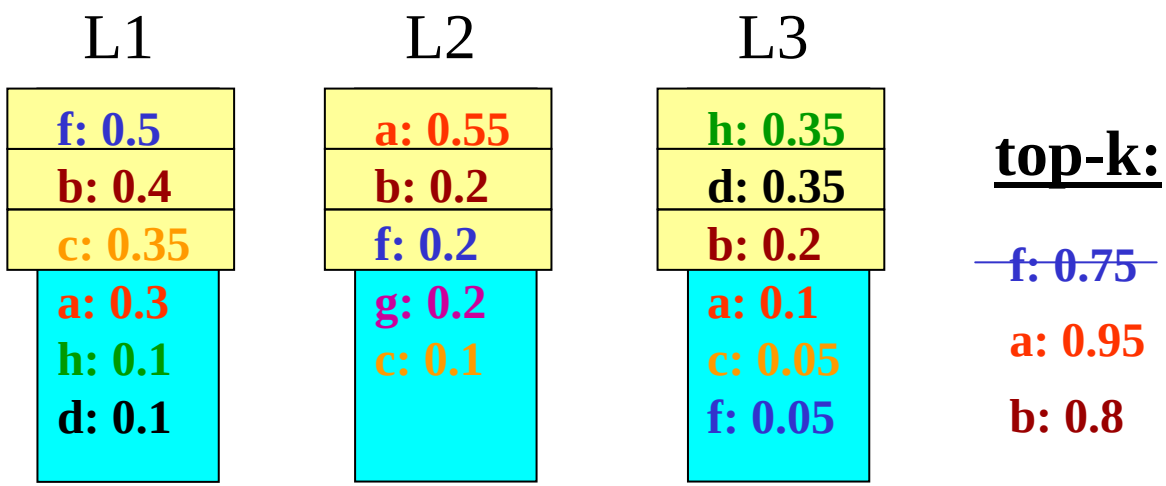
Finde die besten k Datenobjekte (Top-k) bzgl. globaler Scores:

- Traversiere m ***Indexlisten*** Li per ***sortiertem Zugriff (sorted access)*** auf Einträge (d, s_i(q,d)) in absteigender Sortierung von s_i(q,d)
- Verwalte für Kandidaten d eine Menge E(d) evaluierter Dimensionen und einen „Akkumulator“ für den Gesamt-Score
- Für Kandidaten d mit unvollständigem E(d) erwäge Nachschlagen von d in Li für alle $i \in R(d)$ per ***wahlfreiem Zugriff*** (***random access)***
- Terminiere Scans der Indexlisten, wenn genügend Kandidaten gesehen
- Falls nötig, sortiere endgültige Kandidatenliste nach globalem Score

TA: Threshold Algorithm (Fagin; Güntzer et al.)

scan all lists L_i (i=1..m) in parallel:
consider dj at position pos_i in Li;
$high_i := s_i(dj)$;
if dj $\notin$ top-k then {
look up $s_\nu(dj)$ in all lists L_ν with $\nu \neq i$; // random access
compute $s(dj) := aggr \{s_\nu(dj) \mid \nu=1..m\}$;
if s(dj) > min score among top-k then
add dj to top-k and remove min-score d from top-k; };
if min score among top-k $\geq$ aggr $\{high_\nu \mid \nu=1..m\}$ then exit;

m=3
aggr: sum
k=2



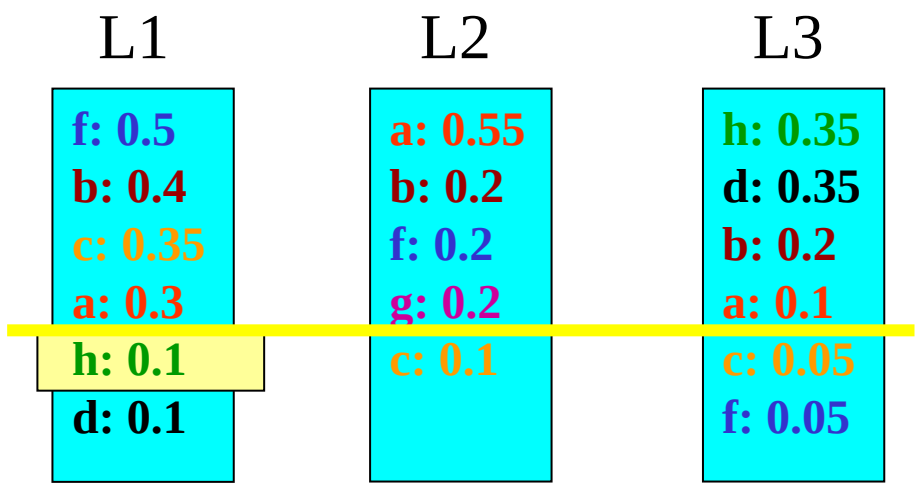
top-k:

~~f: 0.75~~
a: 0.95
b: 0.8

TA-Sorted / NRA (Fagin; Güntzer et al.)

scan index lists in parallel:
consider dj at position pos_i in Li;
E(dj) := E(dj) ∪ {i}; $high_i$:= si(q,dj);
bestscore(dj) := aggr{x1, ..., xm)
with xi := si(q,dj) for i∈E(dj), $high_i$ for i ∉E(dj);
worstscore(dj) := aggr{x1, ..., xm)
with xi := si(q,dj) for i∈E(dj), 0 for i ∉E(dj);
top-k := k docs with largest worstscore;
if min worstscore among top-k ≥ bestscore{d | d not in top-k}
then exit;

m=3
aggr: sum
k=2



top-k:

a: 0.95
b: 0.8

candidates:

~~**f: 0.7 + ? ≤ 0.7 + 0.1**~~
~~**h: 0.45 + ? ≤ 0.45 + 0.2**~~
~~**c: 0.35 + ? ≤ 0.35 + 0.3**~~
~~**d: 0.35 + ? ≤ 0.35 + 0.3**~~
~~**g: 0.2 + ? ≤ 0.2 + 0.4**~~

Datenintensive Anwendungen mit Top-k-Queries

Top-k-Resultate von Anfragen mit Ranking über

- ***Multimedia-Daten***: Aggr. über Features wie Farbe, Kontur, Textur, etc.
- ***Produktkatalogen***: Aggr. über Ähnlichkeits-Scores für
  numerische Attribute wie Jahr, Preis, Bewertung, etc. und
  kategoriale Attribute wie Lage (von Häusern), Schnittstellen (von PCs)
- ***Textdokumente***: Aggr. über Termgewichte
- ***Web-Dokumente***: Aggr. über (Inhalts-) Relevanz, Autorität, Aktualität
- ***Intranet-Dokumente***: Aggr. über Features wie
  Text, Titel, Ankertext, Autorität, Aktualität, URL-Länge, URL-Tiefe,
  URL-Typ (z.B. mit „index.html“ oder „~“ vs. „?“)
- ***Metasuchmaschinen:*** Aggr. über Ranglisten
  verschiedener Suchmaschinen
- ***Verteilte Deep-Web-Quellen***: Aggr. über Eigenschaften von Objekten
  auf dynamischen Webseiten wie z.B. Restaurantbewertungen,
  Restaurantpreisen, Entfernung lt. streetfinder.com u.ä.
- ***Peer-to-Peer-basierte Empfehlungen***

2.5 Grundlagen aus der Linearen Algebra

Eine Menge S von Vektoren heißt **linear unabhängig**, wenn sich kein $x \in S$ als Linearkombination der anderen Vektoren aus S schreiben lässt.
Der **Rang** einer Matrix A ist die maximale Anzahl linear unabhängiger Zeilen- oder Spaltenvektoren.
Eine **Basis** einer n×n-Matrix A ist eine Menge S von Zeilen- bzw. Spaltenvektoren, so dass alle Zeilen bzw. Spalten Linearkombinationen der Vektoren aus S ist.
Eine Menge S von n×1-Vektoren heißt **Orthonormalbasis**, wenn für alle $x, y \in S$ gilt:

$$\|x\|_2 := \sqrt{\sum_{i=1}^{n} X_i^2} = 1 = \|y\|_2 \quad und \quad x \cdot y = 0$$

Eigenwerte und Eigenvektoren

Seien A eine reellwertige n×n-Matrix, x ein reellwertiger n×1-Vektor und λ ein reeller Skalarwert. Die Lösungen x und λ der Gleichung $A \times x = \lambda x$ heißen **Eigenvektor** bzw. **Eigenwert** von A.
Die Eigenvektoren von A sind Vektoren, deren Richtungen bei der durch A beschriebenen Linearabbildung erhalten bleiben.

Die Eigenwerte von A sind die Nullstellen des charakteristischen Polynoms f(λ) von A: $f(\lambda) = |A - \lambda I| = 0$

mit der Determinantenentwicklung nach der i-ten Zeile:

$$|A| = \sum_{j=1}^{n} (-1)^{i+j} a_{ij} \left|A^{(ij)}\right|$$

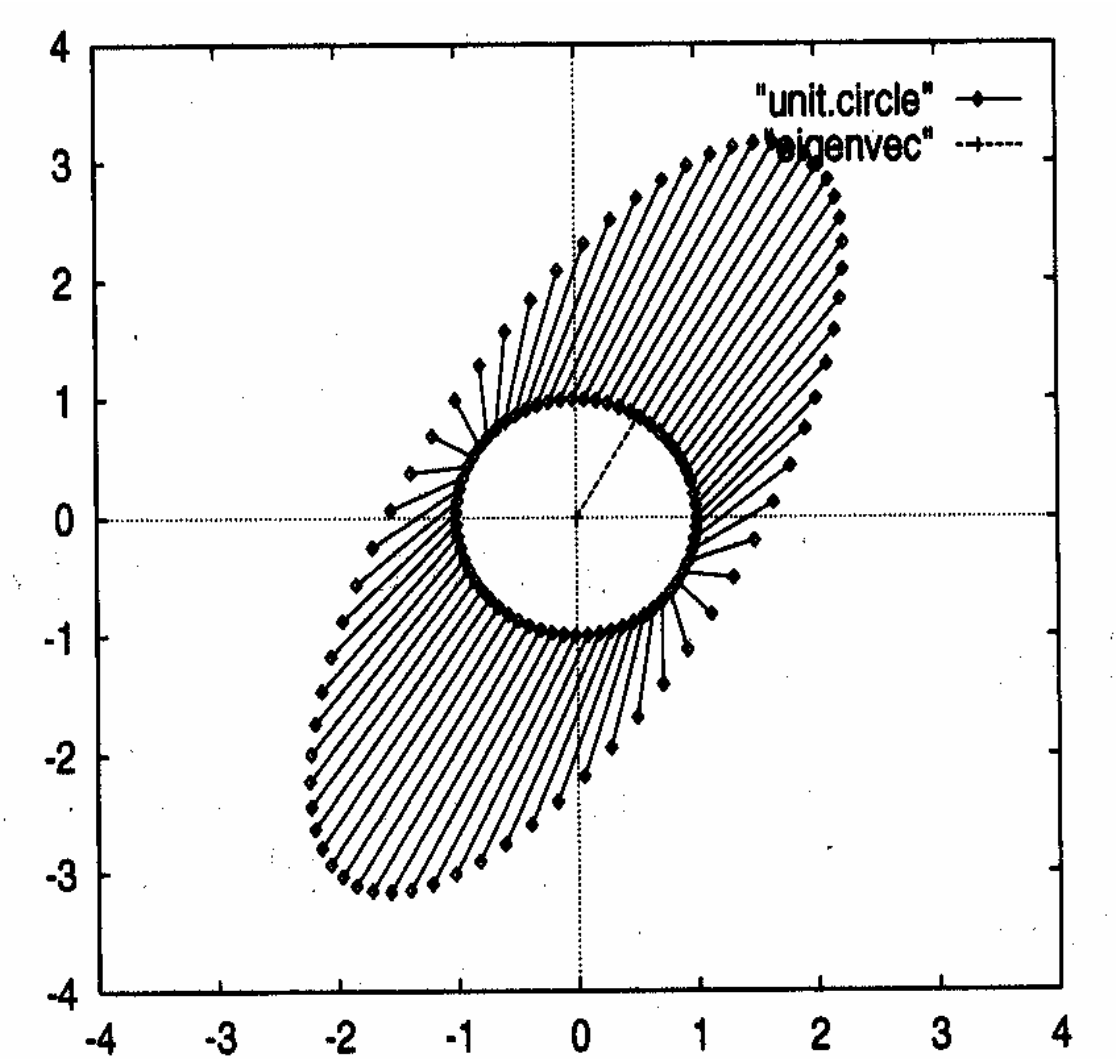
wobei man die Matrix $A^{(ij)}$ aus A durch Streichung der i. Zeile und der j. Spalte erhält

Die reellwertige n×n-Matrix A heißt **symmetrisch**, wenn $a_{ij}=a_{ji}$ für alle i, j. A heißt **positiv definit**, wenn für alle n×1-Vektoren $x \neq \mathcal{O}$ gilt: $x^T \times A \times x > 0$.
Wenn A symmetrisch ist, sind alle Eigenwerte von A reell.
Wenn A symmetrisch und positiv definit ist, sind alle Eigenwerte positiv.

Illustration von Eigenvektoren

Matrix $A = \begin{pmatrix} 2 & 1 \\ 1 & 3 \end{pmatrix}$

beschreibt
affine Abb. $x \mapsto Ax$



Eigenvektor
x1 = $(0.52 \;\; 0.85)^T$
zum Eigenwert $\lambda 1 = 3.62$

Eigenvektor
x2 = $(0.85 \;\; -0.52)^T$
zum Eigenwert $\lambda 2 = 1.38$

Spektralsatz der Linearen Algebra

Spektralsatz

(Hauptachsentransformation, Principal Component Analysis, PCA):

Sei A eine symmetrische n×n-Matrix mit Eigenwerten λ1, ..., λn
und Eigenvektoren x1, ..., xn, so dass $\|x_i\|_2 = 1$ für alle i.
Die Eigenvektoren bilden eine Orthonormalbasis von A.
Dann gilt:

$$D = Q^T \times A \times Q,$$

wobei D eine Diagonalmatrix ist mit den Diagonalelementen λ1, ..., λn
und Q aus den Spaltenvektoren x1, ..., xn besteht.

2.6 Latent Semantic Indexing (LSI): Grundidee

Ziel:

Transformation der Dokumentvektoren vom hochdimensionalen Termvektorraum in einen **Themenvektorraum** niedrigerer Dimensionalität unter

- Ausnutzung von Korrelationen zwischen Termen
  (z.B. „Web“ und „Internet“ häufig zusammen)
- implizite Differenzierung von Polysemen, die sich in ihren Korrelationen mit anderen Termen unterscheiden
  (z.B. „Java“ mit „Library“ vs. „Java“ mit „Kona Blend“ vs. „Java“ mit „Borneo“)

mathematisch:

gegeben: m Terme, n Dokumente (i.d.R. n > m) und eine m×n-Term-Dokument-Ähnlichkeitsmatrix A,

gesucht: möglichst gute – ähnlichkeitsbewahrende – Abbildung der Spaltenvektoren von A in einen k-dimensionalen Vektorraum (k << m) für gegebenes k

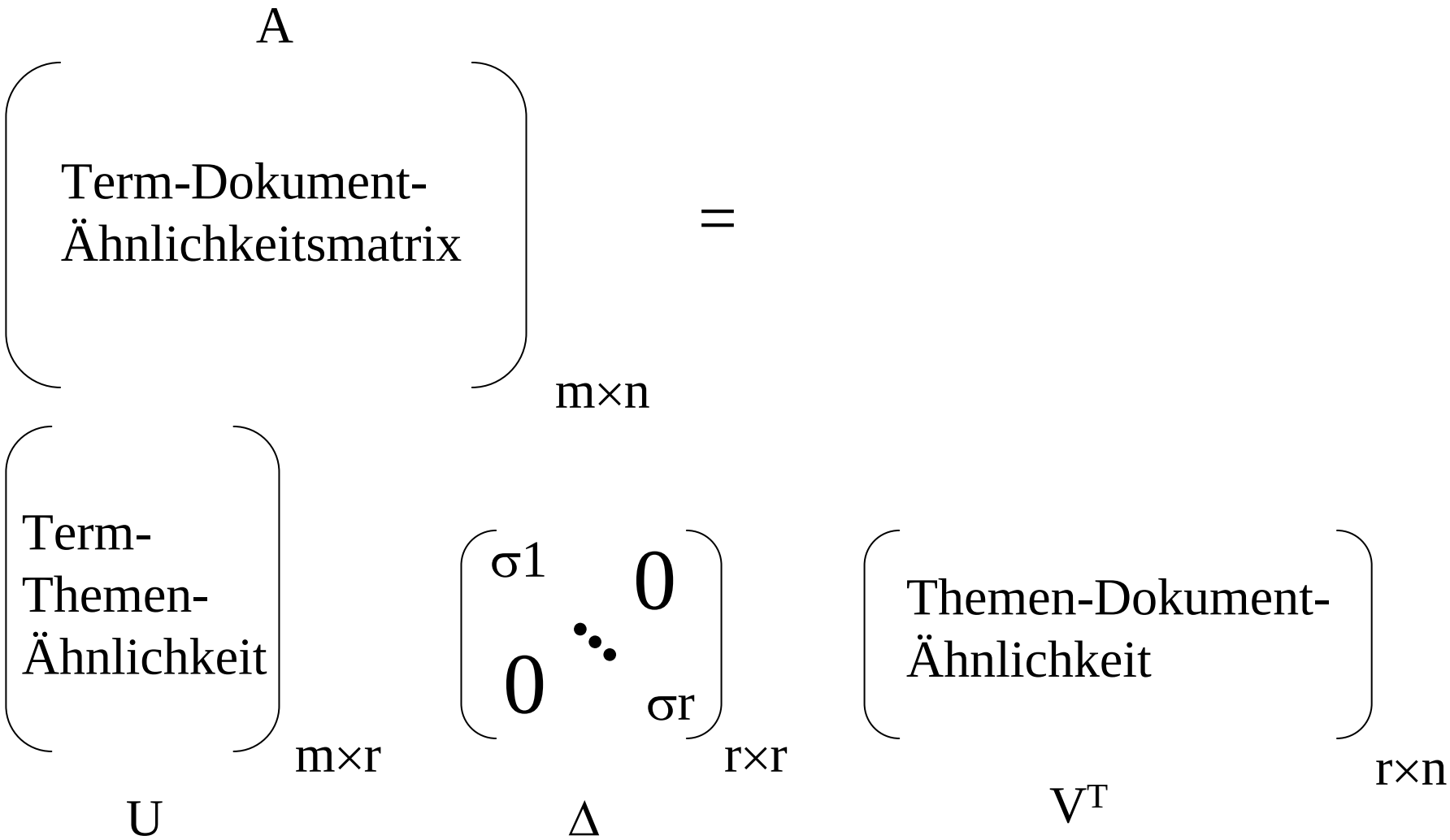
Exkurs: Singulärwertdekomposition (SVD)

Satz:
Jede reellwertige m×n-Matrix A mit Rang r kann zerlegt werden in die Form $\mathbf{A = U \times \Delta \times V^T}$
mit einer **m×r-Matrix U** mit orthonormalen Spaltenvektoren,
einer **r×r-Diagonalmatrix Δ** und
einer **n×r-Matrix V** mit orthonormalen Spaltenvektoren.
Diese Zerlegung heißt Singulärwertdekomposition und ist eindeutig, wenn die Elemente von Δ der Größe nach geordnet werden.

Satz:
In der Singulärwertdekomposition $A = U \times \Delta \times V^T$ der Matrix A sind U, Δ und V wie folgt bestimmt:

- Δ besteht aus den Singulärwerten von A,
  d.h. den positiven Wurzeln der Eigenwerte von $A^T \times A$,
- die Spaltenvektoren von U sind die Eigenvektoren von $A \times A^T$,
- die Spaltenvektoren von V sind die Eigenvektoren von $A^T \times A$.

Schematische Darstellung der SVD



Exkurs: SVD als Regressionsverfahren

Satz:

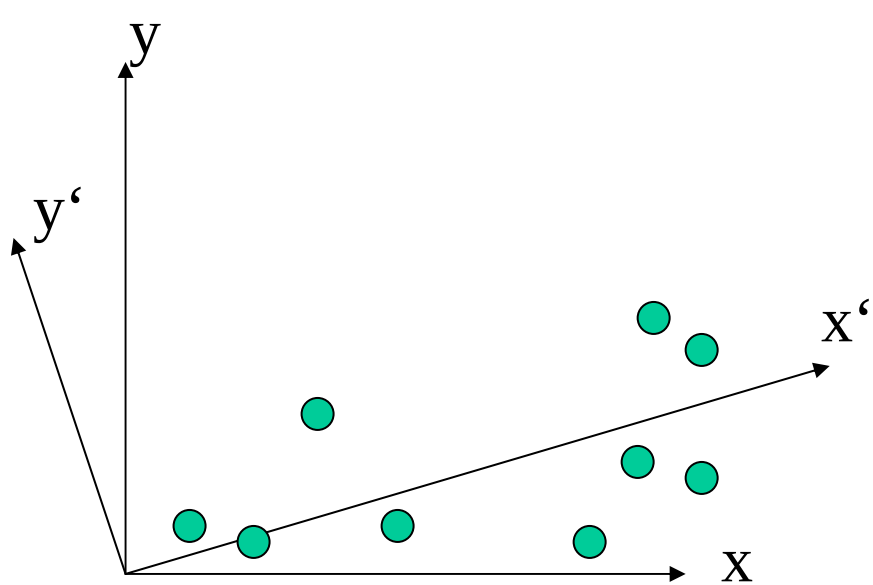
Sei A eine m×n-Matrix mit Rang r und sei Ak = $U_k \times \Delta_k \times V_k^T$, wobei die k×k-Diagonalmatrix Δ_k die k größten Singulärwerte von A enthält und die m×k-Matrix U_k sowie die n×k-Matrix V_k aus den zugehörigen Eigenvektoren der Singulärwertdekomposition von A bestehen.

Unter allen m×n-Matrizen C mit einem Rang, der nicht größer als k ist, ist A_k diejenige Matrix, die den Wert

$$\|A-C\|_F^2 = \sum_{i=1}^{m}\sum_{j=1}^{n}(A_{ij}-C_{ij})^2$$

minimiert (die Frobenius-Norm).

Beispiel:
m=2, n=8, k=1
Projektion auf x‘-Achse
minimiert „Fehler“ bzw.
maximiert Varianz
im k-dimensionalen Raum



Anwendung der SVD auf das Vektorraummodell

A ist die m×n-Term-Dokument-Ähnlichkeitsmatrix. Dann sind:

- U bzw. U_k die m×r- bzw. m×k-Term-Themen-Ähnlichkeitsmatrix,
- V bzw. V_k die n×r- bzw. n×k-Dokument-Themen-Ähnlichkeitsmatrix
- $A \times A^T$ bzw. $A_k \times A_k^T$ die m×m-Term-Term-Ähnlichkeitsmatrix,
- $A^T \times A$ bzw. $A_k^T \times A_k$ die n×n-Dokument-Dokument-Ähnlichkeitsmatrix

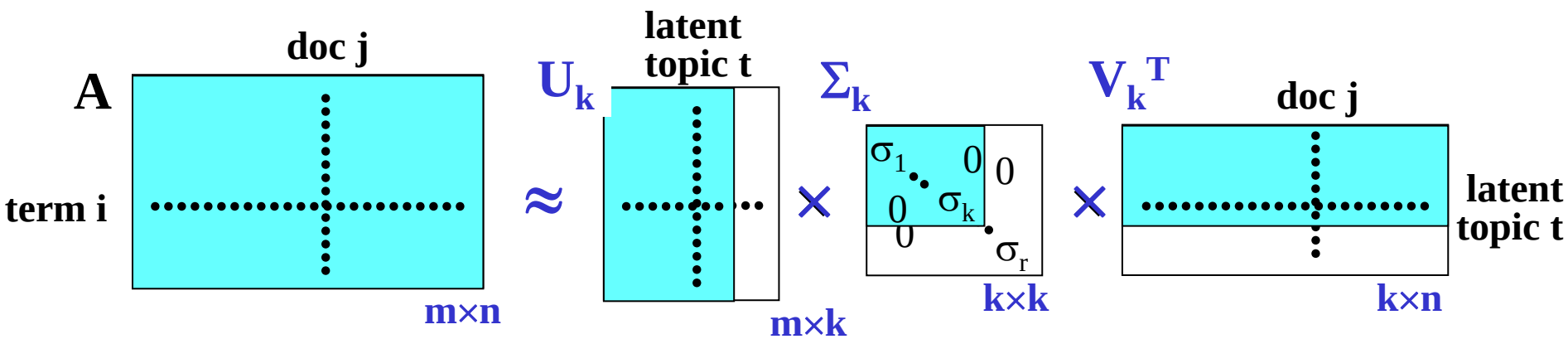


Abbildung von m×1-Vektoren in Themenraum: $d_j \mapsto U_k^T \times d_j =: d_j'$

$q \mapsto U_k^T \times q =: q'$

Skalarprodukt-Ähnlichkeit im Themenraum: $d_j{}^{\prime T} \times q' = ((\Delta_k V_k^T)_{*j})^T \times q'$

Indexierung und Anfrageauswertung

- Die Matrix $\Delta_k V_k^T$ entspricht einem **„Themen-Index“** und ist in einer geeigneten Datenstruktur zu verwalten.
  Statt $\Delta_k V_k^T$ kann man auch vereinfachend **V_k^T als Index** verwenden.
- Zusätzlich muß die **Term-Themen-Abbildung U_k** gespeichert werden.
- Eine **Anfrage q** (ein m×1-Spaltenvektor) im Termvektorraum wird in die Anfrage **$q' = U_k^T \times q$** (ein k×1-Spaltenvektor) transformiert und dann im Themenvektorraum (also V_k) ausgewertet (z.B. mittels Skalarproduktmaß $V_k^T \times q'$ oder Cosinusmaß)
- Ein **neues Dokument d** (ein m×1-Spaltenvektor) wird in **$d' = U_k^T \times d$** (ein k ×1-Spaltenvektor) transformiert und als neue Spalte an den „Index“ V_k^T angefügt („folding-in“)

Beispiel 1 für Latent Semantic Indexing

m=5 (Bush, Schröder, Korea, Klose, Völler), n=7

$$A = \begin{pmatrix} 1&2&1&5&0&0&0 \\ 1&2&1&5&0&0&0 \\ 1&2&1&5&0&0&0 \\ 0&0&0&0&2&3&1 \\ 0&0&0&0&2&3&1 \end{pmatrix} = \underbrace{\begin{pmatrix} 0.58 & 0.00 \\ 0.58 & 0.00 \\ 0.58 & 0.00 \\ 0.00 & 0.71 \\ 0.00 & 0.71 \end{pmatrix}}_{U} \times \underbrace{\begin{pmatrix} 9.64 & 0.00 \\ 0.00 & 5.29 \end{pmatrix}}_{\Delta} \times \underbrace{\begin{pmatrix} 0.18 & 0.36 & 0.18 & 0.90 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.53 & 0.80 & 0.27 \end{pmatrix}}_{V^T}$$

Anfrage $q = (0\ 0\ 1\ 0\ 0)^T$ wird in
$q' = U^T \times q = (0.58\ \ 0.00)^T$ transformiert und gegen V^T evaluiert.

Neues Dokument $d8 = (1\ 1\ 0\ 0\ 0)^T$ wird in
$d8' = U^T \times d8 = (1.16\ \ 0.00)^T$ transformiert und an V^T angefügt.

Beispiel 2 für Latent Semantic Indexing

m=6 terms
- t1: bak(e,ing)
- t2: recipe(s)
- t3: bread
- t4: cake
- t5: pastr(y,ies)
- t6: pie

n=5 documents
- d1: How to bake bread without recipes
- d2: The classic art of Viennese Pastry
- d3: Numerical recipes: the art of scientific computing
- d4: Breads, pastries, pies and cakes: quantity baking recipes
- d5: Pastry: a book of best French recipes

$$A = \begin{pmatrix} 0.5774 & 0.0000 & 0.0000 & 0.4082 & 0.0000 \\ 0.5774 & 0.0000 & 1.0000 & 0.4082 & 0.7071 \\ 0.5774 & 0.0000 & 0.0000 & 0.4082 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.4082 & 0.0000 \\ 0.0000 & 1.0000 & 0.0000 & 0.4082 & 0.7071 \\ 0.0000 & 0.0000 & 0.0000 & 0.4082 & 0.0000 \end{pmatrix}$$

Beispiel 2 für Latent Semantic Indexing (2)

$$A = \begin{pmatrix} 0.2670 & -0.2567 & 0.5308 & -0.2847 \\ 0.7479 & -0.3981 & -0.5249 & 0.0816 \\ 0.2670 & -0.2567 & 0.5308 & -0.2847 \\ 0.1182 & -0.0127 & 0.2774 & 0.6394 \\ 0.5198 & 0.8423 & 0.0838 & -0.1158 \\ 0.1182 & -0.0127 & 0.2774 & 0.6394 \end{pmatrix} \quad \mathrm{U}$$

$$\times \begin{pmatrix} 1.6950 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 1.1158 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.8403 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.4195 \end{pmatrix} \quad \Delta$$

$$\times \begin{pmatrix} 0.4366 & 0.3067 & 0.4412 & 0.4909 & 0.5288 \\ -0.4717 & 0.7549 & -0.3568 & -0.0346 & 0.2815 \\ 0.3688 & 0.0998 & -0.6247 & 0.5711 & -0.3712 \\ -0.6715 & -0.2760 & 0.1945 & 0.6571 & -0.0577 \end{pmatrix} \quad \mathrm{V}^{\mathrm{T}}$$

Beispiel 2 für Latent Semantic Indexing (3)

$$A_3 = \begin{pmatrix} 0.4971 & -0.0330 & 0.0232 & 0.4867 & -0.0069 \\ 0.6003 & 0.0094 & 0.9933 & 0.3858 & 0.7091 \\ 0.4971 & -0.0330 & 0.0232 & 0.4867 & -0.0069 \\ 0.1801 & 0.0740 & -0.0522 & 0.2320 & 0.0155 \\ -0.0326 & 0.9866 & 0.0094 & 0.4402 & 0.7043 \\ 0.1801 & 0.0740 & -0.0522 & 0.2320 & 0.0155 \end{pmatrix} = U_3 \times \Delta_3 \times V_3^T$$

Beispiel 2 für Latent Semantic Indexing (4)

Anfrage q: baking bread

$q = (1\ 0\ 1\ 0\ 0\ 0)^T$

Transformation in den Themenraum mit k=3

$q' = U_k^T \times q = (0.5340\ \ -0.5134\ \ 1.0616)^T$

Skalarprodukt-Ähnlichkeit im Themenraum mit k=3:

$sim(q, d1) = Vk_{*1}^T \times q' \approx 0.86$ $\quad sim(q, d2) = Vk_{*2}^T \times q' \approx -0.12$

$sim(q, d3) = Vk_{*3}^T \times q' \approx -0.24$ $\quad$ usw.

Folding-in eines neuen Dokuments d6:

algorithmic recipes for the computation of pie

$d6 = (0\ \ 0.7071\ \ 0\ \ 0\ \ 0\ \ 0.7071)^T$

Transformation in den Themenraum mit k=3

$d6' = U_k^T \times d6 \approx (0.5\ \ -0.28\ \ -0.15)$

d6‘ als neue Spalte an Vk^T anhängen

Mehrsprachiges Retrieval mit LSI

- Konstruiere LSI-Modell (U_k, Δ_k, V_k^T) anhand von Trainingsdokumenten, die mehrsprachig vorliegen:
  - Betrachte alle Sprachversionen eines Dokuments als ein einziges Dokument und
  - extrahiere zur Indexierung alle Terme oder Wörter unabhängig von der Sprache.
- Indexiere weitere Dokumente durch „folding-in“, also Abbildung in den Themen-Vektorraum und Anhängen an V_k^T.
- Anfragen können dann in beliebiger Sprache gestellt werden und liefern Antworten in allen Sprachen.

Beispiel:

d1: How to bake bread without recipes.
Wie man ohne Rezept Brot backen kann.

d2: Pastry: a book of best French recipes.
Gebäck: eine Sammlung der besten französischen Rezepte.

Terme sind dann z.B. bake, bread, recipe, backen, Brot, Rezept, usw.
Dokumente und Terme werden auf einen kompakten Themenraum abgebildet.

Zusammenfassung zu LSI

+ Elegantes, mathematisch wohlfundiertes Modell
+ „Automatisches Lernen“ von Termkorrelationen
  (inkl. morphologischer Wortvarianten, Mehrsprachigkeit)
+ Impliziter Thesaurus (durch Korrelation von Synonymen)
+ Implizite Diskriminierung der verschiedenen Bedeutungen
  von Polysemen (durch verschiedene Korrelationen)
+ Verbesserte Retrievalgüte auf „geschlossenen“ Korpora
  (z.B. TREC-Benchmark, Finanznachrichten, Patentkollektionen u.ä.)
  mit empirisch günstigstem k in der Größenordnung 100-200
– Schwierige Wahl von günstigem k
– Rechen- und Speicheraufwand für sehr große (z.T. aber dünn
  besetzte) Matrizen
– Keine überzeugenden Resultate für Web-Suchmaschinen