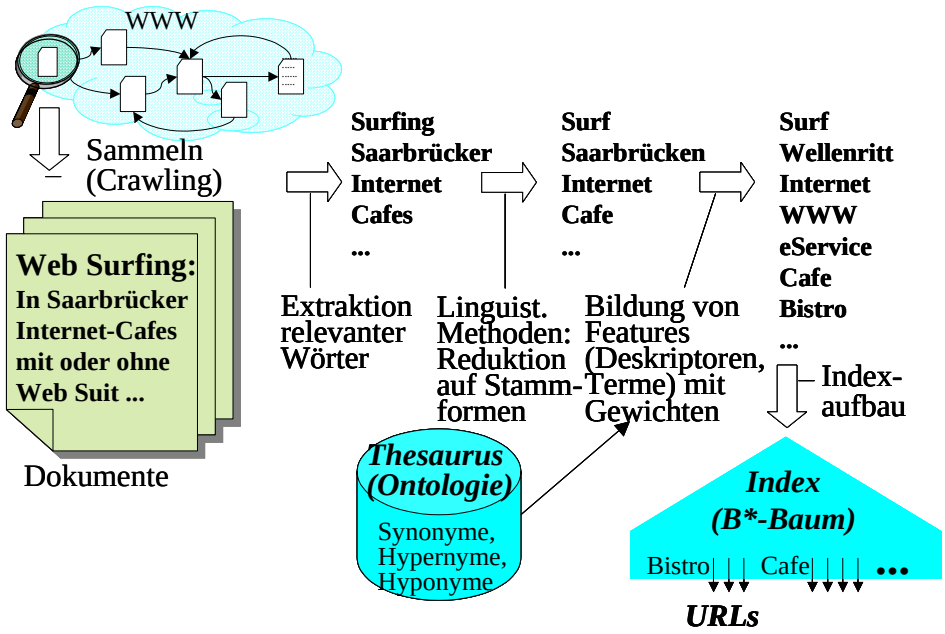
Kapitel 2: Suchmaschinen für Intranets und das Web

2.1 Information-Retrieval-Systeme

Information-Retrieval-Systeme (IRS) verwalten große Sammlungen unstrukturierter Daten oder semistrukturierter Daten, insbesondere Textdokumente und HTML-Dokumente, und unterstützen vor allem die Suche nach relevanten Daten zu einem spezifizierten Thema bzw. ähnlichen Dokumenten zu einer spezifizierten Anfrage (einem "Musterdokument"). IRS-Funktionalität findet sich in Suchmaschinen für das Web und für Intranets, in digitalen Bibliothekssystemen, Mail-Servern und z.T. auch integriert oder als Erweiterungskomponente in objekt-relationalen Datenbanksystemen. Das Prinzip der Ähnlichkeitssuche ist auch für Multimedia-Anwendungen von großer Bedeutung, z.B. für elektronische Museen u.ä.

Die folgende Abbildung gibt einen Überblick über den Aufbau und die Arbeitsweise eines IRS:



Ein *Crawler* traversiert Dokumente, im Web und in Intranets durch Verfolgen von Hyperlinks mittels HTTP-Aufrufen, und lädt sie zur Inhaltserschließung auf den IRS-Server. Eine oberflächliche *Textanalyse* extrahiert aus einem Dokument alle relevanten Wörter, ggf. durch sog. *Stoppwortelimination*, das Entfernen von "Füllwörtern" wie Artikel, Präpositionen, Konjunktionen. Mit einfachen linguistischen Verfahren, sog. *Stemming*-Verfahren, werden Wörter auf ihre jeweiligen Stammformen reduziert (z.B. Plural auf Singular, Dativ auf Nominativ, Partizip Perfekt auf Infinitiv eines Verbs). Die so entstandene Menge von für das Dokument signifikanten Wortstammformen wird auch als Menge von *Termen*, *Deskriptoren* oder *Features* bezeichnet. Ggf. kann diese Menge durch Nachschlagen in einem Wörterbuch, einem sog. *Thesaurus* bzw. einer sog. *Ontologie* (z.B. WordNet), um Synonyme oder eng verwandte Unterbegriffe (Hyponyme) und Oberbegriffe (Hypernyme) erweitert werden. Die in einem Dokument vorkommenden Terme werden typischerweise *gewichtet*, und zwar aufgrund ihrer Vorkommenshäufigkeit und/oder aufgrund ihrer Position und Präsentation im Dokument (z.B. in einer Überschrift, in einem großen Font, usw.); Techniken zur Gewichtung werden in

Abschnitt 2.2 genauer betrachtet. Schließlich wird ein *Index* über diesen Termen als Suchschlüssel aufgebaut, typischerweise in Form eines B*-Baums (oder eines sonstigen Suchbaums für Sekundärspeicher), der mit jedem Term eine Menge von Dokument-Ids bzw. URLs sowie die Gewichte des Terms in den jeweiligen Dokumenten verbindet.

Anfragen an ein IRS sind - in der am meisten verbreiteten Form - zunächst einfache Mengen oder Listen von Termen, z.B. "Java Lava"; bei einer Interpretation als Liste werden die Terme der Anfrage unterschiedlich stark gewichtet. Das Resultat ist entweder eine Menge von Trefferdokumenten, die alle angegebenen Terme enthalten, oder eine Rangliste von zur Anfrage ähnlichen Dokumenten, die nach einem Ähnlichkeitsmaß bzw. einem Relevanz-Score (auch RSV = Retrieval Status Score genannt) absteigend sortiert ist. Ersteres nennt man *Boolesches Retrieval*, letzteres *Ranked Retrieval*. Beim Ranked Retrieval qualifizieren sich oft auch Dokumente, die nur einen Teil der Terme der Anfrage enthalten. Darüber hinaus werden u.U. auch Dokumente berücksichtigt, die keinen der Suchterme wörtlich enthalten, aber Synonyme dazu oder semantisch eng verwandte Begriffe enthalten; für diese Art des Semantischen Retrievals benötigt man einen Thesaurus bzw. eine Ontologie.

Erweiterte Formen von Anfragen erlauben die Angabe negativer Terme, die als Ausschlusskriterium interpretiert werden, z.B. "Java Lava -Coffee", wenn man keine Dokumente über die "heißesten Kaffeesorten" bekommen möchte. Darüber hinaus sind Boolesche Kombinationen von Suchbedingungen möglich und zusätzliche Bedingungen, die auf die Nachbarschaft bzw. textuelle Distanz von Wörtern abzielen, z.B. "Java NEAR Lava". Suchbedingungen dieser Art sind z.T. auch in SQL-Systemen integriert.

Bei einer Anfrage, die n verschiedene Terme enthält, extrahiert das IRS zunächst die zu diesen Termen gehörigen URL- bzw. Dokument-Id-Listen aus seinem Index. Diese bilden eine Kandidatenmenge. Anschließend werden für alle Kandidaten auf der Basis ihrer Termgewichte Ähnlichkeitsmaße zur Anfrage berechnet; dabei wird die Struktur der Termkombinationen in der Anfrage wie z.B. negative Terme oder diskunktive vs. konjunktive Verknüpfung berücksichtigt. Auf diese Weise werden Kandidaten entfernt, und es werden Relevanz-Scores und das Ranking der verbleibenden Trefferdokumente berechnet.

Gütemaße

Die Güte eines IRS, also seines Fähigkeit, zu einer Anfrage möglichst relevante und nur relevante Dokumente zurückzuliefern, wird vor allem mit den folgenden beiden Metriken bewertet:

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

Präzision einer Anfrage (engl.: *precision*) = $\frac{\textit{Anzahl relevanter Dokumente unter Top } r}{r}$

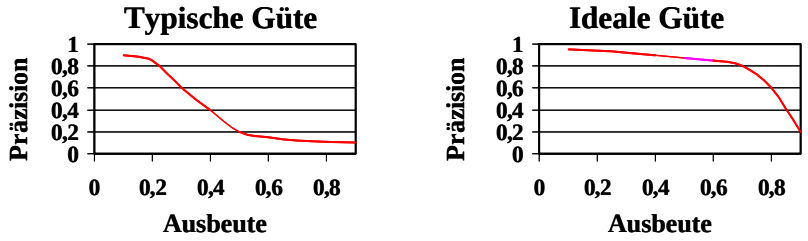
Fähigkeit, zu einer Anfrage *alle* relevanten Dokumente zu liefern:

Ausbeute einer Anfrage (engl.: *recall*) = $\frac{\textit{Anzahl relevanter Dokumente}}{\textit{Anzahl aller relevanten Dokumente im Korpus}}$

Dabei wird r typischerweise auf 10 oder 20 gesetzt, also denjenigen Präfix der Resultatsrangliste, den sich ein Benutzer in der Regel anschaut. Die Auswertung der beiden Gütemaße erfordert eine intellektuelle Bewertung der Anfrageresultate bzw. des gesamten Korpus (aller Dokumente, die das IRS verwaltet). Dies wird für spezielle Benchmark-Dokumentkollektionen (z.B. die TREC-

Benchmarks) gemacht, und verschiedene IRS werden dann anhand von Anfragemixturen auf diesen Daten getestet.

Zwischen Präzision und Ausbeute gibt es einen inhärenten Zielkonflikt. Ein ideales IRS würde für beide Maße Werte in der Nähe von 1,0 erzielen, aber in der Praxis wird eine hohe Präzision oft um den Preis einer schwächeren Ausbeute und eine hohe Ausbeute oft um den Preis einer schlechteren Präzision erkauft.



In Benchmark-Resultaten wird oft die Präzision als Funktion der erzielten Ausbeute dargestellt. Wenn man das Benchmark-Ergebnis auf eine einzige Zahl kompakt reduzieren möchte, wird entweder der Precision-Recall-Breakeven-Point verwendet, also der Punkt auf der Kurve, bei dem Präzision und Ausbeute gleich sind, oder die sog. uninterpolierte durchschnittliche Präzision. Letzeres ist die durchschnittliche Präzision gemittelt über alle Rangplätze, auf denen ein relevanter Treffer steht. Ein weiteres kompaktes Gütemaß ist das sog. F-Maß, das harmonische Mittel aus Präzision und Ausbeute (für eine bestimmte Top-r-Menge): $\frac{1}{\frac{1}{Pr\ddot{a}zision} + \frac{1}{Ausbeute}}$.

2.2 Web-Crawling und Indexierung

Eine Suchmaschine für das Web – oder auch für große Intranets – besteht aus den folgenden Komponenten:

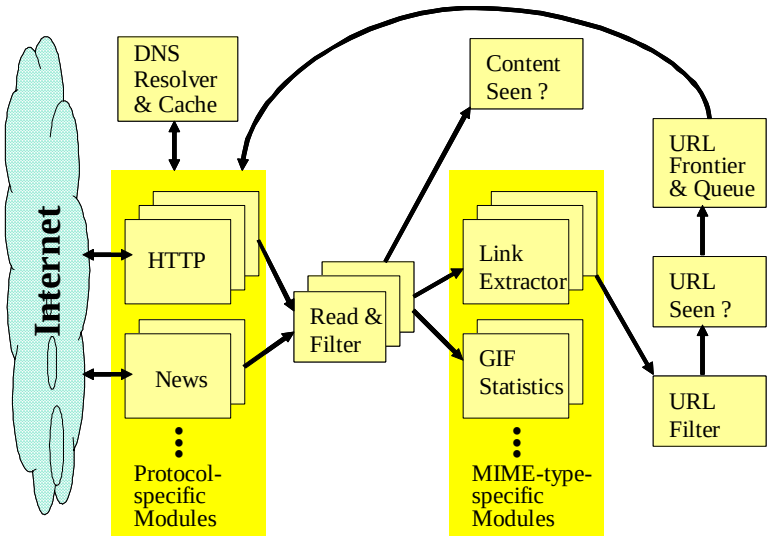
- **URL Queue Server** zur Verwaltung einer Priority-Queue, in der zu Beginn Start-URLs stehen und später die aus Links von besuchten Webseiten extrahierten URLs von Seiten, die selbst noch nicht besucht wurden. Die Priorisierung der URLs hängt von vielen Parametern ab, z.B. von der Zeit des letzten Besuchs, vom Domain-Namen, von Eigenschaften der entsprechenden Webseite (z.B. dem Eingangsgrad, sofern dieser durch frühere Crawls geschätzt werden kann), usw.
- **Crawler (Robot, Spider)** zum Zugriff auf Web-Seiten, in der Regel mit dem HTTP-Protokoll (Hypertext Transfer Protocol) unter Beachtung verschiedener Nebenbedingungen (z.B. Einschränkungen des Typs zu besuchender Web-Seiten, Beachtung der durch das Robot Exclusion Protocol gegebenen „Etikette“, usw.).
- **Repository Server** zur Verwaltung der durch den Crawler geladenen Dokumente (z.B. in einer Tabelle *DocumentRepository* in einer Datenbank).
- **Indexer (inkl. Parser, Stemmer)** zur Analyse von (überwiegend HTML-) Dokumenten und zum Aufbau eines *DocumentIndex* mit Anlegen von Einträgen in *Lexicon* (Wörterbuch für alle Terme) und *Anchors* (Anker von Hyperlinks).
- **URL Resolver** zur Umsetzung von URLs in interne DocIds.

- **Link Analyzer** zur Analyse von Hyperlinkstrukturen und entsprechenden Berechnung von Autoritätsmaßen (siehe Kapitel 4).
- **Query Processor** zur Auswertung von Anfragen, typischerweise einfach Mengen von Keywords**,** durch Nachschlagen im Index und Berechnung einer Rangliste von Treffern mit Hilfe statistischer Maße (siehe Abschnitte 2.3 und 2.4).

Diese Komponenten verwenden die folgenden Datenstrukturen, die z.B. als Tabellen in einer Datenbank realisiert sein könnten. Was genau „Tabellen" einer Datenbank sind, wird in Kapitel 5 erklärt; im Moment genügt eine intuitive Vorstellung für das Verständnis. Große Suchmaschinen wie Google oder Inktomi verwenden eigene, auf Files basierende, Datenstrukturen, die über viele Platten und Rechner verteilt sind.

- **DocumentRepository** ***(DocId, DocContent)*** mit dem vollständigen Text aller (HTML-) Seiten, ggf. in komprimierter Form.
- **Lexicon** ***(TermId, Term)*** mit allen Wortstämmen, die als Indexterme verwendet werden.
- **DocumentIndex** ***(DocId, TermId, Weight, ...)*** mit allen Vorkommen von Termen in Dokumenten sowie entsprechenden Gewichten, die auf Wortstatistiken beruhen (siehe Abschnitt 2.3), optimiert für den schnellen Zugriff über DocIds.
- **TermIndex** ***(TermId, DocId, Weight, ...)*** mit allen Dokumenten zu allen Termen, optimiert für den schnellen Zugriff über TermIds (sog. *invertierte Listen* im IR-Jargon).
- **Anchors** ***(SourceDocId, TargetDocId, AnchorText)*** mit allen bekannten Hyperlinks.
- **URLIndex** ***(URL, DocId)*** zur Umsetzung von URLs auf interne DocIds und umgekehrt.

Die folgende Abbildung illustriert die Arbeitsweise eines Web-Crawlers. Die Architektur ist bezüglich der verwendeten Protokolle (HTTP, News, SOAP für Web Services, etc.) und der Formattypen von Web-Seiten (HTML, PDF, XML, etc.) erweiterbar. Alle Komponenten können im Multithreading-Modus betrieben werden, also mit mehreren parallelen Instantiierungen einer Komponente. Aus Effizienzgründen haben Web-Crawler typischerweise eine eigene Komponente zur Übersetzung von Domain-Namen in IP-Adressen und versuchen, aufwändigere Aufrufe von DNS-Servern (Domain Name Service) zu minimieren. Ebenfalls aus Effizienzgründen ist es sehr wichtig, beim Crawling Seiten zu erkennen, die im aktuellen Crawl bereits besucht wurden. Dazu müssen einerseits besuchte URLs schnell erkannt werden (z.B. mittels Nachschlagen in einer Hashtabelle), und andererseits müssen auch Inhalte mit früher besuchten Seiten verglichen werden (z.B. durch Vergleich mit einer kompakten „Fingerprint"-Signatur), weil etliche Seiten über verschiedene Pfade erreichbar, unter mehreren URLs registriert und vielfach kopiert sind.



Große Web-Suchmaschinen wie Google verwalten in ihrem Index mehr als 4 Milliarden Web-Seiten mit mehr als 10 Millionen Termen. Die Rohdaten umfassen eine Größenordnung von 20 Terabytes, der Index alleine etwa 4 Terabytes. Daten und Index sind über mehr als 10 000 PCs verteilt, partitioniert und aus Verfügbarkeits- und Lastabalancierungsgründen stark repliziert. Eine solche Suchmaschine muss mehr als 200 Millionen Queries pro Tag bedienen, wobei Benutzer eine Antwortzeit von nicht mehr als 1 oder 2 Sekunden erwarten. Der Crawler muss mehr als 1000 Seiten pro Sekunde verarbeiten, und alle Seiten sollten spätestens alle 30 Tage erneut besucht bzw. geprüft werden.

2.3 Vektorraummodell für IR mit Ranking

Im Vektorraummodell nach G. Salton werden Dokumente als Vektoren in einem Feature-Raum repräsentiert. Features, also Vektorraumdimensionen, entsprechen den Termen, die in dem Korpus vorkommen, also den Wörtern nach Stoppwortelimination, Stammformreduktion und ggf. weiteren Abstraktionsschritten (siehe Abschnitt 2.1). Die Komponenten des Vektors eines Dokuments sind entweder binär und signalisieren dann schlicht das Vorkommen eines bestimmten Terms im Dokument oder reelle Zahlen aus dem Intervall [0,1] und signalisieren dann die relative Wichtigkeit oder Häufigkeit des Terms im Dokument. Diese Termgewichte werden nach bestimmten Heuristiken berechnet (siehe unten). Bei einer Feature-Menge F hat der entsprechende Vektorraum die Dimensionalität |F|, und die Vektoren der Form (0...0 1 0 ... 0), bei denen genau ein Feature den Wert 1 hat und alle anderen den Wert 0, bilden eine Basis des Vektorraums.

Anfragen sind in diesem Modell ebenfalls Vektoren im selben Featureraum. Die in der Anfrage vorkommenden Terme werden durch eine 1-Komponente im Vektor repräsentiert, alle anderen Komponenten sind 0. Wenn durch die Reihenfolge der Terme in der Anfrage implizit Gewichte ausgedrückt sein sollen (also die wichtigsten Suchterme vorne stehen müssen), können die Komponenten des Anfragevektors auch geeignet auf das Intervall [0,1] abgebildet werden. Wenn negative Terme angegeben sind, also Wörter, die in den Trefferdokumenten (möglichst) nicht vorkommen sollen, kann man auch das Intervall [-1,1] verwenden. In der Praxis werden solche Negativkriterien aber nicht im Ranking selbst behandelt, sondern als zusätzlicher Filterschritt auf der Rangliste der Kandidaten gemäß der positiven Suchterme realisiert.

Im Vektorraummodell können nun Ähnlichkeiten zwischen Dokumenten und Anfragen (oder auch zwischen verschiedenen Dokumenten) berechnet werden. Diese *Ähnlichkeitswerte*, die oft auch *Relevanz-Scores* oder *Retrieval-Status-Values (RSVs)* genannt werden, sind die Grundlage des Ran-

king: das Resultat ist eine nach Ähnlichkeit zur Anfrage absteigend sortierte Liste von Dokumenten. Das (zumindest als Basis für weitere Variationen) am häufigsten verwendetste Ähnlichkeitsmaß ist das *Cosinus-Maß*:

Für eine Featuremenge F, eine Dokumentmenge D, ein Dokument $d \in D \subseteq [0,1]^{|F|}$ und eine Anfrage $q \in [0,1]^{|F|}$ ist die Ähnlichkeit von d und q: $sim(d,q) = \frac{\sum_{j=1}^{|F|} d_j \, q_j}{\sqrt{\sum_{j=1}^{|F|} d_j^2} \sqrt{\sum_{j=1}^{|F|} q_j^2}}$.

Die Termgewichte der Dokumentvektoren werden in der Regel aufgrund von sog. tf*idf-Formeln bestimmt. Dabei ist tf(t,d) die Häufigkeit von Term t in Dokument d (engl. term frequency = tf), und idf(t) ist der Kehrwert der Häufigkeit des Terms t im gesamten Korpus (engl. inverse document frequency = idf). Die Idee dahinter ist, dass das Gewicht von t in d mit seiner Häufigkeit in d wachsen soll, aber mit der Häufigkeit von t im gesamten Korpus fallen soll. Lokal häufige Terme werden also als signifikant für den Inhalt eines Dokuments angesehen, während global häufige Terme eher inflationären Charakter und daher nur noch geringe Aussagekraft über den Inhalt eines Dokuments haben. In der Praxis werden beide Größen – tf und idf – ggf. noch normalisiert (z.B. auf Werte zwischen 0 und 1), und der idf-Wert wird in der Regel logarithmisch gedämpft, da er sonst das Produkt dominieren würde. Eine typische Variante der tf*idf-Formel für das Termgewicht w_{ij} des i-ten Terms im j-ten Dokument ist z.B.:

$$w_{ij} := \frac{tf_{ij}}{max_k \, tf_{kj}} \; log \frac{N}{df_i}, \text{ wobei}$$

tf_{ij} die Häufigkeit von Term i in Dokument j,

N die Anzahl der Dokumente im Korpus D und

df_i die Häufigkeit von Term i im gesamten Korpus sind.

Wenn man nicht das Cosinus-Maß als Ähnlichkeitsfunktion verwendet, sonderen andere Metriken, kann auch eine Normalisierung der w_{ij}-Werte auf die Dokumentvektorlänge 1 Sinn machen:

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

In Web-Suchmaschinen werden bei den Termgewichten außerdem die Position und Formatierung von Wörtern (z.B. in Überschriften, im Fettdruck oder in einem Hyperlink-Anker) berücksichtigt.

Relevanz-Feedback

Nachdem der Benutzer die ersten r Dokumente der Rangliste einer Query inspiziert hat, kann er durch Markieren der relevanten und irrelevanten Dokumente dem IRS Feedback geben, aufgrund dessen in einer weiteren Verfeinerungs- bzw. Präzisierungs-Query explizite Gewichte in der Query gesetzt werden. Im einfachsten Fall kann dies so realisiert werden, dass der Benutzer den besten Treffer auswählt und eine Query startet, die genau die Deskriptoren dieses Dokuments mit seinen entsprechenden Gewichten enthält.

Die bekannteste Methode für Relevanz-Feedback ist das *Rocchio-Verfahren*. Seien D^+ und D^- Resultatdokumente einer Query q, die der Benutzer positiv bzw. negativ markiert. Dann wird daraus die verfeinerte Query q' mit Gewichten $\alpha, \beta, \gamma \in [0,1]$ (meist mit $\alpha > \beta > \gamma$) wie folgt gebildet:

$$q' = \alpha q + \frac{\beta}{|D^+|} \sum_{d \in D^+} d - \frac{\gamma}{|D^-|} \sum_{d \in D^-} d$$

Alternativ könnten - in einer längeren Recherche-Sitzung auch benutzer- bzw. sitzungsspezifische Dokumentgewichte geführt und je nach Feedback angepasst werden, was aber wesentlich aufwendiger zu implementieren ist.

2.4 Anfrageausführung mit Ranking

Die einfachste, aber mit Abstand wichtigste und häufigste, Form von Anfragen sind Mengen von Keywords $q = t_1 ... t_z$ mit Termen t_i. Im Vektorraummodell sind dies Vektoren, bei denen die Komponenten für $t_1 ... t_z$ Gewicht 1 und alle anderen Komponenten Gewicht 0 haben. Da Treffer mit hohem Score nicht notwendigerweise alle z Terme enthalten müssen, ergibt sich folgender, auf den Indexlisten für $t_1 ... t_z$ arbeitender Algorithmus.

Naiver 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;
return k docs from candidate-docs with highest scores;
```

Der Algorithmus ist naiv, weil er einfach alle möglichen Treffer ermittelt und erst danach Scores berechnet, um am Ende nur die besten k (z.B. k=10) Treffer auszugeben. Bei sehr langen Indexlisten – mit Zehn- oder Hunderttausenden von DocIds – und bei sehr vielen Anfragetermen (z.B. bei vom System erweiterten Anfragen oder Anfragen, die aus markierten Dokumenten konstruiert werden) ist dieser Aufwand unakzeptabel.

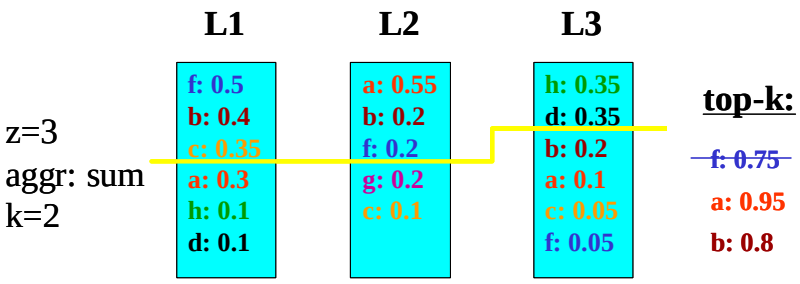
Ein guter Algorithmus sollte es vermeiden, die Indexlisten komplett zu lesen, und der Aufwand sollte in k beschränkt sein. In der IR-Literatur sind zu diesem Zweck zahlreiche Heuristiken vorgeschlagen worden. Ein neuerer, vielseitig einsetzbarer Algorithmus, der garantiert die Top-k-Treffer berechnet und in seiner Effizienz asymptotisch optimal ist, ist der *Threshold-Algorithmus (TA)* von Fagin et al. Dieses Verfahren arbeitet auf z Indexlisten $L_1, ..., L_z$, die jeweils die Treffer zu Anfragetermen $t_1 ... t_z$ enthalten. Zu jeder Indexliste L_i gibt es lokale Scores s_i, und es wird angenommen, dass die Liste nach absteigenden s_i-Werten sortiert ist. Ferner wird angenommen, dass der Gesamt-Score s eines Trefferdokuments eine monotone Aggregationsfunktion aggr seiner lokalen Scores ist, d.h. für alle Dokumente d', d'' gilt:

$s_1(d') \leq s_1(d'') \wedge ... \wedge s_z(d') \leq s_z(d'') \Rightarrow s(d')=aggr(s_1(d'), ..., s_z(d')) \leq aggr(s_1(d''), ..., s_z(d''))=s(d'')$. Letzeres ist z.B. erfüllt, wenn s eine gewichtete Summe der s_i ist, aber auch max und zahlreiche andere Funktionen erfüllen das Monotoniekriterium.

TA traversiert alle z Indexlisten L_1, ..., L_z in verschränkter Form (Round-Robin), liest also erst den besten Treffer von L_1, dann den von L_2, usw., bis hin zum besten Treffer von L_z und dann weiter mit dem zweitbesten von L_1, usw. Die Listen werden also primär mit sortierten Zugriffen (*Sorted Access*) verarbeitet. Zu jedem auf diese Weise gelesenen Dokument d schaut TA in allen anderen Listen per wahlfreiem Zugriff (*Random Access*) die lokalen Scores nach und berechnet dann direkt den Gesamt-Score von d. Die besten k bislang gefundenen Dokumente werden in einer Top-k-Liste festgehalten. In jedem Schritt merkt sich TA außerdem den lokalen Score des zuletzt gelesenen Eintrags der Liste L_i in einer Variablen $high_i$, die eine obere Schranke für den bestmöglichen lokalen Score von noch nicht gelesenen Dokumenten ist. Das Verfahren terminiert, bricht also die Index-Scans ab, wenn der aufgrund der $high_i$-Werte bestmögliche Gesamt-Score eines noch unbekannten Dokuments, $aggr(high_1, ..., high_z)$, nicht besser ist als der schlechteste Gesamt-Score unter den aktuellen Top-k. In der Literatur wurde gezeigt, dass TA bei n Einträgen pro Indexliste L_i mit hoher Wahrscheinlichkeit höchstens $O(n^{m-1/m} \cdot k^{1/m})$ Schritte benötigt und typischerweise sehr viel weniger.

Algorithmus TA:
scan all lists L_i (i=1..z) in parallel:
  consider dj at position pos_i in Li;
  $high_i := s_i(dj)$;
  if dj $\notin$ top-k then {
    look up $s_v(dj)$ in all lists L_v with $v \neq i$; // random access
    compute $s(dj) := aggr \{s_v(dj) \mid v=1..z\}$;
    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 $\geq aggr \{high_v \mid v=1..z\}$ then exit;

Beispiel:



Wenn TA alle Listenelemente oberhalb der gelben Linie gelesen hat, ist der Schwellwert für den Abbruch 0.35+0.2+0.35=0.9. Da dieser Wert höher ist als der schlechteste Gesamt-Score unter den aktuellen Top-k, nämlich 0.8 für Dokument b, muss TA noch ein weiteres Listenelement lesen, und zwar den Eintrag b:0.2 aus Liste L3. Damit wird der Schwellwert neu berechnet, und zwar zu 0.35+0.2+0.2=0.75, und der Algorithmus terminiert.

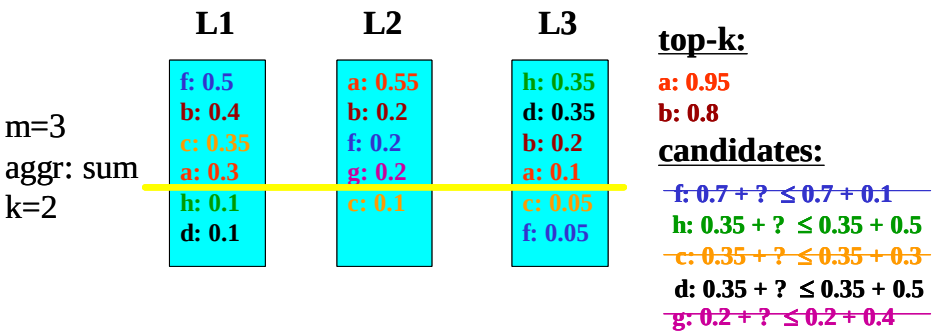
Da wahlfreie Zugriffe auf den Indexlisten in vielen Anwendungssituationen sehr viel teurer sind als die sortierten Zugriffe, sollten sie minimiert werden. Eine Variante von TA, die ganz ohne wahlfreie Zugriffe auskommt, ist *TA-sorted* (TA-NRA für No Random Access). Der Algorithmus führt über jedes gelesene Dokument dj Buch, indem er sich folgende Werte merkt:

- E(dj): die Menge der für den Gesamt-Score von dj ausgewerteten Listen (für die also der lokale Score bekannt ist),
- worstscore(dj) = aggr{x1, ..., xz) mit xi := si(q,dj) für i∈E(dj), 0 für i ∉E(dj)
- bestscore(dj) = aggr{x1, ..., xz) mit xi := si(q,dj) für i∈E(dj), $high_i$ für i ∉E(dj);

TA-sorted kennt in jedem Schritt die aufgrund ihrer worstscore-Werte aktuell besten Top-k-Dokumente und merkt sich alle Kandidaten d, für die bestscore(d) besser ist als der worstscore des schlechtesten Dokuments unter den Top-k. Unter den Kandidaten wird auch ein virtuelles Dokument d mit E(d)=∅ geführt. Kandidaten, deren bestscore unter den worstscore des schlechtesten Top-k-Dokuments fällt, können eliminiert werden. Der Algorithmus terminiert, wenn die Kandidatenliste leer geworden ist.

Algorithmus TA-sorted:
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, ..., xz)
    with xi := si(q,dj) for i∈E(dj), $high_i$ for i ∉E(dj);
  worstscore(dj) := aggr{x1, ..., xz)
    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;

Beispiel:



Die Abbildung gibt den Zustand von TA-sorted an, wenn der Algorithmus alle Listenelemente oberhalb der gelben Linie gelesen hat. Zu diesem Zeitpunkt könnten sich noch h und d für die Top-k qualifizieren. Im nächsten Schritt liest der Algorithmus den L1-Eintrag h:0.1. Dadurch ändern sich der worstscore und der bestscore von h zu 0.35+0.1+? = 0.45+? ≤ 0.45+0.2=0.65, so dass h eliminiert werden kann. Im selben Schritt ergibt sich für d die Änderung 0.35+? ≤ 0.35+0.3=0.65, so dass auch d sich nicht mehr für die Top-k qualifizieren kann. Die Kandidatenliste ist dann leer, und TA-sorted terminiert.

2.5 Grundlagen aus der Linearen Algebra

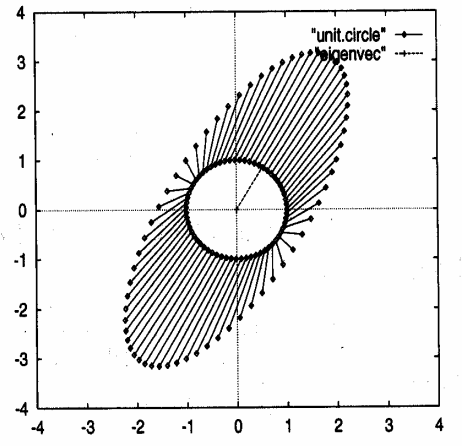
Eine Menge S von Elementen eines Vektorraums heißt **linear unabhängig**, wenn kein Vektor $x \in S$ als Linearkombination anderer Vektoren aus S darstellbar ist, es also keine $y_1, ..., y_k \in S-\{x\}$ gibt, so dass
$x = a_1 y_1 + ... + a_k y_k$ mit reellen Zahlen $a_1, ..., a_k$.
Der **Rang** einer Matrix A ist die maximale Anzahl linear unabhängiger Zeilenvektoren oder linear unabhängiger Spaltenvektoren. Eine **Basis** einer n×n-Matrix A ist eine Menge S linear unabhängiger Zeilenvektoren (oder Spaltenvektoren), so dass alle Zeilen (bzw. Spalten) von A als Linearkombinationen von Vektoren aus S darstellbar sind.
Eine Basis S von A heißt **Orthonormalbasis** wenn für alle x, y ∈S gilt:

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

Sei A eine reellwertige n×n-Matrix, x ein reellwertiger n×1-Vektor und λ ein reellwertiger Skalar. Lösungen x und λ der Gleichung $A \times x = \lambda x$ heißen **Eigenvektor** und **Eigenwert** von A. Eigenvektoren von A sind Vektoren, deren Richtung bei der durch A beschriebenen affinen Abbildung erhalten bleibt.

Beispiel:

Die Matrix $A = \begin{pmatrix} 2 & 1 \\ 1 & 3 \end{pmatrix}$ beschreibt eine affine Abbildung von Vektoren der Ebene: $x \mapsto Ax$. Sie bildet z.B. die Punkte des Einheitskreises auf eine gedreht Ellipse ab. Die Eigenvektoren sind x1 = $(0.52\ 0.85)^T$ zum Eigenwert λ1 = 3.62 und x2 = $(0.85\ -0.52)^T$ zum Eigenwert λ2 = 1.38. Die folgende Abbildung illustriert dieses Beispiel; die Eigenvektoren entsprechen den Richtungen der Ellipsenachsen.



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

mit der Determinante $|A| = \sum_{j=1}^{n} (-1)^{i+j} a_{ij} \left|A^{(ij)}\right|$, wobei $A^{(ij)}$ eine Matrix bezeichnet, die aus A durch Streichen der i-ten Zeile und j-ten Spalte abgeleitet ist.

Eine reellwerte 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 $x \neq 0$: $x^T \times A \times x > 0$.

Satz:
Wenn A symmetrisch ist, dann sind alle Eigenwerte von A reell. Wenn A symmetrisch und positiv definit ist, dann sind alle Eigenwerte positiv.

Spektralsatz (Hauptachsentransformation, Principal Component Analysis, PCA, Karhunen-Loeve-Transformation):
Sei A eine symmetrische n×n-Matrix mit Eigenwerten $\lambda_1, ..., \lambda_n$ und Eigenvektoren $x_1, ..., x_n$, 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 $\lambda_1, ..., \lambda_n$ und Q aus den Spaltenvektoren $x_1, ..., x_n$ besteht. Die Vektoren $x_1, ..., x_n$ heißen in diesem Kontext Hauptachsen (Principal Components).
Man kann leicht zeigen, dass daraus $A = Q \times D \times Q^T$ folgt.

2.6 Latent Semantic Indexing (LSI)

Das einfache Vektorraum-Modell ist in (mindestens) zweierlei Hinsicht zu kritisieren:

- Da es u.U. starke Korrelationen im Vorkommen verschiedener Features in den Dokumenten gibt, ist die Dimensionalität des Feature-Vektorraums eigentlich unnötig hoch.
  Beispiel:
  Wenn "Web" und "Internet" nahezu immer zusammen vorkommen, braucht man nur eines dieser beiden Features zu indexieren.
- Dasselbe Feature kann je nach vorkommender Kombination mit anderen Features eine andere Semantik haben, so daß solche Korrelation explizit berücksichtigt sein sollten.
  Beispiel:
  Wenn "Java" und "Library" zusammen auftreten, ist die Bedeutung von "Java" vermutlich die Programmiersprache Java; das Dokument behandelt, also im weiteren Sinne das Thema Internet. Wenn "Java", "Kona Blend" und "Mokka" zusammen auftreten, handelt das Dokument mit hoher Wahrscheinlichkeit von Kaffee. Wenn schließlich "Java", "Sumatra" und "Borneo" zusammen auftreten, betrifft das Dokument vermutlich das Land Indonesien.

Die Idee von LSI ist, solche Korrelationen auszunutzen, um von den Featurekombinationen auf Themen (Topics) bzw. Konzepte (Concepts) zu schließen, die den Inhalt von Dokumenten charakterisieren. Die Anzahl verschiedener Themen ist typischerweise deutlich kleiner als die der Features. Wenn man das Vektorraum-Modell auf Themen anwendet statt auf Features, erzeugen die Themen einen Vektorraum mit deutlich niedrigerer Dimensionalität. In den Themen kommt also die "latente Semantik" der Dokumente klarer zum Ausdruck.

Gegeben seien:
- eine Featuremenge F mit $|F| = m$
- eine Dokumentmenge D mit $|D|=n$ und $D \subseteq [0,1]^m$
- eine Query $q \in [0,1]^m$

D kann also als m×n-Matrix A mit Werten aus dem Intervall [0,1] interpretiert werden und q als m×1-(Spalten)Vektor.

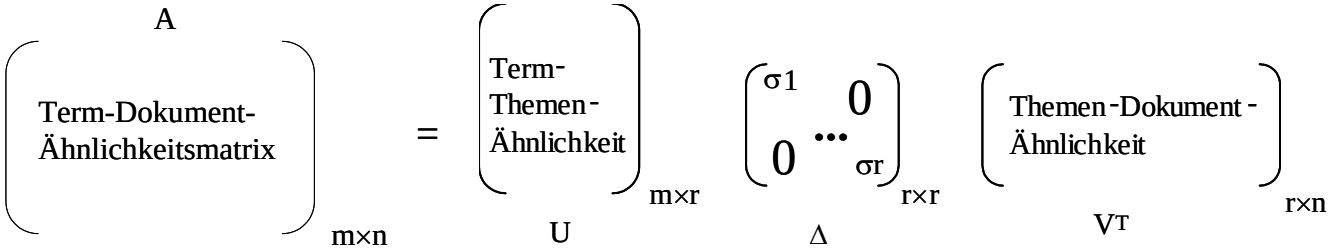
Gesucht wird:
eine "möglichst gute" Abbildung von D und q in einen Themen-Vektorraum
mit Dimensionalität $k << m$

Lösung:
Die Lösung besteht aus einer auf der sogenannten Singulärwertdekomposition von A beruhenden Transformation von A und q in einen automatisch "erzeugten" Vektorraum niedrigerer Dimensionalität. Dabei werden die folgenden mathematischen Eigenschaften der Singulärwertdekomposition ausgenutzt, die den Spektralsatz der Linearen Algebra verallgemeinern.

Satz:
Jede reellwertige n×m-Matrix A mit Rang r kann zerlegt werden in die Form

$$A = U \times \Delta \times V^T$$

mit einer n×r-Matrix U mit orthonormalen Spaltenvektoren, einer r×r-Diagonalmatrix D und einer m×r-Matrix V mit orthonormalen Spaltenvektoren.
Diese Zerlegung heißt *Singulärwertdekomposition* (engl.: Singular Value Decomposition, kurz: SVD) und ist unter der Annahme, daß die Elemente von Δ geordnet sind, eindeutig bestimmt.



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 Zeilenvektoren von V sind die Eigenvektoren von $A^T \times A$.

Dabei kann $A \times A^T$ als Dokument-Dokument-Ähnlichkeitsmatrix interpretiert werden und $A^T \times A$ als Feature-Feature-Ähnlichkeitsmatrix. U ist als *Dokument-Thema-Ähnlichkeitsmatrix* und V als *Feature-Thema-Ähnlichkeitsmatrix* zu interpretieren.

Abbildung von Dokumentvektoren in den Themenraum:

Dokumente d und Queries q, beide als m×1-(Spalten)Vektoren repräsentiert, werden durch Multiplikation mit U^T in den Themenraum abgebildet:

$d \mapsto U^T \times d =: d'$ und $q \mapsto U^T \times q =: q'$.

Die Gesamtheit aller Dokumente, die Spalten von A, wird wie folgt abgebildet: $A \mapsto U^T \times A =: A'$. Dies ist äquivalent zu $A \mapsto U^T \times A = U^T \times \left(U \times \Delta \times V^T\right) = \Delta \times V^T = A'$. Die j-te Spalte von ΔV^T entspricht also dem in den Themenraum transformierten j-ten Dokument.

Die Ähnlichkeit zwischen d_j und q bzw. die Relevanz von d_j für q wird dann z.B. durch das Skalarprodukt $d_j{}'^T \times q'$ = $((\Delta V^T)_{*j})^T \times$ q' bestimmt (wobei M_{*j} die j-te Spalte der Matrix M bezeichnet).

Indexierung von A und Ausführung von Queries q:

Als Index zu verwalten ist die Dokument-Thema-Ähnlichkeitsmatrix ΔV^T sowie - quasi als Domänenwissen - die Term-Thema-Ähnlichkeitsmatrix U. Zur Verwendung von ΔV^T als Index werden in der Literatur auch Variationen betrachtet, insbesondere auch die Variante, bei der einfach V^T als Index verwendet wird und Δ selbst gar nicht berücksichtigt wird. Der Einfachheit halber verwenden wir im folgenden diese simplifizierte Variante. U kann man weitgehend statisch berechnen, sobald man eine repräsentative, große Dokumentenkollektion hat. Ein neu eingefügtes Dokument d (d.h. ein m×1-Spaltenvektor) muß dann zur Indexierung in den Themen-Vektorraum abgebildet werden, und zwar durch die Transformation d' = U^T × D, und der resultierende r×1-Spaltenvektor d' wird zum Index hinzugefügt, d.h. V^T wird um eine Spalte erweitert (*Folding-in*).

Eine Query q, die als m×1-Spaltenvektor aufgefasst werden kann, wird dann zunächst in eine Query q' in den Themen-Vektorraum transformiert, und zwar durch q' = U^T × q.

Dann wird q' aufgrund des Index V^T evaluiert (mit der verwendeten Ähnlichkeitsfunktion des r-dimensionalen Themen-Vektorraums), und die dadurch erzeugte Rangliste von Dokumenten ist das Anfrageresultat. Im einfachsten Fall würde z.B. das Cosinus-Maß, das Skalarprodukt oder die Euklidische Distanz zwischen q' und den Spalten von V^T (also den Dokumenten) verwendet.

Beispiel:

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

$$A = \begin{pmatrix} 1&2&1&5&0&0&0\\ 1&2&1&5&0&0&0\\ 1&2&1&5&0&0&0\\ 0&0&0&0&2&3&1\\ 0&0&0&0&2&3&1 \end{pmatrix} = \begin{pmatrix} 0.58&0.00\\ 0.58&0.00\\ 0.58&0.00\\ 0.00&0.71\\ 0.00&0.71 \end{pmatrix} \times \begin{pmatrix} 9.64&0.00\\ 0.00&5.29 \end{pmatrix} \times \begin{pmatrix} 0.18&0.36&0.18&0.90&0.00&0.00&0.00\\ 0.00&0.00&0.00&0.00&0.53&0.80&0.27 \end{pmatrix}$$

U Δ V^T

Die Anfrage q = $(0\ 0\ 1\ 0\ 0)^T$ wird in
q‘ = U^T q = $(0.58\ \ 0.00)^T$ transformiert und gegen V^T evaluiert.
Ein neues Dokument d8 = $(1\ 1\ 0\ 0\ 0)^T$ wird in
d8‘ = U^T d8 = $(1.16\ \ 0.00)^T$ transformiert und an V^T angefügt.

SVD als Regressionsverfahren und für approximatives LSI

Wenn der Rang r von A nicht hinreichend klein ist, also die Anzahl der “Themen" zu groß ist, kann man die Dimensionalität des Themen-Vektorraums künstlich auf k < r beschränken, indem man nur die k größten Singulärwerte von A und die zugehörigen Eigenvektoren betrachtet. Dadurch berücksichtigt man die “ausgeprägtesten" Themen und “stärksten" Zusammenhänge zwischen Features und Themen, was durch den folgenden Satz formal untermauert wird.

Satz:
Sei A eine m×n-Matrix mit Rang r und sei $A_k = U_k \times D_k \times V_k^T$, wobei die k×k-Diagonalmatrix D_k die k größten Singulärwerte von A enthält und die m×k-Matrix U_k und die k×n-Matrix V_k^T 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 Fehler $\|A-C\|_F^2 = \sum_{i=1}^{m}\sum_{j=1}^{n}(A_{ij}-C_{ij})^2$ minimiert (die Frobenius-Norm).

Zur Illustration:
Die SVD kann als affine Transformation der Dokumentvektoren aufgefaßt werden (also als Kombination von Drehung und Streckung/Stauchung). Bei der Beschränkung auf die k größten Singulärwerte erfolgt quasi eine Projektion auf die k aussagekräftigsten Dimensionen (diejenigen mit der größten Varianz der Datenpunktkoordinaten).
SVD verallgemeinert die Methode der kleinsten Quadrate zur linearen Regression. Im einfachsten Fall betrachtet man m Punkte (x_1, y_1), ..., (x_m, y_m) im R^2 und versucht eine lineare Funktion f(x)=ax+b zu finden, die den quadratischen Fehler $\sum_{i=1..m}(y_i - f(x_i))^2 = \sum_{i=1..m}(y_i - (ax_i + b))^2$ minimiert. Hierbei sind die Werte für alle x_i und y_i gegeben, sozusagen als Messwerte, und die Koeffizienten a und b sind unbekannt. Durch partielle Differenzierung und Nullstellenbestimmung der 1. Ableitungen kann man a und b berechnen, wie es schon Gauß vor mehr als 150 Jahren getan hat. In Matrixschreibweise ist dies mit A=(a b), $X=\begin{pmatrix} x_1 & \dots & x_m \\ 1 & \dots & 1 \end{pmatrix}$ und $Y=(y_1 \dots y_m)$ äquivalent dazu, $\|AX-Y\|_F^2$ zu minieren. Dies zeigt die enge Verwandtschaft zur SVD, die ihrerseits die lineare Regression auf m Punkte des R^n und die Bestimmung einer k-dimensionalen Hyperebene mit kleinestem quadratischem Fehler verallgemeinert.

Bei der Indexierung von Dokumenten und der Ausführung von Queries im - nunmehr k-dimensionalen Themenraum - spielen dann einfach U_k und V_k die Rolle von U und V.

Beispiel:
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

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

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

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

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

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

Anfrage q: baking bread → $q = (1\ 0\ 1\ 0\ 0\ 0)^T$
Transformation in den Themenraum mit k=3 → $q' = U_k^T \times q \approx (0.5340\ \ -0.5134\ \ 1.0616)^T$
Skalarprodukt-Ähnlichkeit im Themenraum mit k=3:
sim (q, d1) = $(V_k^T)_{*1}^T \times q' \approx 0.86$ sim (q, d2) = $(V_k^T)_{*2}^T \times q' \approx -0.12$
sim (q, d3) = $(V_k^T)_{*3}^T \times q' \approx -0.24$ 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 → $d6' = U_k^T \times d6 \approx (0.5\ \ -0.28\ \ -0.15)^T$
d6‘ als neue Spalte an V_k^T anhängen

Zusammenfassende Bewertung von LSI

LSI ist ein elegantes, mathematisch wohlfundiertes Modell, das in Benchmarks auf homogenen Korpora wie z.B. Patentsammlungen, Wirtschaftsnachrichten oder Bibliotheken sehr gute Ergebnisse bzgl. Präzision und Ausbeute liefert. Dabei wird die approximative (Regressions-) Variante mit Werten von k in der Größenordnung von 100 verwendet. Diese Variante liefert deutlich bessere Ergebnisse als die mit dem vollen Rang r, da sie gewissermaßen Rauschen unterdrückt und Korrelationen kompakter zu Themen zusammenfasst.

LSI kann sogar für multilinguales IR verwendet werden, wenn man eine hinreichend große Startmenge von Dokumenten hat, die in mehreren Sprachen verfügbar sind. Dazu werden die mehrsprachigen Varianten desselben Dokument jeweils zu einem einzigen virtuellen Dokument konkateniert, und die SVD wird durch diese Startmenge berechnet. Dadurch kommen die starken Korrelationen zwischen den Wörter desselben Begriffs in mehreren Sprachen implizit im selben Thema zum Ausdruck. Weitere Dokumente können dann auch nur in einer Teilmenge der unterstützten Sprachen

(z.B. einer einzigen Sprache) verfügbar sein und werden per Folding-in in den Index eingefügt. Anfragen in einer beliebigen der unterstützten Sprachen werden auf den latenten Themenraum abgebildet und können somit auch Treffer in anderen Sprachen finden.

Für Web-Retrieval hat sich LSI nicht bewährt, da man es dort mit einem hochgradig heterogenen, terminologisch extrem streuenden Korpus zu tun hat. Hier ist es sehr schwer, einen brauchbaren Wert für die Zahl k der berücksichtigten Singulärwerte zu finden.

Ergänzende Literatur:

A. Arasu, J. Cho, H. Garcia-Molina, A. Paepcke, S. Raghavan: Searching the Web, ACM Transactions on Internet Technology, Vol.1 No.1, 2001

Text REtrieval Conference, http://trec.nist.gov/

S. Brin, L. Page: The Anatomy of a Large-Scale Hypertextual Web Search Engine, WWW Conference, 1998

A.Heydon, M. Najork: Mercator: A Scalable, Extensible Web Crawler, WWW Conference, 1999

Ronald Fagin, Amnon Lotem, Moni Naor: Optimal Aggregation Algorithms for Middleware, Journal of Computer and System Sciences Vol. 66, 2003

M.W. Berry, S.T. Dumais, G.W. O'Brien: Using Linear Algebra for Intelligent Information Retrieval, SIAM Review Vol.37 No.4, 1995

W.H. Press: Numerical Recipes in C, Cambridge University Press, 1993, available online at http://www.nr.com/

G.H. Golub, C.F. Van Loan: Matrix Computations, John Hopkins University Press, 1996

Christos H. Papadimitriou, Prabhakar Raghavan, Hisao Tamaki, Santosh Vempala: Latent Semantic Indexing: A Probabilistic Analysis, PODS Conference, 1998

Yossi Azar, Amos Fiat, Anna R. Karlin, Frank McSherry, Jared Saia: Spectral Analysis of Data, ACM Symposium on Theory of Computing (STOC), 2001