

Information Retrieval in der Bioinformatik

Alexander Hinneburg

5.5.08, SS 2008

1 Überblick über die Vorlesung

Vorlesungsinhalt

- Grundlagen des Information Retrievals
- Technische Unterstützung, Invertierte Listen
- Evaluation von IR-Systemen, Aufbau von Test-Datensätzen
- Themenmodelle
 - Latent Semantic Indexing (LSI)
 - Probabilistic Latent Semantic Indexing (PLSI)
 - Modellierung natürlicher Sprache für IR
- Schnelle Duplikatsuche mit Locally Sensitive Hashing (LSH)
- IR in wissenschaftlichen Texten aus den Lebenswissenschaften
- Verknüpfung von Textsuche und Analyse von Experimentdaten
- IR in Experimentdaten
 - Suche in Genomsequenzen
 - Suche in Proteomics-Experimenten
 - Suche in NMR-Spektren
 - Suche in Molekülstrukturen
 - ...

2 Organisatorisches

Organisation der Vorlesung

- Zeiten: Vorlesung Do. 10:15-11:45, Raum 1.30, Übung alle zwei Wochen Do. 12:15-13:45
- Forschungsprojekt in der vorlesungsfreien Zeit
 - etwa 4 Tage a 6 Stunden,
 - Termin nach Absprache
 - Forschungsprojekt soll in kleinen Teams bearbeitet werden

- Für die Prüfung gibt es keine Zulassungsvoraussetzungen
- Prüfungsrelevant ist der gesamte besprochene Inhalt der Vorlesung, einschließlich des Projektes
- Folien aus der Vorlesung und weiteres Lesematerial werden auf der Vorlesungs-Web-Seite http://www.informatik.uni-halle.de/~hinnebur/Lehre/BioIR_2008SS/ zugänglich sein.
- Projekt und Diplomarbeiten zu diesen Themen werden von mir vergeben und in Zusammenarbeit mit IPB, OntoChem, EBI, ... betreut.

3 Einleitung und Motivation

Überblick über die Einleitung

- Was wird gesucht?
- Informationen versus Daten
- Grundlegende Konzepte
- Suche als Prozess

Was ist Information Retrieval?

- Information Retrieval beschäftigt sich mit
 - der Repräsentation,
 - der Speicherung,
 - der Organisation und
 - dem Zugriff
 auf Informationseinheiten
- Information Retrieval orientiert sich am Bedürfnis des Anwenders
- Charakterisierung dieses Bedürfnisses ist kein einfaches Problem

Informationen versus Daten

- Bereitstellen von Daten (IR)
 - welche Dokumente enthalten die Schlüsselwörter der Anfrage
 - reicht meist nicht um die Anfrage zu beantworten.
- SQL Anfrage: exakt und vollständig
- IR Anfrage: Ähnlichkeiten, kleine Fehler
 - Texte, natürliche Sprache ist nicht gut strukturiert und kann mehrdeutig sein
 - Semantik ist nicht exakt
 - Gegensatz: Datenbank-Schemata sind wohl-definiert

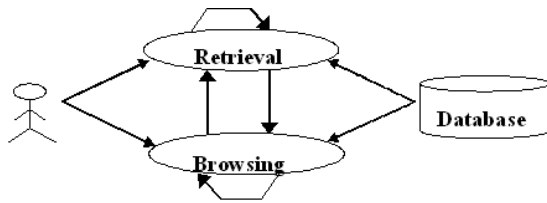


Abbildung 1: Unterscheidung von Information Retrieval und Browsen

Informationen versus Daten

- Bereitstellen von Daten → Datenbanken
- IR System
 - muss Dokumente „interpretieren“ und entsprechend der Relevanz ordnen
 - syntaktische und semantische Informationen müssen aus Dokumenten extrahiert werden
 - Definition von Relevanz ist sehr wichtig
- Probleme
 - Wie werden die relevanten Informationen extrahiert?
 - Wie werden die gefundenen Dokumente im Ergebnis geordnet?
- Ziel
 - Antwort enthält alle relevanten Dokumente mit möglichst wenigen nicht-relevanten Dokumenten
- Effektivität wird experimentell auf Benchmark-Anfragen ermittelt, deren Antworten durch Experten bestimmt wurden
- TREC-Konferenz: ausgereiftes experimentelles Methoden-Arsenal zum Bewerten von IR-Systemen entwickelt.

3.1 Grundlegende Konzepte

Anwender Aufgabe

- Anwender übersetzt dem IR System die Aufgabe in eine Anfrage (Schlüsselwörter+Bedingungen)
- Anwender mit unbestimmten, allgemeinem Interesse
 - will einfach herumstöbern
 - **stöbern** anstatt **suchen** (klare Unterscheidung)

Logische Sicht auf Dokumente

- Texte werden durch Menge von Indexwörtern repräsentiert ⇒ logische Sicht auf die Dokumente
 - handverlesene Indexmenge
 - automatisch generiert
- Falls $\{\text{Indexwörter}\} = \{\text{alle Wörter}\} \Rightarrow \text{Volltext}$

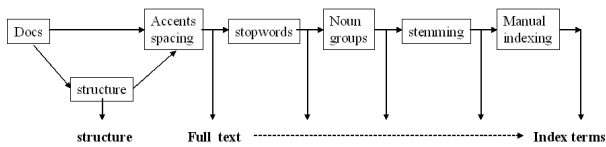


Abbildung 2: Die verschiedenen logischen Sichten variieren schrittweise von Volltext zur Indexterm-Ebene

- Textoperationen
 - Stopwörter entfernen
 - Stemming
 - Substantiv-Gruppen markieren, Part-of-Speech (POS)-Tagging

Dokument Vorverarbeitung

- Lexikalische Analyse
 - Behandlung von Zahlen, Bindestrichen, Punktierungen, Groß/Kleinschreibung
 - Implementierung einfacher Regeln + Liste von Ausnahmen
- Eliminierung von Stop-Wörtern
 - Wenig selektive Wörter entfernen
 - Listen für Standard-IR-Anwendungen
- Stemming
 - Einfache grammatische Regeln (sprachabhängig)
- Auswahl der Indexterme
 - Meist Substantive und Wortgruppen
- Konstruktion eines Thesaurus
 - Hilfe für Anwender, Anfrageerweiterung

Logische Sichten auf Dokumente

Hilfreiche Seiten mit Werkzeugen zur Dokumentvorverarbeitung

- Stopwortlisten, Stemmer für verschiedene Sprachen
<http://www.unine.ch/info/clef/>
- Deutsche und Englische Wörterbücher
<http://dict.tu-chemnitz.de/>
- Thesauri
 - <http://www.openthesaurus.de/>
 - <http://thesaurus.reference.com/>
 - <http://wordnet.princeton.edu/>

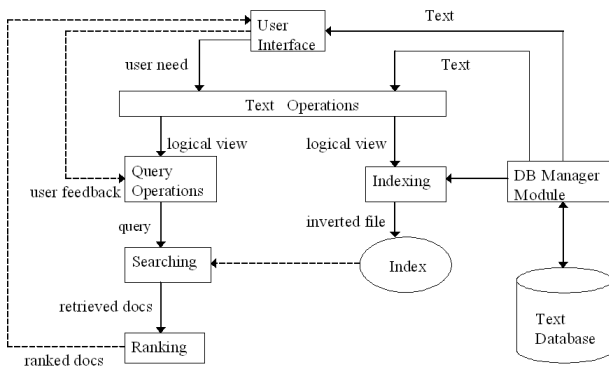


Abbildung 3: Das Zusammenspiel der verschiedenen Teile eines IR-Systems

Textsuche mit Indexstrukturen

- Wann lohnt es sich zu einer Anfrage alle Dokumente sequentiell durchsuchen?
 - Kleine Textsammlungen (ein paar MB)
 - Sehr viele Änderungen
- Indexstrukturen
 - Mehraufwand lohnt sich erst für große Textesammlungen
 - Texte sollten mindestens semi-statisch sein
- Typen
 - Inverted Files
 - Suffix Bäume
 - Signature Files

Suche als Prozess

3.2 Anfragen

Anfrage-Formulierung

- IR Anfragesprachen erlauben Ranking
- Anfragen mit Schlüsselwörtern
 - Einzelwörter
 - Kontext Anfragen
 - Boolesche Anfragen
 - Natürliche Sprache
- Textmuster
- Strukturanfragen

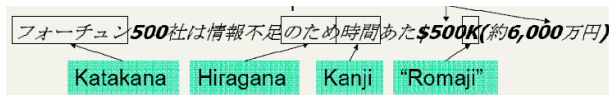


Abbildung 4: Beispiel für japanischen Text

Anfragen mit Schlüsselwörtern

- Text in Einheiten zerlegen
 - Eingabe: „Freunde, Römer, Landsleute ...“
 - Ausgabe: {„Freunde“, „Römer“, „Landsleute“}
- Wörter als Grundeinheit, was ist ein Wort?
- Sprachprobleme
 - Chinesisch, Japanisch: keine Leerzeichen
 - Japanisch: mehrere Alphabete gemischt

Sprachprobleme – Normalisierung

- „rechts-nach-links“ Sprachen, (Hebräisch, Arabisch), gemischt mit „links-nach-rechts“ Text (Datum, Geld, ...)
- Groß/Kleinschreibung, Mehrdeutigkeiten
 - „Ich habe liebe (g/G)enossen.“
- Punktierung
- Zahlen

Rechtschreibprüfung

- Erweitere Anfragen
 - z.B. um Terme innerhalb Edit-Distanz 2
 - Erweitere zur Anfragezeit (Warum nicht zur Indexzeit?)
- Rechtschreibprüfung ist teuer, verlangsamt Anfrage bis zu Faktor 100
 - Wird eingesetzt, wenn fast nichts gefunden wird
 - Was, wenn Dokumente Schreibfehler enthalten?

Soundex

Heuristiken um Anfrage um phonetische Äquivalente zu erweitern. Diese sind sprachspezifisch und hauptsächlich für Namen geeignet. Typischer Algorithmus, → 4 Zeichensignatur

1. Erhalte ersten Buchstaben des Wortes
2. Ersetze
 - A E I O U H W Y zu 0
 - B F P V zu 1

- C G J K Q S X Z zu 2
- D T zu 3
- L zu 4
- M N zu 5
- R zu 6

3. Entferne alle aufeinanderfolgenden Duplikate
4. Entferne alle Nullen
5. Liefere die ersten vier Positionen (bei Bedarf mit Nullen auffüllen)

4 Bio-IR Problemstellungen und Anwendungsbeispiele

IR-Problemstellungen in der Bio- und Chemoinformatik

- Bisher arbeiteten Bio-Wissenschaftler hauptsächlich Einzelfall orientiert, d.h. einzelne Proteine, einzelne Metabolite, ...
- Zunehmend wird mit komplexeren Dateneinheiten gearbeitet: Spektrum, Gen- oder Protein-Expressionsprofil, Metabolitprofil, ...
- IR-Methoden sind zur Suche und Evaluation in Lebenswissenschaften relevant → teilweise Automatisierung von Expertentätigkeiten
- Unscharfe Suche in wissenschaftlichen Datenbanken als wichtiges Werkzeug um Hypothesen zu generieren
- Analogie zu Texten ist häufig eine hilfreiche Metapher um komplexe Situationen verständlich zu machen.

4.1 IR in den Lebenswissenschaften

IR in wissenschaftlichen Texten aus den Lebenswissenschaften

- Wichtige Sammlungen von Literaturdaten
 - Medline/Pubmed: <http://www.ncbi.nlm.nih.gov/pubmed/>
 - DIMDI: Deutsches Institut für Medizinische Dokumentation und Information, <http://www.dimdi.de>
- Verknüpfung von Literaturdaten ist ein spannendes Forschungsgebiet
- GO-Pubmed (TU-Dresden): Verknüpfung von Gene-Ontology (GO) und Pubmed-Suche, die Suchtreffer werden durch den Kategorienbaum der GO zu sinnvollen Unterthemen gruppiert. <http://www.gopubmed.org/>
- Neue Verknüpfung von bekannten Fakten: finden vom unbekannten öffentlichem Wissen
- Schneller Transfer von neuen Methoden quer durch verschiedene Zweige der Lebenswissenschaften

4.2 Textsuche und Analyse von Experimentaldaten

Verbindung von Textsuche und Experimentaldatenanalyse

- In einigen Fällen fehlt bei der Experimentaldatenanalyse Hintergrundwissen
- Idee: nutzte Textsuche in Literaturdaten aus den Lebenswissenschaften als Ersatz für Hintergrundwissen von Experten
 - nutzte einfache Statistiken über Textsuchen um Relevanz von exprimierten Genen, Proteinen, ... für eine bestimmte Krankheitsdiagnose zu schätzen.
 - ähnlich zu Google Count zu bestimmten Themen, z.B. 3.950 für (Informatik OR Computer OR Data) AND “Alexander Hinneburg“, 43.300 für (Informatik OR Computer OR Data) AND “Paul Molitor“ $\Rightarrow$ Prof. Molitor ist ein bekannterer Informatiker als Alexander Hinneburg.

4.3 IR in Experimentaldaten

Textähnliche Suche in Experimentaldaten

- Beispiel Proteomics: ein Experiment zeigt an welche Proteinfragmente in einer Gewebeprobe vorhanden sind
- Anfrage: welche Gewebe passen von ihrem Proteinprofil gut zu einem gegebenen Profil
- Übersetzung der Anfrage in Proteinfragmente
- Behandle ein Experimente wie ein Dokument und die detektierten Proteinfragmente wie Indexterme.
- Gib eine Trefferliste von Proteomics-Experimenten geordnet nach Relevanz aus.

5 Grundlagen des Information Retrievals

5.1 Boolesches Modell

Boolesches Modell

- Dokument wird als Menge von Worten repräsentiert, Reihenfolge und Häufigkeit der Worte im Dokument spielt keine Rolle
- Anfrage besteht aus Stichworten, die logischen Operationen verbunden sein können
- Logisches NOT allein oder mit OR verknüpft ist problematisch, weil die Ergebnismenge sehr groß werden kann.
- Ausweg: NOT nur Kombination mit AND anbieten

Boolesches Modell, Formale Einführung

- Dokumente $d_j \in D = \{d_1, \dots, d_n\}$ werden als Spaltenvektoren einer 0/1-Matrix repräsentiert
- Vokabular $V = \{t_1, \dots, t_m\}$
- Matrix $W \in \{0, 1\}^{m \times n}$.

	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

1 if play contains word, 0 otherwise

Abbildung 5: Beispiel für eine Boolesche Term-Dokument-Matrix

- Term-Dokument-Matrix

$$W = \begin{pmatrix} w_{1,1} & \dots & w_{1,n} \\ \vdots & & \vdots \\ w_{m,1} & \dots & w_{m,n} \end{pmatrix}$$

- Element

$$w_{i,j} = \begin{cases} 1 & \text{Term } t_i \text{ kommt in Dokument } d_j \text{ vor} \\ 0 & \text{sonst} \end{cases}$$

Term-Dokument Inzidenz Matrix

Beispiel

- Welches Stück von Shakespeare enthält die Wörter Brutus, Caesar aber NICHT Calpurnia?
- Ansatz: finde alle Stücke mit Brutus und Caesar und entferne alle die Calpurnia enthalten
 - Langsam für große Sammlungen
 - NICHT Calpurnia ist nicht trivial

Inzidenz Vektoren

- 0/1 Vektoren für jeden Term
- Anfrage beantworten: bit weises AND der Vektoren für Brutus, Caesar und Calpurnia (komplementiert)

$$110100 \wedge 110111 \wedge 101111 = 100100$$

- Antworten
 - Antony and Cleopatra
 - Hamlet

Boolsches Modell, Fazit

- Vorteil: einfach verständlich
- Nachteile
 - Keine Abstufung der Relevanz
 - Informationsbedürfnis schwer in logische Formel zu übersetzen
- Problem: kein teilweises Erfüllen möglich
- Folge: entweder zu wenig oder zu viele Ergebnistreffer

5.2 Vektorraum Modell

Vektorraummodell, Formale Einführung

- Dokumente werden durch Vektoren beschrieben

$$\vec{d}_j = (w_{1,j}, \dots, w_{n,j})^t, w_{i,j} \geq 0$$

- Anfragen sind auch Vektoren

$$\vec{q} = (w_{1,q}, \dots, w_{n,q})^t, w_{i,q} \geq 0$$

- Similarity, Ähnlichkeit
 - Cosinus des Winkels zwischen q und d, Cosine Similarity

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

Vektorraummodell, Fazit

- Vorteile
 - Bessere Retrieval Ergebnisse
 - Teiltreffer möglich
 - Relevante Dokumente nach Ähnlichkeit sortiert
- Nachteil: unabhängige Termgewichte
- Fazit
 - im klassischen Rahmen schwer zu schlagen
 - sehr populär

	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

1 if play contains
word, 0 otherwise

Abbildung 6: Beispiel für eine Boolesche Term-Dokument-Matrix

6 Invertierte Listen

6.1 Motivation für Indexstrukturen

Textsuche mit Indexstrukturen

- Wann lohnt es sich zu einer Anfrage alle Dokumente sequentiell durchsuchen?
 - Kleine Textsammlungen (ein paar MB)
 - Sehr viele Änderungen
- Indexstrukturen
 - Mehraufwand lohnt sich erst für große Textesammlungen
 - Texte sollten mindestens semi-statisch sein
- Typen
 - Inverted Files
 - Suffix Bäume
 - Signature Files

Beispiel für Anfragen

- Welches Stück von Shakespeare enthält die Wörter Brutus, Caesar aber NICHT Calpurnia?
- Ansatz: finde alle Stücke mit Brutus und Caesar und entferne alle die Calpurnia enthalten
 - Langsam für große Sammlungen
 - NICHT Calpurnia ist nicht trivial

Term-Dokument Inzidenz Matrix

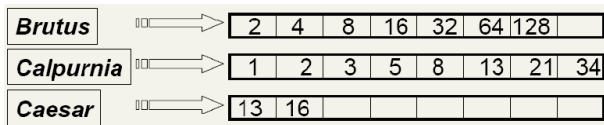


Abbildung 7: Schematische Darstellung eines Beispiels für invertierte Listen

Inzidenz Vektoren

- 0/1 Vektoren für jeden Term
- Anfrage beantworten: bit weises AND der Vektoren für Brutus, Caesar und Calpurnia (komplementiert)

$$110100 \wedge 110111 \wedge 101111 = 100100$$

- Antworten
 - Antony and Cleopatra
 - Hamlet

Größere Sammlungen

- Annahmen
 - $n = 10^6$ Dokumente, jedes mit etwa 10^3 Termen
 - Durchschnittlich 6 Bytes/Term mit Leerzeichen/Punktuation
 - * 6GB Textdaten in den Dokumenten
 - $m = 500 \cdot 10^3$ verschiedene Terme
- Matrix kann nicht materialisiert werden
 - $500 \cdot 10^3 \cdot 10^6 = 1/2 \cdot 10^{12}$ Einträge
 - Nicht mehr als 10^9 Eins-Einträge, Warum?
- Besserer Ansatz: Repräsentiere nur die Einsen

6.2 Invertierte Listen: Darstellung

Invertierte Listen (Inverted Files)

Zu jedem Term t speichere alle Dokumente, die t enthalten.

- Felder oder Listen?
- Was passiert, wenn das Wort Caesar zu Dokument 14 hinzugefügt wird?

Invertierte Listen mit Listenstruktur

Listen werden meist Feldern vorgezogen

- Dynamischer Platz
- Einfügen von Termen in Dokumenten ist leicht
- Preis: Mehrverbrauch an Platz für Zeiger

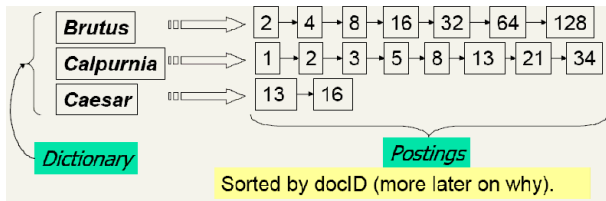


Abbildung 8: Invertierte Liste mit Listenstruktur, Postings sind sortiert.

Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

6.3 Invertierte Listen: Aufbau

Index Schritte

Liste von Paaren (Termen, Dokument ID)

Dokument 1 I did enact Julius Caesar. I was killed i' the Capitol; Brutus killed me.

Dokument 2 So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious.

⇒ Sortierte Liste von (Term, Dokument ID)-Paaren.

Sortieren der Terme

Hauptschritt für Indexing

Mehrfache Einträge

- Mehrfache Einträge in einzelnen Dokumenten zusammenfassen
- Häufigkeiten hinzufügen

Teile Ergebnis in Dictionary und Postings

Term	Doc #	Term	Doc #
I	1	ambitious	2
did	1	be	2
enact	1	brutus	1
julius	1	brutus	2
caesar	1	capitol	1
I	1	caesar	1
was	1	caesar	2
killed	1	caesar	2
i'	1	did	1
the	1	enact	1
capitol	1	hath	1
brutus	1	I	1
killed	1	I	1
me	1	i'	1
so	2	it	2
let	2	julius	1
it	2	killed	1
be	2	killed	1
with	2	let	2
caesar	2	me	1
the	2	noble	2
noble	2	so	2
brutus	2	the	1
hath	2	the	2
told	2	told	2
you	2	you	2
caesar	2	was	1
was	2	was	2
ambitious	2	with	2

Term	Doc #	Term	Doc #	Freq
ambitious	2	ambitious	2	1
be	2	be	2	1
brutus	1	brutus	1	1
brutus	2	brutus	2	1
capitol	1	capitol	1	1
caesar	1	caesar	1	1
caesar	2	caesar	2	2
caesar	2	caesar	2	2
did	1	did	1	1
enact	1	enact	1	1
hath	1	hath	2	1
I	1	I	1	2
I	1	i'	1	1
i'	1	it	2	1
it	2	julius	1	1
julius	1	killed	1	2
killed	1	let	2	1
killed	1	me	1	1
let	2	noble	2	1
me	1	so	2	1
noble	2	the	1	1
so	2	the	2	1
the	1	told	2	1
the	2	you	2	1
told	2	was	1	1
you	2	was	2	1
was	1	with	2	1
with	2			

Term	Doc #	Freq	Term	N docs	Tot Freq	Doc #	Freq
ambitious	2	1	ambitious	1	1	2	1
be	2	1	be	1	1	2	1
brutus	1	1	brutus	1	1	1	1
brutus	2	1	brutus	2	2	2	1
capitol	1	1	capitol	1	1	1	1
caesar	1	1	caesar	2	3	1	1
caesar	2	2	caesar	1	1	2	2
caesar	2	2	caesar	1	1	1	1
did	1	1	did	1	1	1	1
enact	1	1	enact	1	1	1	1
hath	2	1	hath	1	1	2	1
I	1	2	I	1	1	1	2
i'	1	1	i'	1	1	1	1
it	2	1	it	1	1	2	1
julius	1	1	julius	1	1	1	1
killed	1	2	killed	1	2	1	1
let	2	1	let	1	1	1	2
me	1	1	me	1	1	1	1
noble	2	1	noble	1	1	2	1
so	2	1	so	1	1	2	1
the	1	1	the	2	2	2	1
the	2	1	told	1	1	1	1
told	2	1	you	1	1	2	1
you	2	1	you	2	2	2	1
was	1	1	was	1	1	2	1
was	2	1	was	2	2	1	1
with	2	1	with	1	1	2	1

Abbildung 9: Aufteilung der Sortierten Liste in Dictionary und Postings

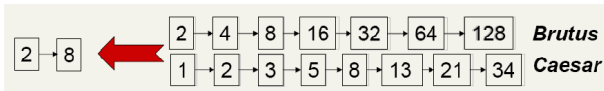


Abbildung 10: Zusammenfassung der sortierten Listen, Sort-Merge

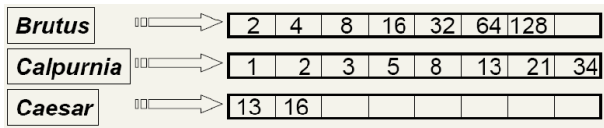


Abbildung 11: Beste Bearbeitungsreihenfolge zum Schneiden der Listen?

6.4 Invertierte Listen: Anfragebearbeitung

Anfrageverarbeitung

Anfrage: Brutus AND Caesar

- Finde Postings von Brutus und Caesar
- Fasse beide sortierte Listen zusammen, durchlaufe beide Listen simultan
- Wenn Listenlängen x und y sind, Mengenschnitt in $O(x + y)$

Allgemeinere Anfragen

- Wie muß das Zusammenfassen geändert werden für
 - Brutus AND NOT Caesar
 - Brutus OR Caesar
 - Brutus OR NOT Caesar
- Bleibt die Laufzeit bei $O(x + y)$
- Was passiert bei allg. Booleschen Anfragen?
 - (Brutus or Caesar) and not (Antony or Cleopatra)
 - Kann man immer in linearer Zeit zusammenfassen
 - Linear in was?

Anfrage Optimierung

- Was ist die beste Bearbeitungsreihenfolge?
- Bearbeite in aufsteigender Häufigkeit (AND-Verknüpfung)
- Ausführung: (Caesar and Brutus) and Calpurnia
- Schätze Größe eines OR als Summe der Einzelhäufigkeiten (Konservativ)

Beispiel Bearbeitungsreihenfolge

Empfehle eine Reihenfolge

<i>(tangerine OR trees) AND (marmalade OR skies) AND (kaleidoscope OR eyes)</i>	Term	Freq
	eyes	213312
	kaleidoscope	87009
	marmalade	107913
	skies	271658
	tangerine	46653
	trees	316812

Abbildung 12: Ermittle eine gute Bearbeitungsreihenfolge.

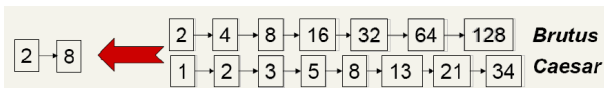


Abbildung 13: Merge-Operation durchläuft beide Listen simultan, lineare Laufzeit in Größe der Listen

7 Invertierte Listen, Optimierungen

7.1 Skippointer

Wiederholung, Invertierte Liste

- Anfrage mit zwei Termen (logisches UND)
 - Merge-Operation
 - Wenn die Listen die Größen x und y haben, ist die Laufzeit $O(x + y)$.
- Kann die Laufzeit verbessert werden?
 - Asymptotisch nicht
 - Reale Laufzeit ja, wenn der Index sich nicht oft ändert

Schnelleres Zusammenfassen durch Skip Pointer

- überspringe Einträge, die nicht in das Ergebnis kommen
- Warum ist das sinnvoll?
- Um Dokument IDs zu überspringen, die nicht im Ergebnis auftauchen werden.
- Wie?
- Wo kommen die Skip Pointer am besten hin?

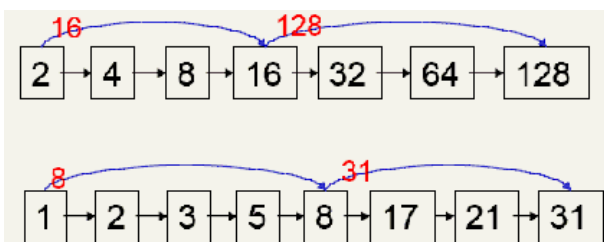


Abbildung 14: Beispiel für Skip-Pointer

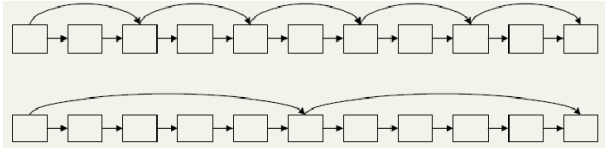


Abbildung 15: Zielkonflikt zwischen Anzahl der erfolgreichen Sprünge über Skippointer und Anzahl der Zusatzvergleiche.

Wohin Skip Pointer plazieren? (1/2)

Zielkonflikt

- Mehr Skips $\Rightarrow$ kürzere Sprünge $\Rightarrow$ wahrscheinlicher zu springen. Aber viele Zusatzvergleiche.
- Weniger Skips $\Rightarrow$ weniger Zusatzvergleiche, aber weitere Sprünge $\Rightarrow$ weniger erfolgreiche Skips

Wohin Skip Pointer plazieren? (2/2)

- Einfache Heuristik
 - Für Postings der Länge L , plaziere $\sqrt{L}$ gleichverteilte Skip Pointer
 - Diese Heuristik ignoriert die Verteilung der Anfrageterme.
 - Einfach zu implementieren, falls es wenig Änderungen im Index gibt
- Wie könnte die Verteilung der Anfrageterme auf die Dokumente genutzt werden?

7.2 Komprimierung der Posting-Listen

Komprimierung der Posting-Listen, Rückblick

- Angenommen, wir haben
 - $n = 10^6$ Dokumente, jedes mit etwa 10^3 Termen.
 - Durchschnittlich werden 6 Bytes pro Term benötigt
 - Dies macht 6GB Daten
- Angenommen, es gibt, $m = 500 \cdot 10^3$ unterschiedliche Terme
- Term-Dokument Matrix kann nicht voll repräsentiert werden
 - $0.5 \cdot 10^{12}$ 0/1-Einträge, aber nur 10^9 Einsen
 - Deshalb werden invertierte Listen genutzt
- Wo investieren wir in Speicherplatz?

Wo investieren wir in Speicherplatz?

Zeiger: zwei widersprüchliche Ziele

- Term Calpurnia kommt möglicherweise nur einem Dokument aus einer Million Dokumenten vor
 - Platz für Zeiger $\log_2 10^6 \approx 20$ Bits
- Term "the" kommt in nahezu jedem Dokument vor
 - 20 Bit sind sehr teuer,
 - 0/1 Vektor wäre in dem Fall besser

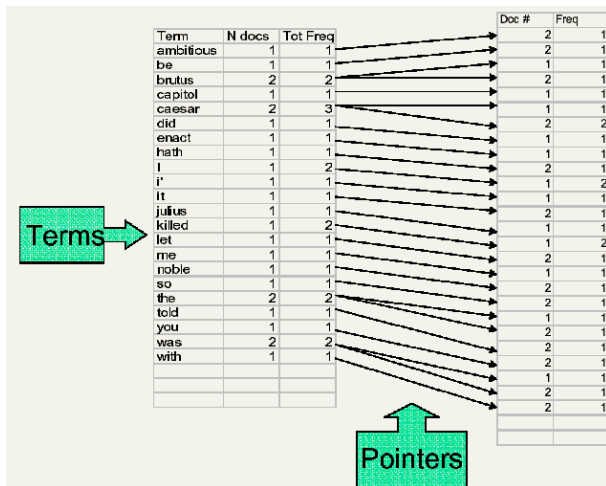


Abbildung 16: Wo wird der meiste Speicherplatz ausgegeben?

Komprimierung der Postings

- Dokumentlisten werden sortiert gespeichert *Brutus*: 33,47,154,159,202
- Deshalb: ausreichend die Lückengröße zu speichern *Brutus*: 33,14,107,5,43
- Hoffnung: Lücken brauchen weniger als 20 Bit

Variable Kodierung der Lücken

- Ziel
 - Für Terme wie Calpurnia ≈ 20 Bits pro Lücke ausgeben
 - Für Terme wie the ≈ 1 Bit pro Lücke ausgeben
 - Wenn die durchschnittliche Lückengröße $\log_2 G$ ist, dann soll $\approx \log_2 G$ Bits pro Lücke ausgeben werden
- Aufgabe
 - Kodiere jeden Integer (Lücke) im Durchschnitt mit so wenig Bits wie möglich

Gamma Kodes

- repräsentiere Lücke G als Paar $\langle L, O \rangle$, (Länge — Offset)
- Länge L wird unär kodiert und braucht $\lfloor \log_2 G \rfloor + 1$ Bits
- Der Offset $O = G - 2^{\lfloor \log_2 G \rfloor}$ wird binär dargestellt.
- Kode für G braucht $2\lfloor \log_2 G \rfloor + 1$ Bits
- z.B. 9 wird repräsentiert als $\langle 0001, 001 \rangle$
- Gamma Kode: Platz für Lücken, Doppelte Länge der Minimal-Repräsentation
- Verteilung der Lücken: Zipf-Verteilung angenommen

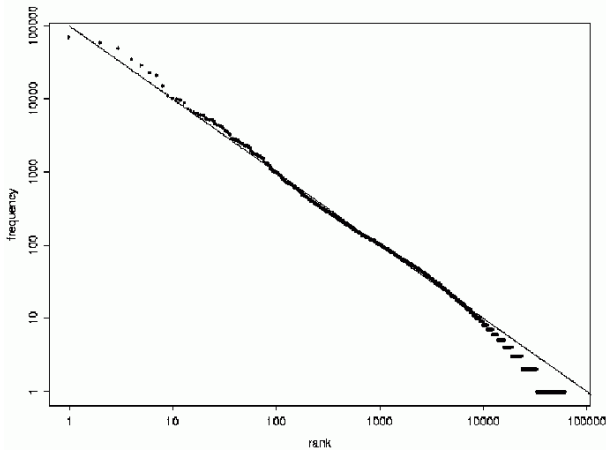


Abbildung 17: Beispiel für eine Zipf-Verteilung

Gamma Kode, Beispiele

$1 = 2^0 + 0 = 1$ $2 = 2^1 + 0 = 010$ $3 = 2^1 + 1 = 011$ $4 = 2^2 + 0 = 00100$ $5 = 2^2 + 1 = 00101$
 $6 = 2^2 + 2 = 00110$ $7 = 2^2 + 3 = 00111$ $8 = 2^3 + 0 = 0001000$ $9 = 2^3 + 1 = 0001001$ $10 = 2^3 + 2 = 0001010$
 $11 = 2^3 + 3 = 0001011$ $12 = 2^3 + 4 = 0001100$ $13 = 2^3 + 5 = 0001101$ $14 = 2^3 + 6 = 0001110$
 $15 = 2^3 + 7 = 0001111$ $16 = 2^4 + 0 = 000010000$ $17 = 2^4 + 1 = 000010001$

Zipf Verteilung

Platz Analyse der Postings für Zipf-verteilte Terme (1/2)

- Zipf's Gesetz
 - k -häufigster Term hat eine normalisierte Häufigkeit von $\approx 1/k$.
- Grobe Analyse des Platz für Postings
 - häufigster Term kommt in n Dokumente vor
 - * n Lücken der Länge 1
 - Zweithäufigster Term kommt in $n/2$ Dokumenten vor
 - * $n/2$ Lücken der Länge 2 ...
 - k -häufigster Term kommt in n/k Dok. vor
 - * n/k Lücken der Länge k ,
 - * Gamma Kodes: $2 \log k + 1$ Bits für jede Lücke
 - * Gesamtplatzbedarf: $\approx 2n/k \log k$ Bits für den k -häufigsten Term

Platz Analyse (2/2)

- Summiere über k von 1 bis $m = 500 \cdot 10^3$.
- Teile die Werte für k in Gruppen
 - Gruppe i besteht aus $2^{i-1} \leq k < 2^i$
 - Gruppe i hat 2^{i-1} Summanden, jeder trägt maximal $n/2^{i-1}$ bei ($n = 10^6$).
- Summiert man i von 1 bis 19, bekommt man 340 MBits ≈ 45 MB für die Postings

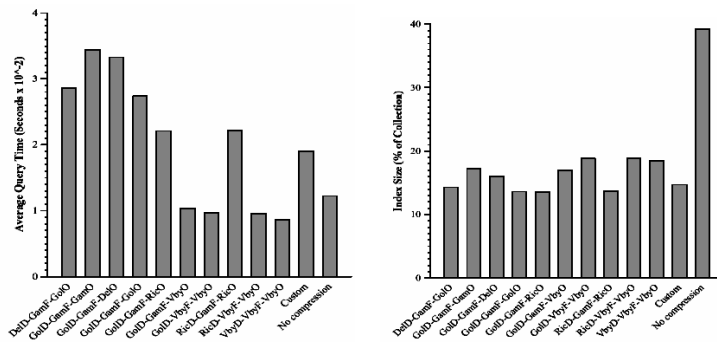


Abbildung 18: Anfragezeiten und Indexgrößen für verschiedene Kodierungen, 500 MB Daten, 10.000 Anfragen

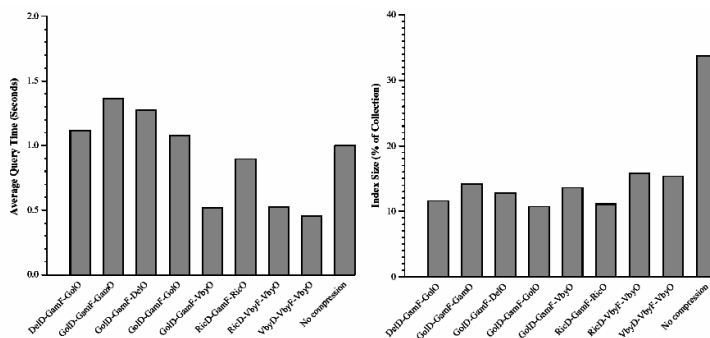


Abbildung 19: Anfragezeiten und Indexgrößen für verschiedene Kodierungen, 20 GB Daten, 25.000 Anfragen

- Gamma Codes sind nicht praktikabel, da Computer mit 16, 32 oder 64 Bits auf einmal rechnen $\Rightarrow$ langsam
- Word-aligned Kompression ist besser

Experimente (Scholer, Zobel, SIGIR 2002)

500 MB Daten, 10.000 Anfragen

Experimente (Scholer, Zobel, SIGIR 2002)

20 GB Daten, 25.000 Anfragen

7.3 Speicherung des Vokabulars

Wie groß ist das Vokabular?

- Wächst langsamer als die Korpusgröße
- Empirisches Model: $m = kN^b$
 - $b = 0.5, k = 30 \dots 100, N = \text{Anzahl der Wörter}$
- Beispiel
 - TREC Disk 1+2, 2GB, 750.000 Artikel, ≈ 500.000 Terme

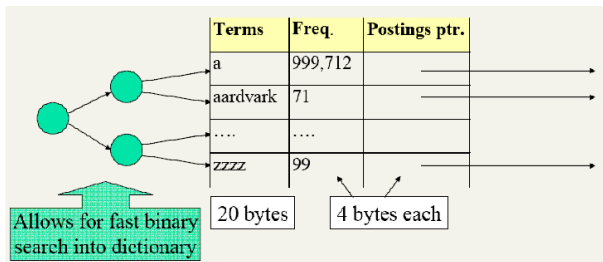


Abbildung 20: Naiver Ansatz zum Aufbau des Dictionary.

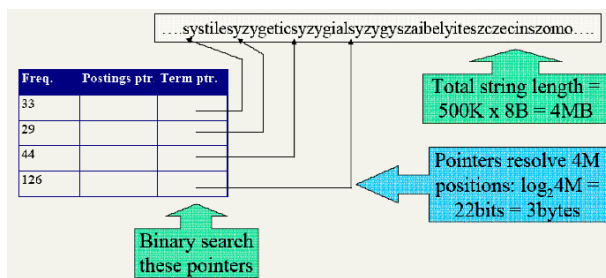


Abbildung 21: Verbessierter Ansatz zum Aufbau des Dictionary.

- Stemming verkleinert das Vokabular
- Indizierung aller Zahlen bläht stark auf
- Schreibfehler vergrößern das Vokabular

Wie groß ist das Dictionary

Erster Ansatz

- Feld mit festen Einträgen, $500 \cdot 10^3$ Terme; 28 Bytes pro Term = 14 MB

Feste Termgröße verschwendet Platz

- Meisten Terme sind kürzer als 20 Zeichen (durchschnittliche 8 Zeichen)
- Lange Terme passen nicht
- Komprimiere Termliste
 - Speichere Dictionary als einen langen String
 - Zeiger zum nächsten Wort zeigt das Ende an

Speicherplatz

- 4 Bytes pro Term für Häufigkeit
- 4 Bytes pro Term für Zeiger auf Postings
- 3 Bytes pro Termzeiger
- Durchschnittlich 8 Bytes pro Term im Termstring

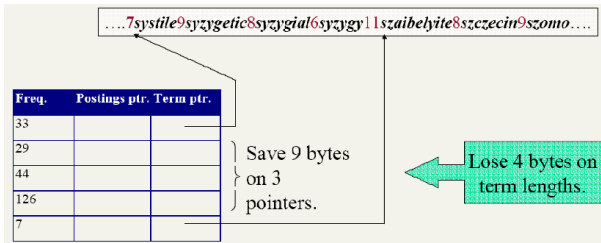


Abbildung 22: Blocking-Ansatz zum Aufbau des Dictionary.

– Jetzt mit durchschnittlich 11 Bytes/Term, statt 20 Bytes/Term

- Gesamt $500 \cdot 10^3$ Terme $\Rightarrow$ 9.5MB

Blocking

Speichere einen Zeiger zu jedem k -ten Term ($k=4$)

- Termlänge muss im String gespeichert werden

Blocking, Gewinn

- Ohne Blocking: 3 Byte/Zeiger
 - $3 * 4 = 12$ Byte für $k=4$ Zeiger
- Mit Blocking: $3+4=7$ Byte für 4 Zeiger
- Einsparung ≈ 0.5 MB
 - mit größerem k kann noch mehr gespart werden
 - Was für Nachteile tauchen mit größerem k auf?

Gesamtverbrauch für Speicher

- Durch größeres k kann der Zeigerspeicherverbrauch im Dictionary weiter gesenkt werden, auf Kosten der Suchzeit $9.5\text{MB} \rightarrow \approx 8\text{MB}$
- Zusammen mit den Postings für 45MB ist der Gesamtverbrauch 53MB für einen einfachen boolschen invertierten Index

Erweiterungen

- Extreme Kompression
 - Sortierte Wörter haben gemeinsames Präfix $\Rightarrow$ speichere nur die Unterschiede
- Große Dokumentsammlungen
 - Rankings als Antworten
 - Speichere nur Dokumente in Postings, die mit einer gewissen Wahrscheinlichkeit in einem Top-k Ranking auftauchen können

8 Evaluation von IR-Systemen

8.1 Allgemeine Ziele und Paradigmen

Evaluierung von IR Systemen

- Theoretische Evaluierung kaum möglich
- Evaluierung von IR Systemen ist experimentell

Zuverlässigkeit: dieselbe Untersuchung im gleichen Kontext sollte stets dieselben Ergebnissen liefern (Wiederholbarkeit)

Validität: die Beobachtungen sollten mit den tatsächlichen Verhältnissen übereinstimmen (Gültigkeit)

Stochastische Experimente: Meßwerte können variieren

- Performanzmaße für Effizienz
 - Speicherplatz
 - CPU-Zeit
 - Anzahl I/O-Operationen
 - Antwortzeiten

Evaluierung von IR Systemen

- Qualität
 - Relevanz der Antwort
- Annahmen
 - Die Systemantwort ist eine Menge von Objekten. Damit werden stärker strukturierte Antworten nicht berücksichtigt. Erweiterung auf lineare Anordnungen (Rangordnungen) leicht möglich
 - Die Relevanz eines Objekts bezüglich der Anfrage (Qualität) hängt nur von der Anfrage ab. Wechselseitige Abhängigkeiten zwischen Objekten bleiben dagegen unberücksichtigt. (wenn z. B. die Bedeutung eines bestimmten Dokumentes erst nach der Lektüre eines anderen Dokumentes erkannt wird).
- Arten von Relevanz
 - Situative Relevanz
 - Pertinenz
 - Objektive Relevanz
 - Systemrelevanz

Evaluierung von IR Systemen

- Standpunkte und Bewertungsmaße
 - IR-System liefert eine Rangordnung von Dokumenten
 - Benutzer geht diese sequentiell durch, bis ein bestimmtes Abbruchkriterium erfüllt ist.
 - Beispiele für Abbruchkriterien und Bewertungsmaße
 - * n Dokumente gesehen: $\#$ gesehene relevante Dokumente

- * n relevante Dokumente gesehen: # gesehene Dokumente
- * n nicht relevante Dokumente gesehen: # gesehene / # gesehene relevante Dokumente
- * n nicht relevante Dokumente in Folge gesehen: # gesehene / # gesehene relevante Dokumente

- Test-Referenzsammlungen
 - Anfragen plus relevante Dokumente

Cranfield-Paradigma der Evaluierung im Information Retrieval

- Objektive Relevanz wird von neutralem Beobachter beurteilt
- Beziehung zwischen dem Informationswunsch und dem Dokument
- Keine individuelle und subjektive Relevanzbewertung
- Bis heute Testaufbau aller wichtigen Evaluierungsinitiativen im Information Retrieval (TREC, CLEF, NTCIR, INEX, ...)

Ziele der Evaluierungsinitiativen

- Einheitliche Bewertungsmaßstäbe für Retrieval-Systeme finden (Standardisierung)
- Vergleich zwischen Systemen ermöglichen
- Weiterentwicklung von IR Systemen vorantreiben
- Anforderungen aus der Community aufnehmen und Methodik weiterentwickeln

8.2 Recall, Precision

Recall, Precision, Fallout

- Precision (Genauigkeit)
 - Anteil der relevanten Dokumente unter den gefundenen Dokumenten
- Recall (Ausbeute)
 - Anteil der gefundenen relevanten Dokumente unter allen relevanten Dokumenten
- Fallout
 - Anteil der gefundenen irrelevanten an allen irrelevanten Dokumenten der Kollektion
- Annahme
 - alle Dokumente der Kollektion wurden vom Anwender bewertet
 - bei Top- r Rankings variiert Recall und Precision mit steigendem r

Recall, Precision

- Einzelne PR-Werte
 - Web-Retrieval
 - * die meisten Nutzer schauen nur die erste Seite an $\Rightarrow$ Prec@10 , Google: Prec@1
 - * Simulation von Benutzerklassen $\Rightarrow$ Prec@5, Prec@10, Prec@30, Prec@100
- Standpunkte
 - einzelner Benutzer
 - * Zusammenfassung der eigenen Anfragen
 - Systembetreiber
 - * Zusammenfassung der Anfragen aller Benutzer

Beispiel: Recall, Precision

- Gegeben eine Anfrage q
- $R = d3, d9, d25, d56, d123$
- Ranking eines IR Systems für Anfrage q
 1. d123
 2. d84
 3. d56
 4. d6
 5. d8
 6. d9
 7. d511
 8. d129
 9. d187
 10. d25
 11. d38
 12. d48
 13. d250
 14. d113
 15. d3

Recall, Precision

- Standard Recall Stufen
 - 0%, 10%, 20%, ..., 100%
- Precision Werte für mehrere Anfragen werden gemittelt
- Interpolation nach Salton
 - Ziehe von jedem PR-Punkte ein waagerechte Linie nach links
 - Ergebnis PR-Graph ist das Maximum

8.3 Probleme und Pooling-Methode

Probleme

- Alle relevanten Dokumente müssen bekannt sein
 - ist für große Dokumentmenge schwer zu garantieren
 - Precision-Recall sind nicht unabhängig
 - Interaktivität wird nicht berücksichtigt

Pooling Methode

1. Juroren erstellen Topics.
2. Systeme liefern die Top 1000 Dokumente für jedes Topic
3. Pooling aller einmal gefundenen Dokumente
4. Relevanz-Bewertung durch Juroren

8.4 Zuverlässigkeit der Evaluation, Bedenken und Antworten

Wie zuverlässig ist die Evaluierung nach dem Cranfield-Paradigma?
Bedenken und Antworten

Einwand Subjektivität

- Hängt das Ergebnis von den Juroren ab, welche die Relevanz der Dokumente bewerten?
 - Juroren bewerten tatsächlich unterschiedlich
 - Dies wirkt sich aber nicht auf die Reihenfolge der Systeme aus
 - Der Vergleich fällt unabhängig von Juroren gleich aus

Buckley und Voorhees 2005

Einwand Anzahl Topics

- Reichen 50 Topics aus, um die Systeme zu vergleichen?
 - Zwischen zwei Systemen muss ein gewisser Unterschied bestehen, um statistisch sicher zu sein, daß eines besser ist als das andere
 - Ab 50 Topics liegt der Unterschied unter 5%
 - Teilweise auch deutlich unter 5% (absolut)

Sanderson und Zobel 2005

Einwand Variabilität

- Der Unterschied in der Performanz zwischen den Systemen ist wesentlich geringer als zwischen den Topics
 - Stabile Performanz über alle Anfragen wichtiger als hohe durchschnittliche Performanz
 - Schwierige Anfragen sollten höheres Gewicht bei der Evaluierung gewinnen

Buckley und Voorhees 2005, Mandl 2006

Einwand Aufgabentyp

- Können Evaluierungsergebnisse auf andere Aufgaben, Korpora und Benutzerintentionen übertragen werden?
 - Nein
 - Evaluierung muss für jeden Anwendungsfall erfolgen

Buckley und Voorhees 2005

Ist die Evaluation relevant für den Benutzer?

- Der Benutzer ist nie in der Rolle der Evaluierenden
- Führt eine Verbesserung der Systeme hinsichtlich der traditionellen Kennzahlen auch zu einer höheren Benutzerzufriedenheit?
- Führt ein gutes Ranking auch zu einer höheren Benutzerzufriedenheit?

Einwand Benutzerzufriedenheit

- Momentan ungeklärt
 - Evaluation: Bei mehreren Bild-Retrieval Systemen konnte kein Zusammenhang zwischen Präferenz und Qualität erkannt werden Maskari et al. 2006
 - Für einfache Suchaufgaben und kontrollierte Retrieval-Qualität ergab sich ebenfalls kein Effekt auf die Präferenz Turpin und Scholer 2006

8.5 IR Psychologie

IR Psychologie

The unhappy customer, on average, will tell 27 other people

Dissatisfied customers tell an average of ten other people about their bad experience. Twelve percent tell up to twenty people.

⇒ Bad news travels fast.

IR Psychologie

On the other hand, satisfied customers will tell an average of five people about their positive experience.

⇒ Good news travels somewhat slower.

- Eine Site-Suche sollte robust funktionieren Ausreißer nach unten müssen vermieden werden!
 - für jede Anfrage
 - für jedes Bewertungskriterium

IR Psychologie

- Sehr gute Ergebnisse, schlechte Interaktion ⇒ Unzufriedener Benutzer
- Sehr gute Interaktion, schlechte Ergebnisse ⇒ Unzufriedener Benutzer

9 Locality Sensitive Hashing

9.1 Ähnlichkeit und Skizzen

Ähnlichkeit und Skizzen

- Sei U der Raum aller möglichen Dokumente
- $D \subseteq U$: Menge von Dokumenten
- Sei $\text{sim}: U \times U \rightarrow [0, 1]$ ein Ähnlichkeitsmaß
 - Falls p, q sehr ähnlich sind, dann ist $\text{sim}(p, q)$ nahe 1
 - Falls p, q sehr unähnlich sind, dann ist $\text{sim}(p, q)$ nahe 0
 - Oft: $\text{sim}(p, q) = 1 - d(p, q)$, wobei $d(p, q)$ eine normalisierte Distanz zwischen p und q ist.
- Sei T eine kleine Menge
- Eine Skizze für sim besteht aus
 - Kompressionsfunktion: eine zufällige Abbildung $\Phi: U \rightarrow T$
 - Rekonstruktionsfunktion: $\Psi: T \times T \rightarrow [0, 1]$
 - Für jedes Paar p, q gilt mit hoher Wahrscheinlichkeit

$$\Psi(\Phi(p), \Phi(q)) \approx \text{sim}(p, q)$$

Anwendung: Dokument-Clustering

- Sei G ein Graph über D : p, q sind durch eine Kante verbunden, falls $\text{sim}(p, q) \geq t$ (t ist ein Schwellwert)
- Ziel: finde die Zusammenhangskomponenten von

```
P ← leere Tabelle mit Größe |D|
G ← leerer Graph mit |D| Knoten
for i ← 1 to |D| do
  Lese Dokument  $p_i$  aus der Menge  $D$ 
   $P[i] \leftarrow \Phi(p_i)$ 
end
for i ← 1 to |D| do
  for j ← 1 to |D| do
    if  $\Psi(P[i], P[j]) \geq t$  then
      füge Kante  $(i, j)$  zu  $G$ 
    end
  end
end
end
Gib Zusammenhangskomponenten von  $G$  aus
```

Analyse

- Skizzen können in einem Durchlauf erzeugt werden
- Tabelle P kann in einer Datei auf einem Rechner gespeichert werden
- Erstellen von G erfordert $|D|^2$ Aufrufe von Φ
 - Schneller als volle Berechnung von sim
 - Quadratische Zeit ist noch ein Problem
- Zusammenhangskomponenten zu finden, ist teuer aber machbar

9.2 Locality Sensitive Hashing

Locality Sensitive Hashing (LSH)

- Eine spezielle Methode um Skizzen zu erzeugen
- $H = \{h | h: U \rightarrow T\}$ ist eine Familie von Hash-Funktionen
- H ist locality sensitive bezüglich sim , falls für alle

$$p, q \in U: Pr[h(p) = h(q)] = sim(p, q)$$

- Wahrscheinlichkeit ist über eine zufällige Wahl von h aus H
- Wahrscheinlichkeit einer Kollision = Ähnlichkeit zwischen p und q

Syntaktisches Clustering mittels LSH

```
 $P \leftarrow$  leere Tabelle der Größe  $|D|$   
 $G \leftarrow$  leerer Graphen mit  $|D|$  Knoten  
for  $i = 1$  to  $|D|$  do  
    lese Dokument  $p_i$  aus der Menge  $D$   
     $P[i] \leftarrow h(p_i)$   
end  
sortiere  $P$  und gruppieren nach Wert  
Gebe Gruppen aus
```

Analyse

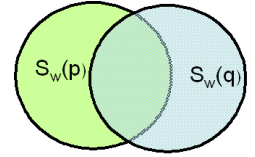
- Skizzen können in einem Durchlauf erzeugt werden
- Tabelle P kann in einer Datei auf einem Rechner gespeichert werden
- Sortieren und Gruppieren erfordert $O(|D| \log |D|)$ Vergleiche
- Eine Gruppe A besteht aus den Dokumenten mit den selben Hash-Wert
 - Wegen LSH Eigenschaft, die Dokument sind mit hoher Wahrscheinlichkeit ähnlich

9.3 LSH für Jaccard-Koeffizient

Shingling und Jaccard-Koeffizient

- Token: Wörter, Zahlen, HTML tags, usw.
- $\text{Token}(p)$: Sequenz der Token eines Dokuments p
- Sei w ein kleiner Integer
- $S_w(p) = w\text{-shingling von } p =$ Menge aller verschiedener fortlaufender Teilsequenzen von $\text{Token}(p)$ mit Länge w .
 - z.B.: $p = \text{'a rose is a rose is a rose'}$, $w = 4$
 - $S_w(p) = \{\text{'a rose is a'}, \text{'rose is a rose'}, \text{'is a rose is'}\}$
- Jaccard-Koeffizient $J_w(p, q) = \frac{|S_w(p) \cap S_w(q)|}{|S_w(p) \cup S_w(q)|}$

LSH für Jaccard-Koeffizient



- $J_w(p, q) = \frac{|S_w(p) \cap S_w(q)|}{|S_w(p) \cup S_w(q)|}$
- π = eine zufällige Permutation von Σ^w
 - π induziert eine zufällige Ordnung auf allen Token-Sequenzen der Länge w
 - π induziert auch eine zufällige Ordnung auf einer beliebigen Teilmenge $X \subseteq \Sigma^w$
 - Für alle solche Teilmengen und für jedes $x \in X$, gilt:

$$\Pr[\min(\pi(X)) = x] = \frac{1}{|X|}$$

- LSH für Jaccard-Koeffizient: $h(p) = \min(\pi(S_w(p)))$

LSH für Jaccard-Koeffizient

- Lemma:

$$\Pr[\min(\pi(S_w(p))) = \min(\pi(S_w(q)))] = \frac{|S_w(p) \cap S_w(q)|}{|S_w(p) \cup S_w(q)|}$$

- Proof:

$$\begin{aligned} & \Pr[\min(\pi(S_w(p))) = \min(\pi(S_w(q)))] \\ &= \Pr[\min(\pi(S_w(p) \cup S_w(q))) \in S_w(p) \cap S_w(q)] \\ &= \sum_{x \in S_w(p) \cap S_w(q)} \Pr[\min(\pi(S_w(p) \cup S_w(q))) = x] \\ &= \sum_{x \in S_w(p) \cap S_w(q)} \frac{1}{|S_w(p) \cup S_w(q)|} \\ &= \frac{|S_w(p) \cap S_w(q)|}{|S_w(p) \cup S_w(q)|} \end{aligned}$$

9.4 LSH für Manhattan-Distanz

Manhattan-Distanz

- Gegeben sei eine Menge von d -dimensionalen Integer-Vektoren X
- Die Koordinatenwerte sind aus der Menge $\{1, \dots, C\}$, d.h. $X \subseteq \{1, \dots, C\}^d$.
- Manhattan-Distanz zwischen zwei Vektoren $\vec{x}, \vec{y} \in X$ ist definiert als

$$d_1(\vec{x}, \vec{y}) = \sum_{i=1}^d |x_i - y_i|$$



Wieviel Punkte sind zu sehen? Auszählen eines $1/25$ der Fläche

Abbildung 23: Beispiel für probabilistisches Zählen

Manhattan- und Hamming-Distanz

- Sei $\vec{x} = (x_1, \dots, x_d)$ ein Vektor aus X dann ist $u(\vec{x}) = \text{Unary}_C(x_1) \dots \text{Unary}_C(x_d)$ eine Transformation von $\vec{x}$ in einen Bitstring.
 - $\text{Unary}_C(a)$ ist ein Bit-String der Länge C , der aus a Einsen gefolgt von $C - a$ Nullen besteht.
 - Z.B. $a = 4, C = 6$ dann ist $u(a) = 111100$

- Für zwei beliebige Vektoren $\vec{x}, \vec{y} \in X$ gilt

$$d_1(\vec{x}, \vec{y}) = d_H(u(\vec{x}), u(\vec{y}))$$

wobei d_H die Hamming-Distanz ist, die die Anzahl der verschiedenen Bits zurück gibt.

- Beispiel

- $C = 6, d = 2, \vec{x} = (3, 4)$ und $\vec{y} = (1, 5)$
- $d_1(\vec{x}, \vec{y}) = |3 - 1| + |4 - 5| = 3 = d_H(u(\vec{x}), u(\vec{y})) = d_H(111000 111100, 100000 111110) = 3$.

LSH-Schema für Manhattan-Distanz

- Zähle probabilistisch den Anteil der übereinstimmenden Bits
- Familie der Hashfunktionen für Manhattan-Distanz

$$H = \{h_i: \text{liefert das } i\text{te Bit}, 1 \leq i \leq C \cdot d\}$$

$$d_1(\vec{x}, \vec{y}) = d_H(u(\vec{x}), u(\vec{y})) = C \cdot d(1 - \Pr[h_i(u(\vec{x})) = h_i(u(\vec{y}))])$$

- Umgehung der Transformation in den Hamming-Raum:

$$h_i = \begin{cases} 1 & \text{falls } (i - 1 \bmod C) + 1 \leq x_{\lfloor \frac{i}{C} \rfloor + 1} \\ 0 & \text{sonst} \end{cases}$$

9.5 LSH für Cosinus-Ähnlichkeit

LSH-Schema für Cosinus-Ähnlichkeit

- Sei V ein Vokabular, $A \subset V$ ein Dokument in Bag-of-Words Darstellung und $\vec{t}_A$ ein Term-Frequency Vektor der A beschreibt.
- Cosinus-Ähnlichkeit zwischen zwei Dokumenten A und B ist

$$\text{sim}_C(A, B) = \frac{\vec{t}_A \cdot \vec{t}_B}{\|\vec{t}_A\| \cdot \|\vec{t}_B\|}$$

- Die Familie der Hash-Funktionen wird von der Menge der möglichen Abbildungen des Vokabulars V auf $\{-1, 1\}^{|V|}$ erzeugt.
- Eine solche Abbildung $m: V \rightarrow \{-1, 1\}^{|V|}$ sei als Vektor $\vec{m}$ dargestellt, dann ist die entsprechende Hash-Funktion

$$h_{\vec{m}}(A) = \begin{cases} 1 & \text{falls } \vec{m} \cdot \vec{t}_A \geq 0 \\ 0 & \text{sonst} \end{cases}$$

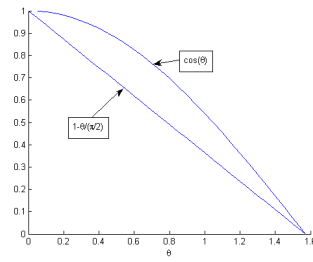


Abbildung 24: Approximation des Cosinus

LSH-Schema für Cosinus-Ähnlichkeit

- Das LSH-Schema ist dann

$$Pr[h_{\vec{m}}(A) = h_{\vec{m}}(B)] = 1 - \frac{\theta(\vec{t}_A, \vec{t}_B)}{\pi/2} \approx sim_C(A, B)$$

– $0 \leq \theta(\vec{t}_A, \vec{t}_B) \leq \pi/2$ ist der Winkel zwischen zwei Dokumentvektoren

- Approximation des Cosinus durch den normalisierten Winkel