Information-Retrieval: Unscharfe Suche

Claes Neuefeind
Fabian Steeg

19. November 2009

Themen des Seminars

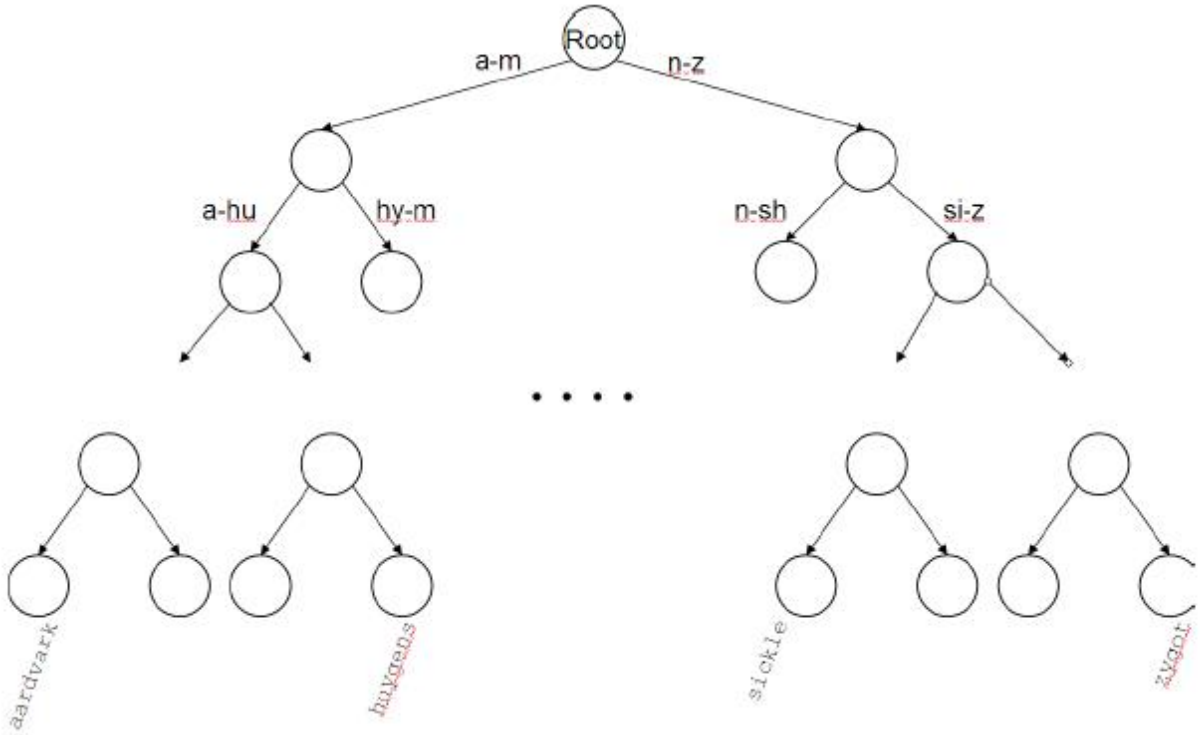
- Boolesches Retrieval-Modell (IIR 1)
- Datenstrukturen (IIR 2)
- Tolerantes Retrieval (IIR 3)
- Vektorraum-Modell (IIR 6)
- Evaluation (IIR 8)
- Web-Retrieval (IIR 19-21)

Wildcard Queries
- Permuterm-Index
- k-gram-Index

Rechtschreibprüfung
- Edit-Distance
- Kontextsensitiver Ansatz

Literatur

Grundlage: Binärer Suchbaum



Warum Wildcards?

- Z.B. bei Unkenntnis / Unklarheit der genauen Schreibweise
- mon* $\rightarrow$ finde Dokumente mit Wörtern, die mit 'mon' beginnen
  - Verarbeitung mittels Suchbaum
    $\rightarrow$ Wörter im Bereich mon $\leq w < moo$

Wildcards

- *mon → finde Dokumente mit Wörtern, die auf 'mon' enden
  - Verarbeitung mit umgekehrtem Suchbaum
    → Wörter im Bereich nom $\leq w < non$
- Was ist mit Wildcards in der Mitte?

Permuterm-Index

- Suchbäume eignen sich nur für * am Ende
- Bei Wildcard in der Mitte:
  - Wildcard-Query so umstellen, dass * am Ende steht
  - Zusätzlicher Hilfsindex mit Varianten
    $\longrightarrow$ Permuterm-Index

Permuterm-Index

- *hello* wird indexiert als *hello$*, *ello$h*, *llo$he*, *lo$hel*, *o$hell*, mit $\$ = $ Wortgrenze

Permuterm-Index

- *hello* wird indexiert als *hello$*, *ello$h*, *llo$he*, *lo$hel*, *o$hell*, mit $\$ =$ Wortgrenze
- Anfragen:

Permuterm-Index

- *hello* wird indexiert als *hello$*, *ello$h*, *llo$he*, *lo$hel*, *o$hell*, mit $\$ = $ Wortgrenze
- Anfragen:
  - X → X$

Permuterm-Index

- *hello* wird indexiert als *hello$*, *ello$h*, *llo$he*, *lo$hel*, *o$hell*, mit $\$ =$ Wortgrenze
- Anfragen:
  - X → X$
  - X* → X*$

Permuterm-Index

- *hello* wird indexiert als *hello$*, *ello$h*, *llo$he*, *lo$hel*, *o$hell*, mit *$* = Wortgrenze
- Anfragen:
  - X → X$
  - X* → X*$
  - *X → X$*

Permuterm-Index

- *hello* wird indexiert als *hello$*, *ello$h*, *llo$he*, *lo$hel*, *o$hell*, mit *$* = Wortgrenze
- Anfragen:
  - X → X$
  - X* → X*$
  - *X → X$*
  - *X* → X*

Permuterm-Index

- *hello* wird indexiert als *hello$*, *ello$h*, *llo$he*, *lo$hel*, *o$hell*, mit *$* = Wortgrenze
- Anfragen:
  - X → X$
  - X* → X*$
  - *X → X$*
  - *X* → X*
  - X*Y → Y$X*

Permuterm-Index

- *hello* wird indexiert als *hello$*, *ello$h*, *llo$he*, *lo$hel*, *o$hell*, mit *$* = Wortgrenze
- Anfragen:
  - X → X$
  - X* → X*$
  - *X → X$*
  - *X* → X*
  - X*Y → Y$X*
  - Beispiel: *hel*o*
    X=hel, Y=o → *o$hel**

Permuterm-Index

- *hello* wird indexiert als *hello$*, *ello$h*, *llo$he*, *lo$hel*, *o$hell*, mit *$* = Wortgrenze
- Anfragen:
  - X → X$
  - X* → X*$
  - *X → X$*
  - *X* → X*
  - X*Y → Y$X*
  - Beispiel: *hel*o*
    X=hel, Y=o → *o$hel**
  - Was machen wir mit X*Y*Z?

Permuterm-Index

- Vorgehen: Anfrage "drehen", bis * am Ende steht
- Problem 1:
  - Bei mehrfachem * wiederholter Abgleich mit Anfrage
  - Beispiel:
    - h*l*o → lookup: o$h*
    - Anschließend Terme filtern, die 'l' enthalten
- Problem 2:
  - Größe des Lexikons vervielfacht sich

k-gram-Index

- k-gram = Sequenz von k Zeichen
- Für alle Terme des Dictionary werden zunächst sämtliche k-Gramme ermittelt
- Beispiel:
  - Bigramme für den Text "April is the cruelest month":

    a,ap,pr,ri,il,l,i,is,s,t,th,he,e,
    c,cr,ru,ue,el,le,es,st,t,m,mo,on,nt,h

k-gram-Index

- Aufbau wie invertierter Index (Term-Dokument):
  - jedem k-gram wird eine postings list zugeordnet
  - enthält alle Terme, die das k-gram enthalten

Verarbeitung mit k-gram-Index

- mon* wird verarbeitet als Boolesche Anfrage:
  - $m UND mo UND on
  - Gibt Terme zurück, die alle 3 Bigramme enthalten
  - Nur verbleibende Terme werden im Index nachgeschlagen
- Problem:
  - Gibt u.a. auch moon zurück
    $\longrightarrow$ Ergebnis muss nochmals gegen die usprüngliche Anfrage gefiltert werden

Verarbeitung von Wildcard-Queries

- Wildcards erfordern vielfache Anfragen auf Hilfsindizes
- Hinzu kommen Anfragen auf Term-Dokument-Index für alle Terme der Ergebnismenge
  - u.U. sehr hoher Verarbeitungsaufwand
- Deshalb: Einsatz nur bei Bedarf
  (als "erweiterte Suche")

Rechtschreibprüfung

- Zwei Anwendungsbereiche:
  - Korrektur von Dokumenten
  - Korrektur von Anfragen
- Zwei Ansätze:
  - Einzelne Wörter
    - Isolierte Betrachtung
    - Problem: Wörter z.T. trotz Fehler 'korrekt'
  - Kontextsensitiv
    - Wortumfeld einbeziehen
      (Flug *noch* Sydney)

Korrektur von Dokumenten

- Korrektur vor Indexierung
- Einsatz v.a. im Zusammenspiel mit OCR
  $\longrightarrow$ spezialisierte Korrekturalgorithmen
- Index soll möglichst wenige OCR-typische Fehler enthalten
- Hier kann domänenspezifisches Wissen genutzt werden:
  - O/D wird bei OCR häufiger verwechselt als O/I
    (= typischer Tippfehler wg. Tastatur-Layout)

Korrektur von Anfragen

- Umgang mit fehlerhaften Anfragen:
  - Dokumente direkt passend zu korrigierter Anfrage zurückgeben
  - korrigierte Alternativen anbieten *("Meinten Sie . . . ?")*
- Voraussetzung ist in beiden Fällen ein Lexikon:
  - standardisiertes Lexikon (Wahrig, DUDEN, etc)
  - Dictionary des indexierten Korpus
    ggf. ergänzt durch Listen mit Namen, Akronymen, etc. (inkl. Schreibweisen, vgl. Normalisierung)

Korrektur von Anfragen

- Vorgehen:
  - Von allen Schreibweisen einer (fehlerhaften) Anfrage soll die 'nächst gelegene' gewählt werden
    $\rightarrow$ Konzept der 'Nähe' nötig
- Ansätze:
  - Edit-Distance
  - k-gram-Überlappung

Edit-Distanz

- Levenshtein-Distanz:
  Geringste Anzahl an Operationen, um String S1 in S2 zu ändern
- Verfügbare Operationen auf Zeichenebene:
  - insert
  - delete
  - replace
- Beispiel:
  - Edit-Distanz von cat zu dog beträgt 3
- Details zum Algorithmus in der Praxissitzung

Gewichtete Edit-Distanz

- Gewichtung in Abhängigkeit der beteiligten Zeichen
- Zielt v.a. auf Tippfehler
  - m-n häufiger als m-q
    $\rightarrow$ *replace*-Operation m-n sollte geringere Edit-Distanz haben
- Alternative Gewichtung für andere Domänen (OCR)
- Gewichtungs-Matrix als Input
  - Algorithmus muss entsprechend angepasst werden

Verwendung der Edit-Distanz

- Für jede Anfrage:

Verwendung der Edit-Distanz

- Für jede Anfrage:
  - Ermitteln aller Terme innerhalb einer festgelegten Edit-Distanz, anschließend Lookup im Index

Verwendung der Edit-Distanz

- Für jede Anfrage:
  - Ermitteln aller Terme innerhalb einer festgelegten Edit-Distanz, anschließend Lookup im Index
  - Edit-Distanz für alle Terme des Dictionary ermitteln?

Verwendung der Edit-Distanz

- Für jede Anfrage:
  - Ermitteln aller Terme innerhalb einer festgelegten Edit-Distanz, anschließend Lookup im Index
  - Edit-Distanz für alle Terme des Dictionary ermitteln?
- Menge an Kandidaten eingrenzen!

Verwendung der Edit-Distanz

- Für jede Anfrage:
  - Ermitteln aller Terme innerhalb einer festgelegten Edit-Distanz, anschließend Lookup im Index
  - Edit-Distanz für alle Terme des Dictionary ermitteln?
- Menge an Kandidaten eingrenzen!
  - Einfachste Heuristik: Terme, die mit gleichem Buchstaben beginnen

Verwendung der Edit-Distanz

- Für jede Anfrage:
  - Ermitteln aller Terme innerhalb einer festgelegten Edit-Distanz, anschließend Lookup im Index
  - Edit-Distanz für alle Terme des Dictionary ermitteln?
- Menge an Kandidaten eingrenzen!
  - Einfachste Heuristik: Terme, die mit gleichem Buchstaben beginnen
  - Besser: k-gram-Index

k-gram-Index

- Ermittlung aller k-Gramme für Anfrage und Terme im Lexikon
  - Vgl. k-gram-Index bei Wildcards
- Retrieval aller Terme, die mehrere k-Gramme mit der Anfrage gemein haben
  - z.B. Mindestanzahl als Schwellwert
  - Variante: Höhere Gewichtung für bestimmte k-Gramme (z.B. anhand Tastatur-Layout)
- Erst dann Edit-Distanz ermitteln

Beispiel: Trigramme

- Text: *november*
  Trigramme: *nov*, *ove*, *vem*, *emb*, *mbe*, *ber*
- Anfrage: *december*
  Trigramme: *dec*, *ece*, *cem*, *emb*, *mbe*, *ber*

- 3 Trigramme von 6 'überlappen'
- Wie lässt sich daraus ein Maß ableiten?

Beispiel: Trigramme

- Text: *november*
  Trigramme: *nov*, *ove*, *vem*, *emb*, *mbe*, *ber*
- Anfrage: *december*
  Trigramme: *dec*, *ece*, *cem*, *emb*, *mbe*, *ber*

- 3 Trigramme von 6 'überlappen'
- Wie lässt sich daraus ein Maß ableiten?

Jaccard-Koeffizient

Seien X und Y die Mengen der k-Gramme für Anfrage und Index-Term, dann ist der JK:

$$\frac{|X \cap Y|}{|X \cup Y|}$$

JK immer zwischen 0 und 1:

- 1, wenn X und Y die gleichen Elemente enthalten
- 0, wenn keine Elemente gemeinsam
- JK als Schwellwert (z.B. JK $>$ 0,5)

Kontextsensitive Korrekturen

- Text: *Flug nach Sydney*
- Anfrage: *Flug noch Sydney*

- Terme für sich scheinen korrekt
- Werden keine/zu wenig Dokumente gefunden, sollten dennoch Korrekturen angeboten werden
  - Kontext muss berücksichtigt werden

Kontextsensitive Korrekturen

- NLP zu aufwendig
- Besser:
  - zu jedem Teil der Anfrage Terme mit geringer Edit-Distanz ermitteln
  - Kombinationen mit je einem korrigierten Term probieren:
    - *Flug nach Sydney*
    - *Flur doch Sydney*
    - *Fluß noch Sydney*
    - etc.
  - Alle abfragen und Kombination mit den meisten Treffern zurückgeben?

Kontextsensitive Korrekturen

- Alternative: Anfrage aufteilen in 2-Wort-Phrasen, nur häufigste zu 3-Wort-Phrasen erweitern
- Evidenz mittels Heuristiken
  - Häufigkeit in Korpus (über positional Index)
  - Nutzungsstatistiken (logs)
- Ähnliche Heuristiken zur Auswahl der Alternative(n), die dem Nutzer präsentiert werden soll(en)
- Allgemein gilt: Rechtschreibprüfung ist sehr verarbeitungsintensiv
  $\longrightarrow$ Nur sinnvoll bei Anfragen, die signifikant wenig Ergebnisse liefern

Wie geht es weiter?

- Vektorraum-Modell (IIR 6)
- Evaluation (IIR 8)
- Web-Retrieval (IIR 19-21)

Manning, C. D., Raghavan, P., and Schütze, H. (2008).
Introduction to Information Retrieval.
Cambridge University Press.

Zum Nachlesen: Kapitel 3 (siehe www.informationretrieval.org)