Chapter 4

Text Search in a Nutshell

Sergej Sizov
Web Retrieval
Summer term 2010
Motivation

Information Retrieval

a) 

b) 

c) 

73 110 102 111 114 109 97 116 105 111 110
82 101 116 114 105 101 118 97 108
Motivation (2)

Stein von Rosetta  Jean-François Champollion
Motivation (3)

Sherlock Holmes

Arthur Conan Doyle
Motivation (4)

73 110 102 111 114 109 97 116 105 111 110
82 101 116 114 105 101 118 97 108
Text Processing: an overview

Analysis of document structure

- Lexical analysis and parsing
- Stopword elimination
- Stemming
- Indexing

- Lexicon / Thesaurus
- Index
- Document Repository

Compress
Stopword Elimination

Lookups in stopword lists
(potentially using domain-specific dictionary - lexikon/thesaurus)
e.g. "definition" or "theorem" for math documents

Common language-specific stopwords (prepositions, conjunctions, pronouns, "overloaded" verbs etc. – several hundreds of stopwords):

a, also, an, and, as, at, be, but, by,
can, could, do, for, from, go,
have, he, her, here, his, how,
I, if, in, into, it, its,
my, of, on, or, our, say, she,
that, the, their, there, therefore, they,
this, these, those, through, to, until,
we, what, when, where, which, while, who, with, would,
you, your, ....
Morphologic Reduction (Lemmatization)

- Grammatical base form:
  Nominative for nouns, infinitive for verbs, plural to singular, passive to active, etc.

Examples:
- „students“ to „student“, „going“ to „go“
- Kontext dependent and phrase dependent
  - „went“ to „go“,
  - „have been“ to „be“

- Linguistical base form
  Tracking of flexion (e.g. declination), composition, substantization, etc.

Examples:
- „nonfood“ to „food“
- „founds“ to „find“
- „Schweinkram“, „Schweinshaxe“ und „Schweinebraten“ to „Schwein“ etc.
Stemming

Ideas:
- use of dictionaries
- recognition through analysis of the linguistic structure
- affix elimination: removal of prefixes and suffixes using (heuristic) rules

Example:

stresses $\rightarrow$ stress, stressing $\rightarrow$ stress, symbols $\rightarrow$ symbol

using rules sses $\rightarrow$ ss, ing $\rightarrow$ e, s $\rightarrow$ e, etc.

Note: the usefulness of stemming in IR is not undisputable

Example:

Bill is operating a company.
On his computer he runs the ... operating system.
Thesaurus

For each concept (word sense) we store:
• the set of synonyms or instances (words)
• the set of generalizations and specializations (hypernyms, hyponyms)
• „part-of“ and „contains“ relationships (meronyms, holonyms)
• concept-example relationships (e.g. fairytale and cinderella)
• the set of antonyms

For each word we store:
• the set of associated concepts (e.g. with some statistics)
  (for disambiguation of polysems or homonymns)

Basic Principles:

• Feature Space: words in documents are reduced to terms.
• Document model: each document is represented as vector in $[0,1]^{|F|}$ whereby $d_{ij}$ is the weight of the $j$-th term in $d_i$.
• Queries: queries are vectors $q_i$ in $[0,1]^{|F|}$
• Relevance: relevance of results is based on similarity function for vector space $[0,1]^{|F|}$
• Indexing: for each term there is a list of Doc-IDs (e.g. URLs) with associated weights, implemented as „inverted file“ (search tree or hash table)
• Query execution: query is decomposed into several index-lookups for particular query terms in order to determine the ranked list of candidates
**Vector Space Model Relevance Ranking**

Ranking by descending relevance

Query \( q \in [0,1]^{|F|} \) (Set of weighted features)

Documents are feature vectors \( d_i \in [0,1]^{|F|} \)

Similarity metric:

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

e.g., using:

\[
d_{ij} := w_{ij} / \sqrt{\sum_k w_{ik}^2}
\]

\[
w_{ij} := \frac{\text{freq}(f_j, d_i)}{\max_k \text{freq}(f_k, d_i)} \cdot \log \frac{\#\text{docs}}{\#\text{docs with } f_i}
\]

tf*idf formula
Term Weighting

We consider following characteristics for $N$ documents and $M$ terms:

- $tf_{ij}$: term frequency - frequency of term $t_i$ in document $d_j$
- $df_i$: document frequency - number of documents that contain $t_i$
- $idf_i$: inverse document frequency = $N / df_i$
- $cf_i$: corpus frequency – frequency of $t_i$ in the corpus (e.g. separate counting of title terms, body terms, etc.)

Basic idea:
- The weight $w_{ij}$ of term $t_i$ in document $d_j$ should increase monotonically with $tf_{ij}$ and $idf_i$

First idea:
- use some tf-idf combination, e.g. $w_{ij} = f_{ij} \cdot idf_i$ (tf-idf formula)
- $w_{ij}$ can be normalized: $d_{ij} = \frac{w_{ij}}{\sqrt{\sum_k w_{k,j}^2}}$
Variations of Term Weighting

Empirical results show that tf and idf values usually must be dampened or normalized

- normalized tf values
  \[ \text{tf}_{ij} = \frac{\text{tf}_{ij}}{\max_k \text{tf}_{kj}} \]

- tf weighting mit dampening
  \[ \text{tf}_{ij} = 1 + \log \text{tf}_{ij} \]

- idf weighting mit dampening
  \[ \text{idf}_i = \log \frac{N}{\text{df}_i} \]

- common combination: (tf*idf formula)
  \[ w_{ij} = \frac{\text{tf}_{ij}}{\max_k \text{tf}_{ij}} \log \frac{N}{\text{df}_i} \]
  \[ d_{ij} = \frac{w_{ij}}{\sqrt{\sum_k w^2_{k,j}}} \]
Term Weighting in Queries

Depending of query interface and user category, simple or advanced term weightings may be used

- simple weighting: \( w_{ij} \in \{0, 1\} \)

- advanced weighting: \( w_{ij} = \left(0.5 + \frac{0.5 \cdot t_{fij}}{\max_k t_{fij}}\right) \cdot \log \frac{N}{df_i} \)

- term ranking: \( w_{ij} = \frac{1}{k} \)

  (when conjunctive query \( q \) contains \( k \) terms and \( t_i \) is in \( k^{th} \) position)
Konzeptionell:
invertierte Dateien (invertierte Listen) mit binärer Suche
nach Suchschlüsseln (Felder von Records, Strings in Texten)

Problem:
Speicherungsorganisation in Plattenblöcken (pagination) und effiziente
Implementierung der (binären) Suche für
  Exact-Match-Suche: search (key) returns ids
  Bereichssuche: search (lowkey, highkey) returns ids
  Präfixstringsuche: search (prefix) returns ids
bei dynamischen Updates
Properties of Media Types

8-32 Mb
1-128 Gb
40-120 Gb each disk
500 Gb-1Tb each changer

CPU Cache (SRAM)
Main Memory (DRAM)
Hard disks
Backup Systems

Gap $10^5$

Room (1 min)
Hamburg (3-5 Std)
Pluto (2 Jahre)
Andromeda (2000 J)
### Database Page (32-64 Kb)

<table>
<thead>
<tr>
<th>Header</th>
<th>Meyer</th>
<th>123</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Schneider</td>
<td>145</td>
<td>...</td>
</tr>
<tr>
<td>Müller</td>
<td>129</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Cache**
- **Forwarding-RID**
- **Slot-Array**
- **Extent Table**
B+ Trees: Example

B+-Tree

record-IDs or doc-IDs
Query
q: Mainz

B+ Tree: Lookup (1)
Query
q: Gießen
DBS-Style Top-k Query Processing

Given: query \( q = t_1 \ t_2 \ldots \ t_z \) with \( z \) (conjunctive) keywords

similarity scoring function \( \text{score}(q,d) \) for docs \( d \in D \), e.g.: \( \bar{q} \cdot \bar{d} \)

with precomputed scores (index weights) \( s_i(d) \) for which \( q_i \neq 0 \)

Find: top \( k \) results w.r.t. \( \text{score}(q,d) = \text{aggr}\{s_i(d)\} \) (e.g.: \( \Sigma_{i \in q} s_i(d) \))

Naive join&sort QP algorithm:

\[
\text{top-k (}
\begin{array}{c}
\sigma[\text{term}=t_1] \ (\text{index}) \\
\sigma[\text{term}=t_2] \ (\text{index}) \\
\ldots \\
\sigma[\text{term}=t_z] \ (\text{index})
\end{array} \times \\
\times \times \\
\times \times \\
\times \times \\
\text{order by } s \text { desc})
\]
Index List Processing by Merge Join

Keep L(i) in **ascending order of doc ids**
Compress L(i) by actually storing the gaps between successive doc ids
(or using some more sophisticated prefix-free code)

QP may start with those **L(i) lists that are short and have high idf**
Candidate results need to be looked up in other lists L(j)
To avoid having to uncompress the entire list L(j),
L(j) is encoded into groups of entries
with a **skip pointer** at the start of each group
→ sqrt(n) evenly spaced skip pointers for list of length n
Efficient Top-k Search

[Buckley85, Güntzer/Balke/Kießling 00, Fagin01]

threshold algorithms: efficient & principled top-k query processing with monotonic score aggr.

Data items: $d_1, \ldots, d_n$

**Query:** $q = (t_1, t_2, t_3)$

### Index lists

<table>
<thead>
<tr>
<th>Index</th>
<th>$d_{78}$</th>
<th>$d_{23}$</th>
<th>$d_{10}$</th>
<th>$d_1$</th>
<th>$d_{88}$</th>
<th>$d_9$</th>
<th>$d_{99}$</th>
<th>$d_{34}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>0.9</td>
<td>0.8</td>
<td>0.8</td>
<td>0.7</td>
<td>0.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_2$</td>
<td>0.8</td>
<td>0.6</td>
<td>0.6</td>
<td>0.2</td>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_3$</td>
<td>0.7</td>
<td>0.5</td>
<td>0.4</td>
<td>0.2</td>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TA with sorted access only (NRA):**
- can index lists; consider $d$ at pos $i$ in $L_i$;
- $E(d) := E(d) \cup \{i\}$; $high_i := s(t_i,d)$;
- $worstscore(d) := aggr\{s(t_\nu,d) | \nu \in E(d)\}$;
- $bestscore(d) := aggr\{worstscore(d), aggr\{high_\nu | \nu \not\in E(d)\}\}$;

if worstscore($d$) > min-k then add $d$ to top-k

- $min-k := min\{worstscore(d') | d' \in top-k\}$;

else if bestscore($d$) > min-k then

- $cand := cand \cup \{d\}; s$

**threshold := max \{bestscore(d') | d' \in cand\}$;

if threshold $\leq$ min-k then exit;

**keep $L(i)$ in descending order of scores**

---

**Scan depth 3**

<table>
<thead>
<tr>
<th>Rank</th>
<th>Doc</th>
<th>Worst-score</th>
<th>Best-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>d10</td>
<td>2.1</td>
<td>2.1</td>
</tr>
<tr>
<td>2</td>
<td>d78</td>
<td>2.0</td>
<td>2.0</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>1.8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>2.0</td>
<td></td>
</tr>
</tbody>
</table>

STOP!
Threshold Algorithm (TA, Quick-Combine, MinPro)
(Fagin’01; Güntzer/Balke/Kießling; Nepal/Ramakrishna)

scan all lists $L_i$ ($i=1..m$) in parallel:
consider $d_j$ at position $pos_i$ in $L_i$;
$high_i := s_i(d_j)$;
if $d_j \not\in$ top-k then {
look up $s_\nu(d_j)$ in all lists $L_\nu$ with $\nu \neq i$; // random access
compute $s(d_j) := \text{aggr} \{s_\nu(d_j) \mid \nu=1..m\}$;
if $s(d_j) > \text{min score among top-k}$ then
add $d_j$ to top-k and remove min-score $d$ from top-k; }
threshold := $\text{aggr}\{high_\nu \mid \nu=1..m\}$;
if min score among top-k $\geq$ threshold then exit;

$m = 3$
\begin{align*}
\text{aggr: sum} \quad & f: 0.5 \quad & a: 0.55 \quad & h: 0.35 \\
& b: 0.4 \quad & b: 0.2 \quad & d: 0.35 \\
& c: 0.35 \quad & f: 0.2 \quad & \quad \\
& a: 0.3 \quad & g: 0.2 \quad & a: 0.1 \\
& h: 0.1 \quad & e: 0.1 \quad & c: 0.05 \\
& d: 0.1 \quad & f: 0.05 \quad & b: 0.8 \\
\end{align*}

top-k: $f: 0.75$
$a: 0.95$
scan index lists in parallel:
  consider \( dj \) at position \( posi \) in \( Li \);
  \( E(dj) := E(dj) \cup \{i\} \); \( high_i := si(q,dj) \);
  \( \text{bestscore}(dj) := \text{aggr}\{x_1, \ldots, x_m\} \)
    with \( xi := si(q,dj) \) for \( i \in E(dj) \), \( high_i \) for \( i \notin E(dj) \);
  \( \text{worstscore}(dj) := \text{aggr}\{x_1, \ldots, x_m\} \)
    with \( xi := si(q,dj) \) for \( i \in E(dj) \), 0 for \( i \notin E(dj) \);
  \( \text{top-k} := k \) docs with largest \( \text{worstscore} \);
  \( \text{threshold} := \text{bestscore}\{d \mid d \not\in \text{top-k}\} \);
  if min \( \text{worstscore} \) among \( \text{top-k} \geq \text{threshold} \) then exit;

\[ m=3 \]
\[ \text{aggr: sum } k=2 \]

\[ \begin{array}{cccc}
  f: & 0.5 & a: & 0.55 \\
  b: & 0.4 & b: & 0.2 \\
  c: & 0.35 & f: & 0.2 \\
  a: & 0.3 & g: & 0.2 \\
  h: & 0.35 & a: & 0.1 \\
  d: & 0.1 & c: & 0.1 \\
  h: & 0.35 & c: & 0.05 \\
  d: & 0.35 & f: & 0.05 \\
\end{array} \]

\[ \begin{array}{l}
  \text{top-k:} \\
  a: 0.95 \\
  b: 0.8 \\
\end{array} \]

\[ \begin{array}{l}
  \text{candidates:} \\
  f: 0.7 + ? \leq 0.7 + 0.1 \\
  h: 0.35 + ? \leq 0.35 + 0.5 \\
  c: 0.35 + ? \leq 0.35 + 0.3 \\
  d: 0.35 + ? \leq 0.35 + 0.5 \\
  g: 0.2 + ? \leq 0.2 + 0.4 \\
\end{array} \]
Approximate Top-k Query Processing

Approximation TA:

A \( \theta \)-approximation \( T' \) for top-k query \( q \) with \( \theta > 1 \) is a set \( T' \) of docs with:

- \( |T'| = k \) and
- for each \( d' \in T' \) and each \( d'' \notin T' \): \( \theta \cdot \text{score}(q,d') \geq \text{score}(q,d'') \)

Modified TA:

... Stop when \( \min_k \geq \text{aggr}(\text{high}_1, ..., \text{high}_m) / \theta \)
Focus on $\text{score}(q,dj) = r(dj) + s(q,dj)$
with normalization $r(\cdot) \leq a$, $s(\cdot) \leq b$ (and often $a+b=1$)
Keep index lists sorted in descending order of „static“ authority $r(dj)$

**Conservative authority-based pruning:**
- $\text{high}(0) := \max \{ r(\text{pos}(i)) \mid i=1..m \}$; $\text{high} := \text{high}(0) + b$;
- $\text{high}(i) := r(\text{pos}(i)) + b$;
- stop scanning $i$-th index list when $\text{high}(i) < \text{min score of top k}$
- terminate algorithm when $\text{high} < \text{min score of top k}$
- effective when total score of top-k results is dominated by $r$

**First-$k'$ heuristics:**
- scan all $m$ index lists until $k' \geq k$ docs have been found that appear in all lists;
- the stopping condition is easy to check because of the sorting by $r$
Text Retrieval: Limitations and Problems with Web Mining Apps

IR necessary but not sufficient for Web search!

- Doesn’t capture authority
  - An article on BBC as good as a copy on john-doe-news.com
- Doesn’t address web navigation
  - Query ibm seeks www.ibm.com
  - www.ibm.com may look less topical than a quarterly report