Information Retrieval & Data Mining

Information Retrieval & Data Mining
Universität des Saarlandes, Saarbrücken
Wintersemester 2013/14
Chapter III: Ranking Principles
Zipf’s Law (after George Kingsley Zipf)

• The **collection frequency** $cf_i$ of the $i$-th most frequent word in the document collection is **inversely proportional** to the rank $i$

  $$ cf_i \propto \frac{1}{i} $$

• For the relative collection frequency with **language-specific constant** $c$ (for English $c \approx 0.1$) we obtain

  $$ \frac{cf_i}{\sum_j cf_j} \propto \frac{c}{i} $$

• In an English document collection, we can thus expect the most frequent word to account for 10% of all term occurrences
Levenshtein Edit Distance

- Levenshtein edit distance between two strings $x$ and $y$ is the minimal number of edit operations (insert, replace, delete) required to transform $x$ into $y$.

- The minimal number of operations $m[i, j]$ to transform the prefix substring $x[1:i]$ into $y[1:j]$ is defined via the recurrence

$$m[i, j] = \min \begin{cases} m[i - 1, j - 1] + (x[i] = y[j] ? 0 : 1) & \text{(replace } x[i]?) \\ m[i - 1, j] + 1 & \text{(delete } x[i]?) \\ m[i, j - 1] + 1 & \text{(insert } y[j]?) \end{cases}$$

and can be computed using dynamic programming.

- Examples: $d(\text{house, rose}) = 2$
Vector Space Model (VSM)

- Boolean retrieval model provides **no (or only rudimentary) ranking of results** – severe limitation for large result sets

- Vector space model views **documents and queries as vectors** in a $|V|$-dimensional vector space (i.e., one dimension per term)

- **Cosine similarity** between two vectors $q$ and $d$ is the cosine of the angle between them

$$sim(q, d) = \frac{q \cdot d}{\|q\| \|d\|}$$

$$= \frac{\sum_{i=1}^{\mid V \mid} q_i d_i}{\sqrt{\sum_{i=1}^{\mid V \mid} q_i^2} \sqrt{\sum_{i=1}^{\mid V \mid} d_i^2}}$$

$$= \frac{q}{\|q\|} \frac{d}{\|d\|}$$
**TF*IDF**

- **Term frequency** $tf_{i,d}$ as the number of times the term $t$ occurs in document $d$

- **Document frequency** $df_t$ as the number of documents that contain the term $t$

- **Inverse document frequency** $idf_t$ as

  $$idf_t = \frac{|D|}{df_t}$$

  with $|D|$ as the number of documents in the collection

- The tf.idf weight of term $t$ in document $d$ is then defined as

  $$tf.idf_{t,d} = tf_{t,d} \times idf_t$$

  favoring terms that **occur often in the document** $d$

  and/or **not in many documents from the collection** $D$
Precision, Recall, and Accuracy

- **Precision** $P$ is the fraction of retrieved documents that is relevant
  \[ P = \frac{tp}{tp + fp} \]

- **Recall** $R$ is the fraction of relevant results that is retrieved
  \[ R = \frac{tp}{tp + fn} \]

- **Accuracy** $A$ is the fraction of correctly classified documents
  \[ A = \frac{tp + tn}{tp + fp + tn + fn} \]

Not appropriate for IR
(Mean) Average Precision

- Precision, recall, and F-measure ignore the order of results

- **Average precision** (AP) averages over retrieved relevant results
  
  - Let \( \{d_1, \ldots, d_{m_j}\} \) be the set of relevant results for the query \( q_j \)
  
  - Let \( R_{jk} \) be the set of ranked retrieval results for the query \( q_j \) from top until you get to the relevant result \( d_k \)

  
  \[
  AP(q_j) = \frac{1}{m_j} \sum_{k=1}^{m_j} \text{Precision}(R_{jk})
  \]

- **Mean average precision** (MAP) averages over multiple queries

  \[
  \text{MAP}(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \text{AP}(q_j)
  \]
(Normalized) Discounted Cumulative Gain

• What if we have **graded labels** as relevance assessments? (e.g., 0 : not relevant, 1 : marginally relevant, 2 : relevant)

• **Discounted cumulative gain** (DCG) for query \( q \)

\[
DCG(q, k) = \sum_{m=1}^{k} \frac{2^{R(q, m)} - 1}{\log(1 + m)}
\]

with \( R(q, m) \in \{0, \ldots, 2\} \) as label of \( m \)-th retrieved result

• **Normalized discounted cumulative gain** (NDCG)

\[
NDCG(q, k) = \frac{DCG(q, k)}{IDCG(q, k)}
\]

normalized by **idealized discounted cumulative gain** (IDCG)
(Normalized) Discounted Cumulative Gain

- $IDCG(q, k)$ is the **best-possible** value $DCG(q, k)$ achievable for the query $q$ on the document collection at hand.

- **Example**: Let $R(q, m) \in \{0, \ldots, 2\}$ and assume that two documents have been labeled with 2, two with 1, all others with 0. The best-possible top-5 result thus has labels $<2, 2, 1, 1, 0>$ and determines the value of $IDCG(q, k)$ for this query.

- $NDCG$ **also considers rank** at which relevant results are retrieved.

- $NDCG$ is typically averaged over **multiple queries**

\[
NDCG(Q, k) = \frac{1}{|Q|} \sum_{q \in Q} NDCG(q, k)
\]
Okapi BM25

• **State-of-the-art retrieval model** (among top-ranked in TREC) having roots in **Probabilistic Information Retrieval**

\[
w_{t,d} = \frac{(k_1 + tf_{t,d})}{k_1((1 - b) + b \frac{|d|}{avdl}) + tf_{t,d}} \ log \ \frac{|D| - df_j + 0.5}{df_j + 0.5}
\]

• \(k_1\) controls **impact of term frequency** (common choice \(k_1 = 1.2\))

• \(b\) controls **impact of document length** (common choice \(b = 0.75\))
Multinomial Language Model

- Query $q$ is seen as a **bag of terms** and generated from document $d$ by **drawing terms** from the bag of terms corresponding to $d$

$$P(q|d) = \left( \frac{|q|}{tf(t_1, q) \ldots tf(t_{|q|}, q)} \right) \prod_{t_i \in q} P(t_i|d)^{tf(t_i, q)}$$

$$\propto \prod_{t_i \in q} P(t_i|d)^{tf(t_i, q)}$$

$$\approx \prod_{t_i \in q} P(t_i|d) \quad \text{(assuming } \forall t_i \in q : tf(t_i, q) = 1)$$

- **Maximum-likelihood estimate** for parameters $P(t_i|d)$

$$P(t_i|d) = \frac{tf(t_i, d)}{|d|}$$
Smoothing

- **Jelinek-Mercer smoothing** as linear combination of document language model $\theta_d$ and document-collection language model $\theta_D$

\[
P(t|d) = \lambda \frac{tf(t, d)}{|d|} + (1 - \lambda) \frac{tf(t, D)}{|D|}
\]

with document $D$ as concatenation of entire document collection

- **Dirichlet-prior smoothing** with a conjugate Dirichlet prior instead of the Maximum-Likelihood Estimation

\[
P(t|d) = \frac{tf(t, d) + \alpha \frac{tf(t, D)}{|D|}}{|d| + \alpha}
\]
Chapter IV: Link Analysis
PageRank

- **Random surfer model**
  - follows a uniform random outgoing link with probability \((1-\varepsilon)\)
  - jumps to a uniform random web page with probability \(\varepsilon\)

- **Matrix** \(T\) captures following of a uniform random outgoing link

\[
T_{ij} = \begin{cases} 
1/\text{out}(i) & : \ (i, j) \in E \\
0 & : \ \text{otherwise}
\end{cases}
\]

- **Vector** \(j\) captures jumping to a uniform random web page

\[
j_i = 1/|V|
\]

- **Transition probability matrix** of Markov chain then obtained as

\[
P = (1 - \varepsilon) T + \varepsilon [1 \ldots 1]^T j
\]
HITS

• Hyperlinked-Induced Topic Search (HITS) identifies
  • authorities as good content sources (~high indegree)
  • hubs as good link sources (~high outdegree)

• HITS [Kleinberg ‘99] considers a web page
  • a good authority if many good hubs link to it
  • a good hub if it links to many good authorities

  ~ mutual reinforcement between hubs & authorities

![Diagram showing mutual reinforcement between hubs and authorities]
**HITS**

- Given (partial) Web graph $G(V, E)$, let $a(v)$ and $h(v)$ denote the **authority score** and **hub score** of the web page $v$

  $$a(v) \propto \sum_{(u,v) \in E} h(u) \quad h(v) \propto \sum_{(v,w) \in E} a(w)$$

- Authority and hub scores in **matrix notation**

  $$a = \alpha A^T h \quad h = \beta A a$$

  with adjacency matrix $A$, hub vector $a$, authority vector $h$, and constants $\alpha$ and $\beta$

- Authority vector $a$ and hub vector $h$ are **eigenvectors** of **cocitation matrix** $A^T A$ and **coreference matrix** $AA^T$
Chapter V: Indexing & Searching
Inverted Index

• Inverted index keeps a **posting list** for each term, which usually reside on secondary storage, with each **posting** capturing information about term’s **occurrences in a specific document**

• **document identifier** (e.g., \(d_{123}, d_{234}, \ldots\))

• **term frequency** (e.g., \(tf(\text{house}, d_{123}) = 2, \ tf(\text{house}, d_{234}) = 4\))

• **score impacts** (e.g., \(tf(\text{house}, d_{123}) \times idf(\text{house}) = 3.75\))

• **offsets** (i.e., absolute positions at which the term occurs in the document)

• Posting lists are usually **compressed** for time and space efficiency
Inverted Index

- **Document-ordered posting lists** for more efficient intersections (e.g., required for Boolean queries and phrase queries)
  
  \[d_{123}, 2, [4, 14] \quad d_{133}, 1, [47] \quad d_{266}, 3, [1, 9, 20]\]

- **Impact-ordered posting lists** for more efficient top-\(k\) queries (i.e., terminate query processing as soon as top-\(k\) result is known)
  
  \[d_{231}, 1.0 \quad d_{12}, 0.9 \quad d_{662}, 0.8 \quad d_{3}, 0.5\]

- **Skip pointers** allow “fast forwarding” in a posting list
  
  \[d_{1}, 2 \quad d_{16}, 2 \quad d_{55}, 2 \quad d_{101}, 2\]
Ziv-Lempel Compression

- **LZ77** (Adaptive Dictionary) and further variants:
  - Scan text and identify in a *lookahead window* the longest string that occurs repeatedly and is contained in *backwards window*
  - Replace this string by a *pointer* to its previous occurrence
  - Encode text into list of **triples** < *back*, *count*, *new* > where
    - *back* is the backward distance to a prior occurrence of the string that starts at the current position
    - *count* is the length of this repeated string
    - *new* is the next symbol that follows the repeated string
  - Triples themselves can be further encoded (with variable length)
  - Variants use explicit dictionary with statistical analysis of text but need to scan text twice (for statistics and compression)
Variable-Byte Encoding

• 32-bit binary code represents 12,038 using 4 bytes as

```
00000000 00000000 00101111 00000110
```

• Variable-byte encoding (aka. 7-bit encoding) uses one bit per byte as a continuation bit indicating whether the current number expands into the next bytes

• Variable-byte encoding represents 12,038 using only 2 bytes as

```
01011110 10000110
```

1 continuation bit
7 data bits

• Byte-aligned, i.e., each number corresponds to sequence of bytes
Gamma Encoding

• Gamma (γ) encoding represents an integer \( x \) as
  
  • \( \text{length} = \text{floor}(\log_2 x) \) in \textit{unary}
  
  • \( \text{offset} = x - 2^{\text{length}} \) in \textit{binary}

results in \((1 + \log_2 x + \log_2 x)\) bits for integer \( x \)

• \text{Not byte-aligned}, i.e., needs to be packed into bytes or words

• Useful when \text{distribution} of numbers is \textit{not known} ahead of time or when \text{small numbers} (e.g., gaps, tf) are \textit{frequent}
Term-at-a-Time Query Processing

- **Term-at-a-Time (TAAT)** query processing
  - reads posting lists for query terms \( \langle t_1, \ldots, t_{|q|} \rangle \) successively
  - maintains an **accumulator** for each result document with value
    \[
    acc(d) = \sum_{i \leq j} score(t_i, d)
    \]
    after the first \( j \) posting lists have been read

<table>
<thead>
<tr>
<th></th>
<th>( d_1 )</th>
<th>( d_4 )</th>
<th>( d_7 )</th>
<th>( d_8 )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>a</strong></td>
<td>1.0</td>
<td>2.0</td>
<td>0.2</td>
<td>0.1</td>
</tr>
<tr>
<td><strong>b</strong></td>
<td>1.0</td>
<td>2.0</td>
<td>0.2</td>
<td>0.1</td>
</tr>
<tr>
<td><strong>c</strong></td>
<td>3.0</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- required **memory** depends on the **number of accumulators** maintained
- **top-k results** can be determined by **sorting accumulators** at the end
Document-at-a-Time Query Processing

- **Document-at-a-Time (DAAT) query processing**
  - assumes **document-ordered posting lists**
  - reads posting lists for query terms $\langle t_1, \ldots, t_{|q|} \rangle$ **concurrently**
  - computes score when **same document** is seen in one or more posting lists

<table>
<thead>
<tr>
<th></th>
<th>$d_1$</th>
<th>$d_4$</th>
<th>$d_7$</th>
<th>$d_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1.0</td>
<td>6.0</td>
<td>3.2</td>
<td>0.3</td>
</tr>
<tr>
<td>b</td>
<td>1.0</td>
<td>2.0</td>
<td>0.2</td>
<td>0.1</td>
</tr>
<tr>
<td>c</td>
<td>3.0</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- always advances posting list with **lowest current document identifier**
- required main memory depends on the **number of results** to be reported
- **top-$k$ results** can be determined by keeping results in **priority queue**
Fagin’s Threshold Algorithms

• **Threshold Algorithm** (TA)
  
  • original version, often used as synonym for entire family of algorithms
  
  • requires eager random access to candidate objects
  
  • worst-case memory consumption: $O(k)$

• **No-Random-Accesses** (NRA)

  • no random access required, may have to scan large parts of the lists
  
  • worst-case memory consumption: $O(m*n + k)$
Fagin’s Threshold Algorithms

• Assume score-ordered posting lists and additional index for score look-ups by document identifier

• Scan posting lists using inexpensive sequential accesses (SA) in round-robin manner

• Perform expensive random accesses (RA) to look up scores for a specific document when beneficial

• Support monotone score aggregation function

\[ \text{aggr} : \mathbb{R}^m \to \mathbb{R} : \forall x_i \geq x'_i \Rightarrow \text{aggr}(x_1, \ldots, x_m) \geq \text{aggr}(x'_1, \ldots, x'_m) \]

• Compute aggregate scores incrementally in candidate queue

• Compute score bounds for candidate results and stop when threshold test guarantees correct top-\(k\) result
No-Random-Accesses Algorithm (NRA)

- **Sequential accesses (SA) only**
- **Worst-case memory consumption** $O(m*n + k)$

No-Random-Accesses Algorithm (NRA):
scan index lists (e.g., round-robin)
consider $d = c\text{did}(i)$ in posting list for $t_i$
$\text{high}(i) = c\text{score}(i)$
$\text{eval}(d) = \text{eval}(d) \cup \{i\}$ // where have we seen $d$?

$\text{worst}(d) = \text{aggr}\{ \text{score}(t_j, d) \mid j \in \text{eval}(d) \}$
$\text{best}(d) = \text{aggr}\{ \text{worst}(d), \text{aggr}\{ \text{high}(j) \mid j \notin \text{eval}(d) \} \}$

- **if worst($d$) $>$ min$_k$ then** // good enough for top-$k$?
  - add $d$ top top-$k$
  - min$_k$ = min\{ worst($d'$) $|$ $d'$ $\in$ top-$k$ \}
- **else if best($d$) $>$ min$_k$ then** // good enough for cand?
  - cand = cand $\cup$ \{ $d$ \}
  - ub = max\{ best($d'$) $|$ $d'$ $\in$ cand \}
- **if ub $\leq$ min$_k$ then**
  - exit

<table>
<thead>
<tr>
<th></th>
<th>$d_{78}$</th>
<th>$d_{23}$</th>
<th>$d_{10}$</th>
<th>$d_1$</th>
<th>$d_{88}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>0.9</td>
<td>0.8</td>
<td>0.8</td>
<td>0.7</td>
<td>0.2</td>
</tr>
<tr>
<td>$b$</td>
<td></td>
<td>0.8</td>
<td>0.6</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>$c$</td>
<td></td>
<td>0.7</td>
<td>0.5</td>
<td>0.3</td>
<td></td>
</tr>
</tbody>
</table>

$ub = 2.4$ $ub = 2.1$ $ub = 2.0$

Top-1

<table>
<thead>
<tr>
<th></th>
<th>$d_{78}$</th>
<th>$d_{88}$</th>
<th>$d_{34}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>worst</td>
<td>0.9</td>
<td>0.4</td>
<td>0.8</td>
</tr>
<tr>
<td>best</td>
<td>2.0</td>
<td>2.9</td>
<td>2.0</td>
</tr>
</tbody>
</table>

SA  RA
Shingling

• Observation: Duplicates on the Web are often slightly perturbed (e.g., due to different boilerplate, minor rewordings, etc.)

• Document fingerprinting (e.g., SHA-1 or MD5) is not effective, since we need to allow for minor differences between documents

• Shingling represents document $d$ as set $S(d)$ of word-level $n$-grams (shingles) and compares documents based on these sets

\[ n = 3 \]

\{ the little brown fox jumps over the green frog \} 

\{ little brown fox, brown fox jumps, fox jumps over \}
Shingling

• Encode shingles by **hash fingerprints** (e.g., using SHA-1), yielding a set of numbers $S(d) \subseteq [1, \ldots, n]$ (e.g., for $n = 2^{64}$)

\[
n = 3
\]

- \{ the little brown \\ little brown fox \\ brown fox jumps \\ fox jumps over \}

\{ 141,944 \\ 13,031,980 \\ 21,111,978 \\ 6,012,014 \}

• Compare suspected near-duplicate documents $d$ and $d'$ by

  - **Resemblance** $\frac{|S(d) \cap S(d')|}{|S(d) \cup S(d')|}$ (Jaccard coefficient)

  - **Containment** $\frac{|S(d) \cap S(d')|}{|S(d)|}$ (Relative overlap)
Min-Wise Independent Permutations

- **Statistical sketch** to estimate the resemblance of $S(d)$ and $S(d')$
  - consider $m$ independent random permutations of the two sets, implemented by applying $m$ independent hash functions
  - keep the minimum value observed for each of the $m$ hash functions, yielding a $m$-dimensional MIPs vector for each document
  - estimate resemblance of $S(d)$ and $S(d')$ based on MIPs($d$) and MIPs($d'$)

  $\hat{r}(d, d') = \frac{|\{1 \leq i \leq m \mid MIPs(d)[i] = MIPs(d')[i]\}|}{m}$

- **Full details**: [Broder et al. ’00]
Min-Wise Independent Permutations

Set of shingle fingerprints

\[ S(d) = \{3, 8, 12, 17, 21, 24\} \]

\[ h_1(x) = 7x + 3 \mod 51 \]
\[ \{24, 8, 36, 20, 48, 18\} \]

\[ h_2(x) = 5x + 6 \mod 51 \]
\[ \{21, 46, 15, 40, 9, 24\} \]

\[ h_m(x) = 3x + 9 \mod 51 \]
\[ \{18, 33, 45, 9, 21, 30\} \]

- MIPs are an **unbiased estimator of resemblance**

\[ P[min\{h(x)|x \in A\} = min\{h(y)|y \in B\}] = |A \cap B|/|A \cup B| \]

- MIPs can be seen as **repeated random sampling** of x,y from A,B

Estimated resemblance: \(2 / 4\)

MIPs vector

\[ \begin{align*}
MIPs(d) & \quad MIPs(d') \\
8 & \quad 8 \\
9 & \quad 9 \\
\ldots & \quad \ldots \\
9 & \quad 2
\end{align*} \]
Chapter VI: Information Extraction
Hidden Markov Models (HMMs)

- Hidden Markov Model (HMM) is a discrete-time, finite-state Markov model consisting of
  - **state space** $S = \{s_1, \ldots, s_n\}$ and the state in step $t$ is denoted as $X(t)$
  - **initial state probabilities** $p_i$ ($i = 1, \ldots, n$)
  - **transition probabilities** $p_{ij} : S \times S \rightarrow [0,1]$, denoted $p(s_i \rightarrow s_j)$
  - **output alphabet** $\Sigma = \{w_1, \ldots, w_m\}$
  - **state-specific output probabilities** $q_{ik} : S \times \Sigma \rightarrow [0,1]$, denoted $q(s_i \uparrow w_k)$

- Probability of emitting output sequence $o_1, \ldots, o_T \in \Sigma^T$

$$
\sum_{x_1,\ldots,x_T \in S} \prod_{i=1}^{T} p(x_{i-1} \rightarrow x_i) q(x_i \uparrow o_i) \text{ with } p(x_0 \rightarrow x_0) = p(x_0)
$$
HMM Example

- **Goal**: Label the tokens in the sequence
  
  *Max-Planck-Institute, Stuhlsatzenhausweg 85*

  with the labels **Name, Street, Number**

\[
\Sigma = \{\text{“MPI”, “St.”, “85”}\} \quad // \text{output alphabet}
\]

\[
S = \{\text{Name, Street, Number}\} \quad // \text{(hidden) states}
\]

\[
p_i = \{0.6, 0.3, 0.1\} \quad // \text{initial state probabilities}
\]
Forward Computation

- Probability of emitting output sequence $o_1, \ldots, o_T \in \Sigma^T$ is
  \[
  \sum_{x_1, \ldots, x_T \in S} \prod_{i=1}^T p(x_{i-1} \to x_i) q(x_i \uparrow o_i) \text{ with } p(x_0 \to x) = p(x)
  \]

- Naïve computation would require $O(n^T)$ operations!

- Iterative forward computation with clever caching and reuse of intermediate results ("memoization") requires $O(n^2 T)$ operations

  - Let $\alpha_i(t) = P[o_1, \ldots, o_{t-1}, X(t) = i]$ denote the probability of being in state $i$ at time $t$ and having already emitted the prefix output $o_1, \ldots, o_{t-1}$

    - Begin: $\alpha_i(1) = p_i$
    - Induction: $\alpha_j(t + 1) = \sum_{i=1}^n \alpha_i(t) p(s_i \to s_j) p(s_i \uparrow o_t)$
Viterbi Algorithm

• **Goal**: Identify state sequence $x_1, \ldots, x_T$ most likely of having generated the observed output $o_1, \ldots, o_T$

• **Viterbi algorithm** (dynamic programming)

\[
\delta_i(1) = p_i \quad // \text{highest probability of being in state } i \text{ at step 1}
\]

\[
\psi_i(1) = 0 \quad // \text{highest-probability predecessor of state } i
\]

\begin{verbatim}
for t = 1, ..., T

\delta_j(t + 1) = \max_{i=1, \ldots, n} \delta_i(t) p(x_i \rightarrow x_j) q(x_i \uparrow o_t) \quad // \text{probability}

\psi_j(t + 1) = \arg \max_{i=1, \ldots, n} \delta_i(t) p(x_i \rightarrow x_j) q(x_i \uparrow o_t) \quad // \text{state}
\end{verbatim}

• Most likely state sequence can be obtained by means of **backtracking** through the memoized values $\delta_i(t)$ and $\psi_i(t)$
Thanks!