Chapter 2:
Text Mining and Retrieval

Dr. Dr. Sergej Sizov
Summer Term 2013
The document ranking problem

- We have a collection of documents
- User issues a query
- A list of documents needs to be returned

**Ranking method is core of an IR system:**
- In what order do we present documents to the user?
- We want the “best” document to be first, second best second, etc.

**Idea: Rank by probability of relevance of the document w.r.t. information need**
- $P(\text{relevant} \mid \text{document})$
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
73 110 102 111 114 109 97 116 105 111 110
82 101 116 114 105 101 118 97 108
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.
Search engine

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

**Ranking** by descending relevance

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

**Similarity metric:**

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

**e.g., using:**

$$d_{ij} := \frac{w_{ij}}{\sqrt{\sum_k w_{ik}^2}}$$

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

**tf*idf formula**
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. $w_{ij} = f_{ij} \times idf_i$ (tf-idf formula)

$w_{ij}$ can be normalized:

$$d_{ij} = \frac{w_{ij}}{\sqrt{\sum_k w_{kj}^2}}$$
Variations of Term Weighting

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

  - normalized tf values

  \[ tf_{ij} = \frac{tf_{ij}}{\max_k tf_{kj}} \]

  - tf weighting mit dampening

  \[ tf_{ij} = 1 + \log tf_{ij} \]

  - idf weighting mit dampening

  \[ idf_i = \log \frac{N}{df_i} \]

  - common combination: (tf*idf formula)

  \[ w_{ij} = \frac{tf_{ij}}{\max_k tf_{ij}} \log \frac{N}{df_i} \]

  \[ d_{ij} = \frac{w_{ij}}{\sqrt{\sum_k w_{kj}^2}} \]
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: 
    \[
    w_{ij} = \left( 0.5 + \frac{0.5 \cdot tf_{ij}}{\max_k tf_{ij}} \right) \cdot \log \frac{N}{df_i}
    \]

  - term ranking: 
    \[
    w_{ij} = \frac{1}{k}
    \]

  (when conjunctive query \( q \) contains \( k \) terms and \( t_i \) is in \( k^{th} \) position)
The document ranking problem

- We have a collection of documents
- User issues a query
- A list of documents needs to be returned
- **Ranking method is core of an IR system:**
  - In what order do we present documents to the user?
  - We want the “best” document to be first, second best second, etc....
- **Idea: Rank by probability of relevance of the document w.r.t. information need**
  - P(relevant|document)
Goal:

Ranking based on \( \text{sim}(\text{doc } d, \text{ query } q) = P[R|d] = P[\text{ doc } d \text{ is relevant for query } q | d \text{ has term vector } X_1,\ldots, X_m] \)

Assumptions:

Document relevance does not depend on other documents

Relevant and irrelevant documents differ in their terms.

Binary Independence Retrieval (BIR) Model:

- Probabilities for term occurrence are pairwise independent for different terms.

- Term weights are binary: \( X_i \in \{0,1\} \).

For terms that do not occur in query \( q \) the probabilities for such a term occurring are the same for relevant and irrelevant documents.
Probabilistic IR with Term Independence

\[ \text{sim}(d, q) = O(R|d) \]

\[ = \frac{P[R|d]}{P[\neg R|d]} \]

\[ = \frac{P[d|R] \cdot P[R]}{P[d]} \]

\[ = \frac{P[d|R]}{P[d|\neg R]} \times \frac{P[R]}{P[\neg R]} \]

\[ \sim \frac{P[d|R]}{P[d|\neg R]} \]

\[ = \prod_{i} \frac{P[X_i|R]}{P[X_i|\neg R]} \]

odds for relevance
(ratio of relevant documents)

Bayes’ theorem

independence or
linked dependence

Xi = 1 if d includes
i-th term, 0 otherwise
\[ \text{sim}(d, q)' = \log \prod_{i \in q} \frac{P[X_i|R]}{P[X_i|\neg R]} \]

\[ p_i = P[X_i = 1|R] \]
\[ q_i = P[X_i = 1|\neg R] \]

\[ = \sum_{i \in q} \log P[X_i|R] - \log P[X_i|\neg R] \]

\[ = \sum_{i \in q} \left\{ \log \left( p_i^{X_i} \cdot (1 - p_i)^{1-X_i} \right) - \log \left( q_i^{X_i} \cdot (1 - q_i)^{1-X_i} \right) \right\} \]

\[ = \sum_{i \in q} \left\{ \log \left( \frac{p_i^{X_i}}{(1 - p_i)^{X_i}} \right) - \log \left( \frac{q_i^{X_i}}{(1 - q_i)^{X_i}} \right) \right\} \]

\[ = \sum_{i \in q} X_i \cdot \log \frac{p_i}{1 - p_i} + \sum_{i \in q} X_i \cdot \log \frac{1 - q_i}{q_i} + \sum_{i \in q} \log \frac{1 - p_i}{1 - q_i} \]

\[ \sim \sum_{i \in q} X_i \cdot \log \frac{p_i}{1 - p_i} + \sum_{i \in q} X_i \cdot \log \frac{1 - q_i}{q_i} = \text{sim}(d, q)'' \]
Assumptions (without training sample or relevance feedback):
• $p_i$ is the same for all $i$.
• Most documents are irrelevant.
• Each individual term $i$ is infrequent.

This implies:

$$
\sum_i X_i \cdot \frac{p_i}{1 - p_i} = c \cdot \sum_i X_i \quad \text{with constant } c
$$

$$
q_i = P[X_i = 1|\neg R] \approx \frac{df_i}{N}
$$

$$
\frac{1 - q_i}{q_i} = \frac{N - df_i}{df_i} \approx \frac{N}{df_i} = idf_i
$$

$$
\text{sim}(d, q)^{'''} = \sum_{i \in q} X_i \cdot \log \frac{p_i}{1 - p_i} + \sum_{i \in q} X_i \cdot \log \frac{1 - q_i}{q_i}
\approx c \cdot \sum_i X_i + \sum_i X_i \cdot idf_i
$$

Scalar product over the product of tf and dampend idf values for query terms.
Estimate \( p_i \) und \( q_i \) based on training sample (query \( q \) on small sample of corpus) or based on intellectual assessment of first round ‘s result (\textit{relevance feedback}):

Let

\begin{align*}
N & \text{ be } \#\text{docs in sample,} \\
R & \text{ be } \#\text{ relevant docs in sample} \\
n_i & \#\text{docs in sample that contain term } i, \\
r_i & \#\text{ relevant docs in sample that contain term } i.
\end{align*}

Then

\[
p_i \approx \frac{r_i}{R} \quad q_i \approx \frac{n_i - r_i}{N - R}
\]

\[p_i \approx \frac{r_i + 0.5}{R + 1} \quad q_i \approx \frac{n_i - r_i + 0.5}{N - R + 1}\]

(Lidstone smoothing with \( \lambda = 0.5 \))

\[
sim(d, q)'' \approx \sum_{i \in q} X_i \cdot \log \frac{r_i + 0.5}{R - r_i + 0.5} + \sum_{i \in q} X_i \cdot \log \frac{N - n_i - R + r_i + 0.5}{n_i - r_i + 0.5}
\]

\[
weight_i(d) = \log \frac{(r_i + 0.5) \cdot (N - n_i - R + r_i + 0.5)}{(R - r_i + 0.5) \cdot (n_i - r_i + 0.5)}
\]
Probabilistic Retrieval: Example

Documents with relevance feedback:

<table>
<thead>
<tr>
<th></th>
<th>t1</th>
<th>t2</th>
<th>t3</th>
<th>t4</th>
<th>t5</th>
<th>t6</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>d1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>d2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>d3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>d4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ni</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ri</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>pi</td>
<td>5/6</td>
<td>1/2</td>
<td>1/2</td>
<td>5/6</td>
<td>1/2</td>
<td>1/6</td>
<td></td>
</tr>
<tr>
<td>qi</td>
<td>1/6</td>
<td>1/6</td>
<td>1/2</td>
<td>1/2</td>
<td>1/2</td>
<td>1/6</td>
<td></td>
</tr>
</tbody>
</table>

Score of new document d5 (with Lidstone smoothing with $\lambda=0.5$):

\[
\text{sim}(d5, q) = \sum_{i} X_i \log \frac{p_i}{1 - p_i} + \sum_{i} X_i \log \frac{1 - q_i}{q_i}
\]

\[
d5 \cap q: <1 1 0 0 0 1> \quad \rightarrow \quad \text{sim}(d5, q) = \log 5 + \log 1 + \log 0.2 + \log 5 + \log 5 + \log 5
\]
BIR: Summary

- The most important classic prob. IR model
- Use only term presence/absence, thus also referred to as Binary Independence Model
- Most natural for relevance/pseudo feedback
- When without relevance judgments, the model parameters must be estimated in an ad hoc way
- Performance isn’t as good as tuned VS model
- Extensions
  - Term correlations
  - Okapi
  - Smoothing
Constructing the Term Dependence Tree

Given:
- complete graph (V, E) with m nodes Xi ∈ V and
  m² undirected edges ∈ E with weights ε (or ρ)

Wanted:
- spanning tree (V, E‘) with maximal sum of weights

Algorithm:
- Sort the m² edges of E in descending order of weight
- E‘ := ∅
- Repeat until |E‘| = m-1
  - E‘ := E‘ ∪ {(i,j) ∈ E | (i,j) has max. weight in E}
  - provided that E‘ remains acyclic;
  - E := E – {(i,j) ∈ E | (i,j) has max. weight in E}

Example:
- Web 0.7 Surf
  - 0.9 Internet
  - 0.1 Swim
- Internet 0.5 Surf
  - 0.1 Swim
- Web 0.7 Surf
  - 0.9 Internet
  - 0.3 Swim
  - 0.1 Swim
Estimation of Multidimensional Probabilities

with Term Dependence Tree

Given is a term dependence tree \( V = \{X_1, \ldots, X_m\}, E' \).
Let \( X_1 \) be the root, nodes are preorder-numbered, and assume that
\( X_i \) and \( X_j \) are independent for \( (i,j) \notin E' \). Then:

\[
P[X_1 = \ldots \land \ldots \land X_m = \ldots] = P[X_1 = \ldots] \cdot P[X_2 = \ldots \land \ldots X_m = \ldots | X_1 = \ldots] \\
= \prod_{i=1}^{m} P[X_i = \ldots | X_1 = \ldots \land X_{i-1} = \ldots] \\
= P[X_1] \cdot \prod_{(i,j) \in E} P[X_j | X_i] \\
= P[X_1] \cdot \prod_{(i,j) \in E} \frac{P[X_j, X_i]}{P[X_i]}
\]

Example:

\[
P[\text{Web, Internet, Surf, Swim}] = \frac{P[\text{Web}]}{P[\text{Web}]} \frac{P[\text{Web, Internet}]}{P[\text{Web}]} \frac{P[\text{Web, Surf}]}{P[\text{Web}]} \frac{P[\text{Surf, Swim}]}{P[\text{Surf}]}\]
Text Indexing and Query Execution

Concept:
Inverted lists for binary search with keys

![Diagram showing inverted lists for binary search with keys]

- **$d_j$: docIDs**
- **$t_i$: termIDs**
B+ Trees: Lookup (1)

Query
q: Mainz

B+ Tree
B⁺ Trees: Lookup (2)

Query
q: Gießen

B⁺ Tree

Aachen Berlin Bonn Erfurt Essen Köln Mainz Merzig Paris Saarbrücken Trier Ulm

Bonn Essen Jena Merzig

Frankfurt Jena
Efficient Top-k Search

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

\[
\begin{array}{c|c|c|c|c|c}
  \text{Doc} & d78 & d23 & d10 & d1 & d88 \\
  \text{Score} & 0.9 & 0.8 & 0.8 & 0.7 & 0.2 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c}
  \text{Doc} & d64 & d23 & d10 & d10 & d78 \\
  \text{Score} & 0.8 & 0.6 & 0.6 & 0.2 & 0.1 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c}
  \text{Doc} & d10 & d78 & d64 & d99 & d34 \\
  \text{Score} & 0.7 & 0.5 & 0.4 & 0.2 & 0.1 \\
\end{array}
\]

TA with sorted access only (NRA):
- can index lists; consider \(d\) at pos \(i\) in \(L_i\);
- \(E(d) := E(d) \cup \{i\}\); \(\text{high}_i := s(t_i, d)\);
- \(\text{worstscore}(d) := \text{aggr}\{s(t_\nu, d) \mid \nu \in E(d)\}\);
- \(\text{bestscore}(d) := \text{aggr}\{\text{worstscore}(d), \text{aggr}\{\text{high}_\nu \mid \nu \notin E(d)\}\}\);
- if \(\text{worstscore}(d) > \text{min-k}\) then add \(d\) to top-k
- \(\text{min-k} := \min\{\text{worstscore}(d') \mid d' \in \text{top-k}\}\);
- else if \(\text{bestscore}(d) > \text{min-k}\) then
  \(\text{cand} := \text{cand} \cup \{d\}\); \(s\)
  \(\text{threshold} := \max\{\text{bestscore}(d') \mid d' \in \text{cand}\}\);
  if \(\text{threshold} \leq \text{min-k}\) then exit;

keep \(L(i)\) in descending order of scores

<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>2.4</td>
<td>2.0</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>1.8</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>2.0</td>
</tr>
</tbody>
</table>
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{x_1, ..., x_m}
    with x_i := si(q,dj) for i ∈ E(dj), high_i for i ∉ E(dj);
worstscore(dj) := aggr{x_1, ..., x_m}
    with x_i := si(q,dj) for i ∈ E(dj), 0 for i ∉ E(dj);
top-k := k docs with largest worstscore;
threshold := bestscore{d | d not in top-k};
if min worstscore among top-k ≥ threshold then exit;

m=3
aggr: sum
k=2

top-k:
  a: 0.95
  b: 0.8

candidates:
  f: 0.7 + ? ≤ 0.7 + 0.1
  h: 0.35 + ? ≤ 0.35 + 0.5
  c: 0.35 + ? ≤ 0.35 + 0.3
  d: 0.35 + ? ≤ 0.35 + 0.5
  g: 0.2 + ? ≤ 0.2 + 0.4
Indexing Text
Inverted Index

- Central datastructure in IR
- Requirements from T-D-matrix view
  - Lookup row vectors (term-vector)
  - Apply bit operations (AND, OR, NOT)
- Additionally:
  - T-D-Matrix is typically very large
    - 1,000,000 documents
    - 100,000 terms
    - each entry 1 bit
    - entire Matrix: 12.5 GB
  - T-D-Matrix is typically very sparse
    - 1,000 terms per document: 99% of entries are 0
  - Compress matrix: store only entries with value 1
Inverted Index

- Data structure consisting of
  - Lookup terms (row vectors)
    - Search tree
      - Term $t_1$
      - Term $t_2$
      - Term $t_3$
      - Term $t_4$
  - Posting-List of non-zero entries in vector
    - Linked list of postings
      - Term $t_i$ → $d_3$ → $d_{15}$ → $d_{42}$ → $d_{43}$ → $d_{58}$
  - Posting: reference to a document
Inverted Index Construction

- Sort terms with document references
- Build search tree over vocabulary
- Compile posting list for each term
Search: single term

Result of a query is a posting list
Search: multiple terms + „AND“
Search. Multiple Terms + "OR"
More complex queries:
- „Term 1“ AND „Term 2“ OR „Term 3“ AND „Term 4“
- Recursive approach

Suits parse tree of query
Intersect short lists first
- Faster to process
- Result lists get shorter
- Empty list serves as stop criterion

Example: query “coffee AND jar AND water”
- Annotate terms with length of posting list
  - List of length = number of documents containing term
  - Document frequency:
Phrase queries: Bi-Gram Index

- Index over word bi-grams
- Phrase search for „jar water“
  - Lookup bi-gram
- Longer phrases:
  - „to be or not“
  - Search for „to be“ AND „be or“ AND „or not“
  - False positives:
    - Filtering on full text
    - Accept errors
Position Index

- Store position of term in postings

- Intersect position-lists with offset
- Also allows for NEAR operator
Flexible search

- Leave out fragments of terms, mark with wildcard
  - Question mark (?): single character
  - Asterisk (*): zero, one or more characters

Examples:

- na?ve \(\rightarrow\) naive, naïve
- universit* \(\rightarrow\) university, universität, universitá
- go* \(\rightarrow\) go, goes, gone
- g?n* \(\rightarrow\) gun, gone, gin, ginger
Wildcard

- Simple case
  - Use sorting in inverted index
  - Example: "col*"

- Further application: Autocomplete
Permuterm Index

- Wildcard in the middle
- Build second index
  - Based on term permutations
  - Mark end of word ($)
  - Link to original term in index
- Example: „hut“
Example: hat, hunt, hut

Permuterm Index

Hat, hunt, hut
Search: $\sigma^*\tau$

Procedure:
- Append $\$: $\sigma^*\tau\$
- Permute to have * at end: $\tau\sigma^*$
- Wildcard search in permuterm index

Example:
- "h*t" $\rightarrow$ "t$h*$"
- "hu*t" $\rightarrow$ "t$hu*$"
N-Gram Index

- Permuterm index: very large search tree
- Alternatively: Index of character n-grams
  - Aim: search on sub-term fragments
  - Split terms in g-grams
  - Mark start, end with $%
  - Link to original terms in index
- Example: „hand“

$hand$

$ha$  han  and  nd$
N-Gram Index

- Example: band, bond, bank, hand
Search for „*and“:
- Search for n-grams
  - „and“
  - „nd$“
- Intersection of result lists
  - band, hand
- Still needed
  - Check order!
  - „ren*“ matches „referencing“
Why using eigenvector?

**Linear algebra:** \( A \mathbf{x} = b \)

**Eigenvector:** \( A \mathbf{x} = \lambda \mathbf{x} \)
Why using eigenvector

- Eigenvectors are orthogonal (seen as being independent)
- Eigenvector represents the basis of the original vector $A$
- Useful for
  - Solving linear equations
  - Determine the natural frequency of bridge
  - …
Latent Semantic Analysis

• Lexical matching at term level inaccurate (claimed)
• Polysemy – words with number of ‘meanings’ – term matching returns irrelevant documents – impacts precision
• Synonymy – number of words with same ‘meaning’ – term matching misses relevant documents – impacts recall
• Fewer dimensions $\rightarrow$ dimension reduction
• Keep k strongest dimensions: remove noise

LSA assumes that there exists a LATENT structure in word usage – obscured by variability in word choice

Word usage defined by term and document co-occurrence – matrix structure
Latent Semantic Indexing (LSI)  
[Deerwester et al. 1990]

A is the m×n term-document matrix. Then:
• U and U_k are the m×r and m×k term-topic similarity matrices,
• V and V_k are the n×r and n×k document-topic similarity matrices,
• A×A^T and A_k×A_k^T are the term-term similarity matrices,
• A^T×A and A_k^T×A_k are the document-document similarity matrices

\[
A \approx \begin{bmatrix}
U_k \\
\Sigma_k \\
V_k^T
\end{bmatrix} \begin{bmatrix}
doc j \\
latent topic t \\
doc j
\end{bmatrix}
\]

mapping of m×1 vectors into latent-topic space:
\[
d_j : U_k^T \times d_j =: d_j'
\]
\[
q : U_k^T \times q =: q'
\]

scalar-product similarity in latent-topic space:
\[
d_j'T \times q' = ((A_k V_k^T)_{j})^T \times q'
\]
Explicit Semantic Analysis

\[ T = w_1 \ldots w_n \]

- input text

\[ < v_i > = (v_1 \ldots v_N) \]

- corresponding TFIDF vector

\[ c_j \in \{c_1 \ldots c_N\} \]

- Wikipedia concepts

\[ < k_j > = (k_1 \ldots k_N) \]

- Associations between words / concepts

the semantic interpretation vector \( V \) for text \( T \) is a vector of length \( N \), in which the weight of each concept \( c_j \) is defined as

\[
\sum_{w_i \in T} v_i \cdot k_j
\]

To compute semantic relatedness of a pair of text fragments we compare their vectors using the cosine metric
Explicit Semantic Analysis: Example

**Apple iPod**
- IPod mini
- IPod photo
- IPod nano
- Apple Computer
- IPod shuffle
- ITunes

**Monetary Policy**
- International Monetary Fund
- Monetary policy
- Economic and Monetary Union
- Hong Kong Monetary Authority
- Monetarism
- Central bank