Fast LSI-based techniques for query expansion in text retrieval systems

L. Laura U. Nanni F. Sarracco

Department of Computer and System Science
University of Rome “La Sapienza”

2nd Workshop on Text-based Information Retrieval
Koblenz, 11 Sept. 2005
1 Preliminaries
   - Classical Text Matching
   - Query expansion by Thesauri
   - Spectral Techniques

2 Our approaches
   - LS-Thesaurus
   - LS-Filter

3 Experimental results

4 Conclusions
Term-Documents matrix

Collection $D = \{d_1, \ldots, d_n\}$ of text documents.

$T = \{t_1, \ldots, t_m\}$: set of distinct index terms in $D$:

$$
A = \begin{pmatrix}
  t_1 & d_1 & d_2 & \cdots & d_n \\
  t_2 & a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  t_m & a_{m,1} & a_{m,2} & \cdots & a_{m,n}
\end{pmatrix}
$$

$a_{i,j}$ is a function of the weight of term $t_i$ in document $d_j$
Collection $D = \{d_1, \ldots, d_n\}$ of text documents. 
$T = \{t_1, \ldots, t_m\}$: set of distinct index terms in $D$:

$$A = \begin{pmatrix}
  t_1 & d_1 & d_2 & \cdots & d_n \\
  t_2 & a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  t_m & a_{m,1} & a_{m,2} & \cdots & a_{m,n}
\end{pmatrix}$$

$a_{i,j}$ is a function of the weight of term $t_i$ in document $d_j$.
Collection $D = \{d_1, \ldots, d_n\}$ of text documents.

$T = \{t_1, \ldots, t_m\}$: set of distinct index terms in $D$:

$$A = \begin{pmatrix}
    d_1 & d_2 & \cdots & d_n \\
    a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\
    a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\
    \vdots & \vdots & \ddots & \vdots \\
    a_{m,1} & a_{m,2} & \cdots & a_{m,n}
\end{pmatrix}$$

$a_{i,j}$ is a function of the weight of term $t_i$ in document $d_j$. 
Collection $D = \{d_1, \ldots, d_n\}$ of text documents.
$T = \{t_1, \ldots, t_m\}$: set of distinct index terms in $D$:

$$A = \begin{pmatrix}
  t_1 & d_1 & d_2 & \cdots & d_n \\
  t_2 & a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  t_m & a_{m,1} & a_{m,2} & \cdots & a_{m,n}
\end{pmatrix}$$

$a_{i,j}$ is a function of the weight of term $t_i$ in document $d_j$
The value of $a_{i,j}$ is a function of two factors:

### A Local Factor $L(i, j)$

Measuring the relevance of term $t_i$ in document $d_j$. We used:

$$L(i, j) = \frac{\text{freq}(i, j)}{\max_{i \in [1, m]} \text{freq}(i, j)}$$

### A Global Factor $G(i)$

To de-amplify the relative weight of terms which are very frequently used in the collection. We used:

$$G(i) = \log \frac{n}{n_i}$$

$n_i$ is the number of documents containing term $t_i$. 
The value of $a_{i,j}$ is a function of two factors.

**A local factor $L(i,j)$**

measuring the relevance of term $t_i$ in document $d_j$. We used:

$$L(i,j) = \frac{\text{freq}(i,j)}{\max_{i\in[1,m]} \text{freq}(i,j)}$$

**A global factor $G(i)$**

to de-amplify the relative weight of terms which are very frequently used in the collection. We used:

$$G(i) = \log \frac{n}{n_i}$$

$n_i$ is the number of documents containing term $t_i$. 

---

L. Laura, U. Nanni, F. Sarracco: Fast LSI-based techniques for query expansion
The value of $a_{i,j}$ is a function of two factors:

**A LOCAL FACTOR $L(i,j)$**
measuring the relevance of term $t_i$ in document $d_j$. We used:

$$L(i,j) = \frac{\text{freq}(i,j)}{\max_{i\in[1,m]} \text{freq}(i,j)}$$

**A GLOBAL FACTOR $G(i)$**
to de-amplify the relative weight of terms which are very frequently used in the collection. We used:

$$G(i) = \log \frac{n}{n_i}$$

$n_i$ is the number of documents containing term $t_i$. 
Text Matching Algorithm

- **Input:** a query vector $\vec{q} = \{q_1, \ldots, q_m\}$
- **Output:** the rank vector $\vec{r} = \vec{q} \cdot A$
- $A$ and $q$ are sparse $\Rightarrow \vec{q} \cdot A$ can be computed very efficiently.
- If the size of the query is limited to $k$ terms $\Rightarrow$ TM has cost $O(kn)$.

Issues

- **Polysemy:** e.g., polo
- **Synonymy:** e.g., automobile, car, machine


Text Matching Algorithm

- **Input**: a query vector \( \vec{q} = \{ q_1, \ldots, q_m \} \)
- **Output**: the rank vector \( \vec{r} = \vec{q} \cdot A \)

- \( A \) and \( q \) are sparse \( \Rightarrow \) \( \vec{q} \cdot A \) can be computed very efficiently.
- If the size of the query is limited to \( k \) terms \( \Rightarrow \) TM has cost \( O(kn) \).

**Issues**

- **Polysemy**: e.g., polo
- **Synonymy**: e.g., automobile, car, machine
Text Matching Algorithm

- **Input**: a query vector \( \vec{q} = \{ q_1, \ldots, q_m \} \)
- **Output**: the rank vector \( \vec{r} = \vec{q} \cdot A \)
- \( A \) and \( q \) are sparse \( \Rightarrow \vec{q} \cdot A \) can be computed very efficiently.
- If the size of the query is limited to \( k \) terms \( \Rightarrow \) TM has cost \( O(kn) \).

**Issues**

- Polysemy: e.g., polo
- Synonymy: e.g., automobile, car, machine
Text Matching Algorithm

- **Input:** a query vector $\vec{q} = \{q_1, \ldots, q_m\}$
- **Output:** the rank vector $\vec{r} = \vec{q} \cdot A$
- $A$ and $q$ are sparse $\Rightarrow \vec{q} \cdot A$ can be computed very efficiently.
- If the size of the query is limited to $k$ terms $\Rightarrow$ TM has cost $O(kn)$.

**Issues**

- *Polysemy:* e.g., polo
- *Synonymy:* e.g., automobile, car, machine
Text Matching Algorithm

- **Input:** a query vector $\vec{q} = \{q_1, \ldots, q_m\}$
- **Output:** the rank vector $\vec{r} = \vec{q} \cdot A$
- $A$ and $q$ are sparse $\Rightarrow \vec{q} \cdot A$ can be computed very efficiently.
- If the size of the query is limited to $k$ terms $\Rightarrow$ TM has cost $O(kn)$.

Issues

- **Polysemy:** e.g., polo
- **Synonymy:** e.g., automobile, car, machine
Query expansion

**QUERY EXPANSION (or QUERY REWEIGHTING)**

The process aimed to alter the weights, and possibly the terms, of a query.

- **Two approaches:**
  1. use relevance feedback
  2. use some knowledge on terms relationship (thesaurus)

- The term-term correlation matrix $AA^T$ gives a statistic estimation of relationships among terms in the collection

$\Rightarrow$ Query expansion: $\vec{q}' \leftarrow \vec{q}AA^T$
Query expansion

**QUERY EXPANSION (or QUERY REWEIGHTING)**

The process aimed to alter the weights, and possibly the terms, of a query.

- Two approaches:
  1. use relevance feedback
  2. use some knowledge on terms relationship (thesaurus)

- The term-term correlation matrix $AA^T$ gives a statistic estimation of relationships among terms in the collection

  $\Rightarrow$ Query expansion: $\bar{q}' \leftarrow \bar{q} AA^T$
Query expansion

**QUERY EXPANSION (or QUERY REWEIGHTING)**

The process aimed to alter the weights, and possibly the terms, of a query.

- Two approaches:
  1. use relevance feedback
  2. use some knowledge on terms relationship (thesaurus)

- The term-term correlation matrix $A A^T$ gives a statistic estimation of relationships among terms in the collection

  $\Rightarrow$ Query expansion: $\tilde{q}' \leftarrow \tilde{q} A A^T$
Qiu and Frei presented an alternative approach to compute $A$
Idea: compute the probability that a document is representative of a term
They propose the following weighting scheme:

$$a_{i,j} = \begin{cases} 
(0.5 + 0.5 \times \frac{freq(i,j)}{\max_j freq(i,j)}) \times itf(j) & \text{if } freq(i,j) > 0 \\
\sqrt{\sum_{l=1}^{n} ((0.5 + 0.5 \times \frac{freq(i,l)}{\max_l freq(i,l)}) \times itf(j))^2} & \text{otherwise}
\end{cases}$$

$\max_j freq(i,j)$: is the maximum frequency of term $t_i$ over all the documents in the collection.
$itf_j = \log \frac{m}{m_j}$, $m_j$ the number of distinct terms in the document $d_j$;
Algorithm

1: Compute $\bar{A}$ with the previous weighting function;
2: Compute similarity thesaurus: $S \leftarrow \bar{A} \bar{A}^T$;
3: Given a query vector $\vec{q}$:
4: $\vec{s} \leftarrow \vec{q} S$;
5: $\vec{s}' \leftarrow \xi(\vec{s}, x_r)$;
6: $\vec{s}'' \leftarrow \frac{\vec{s}'}{|q|}$, where $|q| = \sum_{i=1}^{m} q_i$;
7: $\vec{q}' \leftarrow \vec{q} + \vec{s}''$;

Search quality

Use of S.T. improves Text-Matching but does not completely resolve polysemy and synonymy.
Query expansion by Similarity Thesaurus

Algorithm

1: Compute $\vec{A}$ with the previous weighting function;
2: Compute similarity thesaurus: $S \leftarrow \vec{A} \vec{A}^T$;
3: Given a query vector $\vec{q}$:
4: $\vec{s} \leftarrow \vec{q} S$;
5: $\vec{s}^\prime \leftarrow \xi(\vec{s}, x_r)$;
6: $\vec{s}^{\prime\prime} \leftarrow \frac{\vec{s}^\prime}{|\vec{q}|}$, where $|\vec{q}| = \sum_{i=1}^{m} q_i$;
7: $\vec{q}^\prime \leftarrow \vec{q} + \vec{s}^{\prime\prime}$;

Search quality

Use of S.T. improves Text-Matching but does not completely resolve polysemy and synonymy.
Latent Semantic Indexing

Idea

Use the Singular Value Decomposition to produce a low rank approximation of $A$

$$A = U S V^T$$
Latent Semantic Indexing

Idea

Use the Singular Value Decomposition to produce a low rank approximation of $A$

$$A_k = U_k S_k V_k^T$$
The documents collection is represented in the reduced \( k \)-dimensional subspace:

\[
D = \Sigma_k^{-1} U_k^T A
\]

Also the query vector is projected in the \( k \)-dimensional subspace:

\[
q_k = \Sigma_k^{-1} U_k^T q
\]

The rank of \( i \)-th document is given by

\[
\cos \alpha_i = \frac{q_k \cdot d^i}{|q_k| \cdot |d^i|}
\]

**Issue**

Requires the computation of \( n \) cosines \( \Rightarrow \) very slow, unusable for large collections
LSI Search

- The documents collection is represented in the reduced $k$-dimensional subspace:
  \[ D = \Sigma_k^{-1} U_k^T A \]

- Also the query vector is projected in the $k$-dimensional subspace: $q_k = \Sigma_k^{-1} U_k^T q$

- The rank of $i$-th document is given by
  \[ r_i = \cos \alpha_i = \frac{\vec{q}_k \cdot \vec{d}^i}{|\vec{q}_k| \cdot |\vec{d}^i|} \]

**Issue**

Requires the computation of $n$ cosines $\Rightarrow$ very slow, unusable for large collections
LSI Search

- The documents collection is represented in the reduced \( k \)-dimensional subspace:

\[
D = \Sigma_{k}^{-1} U_{k}^{T} A
\]

- Also the query vector is projected in the \( k \)-dimensional subspace: \( q_{k} = \Sigma_{k}^{-1} U_{k}^{T} q \)

- The rank of \( i \)-th document is given by

\[
r_{i} = \cos \alpha_{i} = \frac{\vec{q}_{k} \cdot \vec{d}_{i}}{|\vec{q}_{k}| \cdot |\vec{d}_{i}|}
\]

Issue

Requires the computation of \( n \) cosines \( \Rightarrow \) very slow, unusable for large collections
LSI Search

- The documents collection is represented in the reduced $k$-dimensional subspace:

\[ D = \Sigma_k^{-1} U_k^T A \]

- Also the query vector is projected in the $k$-dimensional subspace: $q_k = \Sigma_k^{-1} U_k^T q$

- The rank of $i$-th document is given by

\[ r_i = \cos \alpha_i = \frac{q_k \cdot d_i}{|q_k| \cdot |d_i|} \]

**Issue**

Requires the computation of $n$ cosines $\Rightarrow$ very slow, unusable for large collections

---

Fast LSI-based techniques for query expansion

L. Laura, U. Nanni, F. Sarracco
LS-Thesaurus

Idea

Compute the similarity matrix starting from a low rank approximation of $\overline{A}$ (as in LSI).

We can formally define our similarity matrix $S_k$ in the following way:

$$S_k = \overline{A}_k \overline{A}_k^T = U_k \overline{\Sigma}_k V_k^T V_k \overline{\Sigma}_k^T U_k^T = U_k \overline{\Sigma}_k \overline{\Sigma}_k^T U_k^T$$

Note that $U$ and the diagonal elements of $\overline{\Sigma}$ correspond respectively to the eigenvectors and the eigenvalues of matrix $\overline{A} \overline{A}^T$. 

L. Laura, U. Nanni, F. Sarracco

Fast LSI-based techniques for query expansion
Algorithm LS-Thesaurus - Pre-Process

1: **Input:** a collection of documents $D$;
2: **Output:** a Similarity Thesaurus, i.e., a $m \times m$ matrix $S_k$;
3:
4: Compute the term-document matrix $\overline{A}$ from $D$;
5: $(U, \Lambda) \leftarrow EIGEN(\overline{A} \overline{A}^T)$;
6: $U_k \leftarrow U^{(1:m,1:k)}$;
7: $\Lambda_k \leftarrow \Lambda^{(1:k,1:k)}$;
8: $S_k \leftarrow U_k \Lambda_k U_k^T$;
Algorithm LS-Thesaurus - Expand

1: **Input:** a query vector $\vec{q}$, a similarity thesaurus $S_k$,  
2: a positive integer $x_r$;  
3: **Output:** a query vector $\vec{q}'$;  
4:  
5: $\vec{s} \leftarrow \vec{q} S_k$;  
6: $\vec{s}' \leftarrow \xi(\vec{s}, x_r)$;  
7: $\vec{s}'' \leftarrow \frac{\vec{s}'}{|\vec{q}|}$, where $|q| = \sum_{i=1}^{m} q_i$;  
8: $\vec{q}' \leftarrow \vec{q} + \vec{s}''$;
Assumptions

- In LSI algorithm documents and queries are projected (and compared) in a $k$-dimensional subspace.
- The axes of this subspace represent the $k$ most important concepts arising from the documents in the collection.
- The user query tries to catch one (or more) of these concepts by using an appropriate set of terms.

Idea

Let the system select the terms which best represent the required concepts.
Assumptions

- In LSI algorithm documents and queries are projected (and compared) in a $k$-dimensional subspace.
- The axes of this subspace represent the $k$ most important concepts arising from the documents in the collection.
- The user query tries to catch one (or more) of these concepts by using an appropriate set of terms.

Idea

Let the system select the terms which best represent the required concepts.
3. The user inserts a query
4. The query vector is projected in the concepts space
5. Minor concepts are removed
6. Concepts vector is projected back into the terms space
7. Minor terms are removed
8. Remaining terms are added to the original query vector
Algorithm LS-Filter - Pre-Process

1. **Input:** a collection of documents $D$;
2. **Output:** a pair of matrices $(P, P^{-1})$;
3: 
4. Compute the term-document matrix $A$ from $D$;
5: $(U, \Sigma, V) \leftarrow SVD(A)$;
6: $U_k \leftarrow U^{(1:m, 1:k)}$;
7: $\Sigma_k \leftarrow \Sigma^{(1:k, 1:k)}$;
8: $P \leftarrow \Sigma_k^{-1} U_k^T$;
9: $P^{-1} \leftarrow U_k \Sigma_k$;
Algorithm LS-Filter - Expand

1: **Input:** a query vector $\vec{q}$, matrices $(P, P^{-1})$, 
2: two positive integers $x_c$ and $x_t$; 
3: **Output:** a query vector $\vec{q}'$; 
4: 
5: $\vec{p} \leftarrow P \vec{q}$; 
6: $\vec{p}' \leftarrow \xi(\vec{p}, x_c)$; 
7: $\vec{p}'' \leftarrow P^{-1} \vec{p}'$; 
8: $\vec{q}' \leftarrow \xi(\vec{p}'', x_t)$;
We compared the behavior of the following approaches:

- TM: the simple text matching;
- LS-T: text matching with queries previously expanded by \textit{LS-Thesaurus} algorithm;
- LS-F: text matching with queries previously expanded by \textit{LS-Filter} algorithm;
- LSI: the full \textit{LSI} computation.
Dataset

- Three books from O’Reilly in html format about Perl, Unix and Java
- 3000 documents (html pages)
- 150 short queries, with human made collection of relevant documents

publicly available on the web at URL:
http://www.dis.uniroma1.it/~laura/
Fast LSI-based techniques for query expansion
## Examples of LS-Filter

<table>
<thead>
<tr>
<th>job control</th>
<th>shell logout</th>
<th>zip file</th>
</tr>
</thead>
<tbody>
<tr>
<td>job</td>
<td>shell</td>
<td>file</td>
</tr>
<tr>
<td>background</td>
<td>bourne</td>
<td>util</td>
</tr>
<tr>
<td>echo</td>
<td>login</td>
<td>zip</td>
</tr>
<tr>
<td>number</td>
<td>perl</td>
<td>checksum</td>
</tr>
<tr>
<td>list</td>
<td>prompt</td>
<td></td>
</tr>
<tr>
<td>control</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrl</td>
<td></td>
<td></td>
</tr>
<tr>
<td>object</td>
<td></td>
<td></td>
</tr>
<tr>
<td>line</td>
<td></td>
<td></td>
</tr>
<tr>
<td>filename</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What we have done

- Previous techniques:
  - TM: fast but prone to synonymy and polysemy
  - LSI: effective but slow

- We introduced 2 techniques to improve TM search by using query expansion

- We compared them with TM and LSI search:
  - we used a non-standard mid-size dataset
  - generally their performances are better than TM, but not homogeneous
What we have done

- Previous techniques:
  - TM: fast but prone to synonymy and polysemy
  - LSI: effective but slow

- We introduced 2 techniques to improve TM search by using query expansion

- We compared them with TM and LSI search:
  - we used a non-standard mid-size dataset
  - generally their performances are better than TM, but not homogeneous
Previous techniques:
- TM: fast but prone to synonymy and polysemy
- LSI: effective but slow

We introduced 2 techniques to improve TM search by using query expansion

We compared them with TM and LSI search:
- we used a non-standard mid-size dataset
- generally their performances are better than TM, but not homogeneous
What we want to do

- Further experiments on standard datasets
- Exploit user relevance feedback to discriminate relevant concepts in case of ambiguous queries
- Very big data structures: can we decrease spatial cost?
H. Bast and D. Majumdar. Why spectral retrieval works.

Indexing by latent semantic analysis.

C. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala
Latent semantic indexing: A probabilistic analysis.
Thank you