Vintage: A Visual Information Retrieval Interface Based on Latent Semantic Indexing

Shih-Hao Li and Peter B. Danzig
Computer Science Department
University of Southern California
Los Angeles, California 90089-0781
{shih, danzig}@cs.usc.edu

Abstract

A visual information retrieval interface helps searchers browse a large number of documents and locate interesting ones. This article describes a new method that displays the relationships between query terms and documents in a two-dimensional space. Users can refine their query by marking returned documents as relevant or nonrelevant and resubmit for higher recall and precision. We have implemented this method in a Web-based prototype system called Vintage.

1 Introduction

Traditional information retrieval systems return documents in a sorted or ranked list. Such lists, as they get long, become tedious to browse.

By arranging documents in a two-dimensional space based on their inter-relationships, users can more easily identify topics and browse topical clusters. In [1], Gower and Digby introduce several methods, such as principal components analysis, biplot, and correspondence analysis, for expressing complex relationships in two dimensions. These methods use Singular Value Decomposition (SVD) to transform documents into one or more matrices that carry the “compressed” relationships of the original data. In Information Agents [2], Cybenko et. al use Self Organizing Maps (SOM) [3] to map multi-dimensional data onto a two-dimensional space. SOM is based on neural networks and needs iterative training and adjustments. Similarly, Lin [4] uses SOM to generate a two-dimensional map display to show the document distribution in a database. GUIDO (Graphical User Interface for Document Organization) [5] organizes documents in a two-dimensional display according to their distances to two reference terms, called Points of Interest (POI), chosen by the user. In GUIDO, documents lying in the same direction with respect to a given set of POIs are judged similar.

In this paper, we propose a two-dimensional visualization scheme based on Latent Semantic Indexing (LSI) [6] for retrieving documents. Our method is similar to GUIDO, but uses different coordinates to arrange documents. In addition, it needs no training and adjustment in contrast to SOM. Our method displays data by clusters according to their cross-similarities and ranks documents by colors with respect to the user query. We introduce LSI in Section 2 and describe our method and Vintage in Section 3. Section 4 shows the experimental results for relevance feedback and Section 5 presents our conclusions.
2 Latent Semantic Indexing

LSI was originally developed to address the *vocabulary problem* [7], which states that people with different backgrounds or in different contexts describe information differently. LSI assumes some underlying semantic structure exists in the pattern of term usage across documents and uses SVD to capture this structure. LSI represents documents as vectors of term frequencies following Salton’s *Vector Space Model* (VSM) [8]. An entire database is represented as an $m \times n$ term-document matrix, where $m$ and $n$ are the number of terms and documents in the database. To capture the semantic structure among documents, LSI applies SVD to this matrix and generates vectors of $k$ (typically 100 to 300) orthogonal indexing dimensions, where each dimension represents a linearly independent concept. The decomposed vectors are used to represent both documents and terms in the same semantic space, while their values indicate the degrees of association with the $k$ underlying concepts. Because $k$ is chosen much smaller than the number of documents and terms in the database, the decomposed vectors are represented in a compressed dimensional space and are not independent. Therefore, two documents can be relevant without having common terms but with common concepts. Figure 1 shows SVD applied to a term-document matrix and Example 1 illustrates the document representation before and after SVD.

![Diagram](image)

Figure 1: SVD applies to an $m \times n$ term-document matrix, where $m$ and $n$ are the number of terms and documents in the database, and $k$ is the dimensionality of the SVD representation.

Example 1

Let $d_i$ ($1 \leq i \leq 5$), and $t_j$ ($1 \leq j \leq 8$) be a set of documents and their associated terms in an information system, where

\[
\begin{align*}
    d_1 & = \{ t_1, t_3, t_4, t_5 \}, \\
    d_2 & = \{ t_1, t_2, t_3, t_4 \}, \\
    d_3 & = \{ t_2, t_3, t_4 \}, \\
    d_4 & = \{ t_6, t_7, t_8 \}, \\
    d_5 & = \{ t_5, t_6, t_7 \}.
\end{align*}
\]

We apply LSI to these documents and choose the indexing dimension $k$ as 2. Table 1 shows their vector representations in VSM (i.e. before SVD) and LSI (i.e. after SVD). Figure 2 shows the
two-dimensional plot of the decomposed document and term vectors in LSI.

$$\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|}
\hline
\text{document} & \text{term} & \text{VSM} & \text{LSI} \\
\hline
\text{description} & (t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8) & (\text{dim1} & \text{dim2}) \\
\hline
\text{d}_1 & t_1 & t_3 & t_4 & t_4 & 1 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0.863 & -0.508 \\
\text{d}_2 & t_1 & t_2 & t_2 & t_3 & 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0.873 & 0.521 \\
\text{d}_3 & t_2 & t_3 & t_4 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0.702 & 0.005 \\
\text{d}_4 & t_6 & t_7 & t_8 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & -0.010 & 0.603 \\
\text{d}_5 & t_5 & t_6 & t_6 & t_7 & 0 & 0 & 0 & 0 & 1 & 2 & 1 & 0 & -0.017 & 0.974 \\
\hline
\end{array}$$

Table 1: The vector representations of documents $d_i$ ($1 \leq i \leq 5$) in VSM (i.e. before SVD) and LSI (i.e. after SVD). The "$(t_1 \ldots t_8)$" and "$(\text{dim1 dim2})$" columns show the vectors in VSM and LSI, respectively.

Figure 2: Two-dimensional plot of decomposed vectors in Example 1, where documents $d_i$ ($1 \leq i \leq 5$) and terms $t_j$ ($1 \leq j \leq 8$) are represented as "o" and "+", respectively.

The above example shows that documents having a number of common terms are close to each other as well as their common terms. In Figure 2, we can roughly see two clusters, A and B. In cluster A, documents $d_1$, $d_2$, and $d_3$ are close to each other because each pair of them share $t_1$, $t_2$, $t_3$, or $t_4$. Similarly, $d_4$ and $d_5$ are close to each other because both of them have $t_6$ and $t_7$. To present documents and terms in this way, we can easily see their relationships in a two-dimensional space.
In an information retrieval interface based on this technique, users can find interesting documents by surrounding terms and locate the relevant ones by clusters.

3 Visualization Using LSI

Based on the clustering feature in LSI, we develop a new visual interface to display the relationships between query terms and documents for information retrieval. In this section, we describe our method and its prototype system Vintage.

3.1 Methodology

Let $k$ be the indexing dimension used in LSI when transforming the original term-document matrix to decomposed term and document vectors. Those decomposed vectors are only computed once and are stored in the server for further query processing. LSI represents a query as a $k$-dimensional vector, which consists of the average sum of its component term vectors. To compute the answer set of a query, LSI compares the query vector with all document vectors and returns those with the highest cosine coefficient [9].

To display query terms and returned documents in a two-dimensional space, we need to reduce their $k$-dimensional vectors to two dimensions. One solution is to run SVD on those $k$-dimensional vectors by choosing the new indexing dimension as 2. This approach can precisely present the relationships between terms and documents in a two-dimensional space. However, it needs to recompute SVD for each query, which generates a high computation overhead on the server. Another solution is to use the first two of the $k$ dimensions. This approach needs no SVD re-computations and can be viewed as an approximation due to SVD properties. When computing SVD, the singular values (corresponding to the indexing dimensions) are of decreasing importance. The original term-document matrix can be approximated by a linear combination of the top singular values and their corresponding rows and columns from the decomposed document and term matrices. The more singular values are included, the closer the original matrix can be obtained [6]. Based on this concept, we implemented a prototype system called Vintage.

3.2 Vintage

Vintage (A Visual Information Retrieval Interface Based on Latent Semantic Indexing) is a Web-based prototype system built using HTML and CGI scripts.

In Vintage, users can enter a list of query terms and submit them to the server. The server consists of two modules – Server Gateway and Query Processing Unit, as shown in Figure 3. The server gateway parses user requests and transforms them to a standard query format for further processing. The query processing unit composes a query vector by its component term vectors and compares it with precomputed document vectors. All the documents with similarity larger than a predefined threshold are considered relevant and are returned to the server gateway. The server gateway generates a 2-D image and HTML page for the relevant documents, which it forwards to the user as both a list and a two-dimensional image. In the list, documents titles are displayed and sorted in decreasing order of their similarities with respect to the query. In the 2-D image, query
terms are printed with a constant color and documents are represented as squares and painted with different colors showing their degree of similarity to the query (the higher similarities, the darker colors). The numbers inside the squares indicate the ranks of associated documents in the linear list. By doing this, users can see the relationships between query terms and documents by their distances as well as their individual similarities by colors in the same two-dimensional space. In addition, users can click on documents in the list or the 2-D image to see their detailed descriptions.

Vintage also provides relevance feedback. Users can refine a query by changing the original query terms or mark returned documents as relevant or nonrelevant and resubmit them to the server. In Vintage, we adopt the Ide dec-hi relevance feedback method [10], which Salton and Buckley found has the best overall performance among several commonly used methods [11]. In “Ide dec-hi”, a refined query consists of the original query plus all user-judged relevant documents and minus the top-most nonrelevant document. Specifically,

\[
Q_{\text{new}} = Q_{\text{old}} + \sum_{i=1}^{r} R_i - N_1,
\]

where \(Q_{\text{old}}\) and \(Q_{\text{new}}\) are the vectors of the original and the refined query, \(R_i\) and \(N_1\) are the vectors of the relevant documents and the top-most nonrelevant document judged by the user, and \(r\) is the number of relevant documents. This method can be iterated for further query refinement.

Below we present an example of a Vintage user session. The underlying database contains 3204 CACM document abstracts indexed by LSI. Suppose the user submits query terms performance and evaluation, and sets the maximum number of results to 50. The result is shown in Figure 4, where 14 documents are returned and displayed in a ranked list and a 2-D image. In the 2-D image, documents are scattered at different places. Among them, four of the top five documents are closer to term “evaluation” than “performance”. The user looks at the titles of these documents and finds documents 1, 2, and 3 are closer to what he wants. He then refines the query by adding a new term model and marking documents 1, 2, 3, 8, and 11 as relevant, and 4 and 7 as nonrelevant (see Figure 5). The new result set is reduced to nine documents as shown in Figure 6, where seven of them gather in a cluster with roughly equal distance (relevance) to the three query terms. The user clicks on square #1, the document with the highest similarity to the query, and sees its detailed description in Figure 7.
4 Experiments

To measure the performance of relevance feedback in Vintage, we conducted experiments on four standard datasets – CACM, CISI, CRAN, and MED, where queries and relevance judgments are available. In each dataset, documents are indexed with terms occurring in the title and abstract but not on a stop list of 429 common words. While queries are written in natural language, terms in a query are only used if they do not appear on a stop list and if they appear in at least one document. All indexed terms are stored in their original forms without stemming. We apply SVD to each term-document matrix using indexing dimension 100 as suggested in Deerwester’s LSI experiments [6]. Table 2 summarizes the characteristics of each dataset.

During relevance feedback, users need to mark returned documents as relevant or nonrelevant for query refinement. For a long returned list, users are likely to browse only those in the top of the list instead of all of them. Therefore, we only focus on the top-ranked documents in our experiments. We set the maximum number of results to four different values, 10, 25, 50, and 100,
each representing the number of returned documents that users browse and mark their relevancy. In our experiments, we do not have real users to judge each returned document. Instead, we simulate user’s relevance feedback by using the standard relevance judgments of each dataset. The standard relevance judgments contain the perfect results for each query. If a returned document is in this judgment, it is considered as judged relevant by the user. Otherwise, it is nonrelevant.

As described previously, we apply the “Ide dec-hi” method to compute relevance feedback in Vintage. The initial queries are taken from the test queries in each dataset. We compare their results with the standard relevance judgments and calculate the associated precision and recall. For query refinement, we add the vectors of all returned documents that are judged relevant to the current query and subtract the top-most vector of those nonrelevant ones. The whole procedure is repeated five times for each query. Figure 8 shows the average precision and recall over all queries for each dataset.

In Figure 8 we can roughly see both precision and recall are improved during each relevance feedback. In the four datasets we tested, the recall gets higher as the number of returned documents
increases, while the precision becomes lower. The reason is that most relevant documents have higher ranks in the returned list. Therefore, when more documents are returned, more nonrelevant documents are involved rather than the relevant ones.

As shown in Figure 8, the precision and recall increases as more relevance feedbacks are performed. However, the increasing rate between each two contiguous feedbacks does not go up accordingly. Figure 9 shows the average increasing rate of precision and recall for the five relevance feedbacks. In Figure 9 we can see the first feedback has the highest improvement for both precision and recall. As more feedbacks proceed, the increasing rate drops to almost zero. This indicates users can get most improvements from the early feedbacks on these datasets.

The experimental results show Vintage’s relevance feedback can improve precision and recall on the four standard datasets. In Vintage, users can set a small number of returned documents to get high precision on the initial query. Then they can mark relevant documents in the returned list and increase this number on further relevance feedbacks for higher recalls. When the returned number is too large, users can look at the 2-D image to locate interesting documents.
In our experiments, we use the same initial query with different relevant and nonrelevant documents for generating a new query. We do not add or delete query terms during each feedback. In a real situation, the precision and recall could have more improvements by changing query terms during relevance feedback.

5 Conclusions

A visual information retrieval interface can help searchers to browse a large number of documents and locate the interested ones. In this paper, we have proposed a two-dimensional visualization scheme based on Deerwester’s latent semantic indexing. Our method displays data according to their cross-similarities and still retains the rankings with respect to the query. We have implemented a prototype system Vintage and shown that the embedded relevance feedback method can improve both precision and recall on four standard datasets.
Figure 8: Precision-recall curves for different number of returned documents \( n \ (n = 10, 25, 50, \text{ and } 100) \) on four datasets (a) CACM, (b) CISI, (c) CRAN, and (d) MED. The data are obtained by measuring the average precision and recall of all test queries, each consisting of the initial query, denoted as “\( \times \)”, and five successive relevance feedbacks, denoted as “\( \circ \)”. 
<table>
<thead>
<tr>
<th>Characteristic</th>
<th>CACM</th>
<th>CISI</th>
<th>CRAN</th>
<th>MED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of documents</td>
<td>3204</td>
<td>1460</td>
<td>1400</td>
<td>1033</td>
</tr>
<tr>
<td>Number of queries</td>
<td>64</td>
<td>112</td>
<td>225</td>
<td>30</td>
</tr>
<tr>
<td>Number of indexing terms</td>
<td>8739</td>
<td>9240</td>
<td>6694</td>
<td>12256</td>
</tr>
<tr>
<td>Average number of terms per document</td>
<td>23.93</td>
<td>48.83</td>
<td>57.85</td>
<td>57.20</td>
</tr>
<tr>
<td>Average number of terms per query</td>
<td>10.61</td>
<td>30.03</td>
<td>9.19</td>
<td>10.13</td>
</tr>
</tbody>
</table>

Table 2: Characteristics of datasets.

![Graphs](image)

Figure 9: Average increasing rate of relevance feedbacks for (a) precision and (b) recall on four datasets CACM, CISI, CRAN, and MED.
The current version of Vintage puts all computations on the server site, including measuring similarities and generating 2-D images. We believe that moving the visualization part to the client site can not only alleviate the load on the server, but also provide more functionalities to the user. For example, users can zoom in a specific portion of the image or rearrange documents and terms based on their profiles [12]. To achieve this goal, we are developing a new Vintage using Sun Microsystems’ Java language and Hot Java browser.

References


