9 Searching the Internet with the SVD

9.1 Information retrieval

Over the last 20 years the number of internet users has grown exponentially with time; see Figure 1. Trying to extract information from this exponentially growing resource of material can be a daunting task. *Information retrieval* (IR) is an interdisciplinary science, which is concerned with automated storage and retrieval of documents.

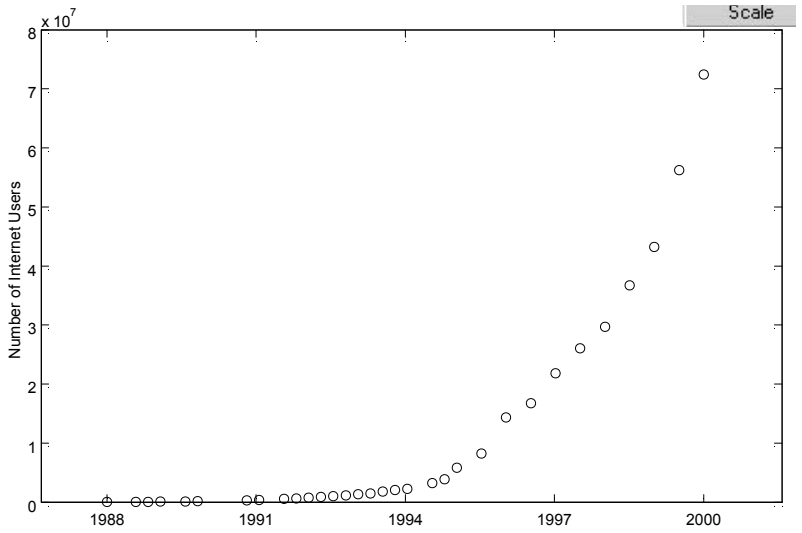

Figure 1: Internet data obtained from the website http://www.isc.org/ds/host-count-history.html.

IR systems are used to handle extremely large sets of digital data and help reduce information overload.[1] One of the largest digital data sets is the Internet with more than 20 billion web pages. Extraction of a small subset of useful information from this huge digital data set requires a web search engine. The grandfather of search engines, which was really not a search engine, was called Archie and was created in 1990 by Peter Deutsch, Alan Emtage, and Bill Wheelan. After Archie, followed a whole array of search engines; World Wide Web Wanderer (1993), Excite (1993), Yahoo (1994), Infoseek (1994), Lycos (1994), AltaVista (1995), and Google (1997) to name a few. In this lecture you will learn how to create a web search engine using a vector space model.

[0]Version November 4, 2013

[1]The term *Information overload* was originated in 1970 by Alvin Toffler in his book Future Shock. It refers to having too much information in order to make a decision.

9.2 A vector space model

Central for the discussion to follow are the term-by-document matrices. A term-by-document matrix $A = [a_{i,j}]$ is constructed by letting the entries $a_{i,j}$ represent the frequency of the term i in document j. We are interested in the situation when the documents are web pages, however, the method described can be applied to other documents as well, such as letters, books, phone calls, etc.

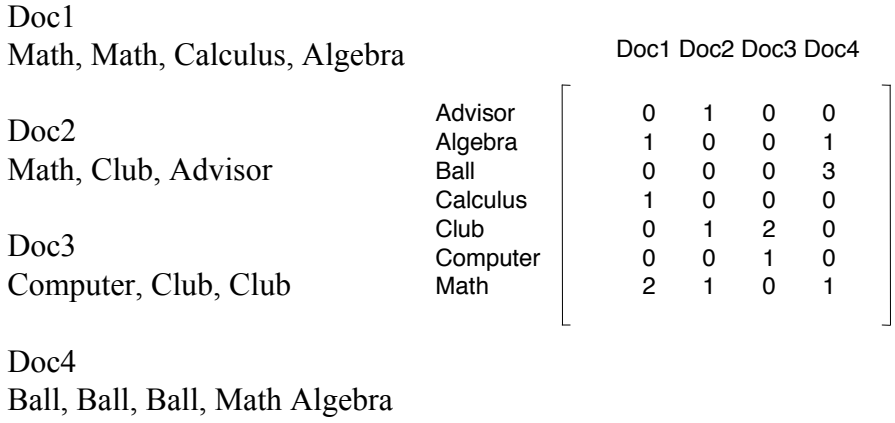

Figure 2: The term-by-document matrix for Example 9.1.

Example 9.1

Consider the term-by-document matrix of Figure 2. The words (=terms) of interest are Advisor, Algebra, Ball, Calculus, Club, Computer, and Math. There are four documents or web pages. Therefore the term-by-document matrix is of size 7×4. Notice that $a_{3,4} = 3$ shows that in document 4, the term Ball occurs 3 times, and $a_{5,2} = 1$ indicates that in document 2 the term Club occurs once. Term-by-document matrices are inherently sparse, since every word does not normally appear in each document. □

Example 9.2

The term-by-document matrix, Hypatia, of the web server of the mathematics department at the University of Rhode Island is of size $11,390 \times 1,265$ with $109,056$ non-zero elements. The matrix is *sparse* because only a small fraction of its entries are nonvanishing (less than 1%). The Matlab command `spy` can be used to visualize the sparsity structure of a matrix. Figure 3 depicts the sparsity structure of Hypatia. □

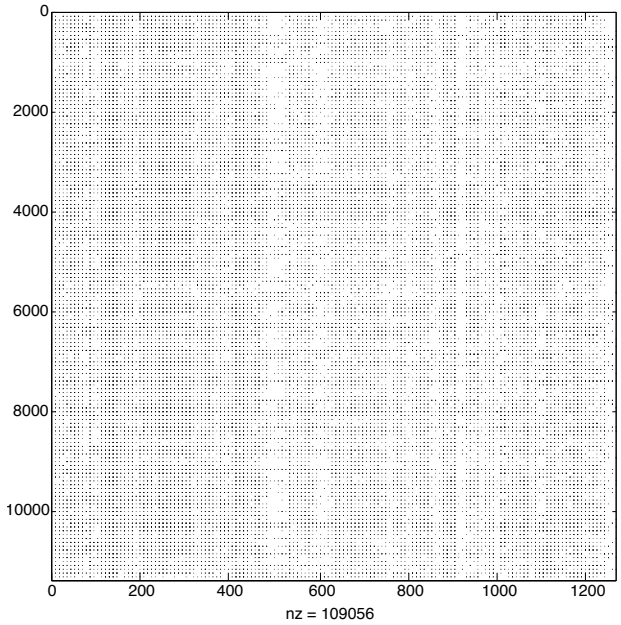

Figure 3: Sparsity pattern of the $11,390 \times 1,265$ term-by-document matrix Hypatia. This matrix is available at http://www.math.uri.edu/∼jbaglama/software/hypatia.gz

How does one search (i.e., query) a system to determine which documents contain the search element? A simple approach would be to return the nonvanishing elements of the row of the term-by-document matrix that match the search element.

Example 9.3

We would like to search for Club using the term-by-document matrix of Figure 2. The row corresponding to Club in this matrix is [0 1 2 0]. This shows that documents 2 and 3 contain the word Club.

Consider instead searching for the term Algebra. The row of the term-by-document matrix corresponding to Algebra is [1 0 0 1]. This identifies the documents 1 and 4. □

However, for very large matrices the approach of Examples 9.2 and 9.3 to identify documents is not practical. A reason for this is that there are many ways to express a given concept (club, organization, society) and the literal terms in a user's query may not match those of relevant documents. For this reason, the number of documents identified may be too small. On the other hand, many words have multiple meanings (club, stick) and therefore terms in a user's query will literally match terms in irrelevant documents. The number of documents identified therefore may be much too large. Moreover, typically one does not only want to know about matches, but one would

like to know the most relevant documents, i.e., one would like to rank the documents according to relevance.

Example 9.4

Consider the matrix A from Figure 2, and a query vector for Club,

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 3 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 0 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{bmatrix} \qquad \mathbf{q} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \begin{matrix} \text{Advisor} \\ \text{Algebra} \\ \text{Ball} \\ \text{Calculus} \\ \leftarrow \text{Club} \\ \text{Computer} \\ \text{Math} \end{matrix} \tag{1}$$

The angle θ_j between the column of $A(:, j)$ of a term-by-document matrix and the query vector q can be computed by

$$\cos(\theta_j) = \frac{\mathbf{q}^T A(:, j)}{\|\mathbf{q}\| \|A(:, j)\|}, \qquad j = 1, 2, , \dots \ . \tag{2}$$

Notice that when q is identical with the jth column of A, the angle between q and this column vanishes and the cosine of this angle is one. Thus, a large value of $\cos(\theta_j)$ suggests that document j may be relevant.

In particular, for the term-by-document matrix (1) and query vector q for Club we have,

$$\cos(\theta_1) = 0, \quad \cos(\theta_2) = 0.5774, \quad \cos(\theta_3) = 0.8944, \quad \cos(\theta_4) = 0. \tag{3}$$

This shows that document 3 is the best match. □

At this stage of the development of the vector space model for a search engine, using vectors and matrices does not seem different from simply returning the row of the term-by-document matrix that matches a term, such as Club. However, it will soon become evident why the Singular Value Decomposition (SVD) is needed.

Let us look at Example 9.1 visually in a two-dimensional plot. In order to be able to do this, we need to project our data into a two-dimensional space. The SVD of the term-by-document matrix A helps us to determine this mapping. Recall that the SVD of a matrix $A \in \mathbb{R}^{m \times n}$ with $m \geq n$ is given by

$$A = U \Sigma V^T, \tag{4}$$

where

$$U = [\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_m] \in \mathbb{R}^{m \times m} \quad \text{and} \quad V = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n] \in \mathbb{R}^{n \times n}$$

are orthogonal and the diagonal entries of

$$\Sigma = \text{diag}[\sigma_1, \sigma_2, \dots, \sigma_n] \in \mathbb{R}^{m \times n}$$

satisfy $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$. In applications of the SVD to document retrieval, the columns $\mathbf{u}_j$ of U commonly are referred to as term vectors and the columns $\mathbf{v}_j$ of V as document vectors.

The decomposition (4) can be expressed in terms of the columns of U and V explicitly,

$$A = \sum_{j=1}^{n} \sigma_j \mathbf{u}_j \mathbf{v}_j^T. \tag{5}$$

According to Exercise 7.7,

$$\|\sigma_j \mathbf{u}_j \mathbf{v}_j^T\| = \sigma_j.$$

Thus, the norm of the terms in the sum (5) are nonincreasing functions of j; generally, the norm decreases as j increases. This is illustrated by Figure 6 below.

Similarly as in Lectures 7 and 8, we approximate A by the matrix

$$A_k = \sum_{j=1}^{k} \sigma_j \mathbf{u}_j \mathbf{v}_j^T, \tag{6}$$

determined by ignoring the last $n - k$ terms in the sum (5). Let

$$U_k = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_k] \in \mathbb{R}^{m \times k}, \qquad V_k = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k] \in \mathbb{R}^{n \times k},$$

and

$$\Sigma_k = \text{diag}[\sigma_1, \sigma_2, \ldots, \sigma_k] \in \mathbb{R}^{k \times k}.$$

Then

$$A_k = U_k \Sigma_k V_k^T. \tag{7}$$

The projection of the scaled documents onto a k-dimensional space is given by

$$V_k \Sigma_k = A^T U_k, \tag{8}$$

see Exercise 9.1, and, similarly, the projection of the scaled terms onto a k-dimensional space can be expressed as

$$U_k \Sigma_k = A V_k. \tag{9}$$

In particular, we can determine a two-dimensional plot of the terms and documents by using the first two columns of $U_k \Sigma_k$ and $V_k \Sigma_k$.

Example 9.5

Regard Figure 4, which shows the two-dimensional projection of the terms and documents of Example 9.1. Let $\mathbf{e}_j = [0, \ldots, 0, 1, 0, \ldots, 0]^T$ denote the jth axis vector. The point

$$\mathbf{e}_5^T U_2 \Sigma_2 = (\sigma_1 u_{1,5}, \sigma_2 u_{2,5}) = (-0.2897, 2.0005)$$

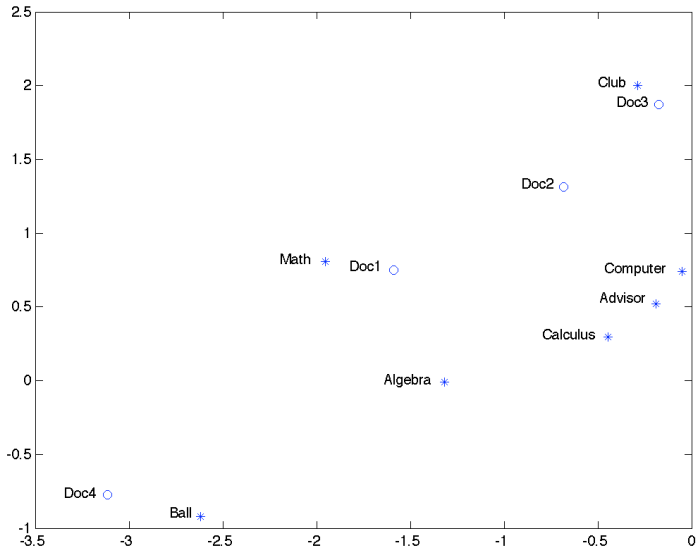

Figure 4: Two-dimensional plot of the projected terms and documents of Example 9.1.

is the projection of the 5th term (Club), where $u_{j,5}$ denotes the 5th entry of the vector u_j; see Tables 1 and 2. Similarly,

$$\mathbf{e}_3^T V_2 \Sigma_2 = (\sigma_1 v_{1,3}, \sigma_2 v_{2,3}) = (-0.1761, 1.8738)$$

	x-coordinates	y-coordinates
Terms	$\sigma_1 u_1$	$\sigma_2 u_2$
Documents	$\sigma_1 v_1$	$\sigma_2 v_2$

Table 1: Data for Example 9.5.

$\sigma_1 \mathbf{u}_1$	$\sigma_2 \mathbf{u}_2$	$\sigma_1 \mathbf{v}_1$	$\sigma_2 \mathbf{v}_2$
−0.1911	0.5194	−1.5889	0.7502
−1.3185	−0.0100	−0.6821	1.3144
−2.6205	−0.9194	−0.1761	1.8738
−0.4450	0.2965	−3.1187	−0.7755
−0.2897	2.0005		
−0.0493	0.7405		
−1.9546	0.8059		

Table 2: Data for Example 9.5.

is the projection of the 3rd document. The projections of the 5th term and 3rd document are close, which indicates that Doc3 is likely to be a relevant document when we search for Club, in agreement with the discussion above.

However, if we search for Algebra (the 2nd term), then it is not so obvious from Figure 4 which document is most relevant. In order find out, we project the query vector for Algebra ($\mathbf{q} = [0\ 1\ 0\ 0\ 0\ 0\ 0\ ]^T$) onto the document space. The scaled projection of the query vector is given by

$$\mathbf{q}^{(p)} = \mathbf{q}^T U_k.$$

A comparison with equation (8) shows that the query vector $\mathbf{q}$ is projected the same way as the documents, i.e., the query vector can be thought of as a column of A, i.e., as another document. If the angle between the vector $\mathbf{q}^{(p)}$ and the projected document is small, then the document may contain information related to the query.

Figure 5 displays the projected query vector $\mathbf{q}^{(p)}$ for Algebra along with plots of the projected terms and documents. Note that Algebra does not appear in Doc2 in Example 9.1, however, Figure 5 shows the angle between $\mathbf{q}^{(p)}$ and the projection of Doc2 to be smaller than 90^o. This indicates that Doc2 may be somewhat relevant to Algebra.

Doc2 contains the words Club, Math, and Advisor, and Doc4 and Doc1 contain the words Math and Algebra. All three documents contain the word Math, and therefore the documents may be relevant to the query. Hence, Doc2 could be relevant. Also, notice that Doc3 does not contain the words Math and Algebra, which indicates that this document is unrelated to the query. In fact, the angle between the query vector for Algebra and Doc3 in Figure 5 is almost 90^o. □

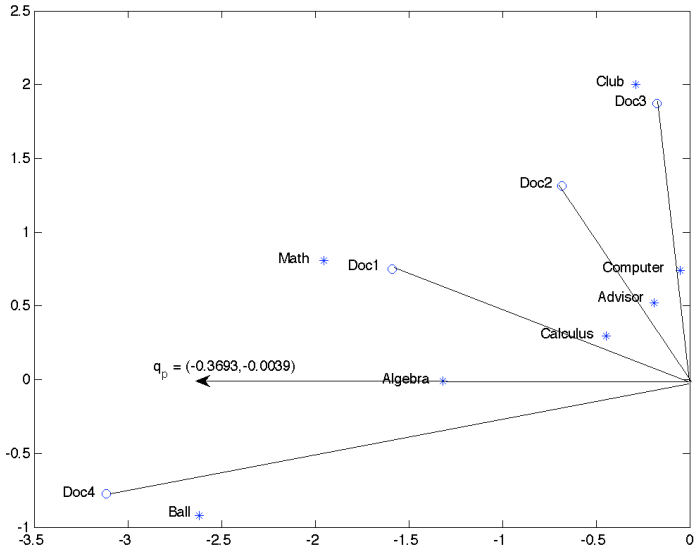

Figure 5: Two-dimensional plot of the projected query vector $\mathbf{q}^{(p)}$ for Algebra and of the projected terms and documents of Example 9.1.

Literally matching terms in documents with a query does not always yield clear answers. A better approach is to match a query with the meaning of a document. *Latent Semantic Indexing* (LSI) does just that. It overcomes the problems of lexical matching by using statistically derived quantities instead of individual words for matching.

9.3 Latent semantic indexing (LSI)

Latent semantic indexing examines the whole document collection to determine which documents contain similar words. LSI considers documents that have words in common to be semantically close, and documents with few words in common to be semantically distant.

Let A_k be the rank k approximation (6) of A. Working with A_k instead of A has several advantages. Since the small terms in the expansion (5) are neglected the important information, which is represented by the first terms in the expansion, is now easier accessible. Moreover, we can store A_k in factored form (7) without explicitly forming the elements of A_k. For large matrices A, storage of A_k in this manner typically requires much less computer memory than storage of the matrix A. We remark that there are methods for computing the matrices U_k, V_k, and Σ_k without forming A; only matrix-vector products with A and A^T have to be evaluated.

The value of k in A_k is typically chosen experimentally. Even for a very large number of documents, $k = 100$ often gives acceptable results. Figure 6 illustrates the magnitude of the

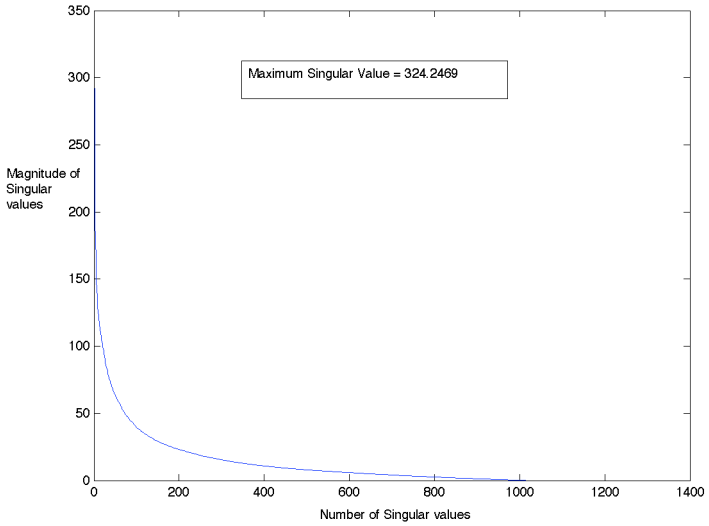

Figure 6: Singular values of the $11,390 \times 1,265$ term-by-document matrix Hypatia.

singular values of the large term-by document matrix Hypatia. Notice that the singular values σ_j decrease to zero very quickly, and a choice of $k = 100$ would be acceptable. We remark that choosing k very large does not always result in better answers to queries. The reason for this is that increasing k increases the search space. The added data is associated with small singular values and does not always contain relevant information.

LSI works with the matrix A_k instead of with A. Specifically, one determines the angle between the query vector q and every column of A_k. These angles can be computed similarly as the angles (2). Thus, replacing A by A_k in (2) yields

$$\cos(\theta_j) = \frac{\mathbf{q}^T A_k \mathbf{e}_j}{\|\mathbf{q}\| \, \|A_k \mathbf{e}_j\|} = \frac{(\mathbf{q}^T U_k)(\Sigma_k V_k^T \mathbf{e}_j)}{\|\mathbf{q}\| \, \|\Sigma_k V_k^T \mathbf{e}_j\|}, \qquad j = 1, 2, \ldots, k, \tag{10}$$

where $\mathbf{e}_j$ denotes the jth axis vector. Note that $\mathbf{q}^T U_k$ and $\Sigma_k V_k$ are projections of the query vector and document vectors into k-dimensional spaces; cf. (8).

Now returning to Example 7.1 and using equation (10) with $k = 2$ for LSI, we obtain the following results. For Club, we have

$$\cos(\theta_1) = 0.4109, \quad \cos(\theta_2) = 0.7391, \quad \cos(\theta_3) = 0.7947, \quad \cos(\theta_4) = -0.1120,$$

which gives the ordering Doc3, Doc2, Doc1, and Doc4 of the documents, with Doc3 being the most relevant and Doc4 the least relevant document. The query vector for Algebra yields

$$\cos(\theta_1) = 0.3323, \quad \cos(\theta_2) = 0.1666, \quad \cos(\theta_3) = 0.0306, \quad \cos(\theta_4) = 0.3593,$$

which determines the ordering Doc4, Doc1, Doc2, and Doc3. Notice that the cosine of the angle between Algebra and Doc3 is close to zero. This indicates that Doc3 is irrelevant for the query. The angles reveal that Doc2 has some connection to Algebra.

Finally, we note that weighting of the terms in a term-by-document matrix can affect query results significantly. Documents that have many words or repeat the same word many times are erroneously given a high ranking without term weighting. The following section discusses this topic.

9.4 Term weighting

The weighting of the terms in a term-by-document matrix A can be carried out as follows. Set

$$a_{i,j} = g_i \cdot t_{i,j} \cdot d_j,$$

where g_i is the global weight of the ith term, $t_{i,j}$ is the local weight of the ith term in the jth document, and d_j is the normalization factor for document j. We will define term frequency $f_{i,j}$ as the number of times the ith term appears in the jth document.

Local weighting depends only on the term within a document and not the occurrence of the term in other documents. Global weighting seeks to give a value to each term, depending on the occurrences of the term in all documents. For instance, global weighting is used to correct for differences in document lengths. In the tables below, n is the number of documents.

$t_{i,j}$	Local weighting scheme
$\chi(f_{i,j})$	1 if the ith term is in the jth document and 0 otherwise
$f_{i,j}$	Frequency of the ith term in the jth document This weighting is used in Example 9.1.
$\log(f_{i,j}+1)$	Logarithmic frequency to reduce the effects of large term frequency (most used local weighting scheme)

Table 3: Examples of local weighting schemes.

Exercise 9.1

Show the formulas (8) and (9). □

Exercise 9.2

a) Reproduce Figure 4.
b) Query for the term Calculus and produce a plot analogous to Figure 5. □

g_i	Global weighting schemes
1	None
$\log(\frac{n}{\sum_{k=1}^{n} \chi(f_{i,k})})$	Inverse document frequency (IDF), emphasizes rare words (most used global weighting scheme)
$\frac{\sum_{k=1}^{n} f_{i,k}}{\sum_{k=1}^{n} \chi(f_{i,k})}$	GFIDF
$1 + \sum_{j=1}^{n} \frac{p_{i,j} \cdot \log(p_{i,j})}{\log(n)}$, $p_{i,j} = \frac{f_{ij}}{\sum_{k=1}^{n} f_{i,k}}$	Entropy, gives higher weights for terms with low frequency

Table 4: Examples of global weighting schemes.

d_j	Normalization schemes
1	None
$\sqrt{\sum_{k=1}^{n} (g_k \cdot t_{k,j})^2}$	Cosine normalization (most used normalization scheme)

Table 5: Examples of normalization schemes.

Exercise 9.3

a) Use the local weighting scheme $t_{i,j} = \log(f_{ij} + 1)$ (logarithmic frequency) for Example 9.1 and produce a two-dimensional plot of the terms and documents similar to Figure 4.
b) Query for the term Calculus and produce a plot similar to Figure 5. □

Exercise 9.4

a) Use the global weighting scheme $\log(\frac{n}{\sum_{k=1}^{n} \chi(f_{i,k})})$ (IDF) for Example 9.1 and produce a two-dimensional plot of the terms and documents similar to Figure 4.
b) Query for the term Calculus and produce a plot similar to Figure 5. □

Exercise 9.5

a) Use both the local weighting scheme $t_{i,j} = \log(f_{ij} + 1)$ (logarithmic frequency) and the global weighting scheme $\log(\frac{n}{\sum_{k=1}^{n} \chi(f_{i,k})})$ (IDF) for Example 9.1 and produce a two-dimensional plot of the terms and documents similar to Figure 4.
b) Query for the term Calculus and produce a plot similar to Figure 5. □

Exercise 9.6

Which was the best method, no weighting, local weighting only, global weighting only, or both local and global weighting? Repeat your searches with a query vector for Calculus and Algebra. What happens? What happens when you query for Calculus and Ball? □

9.5 Project

We will investigate the LSI search engine method for a small set of FAQ for MATLAB. In the directory `http://www.math.uri.edu/~/jbaglama/faq/` there are 68 frequently asked questions about MATLAB. The questions are in individual text files (i.e., documents)

```
http://www.math.uri.edu/~/jbaglama/faq/q1.txt
http://www.math.uri.edu/~/jbaglama/faq/q2.txt
      ⋮     ⋮
http://www.math.uri.edu/~/jbaglama/faq/q68.txt
```

The file `http://www.math.uri.edu/~/jbaglama/faq/documents.txt` contains a list of the above documents. The file `http://www.math.uri.edu/~/jbaglama/faq/terms.txt` contains the list of all relevant terms in alphabetical order. There are 1338 terms. The file `http://www.math.uri.edu/~/jb` contains the entries of the 1338×68 term-by-document matrix. The file `matrix.txt` was created from the alphabetical list of terms and the sequential ordering of the questions. The first couple lines of the file `matrix.txt` are:

```
%term by document matrix for FAQ
1338 68 2374
1 12 1
2 56 1
3 32 1
4 12 1
5 28 1
6 58 1
7 58 1
8 9 2
⋮
```

The first line is a comment line and the second line yields the number of columns, the number of rows, and the number of non-zero entries. Only non-zero entries are stored. The $(2, 56)$ entry of the term-by-document matrix is equal to 1. This can be seen from line 4 of the file. The $(2, 56)$ entry of the term-by-document matrix is associated with the word absolute in question 56. The weighting scheme is the same as in Example 9.1; only local term frequency is used.

Part 1:
Create a MATLAB code that can read the matrix in the file
`http://www.math.uri.edu/~/jbaglama/faq/matrix.txt` into sparse format and plot the sparsity structure using the command `spy`, similarly to Figure 3.

Part 2:
Use both the local weighting scheme $t_{i,j} = \log(f_{ij} + 1)$ (Logarithmic frequency) and the global weighting scheme $\log(\frac{n}{\sum_{k=1}^{n} \chi(f_{i,k})})$ (IDF), and produce a two-dimensional plot of only the questions $q_1, q_2, \ldots, q_{68}$.

Part 3:
Plot the query vector for "variables" on the same axis as Part 2. Determine which documents match the best. Plot the query vector for "global" on the same axis as Part 2 and determine which documents are most relevant.

References

[1] M. W. Berry and M. Browne, *Understanding Search Engines: Mathematical Modeling and Text Retrieval*, 2nd ed., SIAM, Philadelphia, 2005.

[2] N. Polettini, *The Vector Space Model in Information Retrieval - Term Weighting Problem*, http://sra.itc.it/people/polettini/PAPERS/Polettini _Information_Retrieval.pdf, 2004.