Chapter 11
Text Mining

By *text mining* we mean methods for extracting useful information from large and often unstructured collections of texts. Another, closely related term is *information retrieval*. A typical application is searching databases of abstracts of scientific papers. For instance, in a medical application one may want to find all the abstracts in the database that deal with a particular syndrome. So one puts together a search phrase, a *query*, with keywords that are relevant to the syndrome. Then the retrieval system is used to match the query to the documents in the database and presents to the user the documents that are relevant, ranked according to relevance.

Example 11.1. The following is a typical query for search in a collection of medical abstracts:

> 9. *the use of induced hypothermia in heart surgery, neurosurgery, head injuries, and infectious diseases.*

The query is taken from a test collection for information retrieval, called Medline.[22] We will refer to this query as Q9. ∎

Library catalogues are another example of text mining applications.

Example 11.2. To illustrate one issue in information retrieval, we performed a search in the Linköping University library journal catalogue:

Search phrases	Results
computer science engineering	Nothing found
computing science engineering	IEEE: Computing in Science and Engineering

Naturally we would like the system to be insensitive to small errors on the part of the user. Anyone can see that the IEEE journal is close to the query. From

[22] See, e.g., http://www.dcs.gla.ac.uk/idom/ir_resources/test_collections/.

this example we conclude that in many cases straightforward word matching is not good enough. ∎

A very well known area of text mining is Web search engines, where the search phrase is usually very short, and often there are so many relevant documents that it is out of the question to present them all to the user. In that application the ranking of the search result is critical for the efficiency of the search engine. We will come back to this problem in Chapter 12.

For an overview of information retrieval, see, e.g., [43]. In this chapter we will describe briefly one of the most common methods for text mining, namely, the *vector space model* [81]. Here we give a brief overview of the vector space model and some variants: latent semantic indexing (LSI), which uses the SVD of the term-document matrix, a clustering-based method, nonnegative matrix factorization, and LGK bidiagonalization. For a more detailed account of the different techniques used in connection with the vector space model, see [12].

11.1 Preprocessing the Documents and Queries

In this section we discuss the preprocessing that is done to the texts before the vector space model of a particular collection of documents is set up.

In information retrieval, keywords that carry information about the contents of a document are called *terms*. A basic step in information retrieval is to create a list of all the terms in a document collection, a so-called index. For each term, a list is stored of all the documents that contain that particular term. This is called an inverted index.

But before the index is made, two preprocessing steps must be done: elimination of all stop words and stemming.

Stop words are words that one can find in virtually any document. Therefore, the occurrence of such a word in a document does not distinguish this document from other documents. The following is the beginning of one particular stop list:[23]

> a, a's, able, about, above, according, accordingly, across, actually, after, afterwards, again, against, ain't, all, allow, allows, almost, alone, along, already, also, although, always, am, among, amongst, an, and, another, any, anybody, anyhow, anyone, anything, anyway, anyways, anywhere, apart, appear, appreciate, appropriate, are, aren't, around, as, aside, ask, ...

Stemming is the process of reducing each word that is conjugated or has a suffix to its stem. Clearly, from the point of view of information retrieval, no information is lost in the following reduction:

[23] ftp://ftp.cs.cornell.edu/pub/smart/english.stop.

$$\left.\begin{array}{r}\textbf{comput}\text{able}\\ \textbf{comput}\text{ation}\\ \textbf{comput}\text{ing}\\ \textbf{comput}\text{ed}\\ \textbf{comput}\text{ational}\end{array}\right\} \longrightarrow \quad \textbf{comput}$$

Public domain stemming algorithms are available on the Internet.[24]

Table 11.1. *The beginning of the index for the Medline collection. The Porter stemmer and the GTP parser* [38] *were used.*

without stemming	with stemming
action	action
actions	
activation	activ
active	
actively	
activities	
activity	
acts	
actual	actual
actually	
acuity	acuiti
acute	acut
ad	ad
adaptation	adapt
adaptations	
adaptive	
add	add
added	
addition	addit
additional	

Example 11.3. We parsed the 1063 documents (actually 30 queries and 1033 documents) in the Medline collection, with and without stemming, in both cases removing stop words. For consistency it is necessary to perform the same stemming to the stop list. In the first case, the number of terms was 5839 and in the second was 4281. We show partial lists of terms in Table 11.1. ∎

11.2 The Vector Space Model

The main idea in the vector space model is to create a *term-document matrix*, where each document is represented by a column vector. The column has nonzero

[24] http://www.comp.lancs.ac.uk/computing/research/stemming/ and http://www.tartarus.org/~martin/PorterStemmer/.

entries in the positions that correspond to terms that can be found in the document. Consequently, each row represents a term and has nonzero entries in those positions that correspond to the documents where the term can be found; cf. the inverted index in Section 11.1.

A simplified example of a term-document matrix is given in Chapter 1. There we manually counted the frequency of the terms. For realistic problems one uses a *text parser* to create the term-document matrix. Two public-domain parsers are described in [38, 113]. Unless otherwise stated, we have used the one from [113] for the larger examples in this chapter. Text parsers for information retrieval usually include both a stemmer and an option to remove stop words. In addition there are filters, e.g., for removing formatting code in the documents, e.g., HTML or XML.

It is common not only to count the occurrence of terms in documents but also to apply a *term weighting scheme*, where the elements of A are weighted depending on the characteristics of the document collection. Similarly, document weighting is usually done. A number of schemes are described in [12, Section 3.2.1]. For example, one can define the elements in A by

$$a_{ij} = f_{ij} \log(n/n_i), \tag{11.1}$$

where f_{ij} is term frequency, the number of times term i appears in document j, and n_i is the number of documents that contain term i (inverse document frequency). If a term occurs frequently in only a few documents, then both factors are large. In this case the term discriminates well between different groups of documents, and the log-factor in (11.1) gives it a large weight in the documents where it appears.

Normally, the term-document matrix is *sparse*: most of the matrix elements are equal to zero. Then, of course, one avoids storing all the zeros and uses instead a sparse matrix storage scheme; see Section 15.7.

Example 11.4. For the stemmed Medline collection, parsed using GTP [38], the matrix (including 30 query columns) is 4163×1063 with 48263 nonzero elements, i.e., approximately 1%. The first 500 rows and columns of the matrix are illustrated in Figure 11.1. ∎

11.2.1 Query Matching and Performance Modeling

Query matching is the process of finding the documents that are relevant to a particular query q. This is often done using the cosine distance measure: a document a_j is deemed relevant if the angle between the query q and a_j is small enough. Equivalently, a_j is retrieved if

$$\cos(\theta(q, a_j)) = \frac{q^T a_j}{\| q \|_2 \| a_j \|_2} > \text{tol},$$

where tol is a predefined tolerance. If the tolerance is lowered, then more documents are returned, and it is likely that many of those are relevant to the query. But at

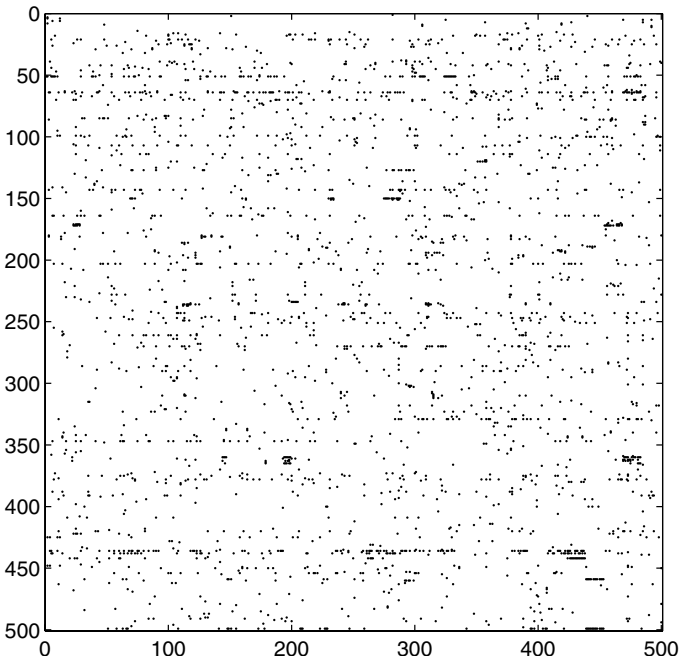


Figure 11.1. *The first* 500 *rows and columns of the Medline matrix. Each dot represents a nonzero element.*

the same time there is a risk that when the tolerance is lowered, more and more documents that are not relevant are also returned.

Example 11.5. We did query matching for query Q9 in the stemmed Medline collection. With tol = 0.19 for the cosine measure, only document 409 was considered relevant. When the tolerance was lowered to 0.17, documents 415 and 467 also were retrieved. ∎

We illustrate the different categories of documents in a query matching for two values of the tolerance in Figure 11.2. The query matching produces a good result when the intersection between the two sets of returned and relevant documents is as large as possible and the number of returned irrelevant documents is small. For a high value of the tolerance, the retrieved documents are likely to be relevant (the small circle in Figure 11.2). When the cosine tolerance is lowered, the intersection is increased, but at the same time, more irrelevant documents are returned.

In performance modeling for information retrieval we define the following measures:

$$\textit{Precision:} \qquad P = \frac{D_r}{D_t}, \tag{11.2}$$

where D_r is the number of relevant documents retrieved and D_t the total number

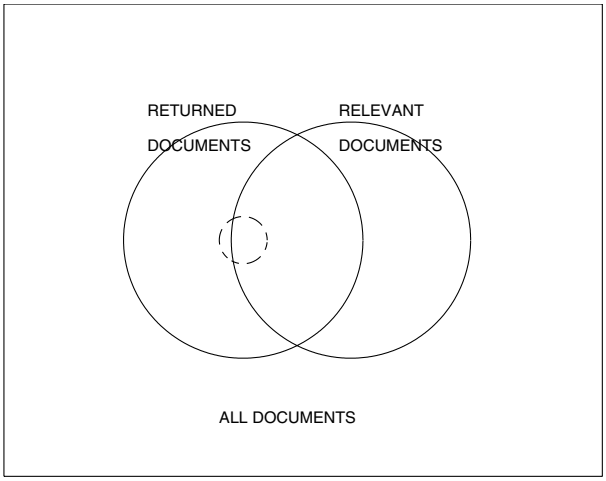


Figure 11.2. *Retrieved and relevant documents for two values of the tolerance. The dashed circle represents the retrieved documents for a high value of the cosine tolerance.*

of documents retrieved, and

$$\textit{Recall}: \qquad R = \frac{D_r}{N_r}, \tag{11.3}$$

where N_r is the total number of relevant documents in the database. With a large value of tol for the cosine measure, we expect to have high precision but low recall. For a small value of tol, we will have high recall but low precision.

In the evaluation of different methods and models for information retrieval, usually a number of queries are used. For testing purposes, all documents have been read by a human, and those that are relevant for a certain query are marked. This makes it possible to draw diagrams of recall versus precision that illustrate the performance of a certain method for information retrieval.

Example 11.6. We did query matching for query Q9 in the Medline collection (stemmed) using the cosine measure. For a specific value of the tolerance, we computed the corresponding recall and precision from (11.2) and (11.3). By varying the tolerance from close to 1 down to zero, we obtained vectors of recall and precision that gave information about the quality of the retrieval method for this query. In the comparison of different methods it is illustrative to draw the recall versus precision diagram as in Figure 11.3. Ideally a method has high recall at the same time as the precision is high. Thus, the closer the curve is to the upper right corner, the higher the retrieval quality.

In this example and the following examples, the matrix elements were computed using term frequency and inverse document frequency weighting (11.1). ∎

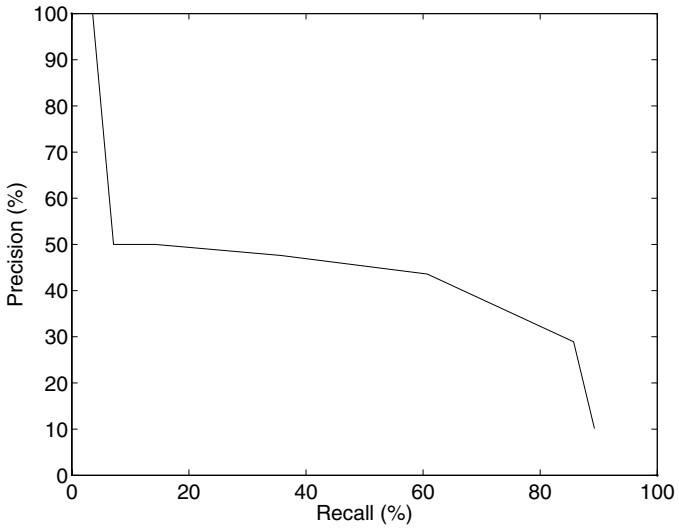
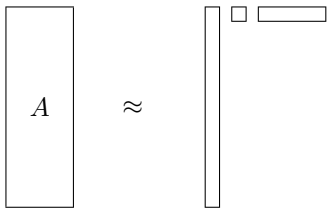


Figure 11.3. *Query matching for* $Q9$ *using the vector space method. Recall versus precision.*

11.3 Latent Semantic Indexing

Latent semantic indexing[25] (LSI) [28, 9] "is based on the assumption that there is some underlying latent semantic structure in the data ... that is corrupted by the wide variety of words used" [76] and that this semantic structure can be discovered and enhanced by projecting the data (the term-document matrix and the queries) onto a lower-dimensional space using the SVD.

Let $A = U\Sigma V^T$ be the SVD of the term-document matrix and approximate it by a matrix of rank k:



$$= U_k \Sigma_k V_k^T =: U_k H_k.$$

The columns of U_k live in the document space and are an orthogonal basis that we use to approximate the documents. Write H_k in terms of its column vectors, $H_k = (h_1, h_2, \ldots, h_n)$. From $A \approx U_k H_k$ we have $a_j \approx U_k h_j$, which means that column j of H_k holds the coordinates of document j in terms of the orthogonal

[25] Sometimes also called latent semantic analysis (LSA) [52].

basis. With this rank-k approximation the term-document matrix is represented by $A_k = U_k H_k$ and in query matching we compute $q^T A_k = q^T U_k H_k = (U_k^T q)^T H_k$. Thus, we compute the coordinates of the query in terms of the new document basis and compute the cosines from

$$\cos\theta_j = \frac{q_k^T h_j}{\|q_k\|_2 \, \|h_j\|_2}, \qquad q_k = U_k^T q. \tag{11.4}$$

This means that the query matching is performed in a k-dimensional space.

Example 11.7. We did query matching for Q9 in the Medline collection, approximating the matrix using the truncated SVD of rank 100.

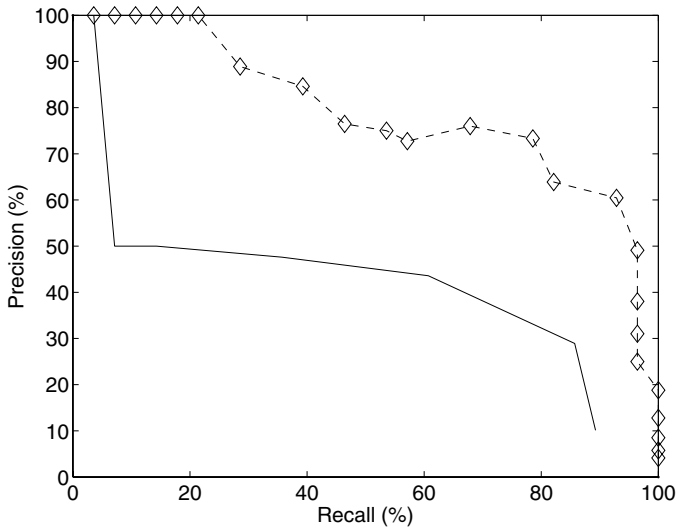


Figure 11.4. *Query matching for Q9. Recall versus precision for the full vector space model (solid line) and the rank 100 approximation (dashed line and diamonds).*

The recall precision curve is given in Figure 11.4. It is seen that for this query, LSI improves the retrieval performance. In Figure 11.5 we also demonstrate a fact that is common to many term-document matrices: it is rather well-conditioned and there is no gap in the sequence of singular values. Therefore, we cannot find a suitable rank of the LSI approximation by inspecting the singular values; it must be determined by retrieval experiments.

Another remarkable fact is that with $k = 100$ the approximation error in the matrix approximation,

$$\frac{\| A - A_k \|_F}{\| A \|_F} \approx 0.8,$$

is large, and we still get *improved retrieval performance*. In view of the large approximation error in the truncated SVD approximation of the term-document matrix,

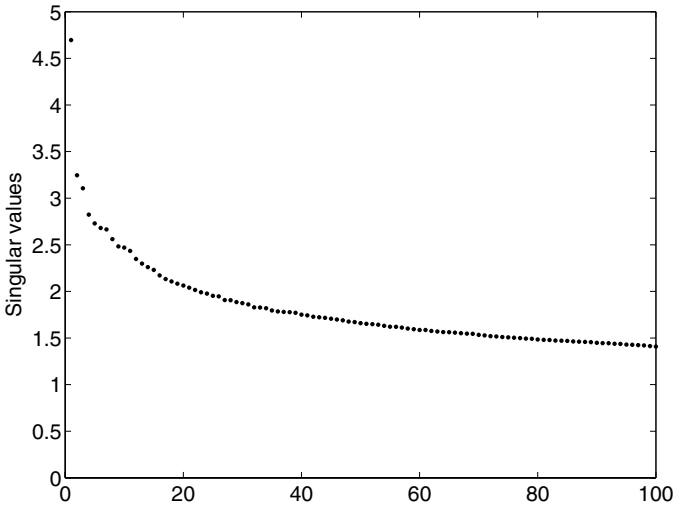


Figure 11.5. *First* 100 *singular values of the Medline* (*stemmed*) *matrix. The matrix columns are scaled to unit Euclidean length.*

one may question whether the "optimal" singular vectors constitute the best basis for representing the term-document matrix. On the other hand, since we get such good results, perhaps a more natural conclusion may be that the Frobenius norm is not a good measure of the information contents in the term-document matrix.

It is also interesting to see what are the most important "directions" in the data. From Theorem 6.6 we know that the first few left singular vectors are the dominant directions in the document space, and their largest components should indicate what these directions are. The MATLAB statement `find(abs(U(:,k))>0.13)`, combined with look-up in the index of terms, gave the following results for `k=1,2`:

`U(:,1)`	`U(:,2)`
cell	case
growth	cell
hormone	children
patient	defect
	dna
	growth
	patient
	ventricular

In Chapter 13 we will come back to the problem of extracting the keywords from texts. ∎

It should be said that LSI does not give significantly better results for all queries in the Medline collection: there are some in which it gives results comparable to the full vector model and some in which it gives worse performance. However, it

is often the average performance that matters.

A systematic study of different aspects of LSI was done in [52]. It was shown that LSI improves retrieval performance for surprisingly small values of the reduced rank k. At the same time, the relative matrix approximation errors are large. It is probably not possible to prove any general results that explain in what way and for which data LSI can improve retrieval performance. Instead we give an artificial example (constructed using similar ideas as a corresponding example in [12]) that gives a partial explanation.

Example 11.8. Consider the term-document matrix from Example 1.1 and the query "**ranking** of **Web pages**." Obviously, Documents 1–4 are relevant with respect to the query, while Document 5 is totally irrelevant. However, we obtain cosines for the query and the original data as

$$\begin{pmatrix} 0 & 0.6667 & 0.7746 & 0.3333 & 0.3333 \end{pmatrix},$$

which indicates that Document 5 (the football document) is as relevant to the query as Document 4. Further, since none of the words of the query occurs in Document 1, this document is orthogonal to the query.

We then compute the SVD of the term-document matrix and use a rank-2 approximation. After projection to the two-dimensional subspace, the cosines, computed according to (11.4), are

$$\begin{pmatrix} 0.7857 & 0.8332 & 0.9670 & 0.4873 & 0.1819 \end{pmatrix}.$$

It turns out that Document 1, which was deemed totally irrelevant for the query in the original representation, is now highly relevant. In addition, the cosines for the relevant Documents 2–4 have been reinforced. At the same time, the cosine for Document 5 has been significantly reduced. Thus, in this artificial example, the dimension reduction enhanced the retrieval performance.

In Figure 11.6 we plot the five documents and the query in the coordinate system of the first two left singular vectors. Obviously, in this representation, the first document is closer to the query than Document 5. The first two left singular vectors are

$$u_1 = \begin{pmatrix} 0.1425 \\ 0.0787 \\ 0.0787 \\ 0.3924 \\ 0.1297 \\ 0.1020 \\ 0.5348 \\ 0.3647 \\ 0.4838 \\ 0.3647 \end{pmatrix}, \qquad u_2 = \begin{pmatrix} 0.2430 \\ 0.2607 \\ 0.2607 \\ -0.0274 \\ 0.0740 \\ -0.3735 \\ 0.2156 \\ -0.4749 \\ 0.4023 \\ -0.4749 \end{pmatrix},$$

and the singular values are $\Sigma = \mathrm{diag}(2.8546, 1.8823, 1.7321, 1.2603, 0.8483)$. The first four columns in A are strongly coupled via the words *Google*, *matrix*, etc., and

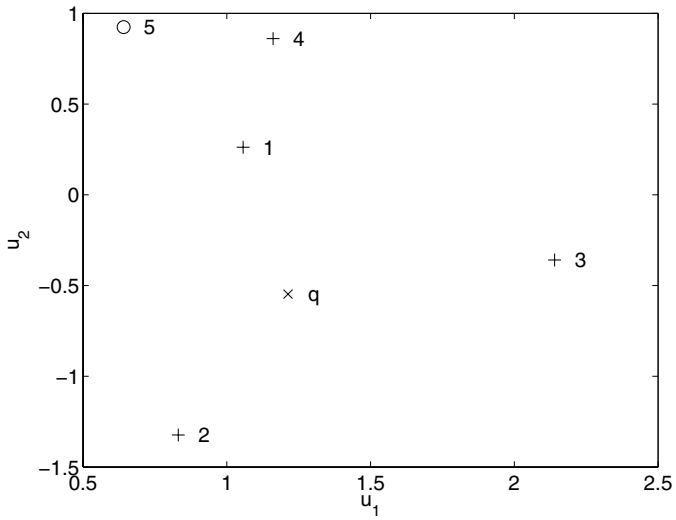


Figure 11.6. *The five documents and the query projected to the coordinate system of the first two left singular vectors.*

those words are the dominating contents of the document collection (cf. the singular values). This shows in the composition of u_1. So even if none of the words in the query are matched by Document 1, that document is so strongly correlated to the dominating direction that it becomes relevant in the reduced representation. ∎

11.4 Clustering

In the case of document collections, it is natural to assume that there are groups of documents with similar contents. If we think of the documents as points in $\mathbb{R}^m$, we may be able to visualize the groups as clusters. Representing each cluster by its mean value, the *centroid*,[26] we can compress the data in terms of the centroids. Thus clustering, using the k-means algorithm, for instance, is another method for low-rank approximation of the term-document matrix. The application of clustering to information retrieval is described in [30, 76, 77].

In analogy to LSI, the matrix $C_k \in \mathbb{R}^{m\times k}$ of (normalized but not orthogonal) centroids can be used as an approximate basis in the "document space." For query matching we then need to determine the coordinates of all the documents in this basis. This can be made by solving the matrix least squares problem,

$$\min_{\hat{G}_k} \| A - C_k \hat{G}_k \|_F.$$

However, it is more convenient first to orthogonalize the columns of C, i.e., compute

[26] Closely related to the *concept vector* [30].