Chapter 1

Vectors and Matrices in Data Mining and Pattern Recognition

1.1 Data Mining and Pattern Recognition

In modern society, huge amounts of data are collected and stored in computers so that useful information can later be extracted. Often it is not known at the time of collection what data will later be requested, and therefore the database is not designed to distill any particular information, but rather it is, to a large extent, unstructured. The science of extracting useful information from large data sets is usually referred to as "data mining," sometimes with the addition of "knowledge discovery."

Pattern recognition is often considered to be a technique separate from data mining, but its definition is related: "the act of taking in raw data and making an action based on the 'category' of the pattern" [31]. In this book we will not emphasize the differences between the concepts.

There are numerous application areas for data mining, ranging from e-business [10, 69] to bioinformatics [6], from scientific applications such as the classification of volcanos on Venus [21] to information retrieval [3] and Internet search engines [11].

Data mining is a truly interdisciplinary science, in which techniques from computer science, statistics and data analysis, linear algebra, and optimization are used, often in a rather eclectic manner. Due to the practical importance of the applications, there are now numerous books and surveys in the area [24, 25, 31, 35, 45, 46, 47, 49, 108].

It is not an exaggeration to state that everyday life is filled with situations in which we depend, often unknowingly, on advanced mathematical methods for data mining. Methods such as linear algebra and data analysis are basic ingredients in many data mining techniques. This book gives an introduction to the mathematical and numerical methods and their use in data mining and pattern recognition.

1.2 Vectors and Matrices

The following examples illustrate the use of vectors and matrices in data mining. These examples present the main data mining areas discussed in the book, and they will be described in more detail in Part II.

In many applications a matrix is just a rectangular array of data, and the elements are scalar, real numbers:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \in \mathbb{R}^{m \times n}.$$

To treat the data by mathematical methods, some mathematical structure must be added. In the simplest case, the columns of the matrix are considered as vectors in $\mathbb{R}^m$.

Example 1.1. *Term-document matrices* are used in *information retrieval*. Consider the following selection of five documents.[1] Key words, which we call *terms*, are marked in boldface.[2]

Document 1:	The **Google™ matrix** P is a model of the **Internet**.
Document 2:	P_{ij} is nonzero if there is a **link** from **Web page** j to i.
Document 3:	The **Google matrix** is used to **rank** all **Web pages**.
Document 4:	The **ranking** is done by solving a **matrix eigenvalue** problem.
Document 5:	**England** dropped out of the top 10 in the **FIFA ranking**.

If we count the frequency of terms in each document we get the following result:

Term	Doc 1	Doc 2	Doc 3	Doc 4	Doc 5
eigenvalue	0	0	0	1	0
England	0	0	0	0	1
FIFA	0	0	0	0	1
Google	1	0	1	0	0
Internet	1	0	0	0	0
link	0	1	0	0	0
matrix	1	0	1	1	0
page	0	1	1	0	0
rank	0	0	1	1	1
Web	0	1	1	0	0

[1] In Document 5, FIFA is the Fédération Internationale de Football Association. This document is clearly concerned with football (soccer). The document is a newspaper headline from 2005. After the 2006 World Cup, England came back into the top 10.

[2] To avoid making the example too large, we have ignored some words that would normally be considered as terms (key words). Note also that only the stem of the word is significant: "ranking" is considered the same as "rank."

Thus each document is represented by a vector, or a point, in $\mathbb{R}^{10}$, and we can organize all documents into a term-document matrix:

$$A = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix}.$$

Now assume that we want to find all documents that are relevant to the query "**ranking** of **Web pages**." This is represented by a *query* vector, constructed in a way analogous to the term-document matrix:

$$q = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} \in \mathbb{R}^{10}.$$

Thus the query itself is considered as a document. The information retrieval task can now be formulated as a mathematical problem: *find the columns of A that are close to the vector q.* To solve this problem we must use some distance measure in $\mathbb{R}^{10}$.

In the information retrieval application it is common that the dimension m is large, of the order 10^6, say. Also, as most of the documents contain only a small fraction of the terms, most of the elements in the matrix are equal to zero. Such a matrix is called *sparse*.

Some methods for information retrieval use linear algebra techniques (e.g., singular value decomposition (SVD)) for data compression and retrieval enhancement. Vector space methods for information retrieval are presented in Chapter 11. ∎

Often it is useful to consider the matrix not just as an array of numbers, or as a set of vectors, but also as a linear operator. Denote the columns of A

$$a_{\cdot j} = \begin{pmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{mj} \end{pmatrix}, \qquad j = 1, 2, \ldots, n,$$

and write

$$A = \begin{pmatrix} a_{\cdot 1} & a_{\cdot 2} & \cdots & a_{\cdot n} \end{pmatrix}.$$

Then the linear transformation is defined

$$y = Ax = \begin{pmatrix} a_{\cdot 1} & a_{\cdot 2} & \dots & a_{\cdot n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \sum_{j=1}^{n} x_j a_{\cdot j}.$$

Example 1.2. The classification of handwritten digits is a model problem in *pattern recognition*. Here vectors are used to represent digits. The image of one digit is a 16×16 matrix of numbers, representing gray scale. It can also be represented as a vector in $\mathbb{R}^{256}$, by stacking the columns of the matrix. A set of n digits (handwritten 3's, say) can then be represented by a matrix $A \in \mathbb{R}^{256 \times n}$, and the columns of A span a subspace of $\mathbb{R}^{256}$. We can compute an approximate basis of this subspace using the SVD $A = U\Sigma V^T$. Three basis vectors of the "3-subspace" are illustrated in Figure 1.1.

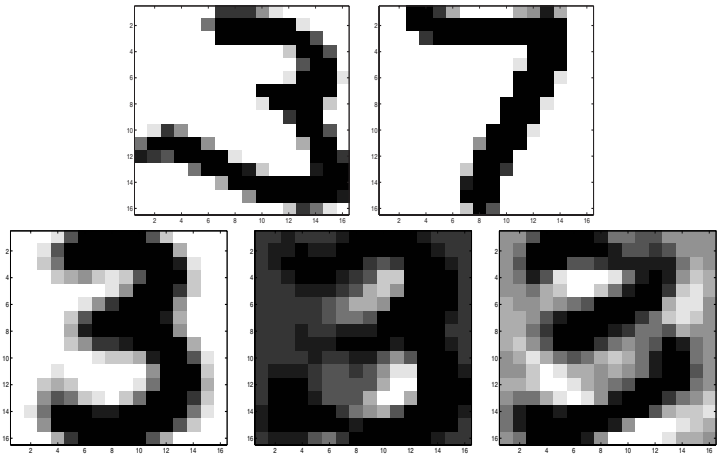


Figure 1.1. *Handwritten digits from the U.S. Postal Service database* [47]*, and basis vectors for* 3*'s* (*bottom*).

Let b be a vector representing an unknown digit. We now want to classify (automatically, by computer) the unknown digit as one of the digits 0–9. Given a set of approximate basis vectors for 3's, $u_1, u_2, \dots, u_k$, we can determine whether b is a 3 by checking if there is a linear combination of the basis vectors, $\sum_{j=1}^{k} x_j u_j$, such that

$$b - \sum_{j=1}^{k} x_j u_j$$

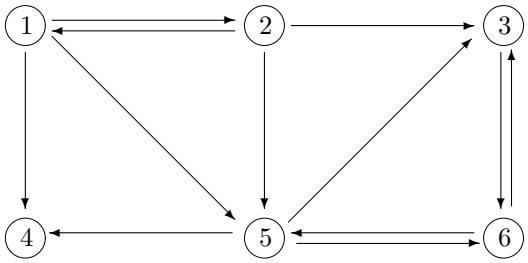
is small. Thus, here we compute the coordinates of b in the basis $\{u_j\}_{j=1}^k$.

In Chapter 10 we discuss methods for classification of handwritten digits. ∎

The very idea of data mining is to extract useful information from large, often unstructured, sets of data. Therefore it is necessary that the methods used are efficient and often specially designed for large problems. In some data mining applications huge matrices occur.

Example 1.3. The task of extracting information from all Web pages available on the Internet is done by *search engines*. The core of the Google search engine is a matrix computation, probably the largest that is performed routinely [71]. The Google matrix P is of the order billions, i.e., close to the total number of Web pages on the Internet. The matrix is constructed based on the link structure of the Web, and element P_{ij} is nonzero if there is a link from Web page j to i.

The following small link graph illustrates a set of Web pages with outlinks and inlinks:



A corresponding *link graph matrix* is constructed so that the columns and rows represent Web pages and the nonzero elements in column j denote outlinks from Web page j. Here the matrix becomes

$$P = \begin{pmatrix} 0 & \frac{1}{3} & 0 & 0 & 0 & 0 \\ \frac{1}{3} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & \frac{1}{2} \\ \frac{1}{3} & 0 & 0 & 0 & \frac{1}{3} & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 1 & 0 & \frac{1}{3} & 0 \end{pmatrix}.$$

For a search engine to be useful, it must use a measure of quality of the Web pages. The Google matrix is used to rank all the pages. The ranking is done by solving an *eigenvalue problem* for P; see Chapter 12. ∎

1.3 Purpose of the Book

The present book is meant to be not primarily a textbook in numerical linear algebra but rather an application-oriented introduction to some techniques in modern

linear algebra, with the emphasis on data mining and pattern recognition. It depends heavily on the availability of an easy-to-use programming environment that implements the algorithms that we will present. Thus, instead of describing in detail the algorithms, we will give enough mathematical theory and numerical background information so that a reader can understand and use the powerful software that is embedded in a package like MATLAB [68].

For a more comprehensive presentation of numerical and algorithmic aspects of the matrix decompositions used in this book, see any of the recent textbooks [29, 42, 50, 92, 93, 97]. The solution of linear systems and eigenvalue problems for large and sparse systems is discussed at length in [4, 5]. For those who want to study the detailed implementation of numerical linear algebra algorithms, software in Fortran, C, and C++ is available for free via the Internet [1].

It will be assumed that the reader has studied introductory courses in linear algebra and scientific computing (numerical analysis). Familiarity with the basics of a matrix-oriented programming language like MATLAB should help one to follow the presentation.

1.4 Programming Environments

In this book we use MATLAB [68] to demonstrate the concepts and the algorithms. Our codes are not to be considered as software; instead they are intended to demonstrate the basic principles, and we have emphasized simplicity rather than efficiency and robustness. *The codes should be used only for small experiments and never for production computations.*

Even if we are using MATLAB, we want to emphasize that any programming environment that implements modern matrix computations can be used, e.g., Mathematica® [112] or a statistics package.

1.5 Floating Point Computations

1.5.1 Flop Counts

The execution times of different algorithms can sometimes be compared by counting the number of *floating point operations*, i.e., arithmetic operations with floating point numbers. In this book we follow the standard procedure [42] and count each operation separately, and we use the term *flop* for one operation. Thus the statement `y=y+a*x`, where the variables are scalars, counts as two flops.

It is customary to count only the highest-order term(s). We emphasize that flop counts are often very crude measures of efficiency and computing time and can even be misleading under certain circumstances. On modern computers, which invariably have memory hierarchies, the data access patterns are very important. Thus there are situations in which the execution times of algorithms with the same flop counts can vary by an order of magnitude.

1.5.2 Floating Point Rounding Errors

Error analysis of the algorithms will not be a major part of the book, but we will cite a few results without proofs. We will assume that the computations are done under the *IEEE floating point standard* [2] and, accordingly, that the following model is valid.

A real number x, in general, cannot be represented exactly in a floating point system. Let $fl[x]$ be the floating point number representing x. Then

$$fl[x] = x(1 + \epsilon) \tag{1.1}$$

for some ϵ, satisfying $|\epsilon| \leq \mu$, where μ is the *unit round-off* of the floating point system. From (1.1) we see that the *relative error* in the floating point representation of any real number x satisfies

$$\left| \frac{fl[x] - x}{x} \right| \leq \mu.$$

In IEEE double precision arithmetic (which is the standard floating point format in MATLAB), the unit round-off satisfies $\mu \approx 10^{-16}$. In IEEE single precision we have $\mu \approx 10^{-7}$.

Let $fl[x \odot y]$ be the result of a floating point arithmetic operation, where $\odot$ denotes any of $+$, $-$, $*$, and $/$. Then, provided that $x \odot y \neq 0$,

$$\left| \frac{x \odot y - fl[x \odot y]}{x \odot y} \right| \leq \mu \tag{1.2}$$

or, equivalently,

$$fl[x \odot y] = (x \odot y)(1 + \epsilon) \tag{1.3}$$

for some ϵ, satisfying $|\epsilon| \leq \mu$, where μ is the unit round-off of the floating point system.

When we estimate the error in the *result* of a computation in floating point arithmetic as in (1.2) we can think of it as a *forward error*. Alternatively, we can rewrite (1.3) as

$$fl[x \odot y] = (x + e) \odot (y + f)$$

for some numbers e and f that satisfy

$$|e| \leq \mu|x|, \quad |f| \leq \mu|y|.$$

In other words, $fl[x \odot y]$ is the *exact* result of the operation on *slightly perturbed data*. This is an example of *backward error analysis*.

The smallest and largest positive real numbers that can be represented in IEEE double precision are 10^{-308} and 10^{308}, approximately (corresponding for negative numbers). If a computation gives as a result a floating point number of magnitude

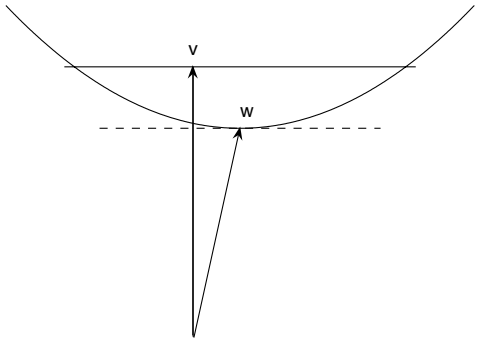


Figure 1.2. *Vectors in the GJK algorithm.*

smaller than 10^{-308}, then a floating point exception called *underflow* occurs. Similarly, the computation of a floating point number of magnitude larger than 10^{308} results in *overflow*.

Example 1.4 (floating point computations in computer graphics). The detection of a collision between two three-dimensional objects is a standard problem in the application of graphics to computer games, animation, and simulation [101]. Earlier fixed point arithmetic was used for computer graphics, but such computations now are routinely done in floating point arithmetic. An important subproblem in this area is the computation of the point on a convex body that is closest to the origin. This problem can be solved by the Gilbert–Johnson–Keerthi (GJK) algorithm, which is iterative. The algorithm uses the stopping criterion

$$S(v, w) = v^T v - v^T w \leq \epsilon^2$$

for the iterations, where the vectors are illustrated in Figure 1.2. As the solution is approached the vectors are very close. In [101, pp. 142–145] there is a description of the numerical difficulties that can occur when the computation of $S(v, w)$ is done in floating point arithmetic. Here we give a short explanation of the computation in the case when v and w are scalar, $s = v^2 - vw$, which exhibits exactly the same problems as in the case of vectors.

Assume that the data are inexact (they are the results of previous computations; in any case they suffer from representation errors (1.1)),

$$\bar{v} = v(1 + \epsilon_v), \quad \bar{w} = w(1 + \epsilon_w),$$

where ϵ_v and ϵ_w are relatively small, often of the order of magnitude of μ. From (1.2) we see that each arithmetic operation incurs a relative error (1.3), so that

$$\begin{aligned} fl[v^2 - vw] &= (v^2(1 + \epsilon_v)^2(1 + \epsilon_1) - vw(1 + \epsilon_v)(1 + \epsilon_w)(1 + \epsilon_2))(1 + \epsilon_3) \\ &= (v^2 - vw) + v^2(2\epsilon_v + \epsilon_1 + \epsilon_3) - vw(\epsilon_v + \epsilon_w + \epsilon_2 + \epsilon_3) + O(\mu^2), \end{aligned}$$

where we have assumed that $|\epsilon_i| \leq \mu$. The relative error in the computed quantity can be estimated by

$$\left| \frac{fl[v^2 - vw] - (v^2 - vw)}{(v^2 - vw)} \right| \leq \frac{v^2(2|\epsilon_v| + 2\mu) + |vw|(|\epsilon_v| + |\epsilon_w| + 2\mu) + O(\mu^2)}{|v^2 - vw|}.$$

We see that if v and w are large, and close, then the relative error may be large. For instance, with $v = 100$ and $w = 99.999$ we get

$$\left| \frac{fl[v^2 - vw] - (v^2 - vw)}{(v^2 - vw)} \right| \leq 10^5((2|\epsilon_v| + 2\mu) + (|\epsilon_v| + |\epsilon_w| + 2\mu) + O(\mu^2)).$$

If the computations are performed in IEEE single precision, which is common in computer graphics applications, then the relative error in $fl[v^2 - vw]$ may be so large that the termination criterion is never satisfied, and the iteration will never stop. In the GJK algorithm there are also other cases, besides that described above, when floating point rounding errors can cause the termination criterion to be unreliable, and special care must be taken; see [101]. ∎

The problem that occurs in the preceding example is called *cancellation*: when we subtract two almost equal numbers with errors, the result has fewer significant digits, and the relative error is larger. For more details on the IEEE standard and rounding errors in floating point computations, see, e.g., [34, Chapter 2]. Extensive rounding error analyses of linear algebra algorithms are given in [50].

1.6 Notation and Conventions

We will consider vectors and matrices with real components. Usually vectors will be denoted by lowercase italic Roman letters and matrices by uppercase italic Roman or Greek letters:

$$x \in \mathbb{R}^n, \quad A = (a_{ij}) \in \mathbb{R}^{m \times n}.$$

Tensors, i.e., arrays of real numbers with three or more indices, will be denoted by a calligraphic font. For example,

$$\mathcal{S} = (s_{ijk}) \in \mathbb{R}^{n_1 \times n_2 \times n_3}.$$

We will use $\mathbb{R}^m$ to denote the vector space of dimension m over the real field and $\mathbb{R}^{m \times n}$ for the space of $m \times n$ matrices.

The notation

$$e_i = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

where the 1 is in position i, is used for the "canonical" unit vectors. Often the dimension is apparent from the context.

The identity matrix is denoted I. Sometimes we emphasize the dimension and use I_k for the $k \times k$ identity matrix. The notation $\text{diag}(d_1, \ldots, d_n)$ denotes a diagonal matrix. For instance, $I = \text{diag}(1, 1, \ldots, 1)$.