Chapter 9

Clustering and Nonnegative Matrix Factorization

An important method for data compression and classification is to organize data points in *clusters*. A cluster is a subset of the set of data points that are close together in some distance measure. One can compute the mean value of each cluster separately and use the means as representatives of the clusters. Equivalently, the means can be used as basis vectors, and all the data points represented by their coordinates with respect to this basis.

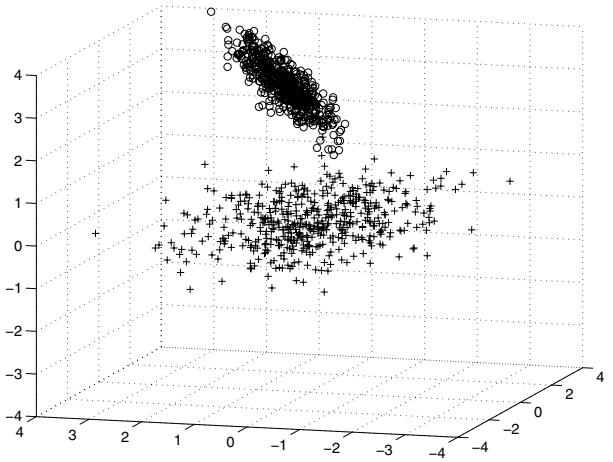

Figure 9.1. *Two clusters in* $\mathbb{R}^3$.

Example 9.1. In Figure 9.1 we illustrate a set of data points in $\mathbb{R}^3$, generated from two correlated normal distributions. Assuming that we know that we have two clusters, we can easily determine visually which points belong to which class. A clustering algorithm takes the complete set of points and classifies them using some distance measure. ∎

There are several methods for computing a clustering. One of the most important is the *k-means algorithm*. We describe it in Section 9.1.

In data mining applications, the matrix is often nonnegative. If we compute a low-rank approximation of the matrix using the SVD, then, due to the orthogonality of the singular vectors, we are very likely to obtain factors with negative elements. It may seem somewhat unnatural to approximate a nonnegative matrix by a low-rank approximation with negative elements. Instead one often wants to compute a low-rank approximation with nonnegative factors:

$$A \approx WH, \qquad W, H \geq 0. \tag{9.1}$$

In many applications, a nonnegative factorization facilitates the interpretation of the low-rank approximation in terms of the concepts of the application.

In Chapter 11 we will apply a clustering algorithm to a nonnegative matrix and use the cluster centers as basis vectors, i.e., as columns in the matrix W in (9.1). However, this does not guarantee that H also is nonnegative. Recently several algorithms for computing such *nonnegative matrix factorizations* have been proposed, and they have been used successfully in different applications. We describe such algorithms in Section 9.2.

9.1 The *k*-Means Algorithm

We assume that we have n data points $(a_j)_{j=1}^n \in \mathbb{R}^m$, which we organize as columns in a matrix $A \in \mathbb{R}^{m\times n}$. Let $\Pi = (\pi_i)_{i=1}^k$ denote a partitioning of the vectors $a_1, a_1, \ldots, a_n$ into k clusters:

$$\pi_j = \{\nu \,|\, a_\nu \text{ belongs to cluster } j\}.$$

Let the mean, or the *centroid*, of the cluster be

$$m_j = \frac{1}{n_j} \sum_{\nu\in\pi_j} a_\nu,$$

where n_j is the number of elements in π_j. We will describe a *k-means algorithm*, based on the Euclidean distance measure.

The tightness or *coherence* of cluster π_j can be measured as the sum

$$q_j = \sum_{\nu\in\pi_j} \|a_\nu - m_j\|_2^2.$$

The closer the vectors are to the centroid, the smaller the value of q_j. The quality of a clustering can be measured as the overall coherence:

$$Q(\Pi) = \sum_{j=1}^k q_j = \sum_{j=1}^k \sum_{\nu\in\pi_j} \|a_\nu - m_j\|_2^2.$$

In the k-means algorithm we seek a partitioning that has optimal coherence, in the sense that it is the solution of the minimization problem

$$\min_{\Pi} Q(\Pi).$$

The basic idea of the algorithm is straightforward: given a provisional partitioning, one computes the centroids. Then for each data point in a particular cluster, one checks whether there is another centroid that is closer than the present cluster centroid. If that is the case, then a redistribution is made.

The *k*-means algorithm

---

1. Start with an initial partitioning $\Pi^{(0)}$ and compute the corresponding centroid vectors, $(m_j^{(0)})_{j=1}^k$. Compute $Q(\Pi^{(0)})$. Put $t = 1$.

2. For each vector a_i, find the closest centroid. If the closest vector is $m_p^{(t-1)}$, assign a_i to $\pi_p^{(t)}$.

3. Compute the centroids $(m_j^{(t)})_{j=1}^k$ of the new partitioning $\Pi^{(t)}$.

4. if $|Q(\Pi^{(t-1)}) - Q(\Pi^{(t)})| < \text{tol}$, then stop; otherwise increment t by 1 and go to step 2.

The initial partitioning is often chosen randomly. The algorithm usually has rather fast convergence, but one cannot guarantee that the algorithm finds the global minimum.

Example 9.2. A standard example in clustering is taken from a breast cancer diagnosis study [66].[18] The matrix $A \in \mathbb{R}^{9\times 683}$ contains data from breast cytology tests. Out of the 683 tests, 444 represent a diagnosis of benign and 239 a diagnosis of malignant. We iterated with $k = 2$ in the k-means algorithm until the relative difference in the function $Q(\Pi)$ was less than 10^{-10}. With a random initial partitioning the iteration converged in six steps (see Figure 9.2), where we give the values of the objective function. Note, however, that the convergence is not monotone: the objective function was smaller after step 3 than after step 6. It turns out that in many cases the algorithm gives only a local minimum.

As the test data have been manually classified, it is known which patients had the benign and which the malignant cancer, and we can check the clustering given by the algorithm. The results are given in Table 9.1. Of the 239 patients with malignant cancer, the k-means algorithm classified 222 correctly but 17 incorrectly. ∎

In Chapter 11 and in the following example, we use clustering for information retrieval or text mining. The centroid vectors are used as basis vectors, and the documents are represented by their coordinates in terms of the basis vectors.

[18] See http://www.radwin.org/michael/projects/learning/about-breast-cancer-wisconsin.html.

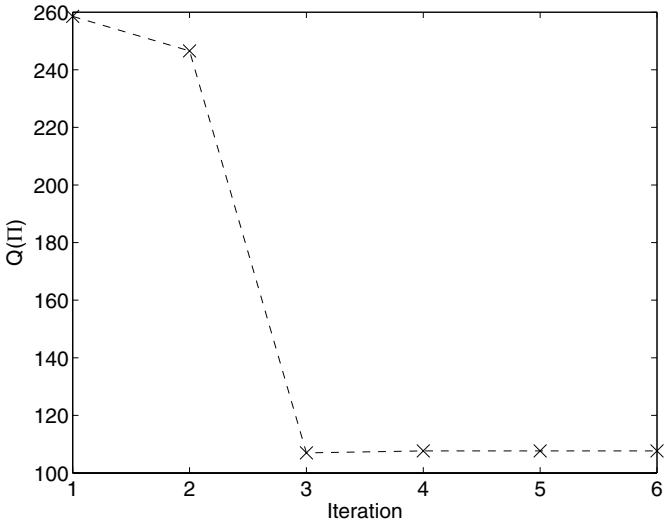

Figure 9.2. *The objective function in the k-means algorithm for the breast cancer data.*

Table 9.1. *Classification of cancer data with the k-means algorithm. B stands for benign and M for malignant cancer.*

	k-means	
	M	B
M	222	17
B	9	435

Example 9.3. Consider the term-document matrix in Example 1.1,

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

and recall that the first four documents deal with Google and the ranking of Web pages, while the fifth is about football. With this knowledge, we can take the average of the first four column vectors as the centroid of that cluster and the fifth

as the second centroid, i.e., we use the normalized basis vectors

$$C = \begin{pmatrix} 0.1443 & 0 \\ 0 & 0.5774 \\ 0 & 0.5774 \\ 0.2561 & 0 \\ 0.1443 & 0 \\ 0.1443 & 0 \\ 0.4005 & 0 \\ 0.2561 & 0 \\ 0.2561 & 0.5774 \\ 0.2561 & 0 \end{pmatrix}.$$

The coordinates of the columns of A in terms of this approximate basis are computed by solving

$$\min_D \|A - CH\|_F.$$

Given the thin QR decomposition $C = QR$, this least squares problem has the solution $H = R^{-1}Q^TA$ with

$$H = \begin{pmatrix} 1.7283 & 1.4168 & 2.8907 & 1.5440 & 0.0000 \\ -0.2556 & -0.2095 & 0.1499 & 0.3490 & 1.7321 \end{pmatrix}.$$

We see that the first two columns have negative coordinates in terms of the second basis vector. This is rather difficult to interpret in the term-document setting. For instance, it means that the first column a_1 is approximated by

$$a_1 \approx Ch_1 = \begin{pmatrix} 0.2495 \\ -0.1476 \\ -0.1476 \\ 0.4427 \\ 0.2495 \\ 0.2495 \\ 0.6921 \\ 0.4427 \\ 0.2951 \\ 0.4427 \end{pmatrix}.$$

It is unclear what it may signify that this "approximate document" has negative entries for the words *England* and *FIFA*.

Finally we note, for later reference, that the relative approximation error is rather high:

$$\frac{\|A - CH\|_F}{\|A\|_F} \approx 0.596. \quad \blacksquare \tag{9.2}$$

In the next section we will approximate the matrix in the preceding example, making sure that both the basis vectors and the coordinates are nonnegative.

9.2 Nonnegative Matrix Factorization

Given a data matrix $A \in \mathbb{R}^{m\times n}$, we want to compute a rank-k approximation that is constrained to have nonnegative factors. Thus, assuming that $W \in \mathbb{R}^{m\times k}$ and $H \in \mathbb{R}^{k\times n}$, we want to solve

$$\min_{W\geq 0,\, H\geq 0} \|A - WH\|_F. \tag{9.3}$$

Considered as an optimization problem for W and H at the same time, this problem is nonlinear. However, if one of the unknown matrices were known, W, say, then the problem of computing H would be a standard, nonnegatively constrained, least squares problem with a matrix right-hand side. Therefore the most common way of solving (9.3) is to use an *alternating least squares (ALS)* procedure [73]:

Alternating nonnegative least squares algorithm

---

1. Guess an initial value $W^{(1)}$.
2. for $k = 1, 2, \ldots$ until convergence
   (a) Solve $\min_{H\geq 0} \|A - W^{(k)}H\|_F$, giving $H^{(k)}$.
   (b) Solve $\min_{W\geq 0} \|A - WH^{(k)}\|_F$, giving $W^{(k+1)}$.

However, the factorization WH is not unique: we can introduce any diagonal matrix D with positive diagonal elements and its inverse between the factors,

$$WH = (WD)(D^{-1}H).$$

To avoid growth of one factor and decay of the other, we need to normalize one of them in every iteration. A common normalization is to scale the columns of W so that the largest element in each column becomes equal to 1.

Let a_j and h_j be the columns of A and H. Writing out the columns one by one, we see that the *matrix least squares problem* $\min_{H\geq 0} \|A - W^{(k)}H\|_F$ is equivalent to n independent *vector least squares problems*:

$$\min_{h_j\geq 0} \|a_j - W^{(k)}h_j\|_2, \qquad j = 1, 2, \ldots, n.$$

These can be solved by an active-set algorithm[19] from [61, Chapter 23]. By transposing the matrices, the least squares problem for determining W can be reformulated as m independent vector least squares problems. Thus the core of the ALS algorithm can be written in pseudo-MATLAB:

[19] The algorithm is implemented in MATLAB as a function `lsqnonneg`.

```
while (not converged)
    [W]=normalize(W);
    for i=1:n
        H(:,i)=lsqnonneg(W,A(:,i));
    end
    for i=1:m
        w=lsqnonneg(H',A(i,:)');
        W(i,:)=w';
    end
end
```

There are many variants of algorithms for nonnegative matrix factorization. The above algorithm has the drawback that the active set algorithm for nonnegative least squares is rather time-consuming. As a cheaper alternative, given the thin QR decomposition $W = QR$, one can take the unconstrained least squares solution,

$$H = R^{-1}Q^T A,$$

and then set all negative elements in H equal to zero, and similarly in the other step of the algorithm. Improvements that accentuate sparsity are described in [13].

A multiplicative algorithm was given in [63]:

```
while (not converged)
    W=W.*(W>=0);
    H=H.*(W'*V)./((W'*W)*H+epsilon);
    H=H.*(H>=0);
    W=W.*(V*H')./(W*(H*H')+epsilon);
    [W,H]=normalize(W,H);
end
```

(The variable `epsilon` should be given a small value and is used to avoid division by zero.) The matrix operations with the operators `.*` and `./` are equivalent to the componentwise statements

$$H_{ij} := H_{ij}\frac{(W^T A)_{ij}}{(W^T WH)_{ij}+\epsilon}, \qquad W_{ij} := W_{ij}\frac{(AH^T)_{ij}}{(WHH^T)_{ij}+\epsilon}.$$

The algorithm can be considered as a gradient descent method.

Since there are so many important applications of nonnegative matrix factorizations, algorithm development is an active research area. For instance, the problem of finding a termination criterion for the iterations does not seem to have found a good solution. A survey of different algorithms is given in [13].

A nonnegative factorization $A \approx WH$ can be used for clustering: the data vector a_j is assigned to cluster i if h_{ij} is the largest element in column j of H [20, 37].

Nonnegative matrix factorization is used in a large variety of applications: document clustering and email surveillance [85, 8], music transcription [90], bioinformatics [20, 37], and spectral analysis [78], to mention a few.

9.2.1 Initialization

A problem with several of the algorithms for nonnegative matrix factorization is that convergence to a global minimum is not guaranteed. It often happens that convergence is slow and that a suboptimal approximation is reached. An efficient procedure for computing a good initial approximation can be based on the SVD of A [18]. We know that the first k singular triplets $(\sigma_i, u_i, v_i)_{i=1}^k$ give the best rank-k approximation of A in the Frobenius norm. It is easy to see that if A is a nonnegative matrix, then u_1 and v_1 are nonnegative (cf. Section 6.4). Therefore, if $A = U\Sigma V^T$ is the SVD of A, we can take the first singular vector u_1 as the first column in $W^{(1)}$ (and v_1^T as the first row in an initial approximation $H^{(1)}$, if that is needed in the algorithm; we will treat only the approximation of $W^{(1)}$ in the following).

The next best vector, u_2, is very likely to have negative components, due to orthogonality. But if we compute the matrix $C^{(2)} = u_2 v_2^T$ and replace all negative elements with zero, giving the nonnegative matrix $C_+^{(2)}$, then we know that the first singular vector of this matrix is nonnegative. Furthermore, we can hope that it is a reasonably good approximation of u_2, so we can take it as the second column of $W^{(1)}$.

The procedure can be implemented by the following, somewhat simplified, MATLAB script:

```
[U,S,V]=svds(A,k);   % Compute only the k largest singular
                     %  values and the corresponding vectors
W(:,1)=U(:,1);
for j=2:k
    C=U(:,j)*V(:,j)';
    C=C.*(C>=0);
    [u,s,v]=svds(C,1);
    W(:,j)=u;
end
```

The MATLAB `[U,S,V]=svds(A,k)` computes only the k-largest singular values and the corresponding singular vectors using a Lanczos method; see Section 15.8.3. The standard SVD function `svd(A)` computes the full decomposition and is usually considerably slower, especially when the matrix is large and sparse.

Example 9.4. We computed a rank-2 nonnegative factorization of the matrix A in Example 9.3, using a random initialization and the SVD-based initialization. With the random initialization, convergence was slower (see Figure 9.3), and after 10 iterations it had not converged. The relative approximation error of the algorithm

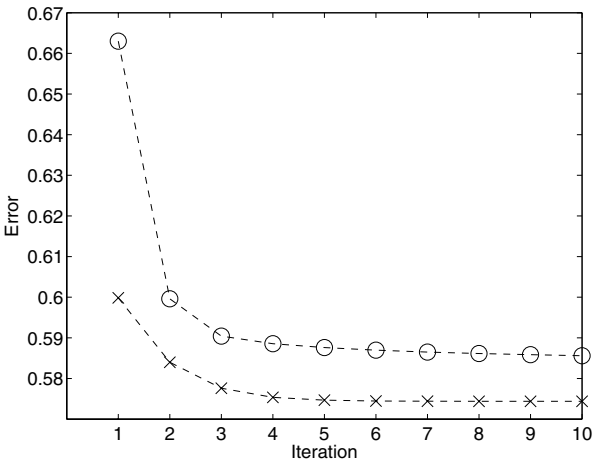

Figure 9.3. *Relative approximation error in the nonnegative matrix factorization as a function of the iteration number. The upper curve is with random initialization and the lower with the SVD-based initialization.*

with SVD initialization was 0.574 (cf. the error 0.596 with the k-means algorithm (9.2)). In some runs, the algorithm with the random initialization converged to a local, suboptimal minimum. The factorization with SVD initialization was

$$WH = \begin{pmatrix} 0.3450 & 0 \\ 0.1986 & 0 \\ 0.1986 & 0 \\ 0.6039 & 0.1838 \\ 0.2928 & 0 \\ 0 & 0.5854 \\ 1.0000 & 0.0141 \\ 0.0653 & 1.0000 \\ 0.8919 & 0.0604 \\ 0.0653 & 1.0000 \end{pmatrix} \begin{pmatrix} 0.7740 & 0 & 0.9687 & 0.9120 & 0.5251 \\ 0 & 1.0863 & 0.8214 & 0 & 0 \end{pmatrix}.$$

It is now possible to interpret the decomposition. The first four documents are well represented by the basis vectors, which have large components for Google-related keywords. In contrast, the fifth document is represented by the first basis vector only, but its coordinates are smaller than those of the first four Google-oriented documents. In this way, the rank-2 approximation accentuates the Google-related contents, while the "football-document" is de-emphasized. In Chapter 11 we will see that other low-rank approximations, e.g., those based on SVD, have a similar effect.

On the other hand, if we compute a rank-3 approximation, then we get

$$WH = \begin{pmatrix} 0.2516 & 0 & 0.1633 \\ 0 & 0 & 0.7942 \\ 0 & 0 & 0.7942 \\ 0.6924 & 0.1298 & 0 \\ 0.3786 & 0 & 0 \\ 0 & 0.5806 & 0 \\ 1.0000 & 0 & 0.0444 \\ 0.0589 & 1.0000 & 0.0007 \\ 0.4237 & 0.1809 & 1.0000 \\ 0.0589 & 1.0000 & 0.0007 \end{pmatrix} \begin{pmatrix} 1.1023 & 0 & 1.0244 & 0.8045 & 0 \\ 0 & 1.0815 & 0.8314 & 0 & 0 \\ 0 & 0 & 0.1600 & 0.3422 & 1.1271 \end{pmatrix}.$$

We see that now the third vector in W is essentially a "football" basis vector, while the other two represent the Google-related documents.