9 Dimension reduction techniques

Large datasets, as well as data consisting of a large number of features, present computational problems in the training of predictive models. In this chapter we discuss several useful techniques for reducing the dimension of a given dataset, that is reducing the number of data points or number of features, often employed in order to make predictive learning methods scale to larger datasets. More specifically, we discuss widely used methods for reducing the *data dimension*, that is the number of data points, of a dataset including random subsampling and K-means clustering. We then detail a common way of reducing the *feature dimension*, or number features, of a dataset as explained in Fig. 9.1. A classical approach for feature dimension reduction, principal component analysis (PCA), while often used for general data analysis is a relatively poor tool for reducing the feature dimension of predictive modeling data. However, PCA presents a fundamental mathematical archetype, the *matrix factorization*, that provides a very useful way of organizing our thinking about a wide array of important learning models (including linear regression, K-means, recommender systems – introduced after detailing PCA in this chapter), all of which may be thought of as variations of the simple theme of matrix factorization.

9.1 Techniques for data dimension reduction

In this section we detail two common ways of reducing the data dimension of a dataset: random subsampling and K-means clustering.

9.1.1 Random subsampling

Random subsampling is a simple and intuitive way of reducing the data dimension of a dataset, and is often the first approach employed when performing regression/classification on datasets too large for available computational resources. Given a set of P points we keep a random subsample of $S < P$ of the entire set. Clearly the smaller we choose S the larger the chance we may loose an important structural characteristic of the underlying dataset (for example the geometry of the separating boundary between two classes of data). While there is no formula or hard rule for how large S should be, a simple guideline used in practice is to choose S as large as possible given the computational resources available so as to minimize this risk.

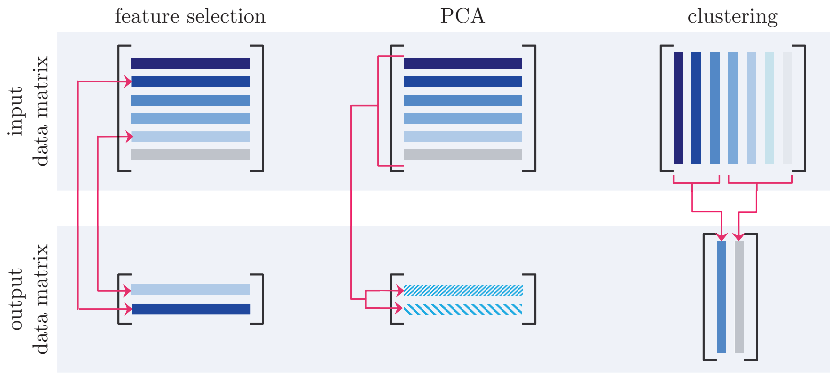

Fig. 9.1 Comparison of feature selection, PCA, and clustering as dimension reduction schemes on an arbitrary data matrix, like those we have discussed in previous chapters for predictive modeling, whose rows contain features and columns individual data points. The former two methods reduce the dimension of the feature space, or in other words the number of rows in a data matrix. However, the two methods work differently: while feature selection literally selects rows from the original matrix to keep, PCA uses the geometry of the feature space to produce a new data matrix based on a lower feature dimensional version of the data. K-means, on the other hand, reduces the dimension of the data/number of data points, or equivalently the number of columns in the input data matrix. It does so by finding a small number of new averaged representatives or "centroids" of the input data, forming a new data matrix whose fewer columns (which are not present in the original data matrix) are precisely these centroids.

9.1.2 K-means clustering

With K-means clustering we reduce the data dimension by finding suitable representatives or *centroids* for clusters, or groups, of data points. All members of each cluster are then represented by their cluster's respective centroid. Hence the problem of clustering is that of partitioning data into clusters of points with similar characteristics, and with K-means specifically this characteristic is geometric closeness in the feature space.[1] Figure 9.2 illustrates K-means clustering performed on a 2-D toy dataset with $P = 10$ data points, where in the right panel data points are clustered into $K = 3$ clusters.

With K-means we look to partition P data points, each of dimension N, into K clusters and find a representative centroid denoted for each cluster. For the moment we will assume that we know the location of these K cluster centroids, as illustrated figuratively in the toy example in the right panel of Fig. 9.2, in order to derive formally the desired relationship between the data and centroids. Once this is expressed clearly we will use it in order to form a learning problem for the accurate recovery of cluster centroids, dropping the unrealistic notion that we have pre-conceived knowledge of their location.

Denoting by $\mathbf{c}_k$ the centroid of the kth cluster and $\mathcal{S}_k$ the set of indices of the subset of those P data points, denoted $\mathbf{x}_1 \ldots \mathbf{x}_P$, belonging to this cluster, the desire that points in the kth cluster should lie close to its centroid may be written mathematically as

[1] Although it is possible to adopt a variety of different ways to define similarity between data points in the feature space (e.g., spectral clustering and subspace clustering), proximity in the Euclidean sense is the most popular measure for clustering data.

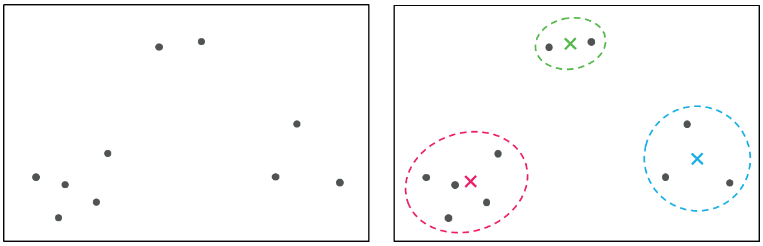

Fig. 9.2 (left) A 2-dimensional toy dataset with $P = 10$ data points. (right) Original data clustered into $K = 3$ clusters where each cluster centroid is marked by a $\times$ symbol. Points that are geometrically close to one another belong to the same cluster.

$$\mathbf{c}_k \approx \mathbf{x}_p \quad \text{for all } p \in \mathcal{S}_k, \tag{9.1}$$

for all $k = 1 \ldots K$. These desired relations can be written more conveniently by first stacking the centroids column-wise into the *centroid matrix* $\mathbf{C} = \begin{bmatrix} \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_K \end{bmatrix}$. Then denoting by $\mathbf{e}_k$ the kth standard basis vector (that is a $K \times 1$ vector with a 1 in the kth slot and zeros elsewhere), we may write $\mathbf{C}\mathbf{e}_k = \mathbf{c}_k$, and hence the relations in Equation (9.1) may be written equivalently for each k as

$$\mathbf{C}\mathbf{e}_k \approx \mathbf{x}_p \quad \text{for all } p \in \mathcal{S}_k. \tag{9.2}$$

Next, to write these equations even more conveniently we stack the data column-wise into the *data matrix* $\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_P \end{bmatrix}$ and form a $K \times P$ *assignment matrix* $\mathbf{W}$. The pth column of this matrix, denoted as $\mathbf{w}_p$, is the standard basis vector associated with the cluster to which the pth point belongs: i.e., $\mathbf{w}_p = \mathbf{e}_k$ if $p \in \mathcal{S}_k$. With this $\mathbf{w}_p$ notation we may write each equation in (9.2) as $\mathbf{C}\mathbf{w}_p \approx \mathbf{x}_p$ for all $p \in \mathcal{S}_k$, or using matrix notation all K such relations simultaneously as

$$\mathbf{C}\mathbf{W} \approx \mathbf{X}. \tag{9.3}$$

Figure 9.3 illustrates the compactly written desired K-means relationship in (9.3) for the dataset shown in Fig. 9.2. Note that the location of the only nonzero entry in each column of the assignment matrix $\mathbf{W}$ determines the cluster membership of its corresponding data point in $\mathbf{X}$.

We now drop the assumption that we know the locations of cluster centroids and have knowledge of which points are assigned to them, i.e., the exact description of the centroid matrix $\mathbf{C}$ and assignment matrix $\mathbf{W}$. We want to *learn* the right values for these two matrices. Specifically, we know that the ideal $\mathbf{C}$ and $\mathbf{W}$ satisfy the compact relations described in Equation (9.3), i.e., that $\mathbf{C}\mathbf{W} \approx \mathbf{X}$ or in other words that $\|\mathbf{C}\mathbf{W} - \mathbf{X}\|_F^2$ is small, while $\mathbf{W}$ consists of properly chosen standard basis vectors relating the data points to their respective centroids. Thus we phrase a K-means optimization problem whose solution precisely satisfies these requirements as

$$\begin{aligned} \underset{\mathbf{C},\mathbf{W}}{\text{minimize}} \quad & \|\mathbf{C}\mathbf{W} - \mathbf{X}\|_F^2 \\ \text{subject to} \quad & \mathbf{w}_p \in \{\mathbf{e}_k\}_{k=1}^K \quad p = 1, \ldots, P. \end{aligned} \tag{9.4}$$

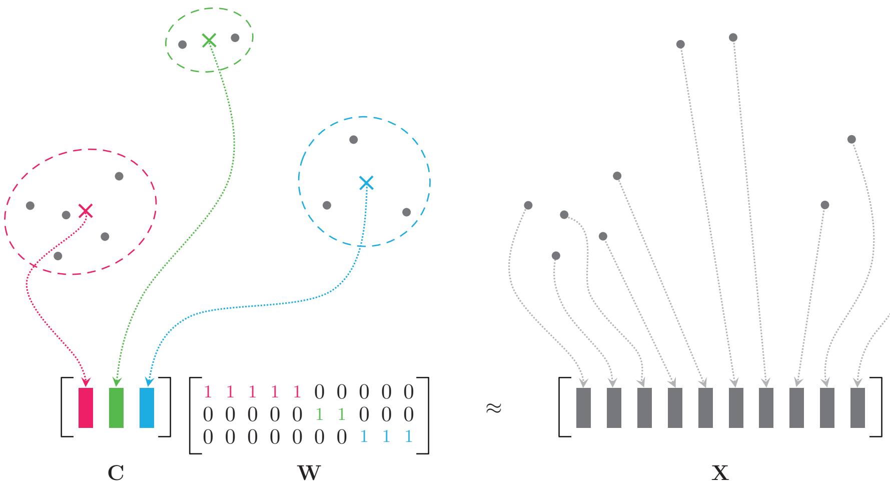

Fig. 9.3 K-means clustering relations described in a compact matrix form. Cluster centroids in $\mathbf{C}$ lie close to their corresponding cluster points in $\mathbf{X}$. The pth column of the assignment matrix $\mathbf{W}$ contains the standard basis vector corresponding to the data point's cluster centroid.

Note that the objective here is non-convex, and because we cannot minimize over both $\mathbf{C}$ and $\mathbf{W}$ simultaneously, it is solved via *alternating minimization*, that is by alternately minimizing the objective function in (9.4) over one of the variables ($\mathbf{C}$ or $\mathbf{W}$), while keeping the other variable fixed. We derive the steps corresponding to this procedure in the following section, and for convenience summarize the resulting simple procedure (often called the *K-means algorithm*) in Algorithm 9.1.

Algorithm 9.1 The K-means algorithm

```
Input(s): Data matrix $\mathbf{X}$, centroid matrix $\mathbf{C}$ initialized (e.g., randomly),
          and assignment matrix $\mathbf{W}$ initialized at zero
Output(s): Optimal centroid matrix $\mathbf{C}^\star$ and assignment matrix $\mathbf{W}^\star$
Repeat until convergence: (e.g., until $\mathbf{C}$ does not change)
   (1) Update $\mathbf{W}$ (assign each data point to its closest centroid)
    for $p = 1 \ldots P$
        Set $\mathbf{w}_p = \mathbf{e}_{k^\star}$ where $k^\star = \underset{k=1 \ldots K}{\operatorname{argmin}} \left\| \mathbf{c}_k - \mathbf{x}_p \right\|_2^2$
   (2) Update $\mathbf{C}$ (assign each centroid the average of its current points)
    for $k = 1 \ldots K$
        Denote $\mathcal{S}_k$ the index set of points $\mathbf{x}_p$ currently assigned to the $k$th cluster
        Set $\mathbf{c}_k = \frac{1}{|\mathcal{S}_k|} \sum_{p \in \mathcal{S}_k} \mathbf{x}_p$
```

Before showing alternating minimization derivation, note that because the objective in (9.4) is non-convex it is possible for the procedure to find non-global minima of the objective function. As with all non-convex problems, this depends on the initializations of our optimization variables (or in this instance just the $\mathbf{C}$ matrix initialization since the procedure begins by updating it independently of $\mathbf{W}$). The result of the algorithm

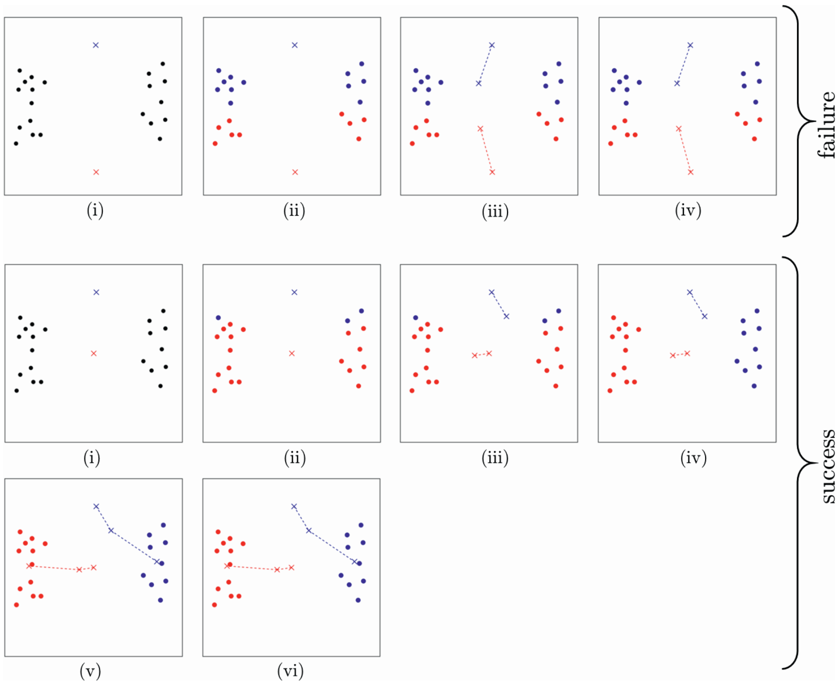

Fig. 9.4 Success or failure of K-means depends on the centroids' initialization. (top) (i) two centroids are initialized, (ii) cluster assignment is updated, (iii) centroid locations are updated, (iv) no change in the cluster assignment of the data points leads to stopping of the algorithm. (below) (i) two centroids are initialized with the red one being initialized differently, (ii) cluster assignment is updated, (iii) centroid locations are updated, (iv) cluster assignment is updated, (v) centroid locations are updated, (vi) no change in the cluster assignment of the data points leads to stopping of the algorithm.

reaching poor minima can have significant impact on the quality of the clusters learned. For example, in Fig. 9.4 we use a 2-D toy dataset with $K = 2$ clusters. With the initial centroid positions shown in the top panel, the K-means algorithm gets stuck in a local minimum and consequently fails to cluster the data properly. A different initialization for one of the centroids, however, leads to a successful clustering of the data, as shown in the lower panel of Fig. 9.4. To overcome the issue of non-convexity of K-means in practice we usually run the algorithm multiple times with different initializations, seeking out the lowest possible minimum of the objective, and the solution resulting in the smallest value of the objective function is selected as the final solution.

9.1.3 Optimization of the K-means problem

Over $\mathbf{W}$ the problem in Equation (9.5) reduces to

$$\begin{aligned} &\underset{\mathbf{W}}{\text{minimize}} \quad \|\mathbf{C}\mathbf{W} - \mathbf{X}\|_F^2 \\ &\text{subject to} \quad \mathbf{w}_p \in \{\mathbf{e}_k\}_{k=1}^K \quad p = 1, \ldots, P. \end{aligned} \tag{9.5}$$

Noting that the objective in (9.5) can be equivalently written as $\sum_{p=1}^{P} \left\| \mathbf{C}\mathbf{w}_p - \mathbf{x}_p \right\|_2^2$ (again $\mathbf{C}$ is fixed) and that each $\mathbf{w}_p$ appears in only one summand, we can recover each column $\mathbf{w}_p$ independently by solving

$$\begin{aligned} &\underset{\mathbf{w}_p}{\text{minimize}} \;\; \left\| \mathbf{C}\mathbf{w}_p - \mathbf{x}_p \right\|_2^2 \\ &\text{subject to } \mathbf{w}_p \in \{\mathbf{e}_k\}_{k=1}^{K}, \end{aligned} \tag{9.6}$$

for each $p = 1, \ldots, P$. Note that this is precisely the problem of assigning a data point $\mathbf{x}_p$ to its closest centroid, i.e., finding k such that $\left\| \mathbf{c}_k - \mathbf{x}_p \right\|_2^2$ is smallest! We can see that this is precisely the problem above by unravelling our compact notation: given the constraint on $\mathbf{w}_p$, we have $\mathbf{C}\mathbf{w}_p = \mathbf{c}_k$ whenever $\mathbf{w}_p = \mathbf{e}_k$ and so the objective may be written as $\left\| \mathbf{C}\mathbf{w}_p - \mathbf{x}_p \right\|_2^2 = \left\| \mathbf{c}_k - \mathbf{x}_p \right\|_2^2$. Hence the problem of finding $\mathbf{w}_p$, or finding the closest centroid to $\mathbf{x}_p$, may be written as

$$\underset{k=1...K}{\text{minimize}} \;\; \left\| \mathbf{c}_k - \mathbf{x}_p \right\|_2^2, \tag{9.7}$$

which can be solved by simply computing the objective for each k and finding the smallest value. Then for whichever $k^\star$ minimizes the above we may set $\mathbf{w}_p = \mathbf{e}_{k^\star}$.

Now minimizing (9.4) over $\mathbf{C}$, we have no constraints (they being applied only to $\mathbf{W}$) and have the problem

$$\underset{\mathbf{C}}{\text{minimize}} \;\; \|\mathbf{C}\mathbf{W} - \mathbf{X}\|_F^2. \tag{9.8}$$

Here we may use the first order condition: setting the derivative of $g(\mathbf{C}) = \|\mathbf{C}\mathbf{W} - \mathbf{X}\|_F^2$ to zero gives the linear system

$$\mathbf{C}\mathbf{W}\mathbf{W}^T = \mathbf{X}\mathbf{W}^T, \tag{9.9}$$

for the optimal $\mathbf{C}$ denoted as $\mathbf{C}^\star$. It is easy to show (see exercises) that $\mathbf{W}\mathbf{W}^T$ is a $K \times K$ diagonal matrix whose kth diagonal entry is equal to the number of data points assigned to the kth cluster, and that the kth column of $\mathbf{X}\mathbf{W}^T$ is the sum of all data points in the kth cluster. Hence each column of $\mathbf{C}^\star$ can therefore be calculated independently as

$$\mathbf{c}_k^\star = \frac{1}{|\mathcal{S}_k|} \sum_{p \in \mathcal{S}_k} \mathbf{x}_p \quad \forall k. \tag{9.10}$$

In other words, $\mathbf{c}_k^\star$ is the average of all data points in the kth cluster.

Finally, we repeat these alternating updates until neither matrix changes too much from iteration to iteration. We present the resulting simple column-wise updates for $\mathbf{C}$ and $\mathbf{W}$ in Algorithm 9.1.

9.2 Principal component analysis

As shown abstractly in Fig. 9.1, feature selection is a reasonable and general approach to lowering the dimension of the feature space when working on predictive modeling

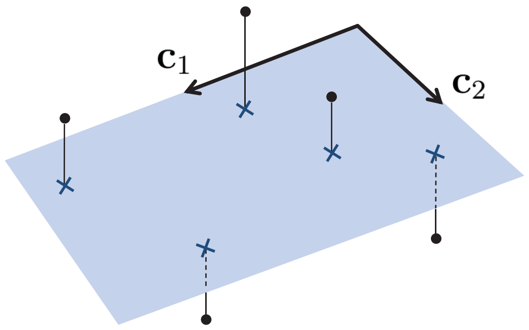

Fig. 9.5 The general PCA dimension reduction scheme visualized in 3D. We begin with three-dimensional data points (black circles) and locate a fitting set of vectors $\mathbf{c}_1$ and $\mathbf{c}_2$ that span a proper lower dimensional subspace for the data. We then project the data onto the subspace (in blue X_s).

problems. For less crucial tasks such as general data exploration where only input values are known (i.e., there is no associated output/labels), other techniques are used to reduce feature space dimension (if it is cumbersomely large). Principal component analysis (PCA), discussed in this section, is one such common technique. PCA works by simply projecting the data onto a suitable lower dimensional feature subspace, that is one which hopefully preserves the essential geometry of the original data. This subspace is found by determining one of its *spanning sets* (e.g., a basis) of vectors which spans it. The basic setup is illustrated in Fig. 9.5.

More formally, suppose that we have P data points $\mathbf{x}_1 \ldots \mathbf{x}_P$, each of dimension N. The goal with PCA is, for some user chosen dimension $K < N$, to find a set of K vectors $\mathbf{c}_1 \ldots \mathbf{c}_K$ that represent the data fairly well. Put formally, we want for each $p = 1 \ldots P$

$$\sum_{k=1}^{K} \mathbf{c}_k w_{k,p} \approx \mathbf{x}_p. \tag{9.11}$$

Note how this is analogous to the motivation for K-means discussed in Section 9.1.2, only here we wish to determine a small set of basis vectors which together explain the dataset. Stacking the desired spanning vectors column-wise into the $N \times K$ matrix $\mathbf{C}$ as $\mathbf{C} = [\mathbf{c}_1|\mathbf{c}_2|\cdots|\mathbf{c}_K]$ and denoting $\mathbf{w}_p = \begin{bmatrix} w_{1,p} & w_{2,p} & \cdots & w_{K,p} \end{bmatrix}^T$ this can be written equivalently for each p as

$$\mathbf{C}\mathbf{w}_p \approx \mathbf{x}_p. \tag{9.12}$$

Note: once $\mathbf{C}$ and $\mathbf{w}_p$ are learned the new K-dimensional feature representation of $\mathbf{x}_p$ is then the vector $\mathbf{w}_p$ (i.e., the weights over which $\mathbf{x}_p$ is represented over the spanning set). By denoting $\mathbf{W} = [\mathbf{w}_1|\mathbf{w}_2|\cdots|\mathbf{w}_P]$ the $K \times P$ matrix of weights to learn, and $\mathbf{X} = [\mathbf{x}_1|\mathbf{x}_2|\cdots|\mathbf{x}_P]$ the $N \times P$ data matrix, all P of these (and equivalently Equation (9.11)) can be written compactly as

$$\mathbf{C}\mathbf{W} \approx \mathbf{X}. \tag{9.13}$$

Table 9.1 Common matrix factorization problems (i.e., variants of PCA) subject to possible constraints on **C** and **W** including: linear regression (discussed in Chapter 3), K-means (discussed in Subsection 9.1.2), Recommender Systems (discussed in Section 9.3), nonnegative matrix factorization (often used for dimension reduction with images and text where the data is naturally nonnegative, see e.g., [45]), and sparse coding (commonly used as a model for low-level image processing in the mammalian visual cortex, see e.g., [60]).

Problem	Constraints
PCA/SVD	None
Linear regression	$\mathbf{C}$ fixed
K-means	$\mathbf{w}_i$ a standard basis vector for all i
Recommender systems	Entries in index set Ω are known i.e., $(\mathbf{CW})_\Omega = (\mathbf{X})_\Omega$
Nonnegative matrix factorization	Both $\mathbf{C}$ and $\mathbf{W}$ nonnegative
Sparse coding	k permissible nonzero entries in each $\mathbf{w}_i$ and columns of $\mathbf{C}$ have unit length

The goal of PCA, compactly stated in Equation (9.13), naturally leads to determining **C** and **W** by minimizing $\|\mathbf{CW}-\mathbf{X}\|_F^2$ i.e., by solving

$$\underset{\mathbf{C},\mathbf{W}}{\text{minimize}} \ \|\mathbf{CW}-\mathbf{X}\|_F^2 . \tag{9.14}$$

Note that this is a simpler version of the K-means problem in Equation (9.4). Note also the similarities between the PCA matrix factorization problem in Equation (9.14), and the Least Squares cost function for linear regression, K-means, recommender systems, and more: each of these problems may be thought of as variations of the basic *matrix factorization* problem in Equation (9.14) (see Table 9.1). Before discussing optimization procedures for the PCA problem we look at several examples.

Example 9.1 PCA on simulated data

In Fig. 9.6 we show the result of applying PCA on two simple 2-dimensional datasets using the solution to (9.14) shown in Equation (9.17). In the top panel dimension reduction via PCA retains much of the structure of the original data. Conversely, the more structured square dataset loses much of its original characteristic after projection onto the PCA subspace.

Example 9.2 PCA and classification data

While PCA can technically be used for preprocessing data in a predictive modeling scenario, it can cause severe problems in the case of classification. In Fig. 9.7 we illustrate

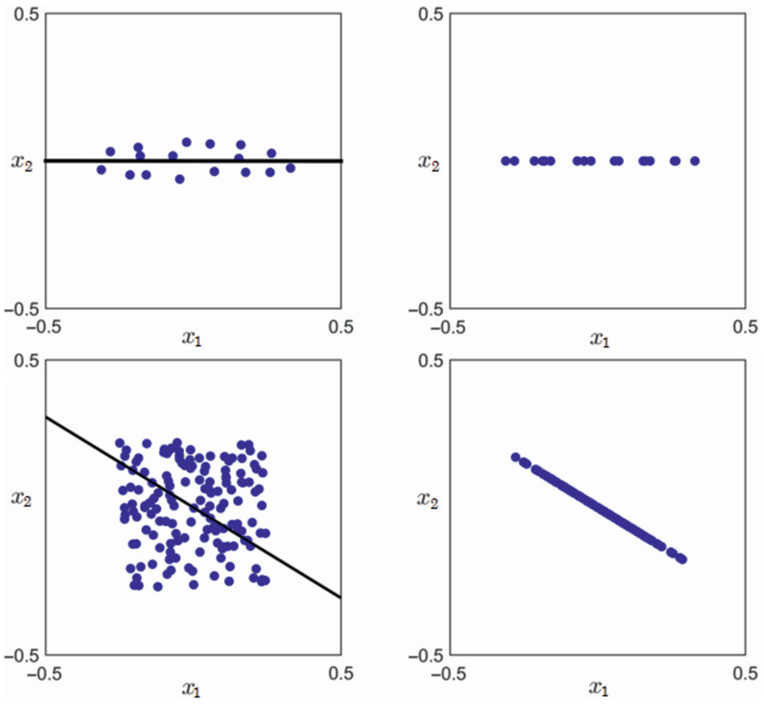

Fig. 9.6 (top panels) A simple 2-D data set (left, in blue) where dimension reduction via PCA retains much of the structure of the original data. The ideal subspace found via solving (9.14) is shown in black in the left panel, and the data projected onto this subspace is shown in blue on the right. (bottom panels) Conversely, the more structured square data-set loses much of its original structure after projection onto the PCA subspace.

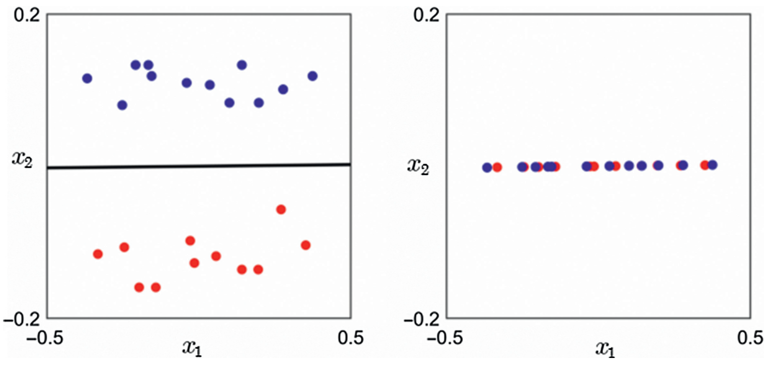

Fig. 9.7 (left) A toy classification dataset consisting of two linearly separable classes. The ideal subspace produced via PCA is shown in black. (right) Projecting the data onto this subspace (in other words reducing the feature space dimension via PCA) destroys completely the original separability of the data.

feature space dimension reduction via PCA on a simulated two-class dataset where the two classes are linearly separable. Because the ideal one-dimensional subspace in this instance runs parallel to the longer length of each class, projecting the complete dataset onto it completely destroys the separability.

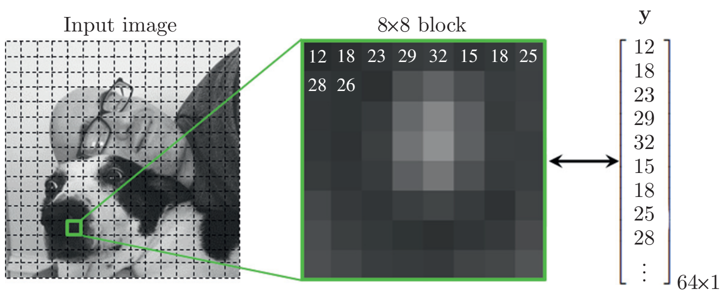

Fig. 9.8 In a prototypical image compression scheme the input image is cut into 8×8 blocks. Each block is then vectorized to make a 64×1 column vector $\mathbf{y}$ which will be input to the compression algorithm.

Example 9.3 Role of efficient bases in digital image compression

Digital image compression aims at reducing the size of digital images without adversely affecting their quality. Without compression a natural photo[2] taken by a digital camera would require one to two orders of magnitude more storage space. As shown in Fig. 9.8, in a typical image compression scheme an input image is first cut up into small square (typically 8×8 pixel) blocks. The values of pixels in each block (which are integers between 0 and 255 for an 8-bit grayscale image) are stacked into a column vector $\mathbf{y}$, and compression is then performed on these individual vectorized blocks.

The primary idea behind many digital image compression algorithms is that with the use of specific bases, we only need very few of their elements to very closely approximate any natural image. One such basis, the 8×8 discrete cosine transform (DCT) which is the backbone of the popular JPEG compression scheme, consists of two-dimensional cosine waves of varying frequencies, and is shown in Fig. 9.9 along with its analogue standard basis. Most natural image blocks can be well approximated using only a few elements of the DCT basis. The reason is, as opposed to bases with more locally defined elements (e.g., the standard basis), each DCT basis element represents a fluctuation commonly seen across the entirety of a natural image block. Therefore with just a few of these elements, properly weighted, we can approximate a wide range of image blocks. In other words, instead of seeking out a basis (as with PCA), here we have a fixed basis over which image data can be very efficiently represented (the same holds, in fact, for other natural signals as well like audio data, see e.g., Section 4.6.3).

To perform compression, DCT basis patches in Fig. 9.9 are vectorized into a sequence of $P = 64$ *fixed* basis vectors $\left\{\mathbf{c}_p\right\}_{p=1}^{P}$ in the same manner as the input image blocks. Concatenating these patches column-wise into a matrix $\mathbf{C}$ and supposing there are K

[2] Pictures of natural subjects such as cities, meadows, people, and animals, etc., as opposed to synthetic images.

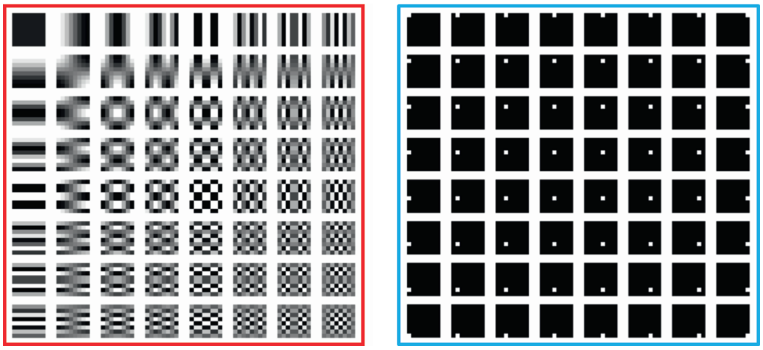

Fig. 9.9 (left) The set of 64 DCT basis elements used for compression of 8 × 8 image blocks. For visualization purposes pixel values in each basis patch are gray-coded so that white and black colors correspond to the minimum and maximum value in that patch, respectively. (right) The set of 64 standard basis elements, each having only one nonzero entry. Pixel values are gray-coded so that white and black colors correspond to entry values of 1 and 0, respectively. Most natural image blocks can be approximated as a linear combination of just a few DCT basis elements while the same cannot be said of the standard basis.

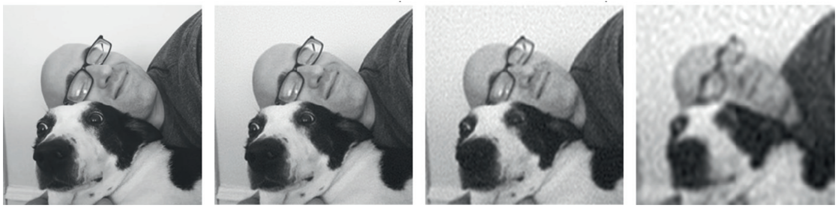

Fig. 9.10 From left to right, the original 256 × 256 input image along with its three compressed versions where we keep only the largest 20%, 5%, and 1% of the DCT coefficients to represent the image, resulting in compression by a factor of 5, 20, and 100, respectively. Although, as expected, the visual quality deteriorates as the compression factor increases, the 1% image still captures a considerable amount of information. This example is a testament to the efficiency of DCT basis in representing natural image data.

blocks in the input image, denoting by $\mathbf{x}_k$ its kth vectorized block, to represent the entire image over the basis we solve K linear systems of equations of the form

$$\mathbf{C}\mathbf{w}_k = \mathbf{x}_k. \tag{9.15}$$

Each vector $\mathbf{w}_k$ in (9.15) stores the DCT coefficients (or weights) corresponding to the image block $\mathbf{x}_k$. Most of the weights in the coefficient vectors $\{\mathbf{w}_k\}_{k=1}^{K}$ are typically quite small. Therefore, as illustrated by an example image in Fig. 9.10, setting even 80% of the smallest weights to zero gives an approximation that is essentially indistinguishable from the original image. Even setting 99% of the smallest weights to zero gives an approximation to the original data wherein we can still identify the objects in the original image. To compress the image, instead of storing each pixel value, only these few remaining nonzero coefficients are kept.

9.2.1 Optimization of the PCA problem

Because problem (9.14) is convex in each variable $\mathbf{C}$ and $\mathbf{W}$ individually (but non-convex in both simultaneously) a natural approach to solving this problem is to alternately minimize (9.14) over $\mathbf{C}$ and $\mathbf{W}$ independently.[3] However, while it is not obvious at first sight, there is in fact a closed-form solution to Equation (9.14) based on the singular value decomposition (SVD) of the matrix $\mathbf{X}$. Denoting the SVD of $\mathbf{X}$ as $\mathbf{X} = \mathbf{USV}^T$, this solution is given as

$$\begin{aligned} \mathbf{C}^\star &= \mathbf{U}_K \mathbf{S}_{K,K} \\ \mathbf{W}^\star &= \mathbf{V}_K^T, \end{aligned} \tag{9.17}$$

where $\mathbf{U}_K$ and $\mathbf{V}_K$ denote the matrices formed by the first K columns of the left and right singular matrices $\mathbf{U}$ and $\mathbf{V}$ respectively, and $\mathbf{S}_{K,K}$ denotes the upper $K \times K$ submatrix of the singular value matrix $\mathbf{S}$. Note that since $\mathbf{U}_K$ is an orthogonal matrix, the recovered basis (for the low dimensional subspace) is indeed orthogonal.

Further, it can be shown that these particular basis elements span the so-called orthogonal directions of variance (or spread) of the original dataset (see Exercise 9.3). While this characteristic is not particularly useful when using PCA as a preprocessing technique (since all we care about is reducing the dimension of the feature space, and so any basis spanning a proper subspace will suffice), it is often used in exploratory data analysis in fields like statistics and the social sciences (see e.g., factor analysis).

9.3 Recommender systems

Recommender systems are heavily used in e-commerce today, providing customers with personalized recommendations for products and services by using a consumer's previous purchasing and rating history, along with those of similar customers. For instance, a movie provider like Netflix with millions of users and tens of thousands of movies, records users' reviews and ratings (typically in the form of a number on a scale of 1–5 with 5 being the most favorable rating) in a large matrix such as the one illustrated in Fig. 9.11. These matrices are very sparsely populated, since an individual consumer has likely rated only a few of the movies available. With this data available, online movie providers can use machine learning techniques to make personalized recommendations

[3] Beginning at an initial value for both $\left(\mathbf{C}^{(0)}, \mathbf{W}^{(0)}\right)$ this produces a sequence of iterates $\left(\mathbf{C}^{(k)}, \mathbf{W}^{(k)}\right)$ where

$$\begin{aligned} \mathbf{C}^{(k)} &= \underset{\mathbf{C}}{\operatorname{argmin}} \left\| \mathbf{C}\mathbf{W}^{(k-1)} - \mathbf{X} \right\|_F^2 \\ \mathbf{W}^{(k)} &= \underset{\mathbf{W}}{\operatorname{argmin}} \left\| \mathbf{C}^{(k)}\mathbf{W} - \mathbf{X} \right\|_F^2, \end{aligned} \tag{9.16}$$

and where each may be expressed in closed form. Setting the gradient in each case to zero and solving gives $\mathbf{C}^{(k)} = \mathbf{X}\left(\mathbf{W}^{(k-1)}\right)^T \left(\mathbf{W}^{(k-1)}\left(\mathbf{W}^{(k-1)}\right)^T\right)^\dagger$ and $\mathbf{W}^{(k)} = \left(\left(\mathbf{C}^{(k)}\right)^T \mathbf{C}^{(k)}\right)^\dagger \left(\mathbf{C}^{(k)}\right)^T \mathbf{X}$ respectively, where $(\cdot)^\dagger$ denotes the pseudo-inverse. The procedure is stopped when the subsequent iterations do not change significantly (see exercises).

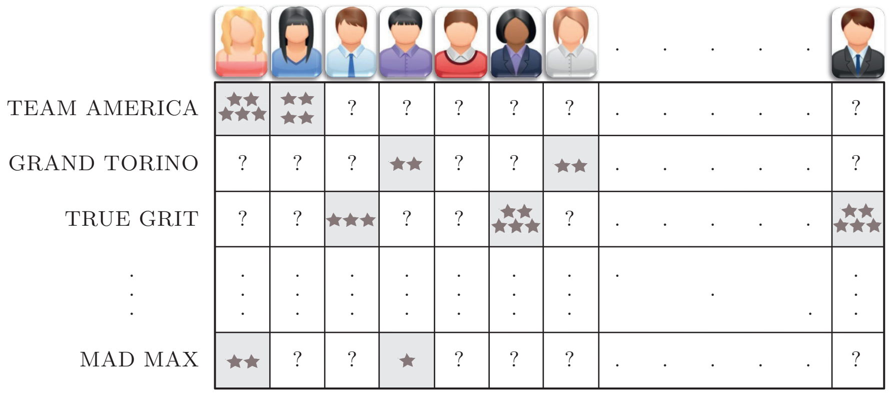

Fig. 9.11 A prototypical movie rating matrix is very sparsely populated, with each user having rated only a very small number of films. In this diagram movies are listed along rows while users are listed across the columns of the rating matrix.

to customers regarding what they might like to watch next. Producing these personalized recommendations requires *completing* the movie rating matrix, or in other words filling in the many missing entries with smart guesses of how much users would like films they have not yet seen.

In completing the movie rating matrix a helpful modeling tool often used is based on the assumption that only a few factors contribute to a user's taste or interest. For instance, users typically fall into a few categories when it comes to their movie preferences. Some only like horror movies and some are only interested in action films, yet there are those who enjoy both. There are users who passionately follow documentaries, and others who completely despise romantic comedies, and so on. The relatively small number of such categories or user types compared to the total number of users or movies in a rating matrix, provides a framework to fill in the missing values. Once the matrix is completed, those movies with the highest estimated ratings are recommended to the respective users. This same concept is used broadly by digital retailers like Amazon and eBay to recommend products of all kinds to their customers.

9.3.1 Matrix completion setup

In a simple model for a recommender system we can imagine an e-vendor who sells N goods and/or services to P users. The vendor collects and stores individual user-ratings on a 1 (= bad) to R (= great) integer-valued scale for each product purchased by each customer. If a customer has not reviewed a product the entry is set to 0. Looking at the entire customer–product database as an $N \times P$ matrix $\mathbf{X}$, we assume that there are just a few, say K, fundamental factors that each customer's behavior can be explained by.

Imagine for a moment that $\mathbf{X}$ were completely filled in. Then the assumption of K fundamental factors explaining $\mathbf{X}$ translates, in linear algebra terms, to the assumption that $\mathbf{X}$ can be factorized as

$$\mathbf{CW} \approx \mathbf{X}, \tag{9.18}$$

where $\mathbf{C}$ and $\mathbf{W}$ are $N \times K$ and $K \times P$ matrices, respectively. The K columns of $\mathbf{C}$ act as a fundamental spanning set for $\mathbf{X}$ and are interpretable in this instance as the basic factors that represent customers' preferences.

Now in the realistic case where $\mathbf{X}$ is incomplete we know only a fraction of the entries of $\mathbf{X}$. Denoting by $\Omega = \left\{(i,j) \,|\, x_{ij} \neq 0\right\}$ the set of index pairs (i,j) of rated products, where the jth customer has rated the ith product, in the database $\mathbf{X}$ our desire is to recover a factorization for $\mathbf{X}$ given only the known (rated) entries in the set Ω. That is, for each rating with index $(i,j) \in \Omega$,

$$\mathbf{CW}|_{\Omega} \approx \mathbf{X}|_{\Omega}, \tag{9.19}$$

which for a single $(i,j) \in \Omega$ can be written as

$$\mathbf{c}^i \mathbf{w}_j \approx x_{ij}, \tag{9.20}$$

where $\mathbf{c}^i$ is the ith row of $\mathbf{C}$. We then aim to learn $\mathbf{C}$ and $\mathbf{W}$ which minimize $\left(\mathbf{c}^i\mathbf{w}_j - x_{ij}\right)^2$ over all index pairs in Ω, i.e., by solving the optimization problem

$$\underset{\mathbf{C},\mathbf{W}}{\text{minimize}} \sum_{(i,j)\in\Omega} \left(\mathbf{c}^i\mathbf{w}_j - x_{ij}\right)^2. \tag{9.21}$$

9.3.2 Optimization of the matrix completion model

The matrix completion problem in (9.21) can be solved following an alternating minimization approach, in a similar sense as with K-means. The gradient of the objective function $g\left(\mathbf{C},\mathbf{W}\right) = \sum_{(i,j)\in\Omega} \left(\mathbf{c}^i\mathbf{w}_j - x_{ij}\right)^2$ with respect to the pth column of $\mathbf{W}$ is given by

$$\nabla_{\mathbf{w}_p} g = 2 \sum_{(i,p)\in\Omega} \left(\mathbf{c}^i\mathbf{w}_p - x_{ip}\right)\left(\mathbf{c}^i\right)^T. \tag{9.22}$$

Setting the gradient to zero, we can recover the optimal $\mathbf{w}_p$ by solving the following linear system of equations:

$$\left(\sum_{(i,p)\in\Omega} \mathbf{c}^i \left(\mathbf{c}^i\right)^T\right)\mathbf{w}_p = \sum_{(i,p)\in\Omega} x_{ip}\left(\mathbf{c}^i\right)^T. \tag{9.23}$$

Similarly, the optimal $\mathbf{c}^n$ can be found as a solution to

$$\mathbf{c}^n\left(\sum_{(n,j)\in\Omega} \mathbf{w}_j\mathbf{w}_j^T\right) = \sum_{(n,j)\in\Omega} x_{nj}\mathbf{w}_j^T. \tag{9.24}$$

By alternately solving these linear system until the values of $\mathbf{C}$ and $\mathbf{W}$ do not change very much we can solve (9.21).

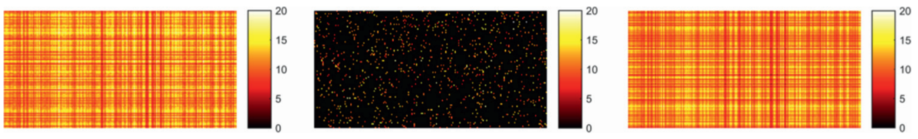

Fig. 9.12 (left panel) Simulated matrix $\mathbf{X}$ with integer-valued entries between 1 and 20. (middle panel) Matrix $\mathbf{X}$ after 95% of entries are removed. (right panel) Recovered matrix $\mathbf{X}^{\star}$ resembles the original matrix $\mathbf{X}$ and matches the original very well.

Example 9.4 Matrix completion of simulated dataset

Here we show an experiment with a simulated data matrix $\mathbf{X}$ with $N = 100$ rows (or products) and $P = 200$ columns (or users). Each entry in $\mathbf{X}$ is an integer-valued rating between 1 and 20, and the complete matrix $\mathbf{X}$ has a rank of $K = 5$, or in other words a matrix rightly approximated using $N \times K$ matrix $\mathbf{C}$ and $K \times P$ matrix $\mathbf{W}$. In the left panel of Fig. 9.12 we show the original matrix $\mathbf{X}$ with each entry color-coded depending on its value. The middle panel shows the matrix $\mathbf{X}$ after 95% of its entries have been randomly removed. Applying matrix completion to this corrupted version of $\mathbf{X}$, by using alternating minimization, we can recover the original matrix (with a small root mean square error of 0.05), as illustrated in the right panel of Fig. 9.12.

9.4 Summary

In this chapter we have described several methods for dimension reduction, beginning in Section 9.1 describing commonly used data dimension reduction techniques (or ways of properly reducing the size of a dataset). Random subsampling, described first, is the most commonly used method for slimming down a regression/classification dataset that simply involves keeping a random selection of points from an original dataset. Here one typically keeps as much of the original data as computational resources permit. A popular alternative to random subsampling, also used for various "data analysis" tasks beyond predictive modeling, is the K-means clustering method. This approach involves the computation of cluster "centroids" of a dataset that best describe its overall structure.

In Section 9.2 we described a classic technique for feature dimension reduction (that is shrinking the dimension of each data point) referred to as principal component analysis (PCA). While not particularly useful for predictive modeling (see Example 9.2 for how it can in fact be destructive when applied to classification data), PCA is the fundamental matrix factorization problem which can help frame our understanding of a wide array of models/problems including: Least Squares for linear regression, K-means, and more. For example in the final section of the chapter we detailed a common matrix factorization approach to recommender systems, or algorithms that recommend products/services to a common base of users.

9.5 Exercises

Section 9.1 exercises

Exercises 9.1 Code up K-means

In this exercise you will reproduce the results shown in Fig. 9.4 by coding up the K-means algorithm (shown in Algorithm 9.1).

a) Place your K-means code in the function

$$[\mathbf{C},\,\mathbf{W}] = \text{your\_}K\text{\_means}\left(\mathbf{X},\,K\right), \tag{9.25}$$

located inside the wrapper *kmeans_demo* (this wrapper together with the assocated dataset *kmeans_demo_data.csv* may be downloaded from the book website). All of the additional code necessary to generate the associated plots is already provided in the wrapper. Here **C** and **W** are the centroid and assignment matrices output by the algorithm, while **X** and K are the data matrix and number of desired centroids, respectively.

b) Run the wrapper with $K = 2$ centroids using the initialization $\mathbf{C} = \begin{bmatrix} 0 & 0 \\ -0.5 & 0.5 \end{bmatrix}$. This should reproduce the successful run of K-means shown in the bottom panels of the figure.

c) Run the wrapper with $K = 2$ centroids using the initialization $\mathbf{C} = \begin{bmatrix} 0 & 0 \\ 0 & 0.5 \end{bmatrix}$. This should reproduce the unsuccessful run of K-means shown in the top panels of the figure.

Section 9.2 exercises

Exercises 9.2 Code up PCA

In this exercise you will reproduce the results shown in Fig. 9.6 by coding up PCA.

a) Implement a singular value decomposition approach to PCA described in Section 9.2.1, placing the resulting code in the function

$$[\mathbf{C},\,\mathbf{W}] = \text{your\_PCA}\left(\mathbf{X},\,K\right), \tag{9.26}$$

located inside the wrapper *PCA_demo* (this wrapper together with the assocated dataset *PCA_demo_data.csv* may be downloaded from the book website). Here **C** and **W** are the spanning set and weight matrices output by the algorithm, while **X** and K are the data matrix and number of desired basis elements, respectively.

All of the additional code necessary to generate the associated plots is already provided in the wrapper. Run the wrapper to ensure that you have coded up PCA correctly.

b) Using the wrapper and data from part a) implement the alternating directions solution to PCA described in footnote 3, once again placing this code inside the function *your_PCA* described previously.

Exercises 9.3 Deriving principal components as orthogonal directions of variance

In this exercise you will show how to derive principal component analysis as the orthogonal directions of largest variance of a dataset. Given P points $\left\{\mathbf{x}_p\right\}_{p=1}^{P}$ of dimension N we may calculate the variance in a unit direction $\mathbf{d}$ (i.e., how much the dataset spreads out in the direction $\mathbf{d}$) with respect to the data as the average squared inner product of the data against $\mathbf{d}$,

$$\frac{1}{P}\sum_{p=1}^{P}\left\langle \mathbf{x}_p, \mathbf{d}\right\rangle^2 . \tag{9.27}$$

This can be written more compactly as

$$\frac{1}{P}\|\mathbf{X}^T\mathbf{d}\|_2^2 = \frac{1}{P}\mathbf{d}^T\mathbf{X}\mathbf{X}^T\mathbf{d}. \tag{9.28}$$

Note that the outer product $\mathbf{X}\mathbf{X}^T$ is a symmetric positive semi-definite matrix.

a) Compute the largest direction of variance of the data, i.e., the unit vector $\mathbf{d}$ that maximizes the value $\mathbf{d}^T\mathbf{X}\mathbf{X}^T\mathbf{d}$. *Hint: use the eigen-decomposition of XX^T.*

b) Compute the second largest direction of variance of the matrix $\mathbf{X}\mathbf{X}^T$, i.e., the unit vector $\mathbf{d}$ that maximizes the value of $\mathbf{d}^T\mathbf{X}\mathbf{X}^T\mathbf{d}$ but where $\mathbf{d}$ is also orthogonal to the first largest direction of variance. *Hint: use the eigen-decomposition of XX^T.*

c) Conclude from part a) and b) that the orthogonal directions of variance of the data are precisely the singular value solution given in Equation (9.17).

Section 9.3 exercises

Exercises 9.4 Code up the matrix completion recommender system

In this exercise you will reproduce the matrix completion recommender system results shown in Fig. 9.12.

Code up the alternating minimization algorithm described in Section 9.3.2, placing the resulting code in the function

$$[\mathbf{C},\ \mathbf{W}] = \text{matrix\_complete}\left(\mathbf{X},\ K\right), \tag{9.29}$$

located inside the wrapper *recommender_demo* (this wrapper and the assocated dataset *recommender_demo_data.csv* may be downloaded from the book website). Here $\mathbf{C}$ and

W are the spanning set and weight matrices output by the algorithm, while **X** and K are the data matrix and number of desired basis elements, respectively.

All of the additional code necessary to generate the associated plots is already provided in the wrapper. Run the wrapper to ensure that you have coded up the matrix completion algorithm correctly.