9.2 Dimension Reduction and Feature Selection

The *curse of dimensionality* is a general observation that statistical tasks get exponentially harder as the dimensions increase. In learning from data, this manifests itself in many ways, the most immediate being computational. Simple algorithms, such as optimal (or near-optimal) k-means clustering, or determining the optimal linear separator, have a computational complexity which scales exponentially with dimensionality. Furthermore, a fixed number, N, of data points only sparsely populates a space whose volume is growing exponentially with d. So, simple rules like nearest neighbor get adversely affected because a test point's 'nearest' neighbor will likely be very far away and will not be a good representative point for predicting on the test point. The complexity of a hypothesis set, as could be measured by the VC dimension, will typically increase with d (recall that, for the simple linear perceptron, the VC dimension is $d + 1$), affecting the generalization from in-sample to out-of-sample. The bottom line is that more data are needed to learn in higher-dimensional input spaces.

Exercise 9.5

Consider a data set with two examples,

$$(\mathbf{x}_1^{\mathrm{T}} = [-1, a_1, \ldots, a_d],\ y_1 = +1); \qquad (\mathbf{x}_2^{\mathrm{T}} = [1, b_1, \ldots, b_d],\ y_2 = -1),$$

where a_i, b_i are independent random ± 1 variables. Let $\mathbf{x}_{\text{test}}^{\mathrm{T}} = [-1, -1, \ldots, -1]$. Assume that only the first component of $\mathbf{x}$ is relevant to f. However, the actual measured $\mathbf{x}$ has additional random components in the additional d dimensions. If the nearest neighbor rule is used, show, either mathematically or with an experiment, that the probability of classifying $\mathbf{x}_{\text{test}}$ correctly is $\frac{1}{2} + O(\frac{1}{\sqrt{d}})$ (d is the number of irrelevant dimensions).

What happens if there is a third data point ($\mathbf{x}_3^{\mathrm{T}} = [1, c_1, \ldots, c_d], y_3 = -1$)?

The exercise illustrates that as you have more and more spurious (random) dimensions, the learned final hypothesis becomes useless because it is dominated by the random fluctuations in these spurious dimensions. Ideally, we should remove all such spurious dimensions before proceeding to the learning. Equivalently, we should retain only the few informative features. For the digits data from Chapter 3, we were able to obtain good performance by extracting just two features from the raw 16×16-pixel input image (size and symmetry), a dimension reduction from 256 raw features to 2 informative ones.

The features $\mathbf{z}$ are simply a transformation of the input $\mathbf{x}$,

$$\mathbf{z} = \Phi(\mathbf{x}),$$

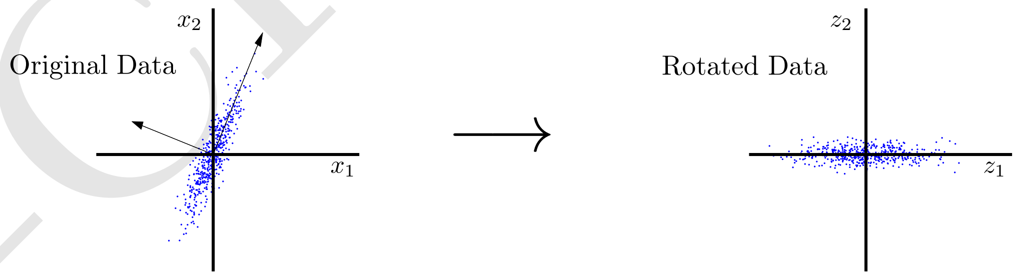
where the number of features is the dimension of $\mathbf{z}$. If the dimension of $\mathbf{z}$ is less than the dimension of $\mathbf{x}$, then we have accomplished dimension reduction. The ideal feature is the target function itself, $z = f(\mathbf{x})$, since if we had this feature, we are done. This suggests that quality feature selection may be as hard as the original learning problem of identifying f.

We have seen features and feature transforms many times before, for example, in the context of linear models and the non-linear feature transform in Chapter 3. In that context, the non-linear feature transform typically *increased* the dimension to handle the fact that the linear hypothesis was not expressive enough to fit the data. In that setting, increasing the dimension through the feature transform was attempting to improve E_{in}, and we did pay the price of poorer generalization. The reverse is also true. If we can *lower* the dimension without hurting E_{in} (as would be the case if we retained all the important information), then we will also improve generalization.

9.2.1 Principal Components Analysis (PCA)

Centering, scaling and whitening all attempt to correct for arbitrary choices that may have been made during data collection. Feature selection, such as PCA, is conceptually different. It attempts to get rid of redundancy or less informative dimensions to help, among other things, generalization. For example, the top right pixel in the digits data is almost always white, so it is a dimension that carries almost no information. Removing that dimension will not hurt the fitting, but will improve generalization.

PCA constructs a small number of *linear* features to summarize the input data. The idea is to rotate the axes (a linear transformation that defines a new coordinate system) so that the important dimensions in this new coordinate system become self evident and can be retained while the less important ones get discarded. Our toy data set can help crystallize the notion.

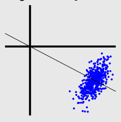
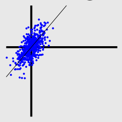


Once the data are rotated to the new 'natural' coordinate system, z_1 stands out as the important dimension of the transformed input. The second dimension, z_2, looks like a bunch of small fluctuations which we ought to ignore, in comparison to the apparently more informative and larger z_1. Ignoring the z_2 dimension amounts to setting it to zero (or just throwing away that coordinate), producing a 1-dimensional feature.

Exercise 9.6

Try to build some intuition for what the rotation is doing by using the illustrations in Figure 9.1 to qualitatively answer these questions.

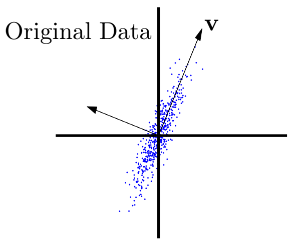
(a) If there is a large offset (or bias) in both measured variables, how will this affect the 'natural axes', the ones to which the data will be rotated? Should you perform input centering before doing PCA?

[*Hint: Consider*  *versus* ]

(b) If one dimension (say x_1) is inflated disproportionately (e.g., income is measured in dollars instead of thousands of dollars). How will this affect the 'natural axes', the ones to which the data should be rotated? Should you perform input normalization before doing PCA?

(c) If you do input whitening, what will the 'natural axes' for the inputs be? Should you perform input whitening before doing PCA?

What if the small fluctuations in the z_2 direction were the actual important information on which f depends, and the large variability in the z_1 dimension are random fluctuations? Though possible, this rarely happens in practice, and if it does happen, then your input is corrupted by large random noise and you are in trouble anyway. So, let's focus on the case where we have a chance and discuss how to find this optimal rotation.



Intuitively, the direction $\mathbf{v}$ captures the largest fluctuations in the data, which could be measured by variance. If we project the input $\mathbf{x}_n$ onto $\mathbf{v}$ to get $z_n = \mathbf{v}^{\mathrm{T}}\mathbf{x}_n$, then the variance of z is $\frac{1}{N}\sum_{n=1}^{N} z_n^2$ (remember $\mathbf{x}_n$ and hence z_n have zero mean).

$$\begin{aligned}\text{var}[z] &= \frac{1}{N}\sum_{n=1}^{N} z_n^2 = \frac{1}{N}\sum_{n=1}^{N} \mathbf{v}^{\mathrm{T}}\mathbf{x}_n\mathbf{x}_n^{\mathrm{T}}\mathbf{v} \\ &= \mathbf{v}^{\mathrm{T}}\left(\frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n\mathbf{x}_n^{\mathrm{T}}\right)\mathbf{v} \\ &= \mathbf{v}^{\mathrm{T}}\Sigma\mathbf{v}.\end{aligned}$$

To maximize var$[z]$, we should pick $\mathbf{v}$ as the top eigenvector of Σ, the one with the largest eigenvalue. Before we get more formal, let us address an apparent conflict of interest. Whitening is a way to put your data into a spherically symmetric form so that all directions are 'equal'. This is recommended when you have no evidence to the contrary; you whiten the data because most learning algorithms treat every dimension equally (nearest neighbor, weight decay, etc.). PCA, on the other hand is highlighting specific directions which contain more variance. There is no use doing PCA after doing whitening, since every direction will be on an equal footing after whitening. You use

PCA precisely because the directions are not to be treated equally. PCA helps to identify and throw away the directions where the fluctuations are a result of small amounts of noise. After deciding which directions to throw away, you can now use whitening to put all the retained directions on an equal footing, if you wish.

Our visual intuition works well in 2-dimensions, but in higher dimension, when no single direction captures most of the fluctuation, we need a more principled approach, starting with a mathematical formulation of the task. We begin with the observation that a rotation of the data exactly corresponds to representing the data in a new (rotated) coordinate system.

Coordinate Systems A coordinate system is defined by an orthonormal basis, a set of mutually orthogonal unit vectors. The standard Euclidean coordinate system is defined by the Euclidean basis in d dimensions, $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_d$, where $\mathbf{u}_i$ is the ith standard basis vector which has a 1 in coordinate i and 0 for all other coordinates. The input vector $\mathbf{x}$ has the components $x_i = \mathbf{x}^{\mathrm{T}}\mathbf{u}_i$, and we can write

$$\mathbf{x} = \sum_{i=1}^{d} x_i \mathbf{u}_i = \sum_{i=1}^{d} (\mathbf{x}^{\mathrm{T}}\mathbf{u}_i)\mathbf{u}_i.$$

This can be done for any orthonormal basis $\mathbf{v}_1, \ldots, \mathbf{v}_d$,

$$\mathbf{x} = \sum_{i=1}^{d} z_i \mathbf{v}_i = \sum_{i=1}^{d} (\mathbf{x}^{\mathrm{T}}\mathbf{v}_i)\mathbf{v}_i,$$

where the *coordinates* in the basis $\mathbf{v}_1, \ldots, \mathbf{v}_d$ are $z_i = (\mathbf{x}^{\mathrm{T}}\mathbf{v}_i)$. The goal of PCA is to construct a more intuitive basis where some (hopefully the majority) of the coordinates are small and can be treated as small random fluctuations. These coordinates are going to be discarded, i.e., set to zero. The hope is that we have reduced the dimensionality of the problem while retaining most of the important information.

So, given the vector $\mathbf{x}$ (in the standard coordinate system) and some other coordinate system $\mathbf{v}_1, \ldots, \mathbf{v}_d$, we can define the transformed feature vector whose components are the coordinates $z_1, \ldots, z_d$ in this new coordinate system. Suppose that the first $k \le d$ of these transformed coordinates are the informative ones, so we throw away the remaining coordinates to arrive at our *dimension-reduced* feature vector

$$\mathbf{z} = \begin{bmatrix} z_1 \\ \vdots \\ z_k \end{bmatrix} = \begin{bmatrix} \mathbf{x}^{\mathrm{T}}\mathbf{v}_1 \\ \vdots \\ \mathbf{x}^{\mathrm{T}}\mathbf{v}_k \end{bmatrix} = \Phi(\mathbf{x}).$$

Exercise 9.7

(a) Show that $\mathbf{z}$ is a linear transformation of $\mathbf{x}$, $\mathbf{z} = \mathrm{V}^{\mathrm{T}}\mathbf{x}$. What are the dimensions of the matrix V and what are its columns?

(b) Show that the transformed data matrix is $\mathrm{Z} = \mathrm{XV}$.

(c) Show that $\sum_{i=1}^{d} z_i^2 = \sum_{i=1}^{d} x_i^2$ and hence that $\|\mathbf{z}\| \leq \|\mathbf{x}\|$.

If we kept all the components $z_1, \ldots, z_d$, then we can reconstruct $\mathbf{x}$ via

$$\mathbf{x} = \sum_{i=1}^{d} z_i \mathbf{v}_i.$$

Using only the first k components, the best reconstruction of $\mathbf{x}$ is

$$\hat{\mathbf{x}} = \sum_{i=1}^{k} z_i \mathbf{v}_i.$$

We have lost that part of $\mathbf{x}$ represented by the trailing coordinates of $\mathbf{z}$. The magnitude of the part we lost is captured by the reconstruction error

$$\|\mathbf{x} - \hat{\mathbf{x}}\|^2 = \left\| \sum_{i=k+1}^{d} z_i \mathbf{v}_i \right\|^2 = \sum_{i=k+1}^{d} z_i^2$$

(because $\mathbf{v}_1, \ldots, \mathbf{v}_d$ are *orthonormal*). The new coordinate system is good if the sum of the reconstruction errors over the data points is small. That is, if

$$\sum_{n=1}^{N} \|\mathbf{x}_n - \hat{\mathbf{x}}_n\|^2$$

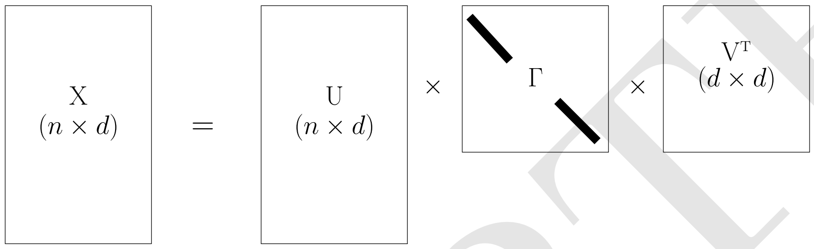
is small. If the $\mathbf{x}_n$ are reconstructed with small error from $\mathbf{z}_n$ (i.e. $\hat{\mathbf{x}}_n \approx \mathbf{x}_n$), then not much information was lost. PCA finds a coordinate system that *minimizes* this total reconstruction error. The trailing dimensions will have the least possible information, and so even after throwing away those trailing dimensions, we can still almost reconstruct the original data. PCA is optimal, which means that no other *linear* method can produce coordinates with a smaller reconstruction error. The first k basis vectors, $\mathbf{v}_1, \ldots, \mathbf{v}_k$, of this optimal coordinate basis are called the top-k principal directions.

So, how do we find this optimal coordinate basis $\mathbf{v}_1, \ldots, \mathbf{v}_d$ (of which we only need $\mathbf{v}_1, \ldots, \mathbf{v}_k$ to compute our dimensionally reduced feature)? The solution to this problem has been known since 1936 when the remarkable *singular value decomposition* (SVD) was invented. The SVD is such a useful tool for learning from data that time spent mastering it will pay dividends (additional background on the SVD is given in Appendix B.2).

The Singular Value Decomposition (SVD). Any matrix, for example, our data matrix X, has a very special representation as the product of three matrices. Assume that $\mathrm{X} \in \mathbb{R}^{N\times d}$ with $N \geq d$ (a similar decomposition holds for $\mathrm{X}^{\scriptscriptstyle T}$ if $N < d$). Then,

$$\mathrm{X} = \mathrm{U}\Gamma\mathrm{V}^{\scriptscriptstyle T}$$

where $\mathrm{U} \in \mathbb{R}^{n\times d}$ has orthonormal columns, $\mathrm{V} \in \mathbb{R}^{d\times d}$ is an orthogonal matrix[3] and Γ is a non-negative diagonal matrix.[4] The diagonal elements $\gamma_i = \Gamma_{ii}$, where $\gamma_1 \geq \gamma_2 \geq \cdots \geq \gamma_d \geq 0$, are the *singular values* of X, ordered from largest to smallest. The number of non-zero singular values is the rank of X, which we will assume is d for simplicity. Pictorially,



The matrix U contains (as its columns) the *left singular vectors* of X, and similarly V contains (as its columns) the *right singular vectors* of X. Since U consists of orthonormal columns, $\mathrm{U}^{\scriptscriptstyle T}\mathrm{U} = \mathrm{I}_d$. Similarly, $\mathrm{V}^{\scriptscriptstyle T}\mathrm{V} = \mathrm{V}\mathrm{V}^{\scriptscriptstyle T} = \mathrm{I}_d$. If X is square, then U will be square. Just as a square matrix maps an eigenvector to a multiple of itself, a more general non-square matrix maps a left (resp. right) singular vector to a multiple of the corresponding right (resp. left) singular vector, as verified by the following identities:

$$\mathrm{U}^{\scriptscriptstyle T}\mathrm{X} = \Gamma\mathrm{V}^{\scriptscriptstyle T}; \qquad \mathrm{X}\mathrm{V} = \mathrm{U}\Gamma.$$

It is convenient to have column representations of U and V in terms of the singular vectors, $\mathrm{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_d]$ and $\mathrm{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_d]$.

Exercise 9.8

Show $\mathrm{U}^{\scriptscriptstyle T}\mathrm{X} = \Gamma\mathrm{V}^{\scriptscriptstyle T}$ and $\mathrm{X}\mathrm{V} = \mathrm{U}\Gamma$, and hence $\mathrm{X}^{\scriptscriptstyle T}\mathbf{u}_i = \gamma_i\mathbf{v}_i$ and $\mathrm{X}\mathbf{v}_i = \gamma_i\mathbf{u}_i$.

(The ith singular vectors and singular value $(\mathbf{u}_i, \mathbf{v}_i, \gamma_i)$ play a similar role to eigenvector-eigenvalue pairs.)

Computing the Principal Components via SVD. It is no coincidence that we used $\mathbf{v}_i$ for the right singular vectors of X and $\mathbf{v}_i$ in the mathematical formulation of the PCA task. The right singular vectors $\mathbf{v}_1, \ldots, \mathbf{v}_d$ are our optimal coordinate basis so that by ignoring the trailing components we incur the least reconstruction error.

[3]An orthogonal matrix is a square matrix with orthonormal columns, and therefore its inverse is the same as its transpose. So, $\mathrm{V}^{\scriptscriptstyle T}\mathrm{V}=\mathrm{V}\mathrm{V}^{\scriptscriptstyle T}=\mathrm{I}$.

[4]In the traditional linear algebra literature, Γ is typically denoted by Σ and its diagonal elements are the singular values $\sigma_1 \geq \cdots \geq \sigma_d$. We use Γ because we have reserved Σ for the covariance matrix of the input distribution and σ^2 for the noise variance.

Theorem 9.1 (Eckart and Young, 1936)**.** For any k, $\mathbf{v}_1, \ldots, \mathbf{v}_k$ (the top-k right singular vectors of the data matrix X) are a set of top-k principal component directions and the optimal reconstruction error is $\sum_{i=k+1}^{d} \gamma_i^2$.

The components of the dimensionally reduced feature vector are $z_i = \mathbf{x}^{\mathsf{T}}\mathbf{v}_i$ and the reconstructed vector is $\hat{\mathbf{x}} = \sum_{i=1}^{k} z_i \mathbf{v}_i$, which in matrix form is

$$\hat{\mathrm{X}} = \mathrm{X}\mathrm{V}_k\mathrm{V}_k^{\mathsf{T}}, \tag{9.2}$$

where $\mathrm{V}_k = [\mathbf{v}_1, \ldots, \mathbf{v}_k]$ is the matrix of top-k right singular vectors of X.

PCA Algorithm:

Inputs: The *centered* data matrix X and $k \geq 1$.

```
1: Compute the SVD of X: $[\mathrm{U}, \Gamma, \mathrm{V}] = \mathsf{svd}(\mathrm{X})$.
2: Let $\mathrm{V}_k = [\mathbf{v}_1, \ldots, \mathbf{v}_k]$ be the first $k$ columns of V.
3: The PCA-feature matrix and the reconstructed data are
       $\mathrm{Z} = \mathrm{X}\mathrm{V}_k, \qquad \hat{\mathrm{X}} = \mathrm{X}\mathrm{V}_k\mathrm{V}_k^{\mathsf{T}}.$
```

Note that PCA, along with the other input pre-processing tools (centering, rescaling, whitening) are all *unsupervised* - you do not need the y-values. The Eckart-Young theorem is quite remarkable and so fundamental in data analysis that it certainly warrants a proof.

Begin safe skip: You may skip the proof without compromising the logical sequence. A similar green box will tell you when to rejoin.

We will need some matrix algebra preliminaries which are useful general tools. Recall that the reconstructed data matrix $\hat{\mathrm{X}}$ is similar to the data matrix, having the reconstructed input vectors $\hat{\mathbf{x}}_n$ as its rows. The Frobenius norm $\|\mathrm{A}\|_F^2$ of a matrix $\mathrm{A} \in \mathbb{R}^{N\times d}$ is the analog of the Euclidean norm, but for matrices:

$$\|\mathrm{A}\|_F^2 \overset{\text{def}}{=} \sum_{n=1}^{N}\sum_{i=1}^{d} \mathrm{A}_{ij}^2 = \sum_{n=1}^{N} \|\mathrm{row}_n(\mathrm{A})\|^2 = \sum_{i=1}^{d} \|\mathrm{column}_i(\mathrm{A})\|^2.$$

The reconstruction error is exactly the Frobenius norm of the matrix difference between the original and reconstructed data matrices, $\|\mathrm{X} - \hat{\mathrm{X}}\|_F^2$.

Exercise 9.9

Consider an arbitrary matrix A, and any matrices U, V with orthonormal columns ($\mathrm{U}^{\mathsf{T}}\mathrm{U} = \mathrm{I}$ and $\mathrm{V}^{\mathsf{T}}\mathrm{V} = \mathrm{I}$).

(a) Show that $\|\mathrm{A}\|_F^2 = \mathrm{trace}(\mathrm{A}\mathrm{A}^{\mathsf{T}}) = \mathrm{trace}(\mathrm{A}^{\mathsf{T}}\mathrm{A})$.

(b) Show that $\|\mathrm{U}\mathrm{A}\mathrm{V}^{\mathsf{T}}\|_F^2 = \|\mathrm{A}\|_F^2$ (assume all matrix products exist). *[Hint: Use part (a).]*

Proof of the Eckart-Young Theorem. The preceding discussion was for general A. We now set A to be any orthonormal basis A, with columns $\mathbf{a}_1, \ldots, \mathbf{a}_d$,

$$\mathrm{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_d].$$

Since V is an orthonormal basis, we can write

$$\mathrm{A} = \mathrm{V}\boldsymbol{\Psi},$$

where $\boldsymbol{\Psi} = [\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_d]$. Since

$$\mathrm{I} = \mathrm{A}^{\mathrm{T}}\mathrm{A} = \boldsymbol{\Psi}^{\mathrm{T}}\mathrm{V}^{\mathrm{T}}\mathrm{V}\boldsymbol{\Psi} = \boldsymbol{\Psi}^{\mathrm{T}}\boldsymbol{\Psi},$$

we see that $\boldsymbol{\Psi}$ is orthogonal. Suppose (without loss of generality) that we will use the first k basis vectors of A for reconstruction. So, define

$$\mathrm{A}_k = [\mathbf{a}_1, \ldots, \mathbf{a}_k] = \mathrm{V}\boldsymbol{\Psi}_k,$$

where $\Psi_k = [\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_k]$. We use A_k to approximate X using the reconstruction $\hat{\mathrm{X}} = \mathrm{X}\mathrm{A}_k\mathrm{A}_k^{\mathrm{T}}$ from (9.2). Then,

$$\begin{aligned}
\|\mathrm{X} - \hat{\mathrm{X}}\|_F^2 &= \|\mathrm{X} - \mathrm{X}\mathrm{A}_k\mathrm{A}_k^{\mathrm{T}}\|_F^2 \\
&= \|\mathrm{U}\Gamma\mathrm{V}^{\mathrm{T}} - \mathrm{U}\Gamma\mathrm{V}^{\mathrm{T}}\mathrm{V}\boldsymbol{\Psi}_k\boldsymbol{\Psi}_k^{\mathrm{T}}\mathrm{V}^{\mathrm{T}}\|_F^2 \\
&= \|\mathrm{U}(\Gamma - \Gamma\boldsymbol{\Psi}_k\boldsymbol{\Psi}_k^{\mathrm{T}})\mathrm{V}^{\mathrm{T}}\|_F^2 \\
&= \|\Gamma(\mathrm{I} - \boldsymbol{\Psi}_k\boldsymbol{\Psi}_k^{\mathrm{T}})\|_F^2,
\end{aligned}$$

where we have used $\mathrm{V}^{\mathrm{T}}\mathrm{V} = \mathrm{I}_d$ and Exercise 9.9(b). By Exercise 9.9(a),

$$\|\Gamma(\mathrm{I} - \boldsymbol{\Psi}_k\boldsymbol{\Psi}_k^{\mathrm{T}})\|_F^2 = \mathrm{trace}(\Gamma(\mathrm{I} - \boldsymbol{\Psi}_k\boldsymbol{\Psi}_k^{\mathrm{T}})^2\Gamma).$$

Now, using the linearity and cyclic properties of the trace and the fact that $\mathrm{I} - \boldsymbol{\Psi}_k\boldsymbol{\Psi}_k^{\mathrm{T}}$ is a projection,

$$\begin{aligned}
\mathrm{trace}\left(\Gamma(\mathrm{I} - \boldsymbol{\Psi}_k\boldsymbol{\Psi}_k^{\mathrm{T}})^2\Gamma\right) &= \mathrm{trace}\left(\Gamma(\mathrm{I} - \boldsymbol{\Psi}_k\boldsymbol{\Psi}_k^{\mathrm{T}})\Gamma\right) \\
&= \mathrm{trace}(\Gamma^2) - \mathrm{trace}\left(\Gamma\boldsymbol{\Psi}_k\boldsymbol{\Psi}_k^{\mathrm{T}}\Gamma\right) \\
&= \mathrm{trace}(\Gamma^2) - \mathrm{trace}\left(\boldsymbol{\Psi}_k^{\mathrm{T}}\Gamma^2\boldsymbol{\Psi}_k\right).
\end{aligned}$$

The first term is independent of $\boldsymbol{\Psi}_k$, so we must maximize the second term. Since $\boldsymbol{\Psi}_k$ has k orthonormal columns, $\|\boldsymbol{\Psi}_k\|_F^2 = k$ (because the Frobenius norm is the sum of squared column norms). Let the rows of $\boldsymbol{\Psi}_k$ be $\mathbf{q}_1^{\mathrm{T}}, \ldots, \mathbf{q}_d^{\mathrm{T}}$. Then $0 \le \|\mathbf{q}_i\|^2 \le 1$ ($\boldsymbol{\Psi}$ has orthonormal rows and $\mathbf{q}_i$ are truncated rows of $\boldsymbol{\Psi}$), and $\sum_{i=1}^d \|\mathbf{q}_i\|^2 = k$ (because the Frobenius norm is the sum of squared row-norms). We also have that

$$\boldsymbol{\Psi}_k^{\mathrm{T}}\Gamma^2\boldsymbol{\Psi}_k = [\mathbf{q}_1, \ldots, \mathbf{q}_d]\begin{bmatrix}\gamma_1^2 & & \\ & \ddots & \\ & & \gamma_d^2\end{bmatrix}\begin{bmatrix}\mathbf{q}_1^{\mathrm{T}} \\ \vdots \\ \mathbf{q}_d\end{bmatrix} = \sum_{i=1}^d \gamma_i^2\mathbf{q}_i\mathbf{q}_i^{\mathrm{T}}.$$

Taking the trace and using linearity of the trace gives

$$\text{trace}(\mathbf{\Psi}_k^{\mathrm{T}}\Gamma^2\mathbf{\Psi}_k) = \sum_{i=1}^{d} \gamma_i^2 \|\mathbf{q}_i\|^2.$$

To maximize $\text{trace}(\mathbf{\Psi}_k^{\mathrm{T}}\Gamma^2\mathbf{\Psi}_k)$, the best possible way to spend our budget of k for $\sum_i \|\mathbf{q}_i\|^2$ is to put as much as possible into $\|\mathbf{q}_1\|$ and then $\|\mathbf{q}_2\|$ and so on, because $\gamma_1 \ge \gamma_2 \ge \cdots$. This will result in the $\|\mathbf{q}_1\|^2 = \cdots = \|\mathbf{q}_k\|^2 = 1$ and all the remaining $\mathbf{q}_i = \mathbf{0}$. Thus,

$$\text{trace}(\mathbf{\Psi}_k^{\mathrm{T}}\Gamma^2\mathbf{\Psi}_k) \le \sum_{i=1}^{k} \gamma_i^2.$$

We conclude that

$$\begin{aligned}
\|\mathrm{X} - \hat{\mathrm{X}}\|_F^2 &= \text{trace}(\Gamma^2) - \text{trace}\left(\mathbf{\Psi}_k^{\mathrm{T}}\Gamma^2\mathbf{\Psi}_k\right) \\
&= \sum_{i=1}^{d}\gamma_i^2 - \text{trace}\left(\mathbf{\Psi}_k^{\mathrm{T}}\Gamma^2\mathbf{\Psi}_k\right) \\
&\ge \sum_{i=1}^{d}\gamma_i^2 - \sum_{i=1}^{k}\gamma_i^2 \\
&= \sum_{i=k+1}^{d}\gamma_i^2.
\end{aligned}$$

If $\mathrm{A} = \mathrm{V}$ so that $\mathbf{\Psi} = \mathrm{I}$, then indeed $\|\mathbf{q}_1\|^2 = \cdots = \|\mathbf{q}_k\|^2 = 1$ and we attain equality in the bound above, showing that the top-k right singular vectors of X do indeed give an optimal basis, which concludes the proof. ■

The proof of the theorem also shows that the optimal reconstruction error is the sum of squares of the trailing singular values of X, $\|\mathrm{X} - \hat{\mathrm{X}}\|_F^2 = \sum_{i=k+1}^{d}\gamma_i^2$.

End safe skip: Those who skipped the proof are now rejoining us for some examples of PCA in action.

Example 9.2. It is instructive to see PCA in action. The digits training data matrix has 7291 (16×16)-images $(d = 256)$, so $\mathrm{X} \in \mathbb{R}^{7291\times 256}$. Let $\bar{\mathbf{x}}$ be the average image. First center the data, setting $\mathbf{x}_n \leftarrow \mathbf{x}_n - \bar{\mathbf{x}}$, to get a centered data matrix X.

Now, let's compute the SVD of this centered data matrix, $\mathrm{X} = \mathrm{U}\Gamma\mathrm{V}^{\mathrm{T}}$. To do so we may use a built in SVD package that is pretty much standard on any numerical platform. It is also possible to only compute the top-k singular vectors in most SVD packages to obtain $\mathrm{U}_k, \Gamma_k, \mathrm{V}_k$, where U_k and V_k

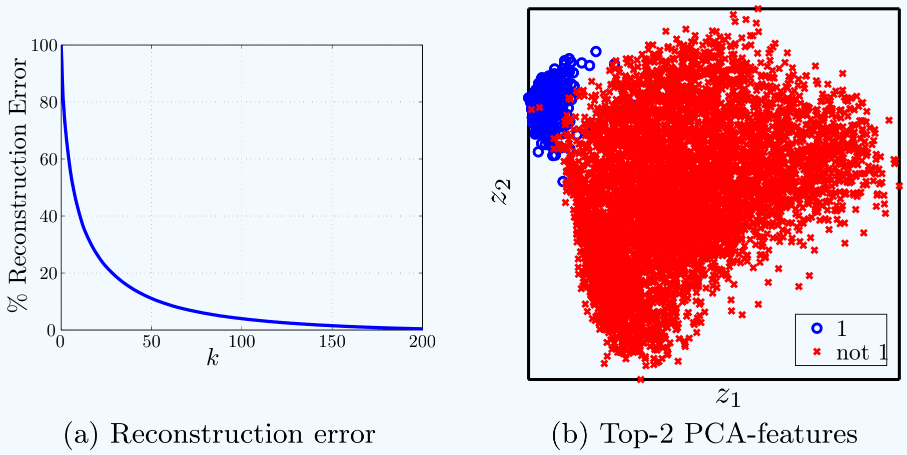


Figure 9.2: PCA on the digits data. (a) Shows how the reconstruction error depends on the number of components k; about 150 features suffice to reconstruct the data almost perfectly. If all principal components are equally important, the reconstruction error would decrease linearly with k, which is not the case here. (b) Shows the two features obtained using the top two principal components. These features look as good as our hand-constructed features of symmetry and intensity from Example 3.5 in Chapter 3.

contain only the top-k singular vectors, and Γ_k the top-k singular values. The reconstructed matrix is

$$\hat{\mathrm{X}} = \mathrm{X}\mathrm{V}_k\mathrm{V}_k^{\mathrm{T}} = \mathrm{U}_k\Gamma_k\mathrm{V}_k^{\mathrm{T}}.$$

By the Eckart-Young theorem, $\hat{\mathrm{X}}$ is the best rank-k approximation to X. Let's look at how the reconstruction error depends on k, the number of features used. To do this, we plot the reconstruction error using k principle components as a percentage of the reconstruction error with zero components. With zero components, the reconstruction error is just $\|\mathrm{X}\|_F^2$. The result is shown in Figure 9.2(a). As can be seen, with just 50 components, the reconstruction error is about 10%. This is a rule of thumb to determine how many components to use: choose k to obtain a reconstruction error of less than 10%. This amounts to treating the bottom 10% of the fluctuations in the data as noise.

Let's reduce the dimensionality to 2 by projecting onto the top two principal components in V_2. The resulting features are shown in Figure 9.2(b). These features can be compared to the features obtained using intensity and symmetry in Example 3.5 on page 106. The features appear to be quite good, and suitable for solving this learning problem. Unlike the size and intensity features which we used in Example 3.5, the biggest advantage of these PCA features is that their construction is fully automated – you don't need to know anything about the digit recognition problem to obtain them. This is also their biggest disadvantage – the features are provably good at reconstructing the input data, but there is no guarantee that they will be useful for solving the learning problem. In practice, a little thought about constructing features, together with such automated methods to then reduce the dimensionality, usually works best.

Finally, suppose you use the feature vector $\mathbf{z}$ in Figure 9.2(b) to build a classifier $\tilde{g}(\mathbf{z})$ to distinguish between the digit 1 and all the other digits. The final hypothesis to be applied to a test input $\mathbf{x}_{\text{test}}$ is

$$g(\mathbf{x}_{\text{test}}) = \tilde{g}(\mathrm{V}_2^{\mathrm{T}}(\mathbf{x}_{\text{test}} - \bar{\mathbf{x}})),$$

where $\tilde{g}$, V_2 and $\bar{\mathbf{x}}$ were constructed from the data: V_2 and $\bar{\mathbf{x}}$ from the data inputs, and $\tilde{g}$ from the targets and PCA-reduced inputs $(\mathrm{Z}, \mathbf{y})$ $\square$

We end this section by justifying the "maximize the variance" intuition that began our discussion of PCA. The next exercise shows that the principal components do indeed represent the high variance directions in the data.

Exercise 9.10

Assume that the data matrix X is centered, and define the covariance matrix $\Sigma = \frac{1}{N}\mathrm{X}^{\mathrm{T}}\mathrm{X}$. Assume all the singular values of X are distinct. (What does this mean for the eigenvalues of Σ?) For a potential principal direction $\mathbf{v}$, we defined $z_n = \mathbf{x}_n^{\mathrm{T}}\mathbf{v}$ and showed that $\text{var}(z_1, \ldots, z_N) = \mathbf{v}^{\mathrm{T}}\Sigma\mathbf{v}$.

(a) Show that the direction which results in the highest variance is $\mathbf{v}_1$, the top right singular vector of X.

(b) Show that we obtain the top-k principal directions, $\mathbf{v}_1, \ldots, \mathbf{v}_k$, by selecting k directions sequentially, each time obtaining the direction with highest variance that is orthogonal to all previously selected directions.

This shows that the top-k principal directions are the directions of highest variance.

(c) If you don't the data matrix X, but only know the covariance matrix Σ, can you obtain the principal directions? If so, how?

9.2.2 Nonlinear Dimension Reduction

Figure 9.3 illustrates one thing that can go wrong with PCA. You can see that the data in Figure 9.3(a) approximately lie on a one-dimensional surface (a curve). However, when we try to reconstruct the data using top-1 PCA, the result is a disaster, shown in Figure 9.3(b). This is because, even though the data lie on a curve, that curve is not linear. PCA can only construct linear features. If the data do not live on a lower dimensional 'linear manifold', then PCA will not work. If you are going to use PCA, here is a checklist that will help you determine whether PCA will work.

1. Do you expect the data to have *linear* structure, for example does the data lie in a linear subspace of lower dimension?

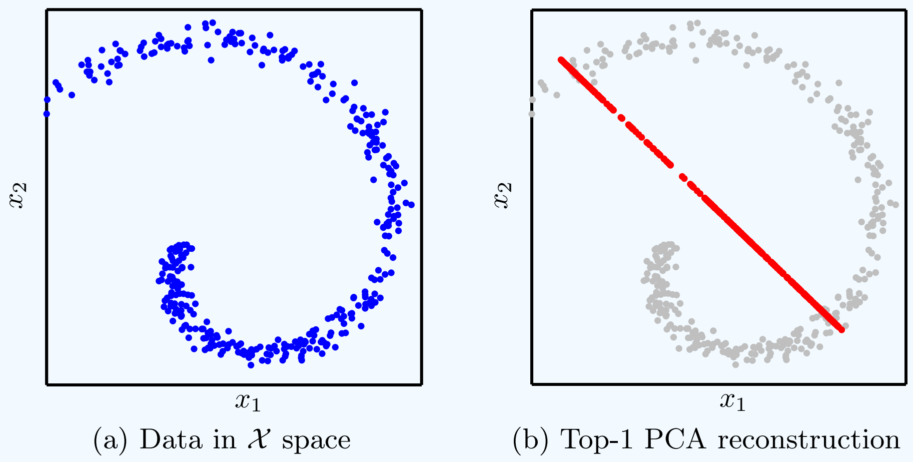


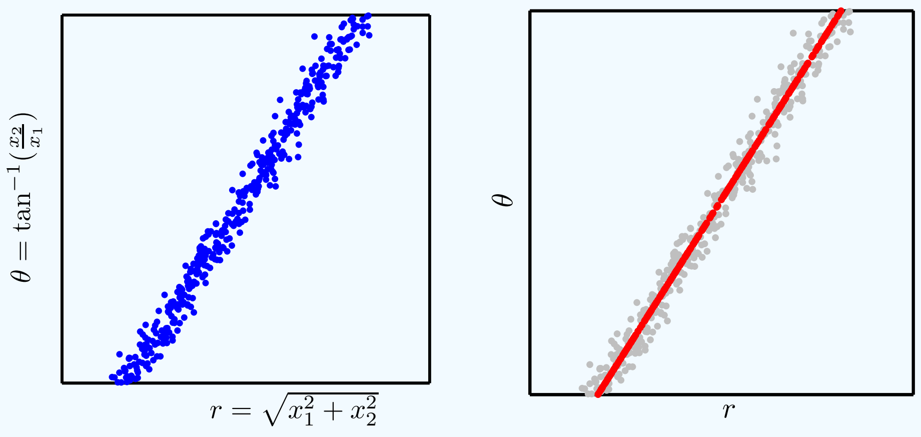
Figure 9.3: (a) The data in $\mathcal{X}$ space does not 'live' in a lower dimensional *linear* manifold. (b) The reconstructed data using top-1 PCA data must lie on a line and therefore cannot accurately represent the original data.

2. Do the bottom principal components contain primarily small random fluctuations that correspond to noise and should be thrown away? The fact that they are small can be determined by looking at the reconstruction error. The fact that they are noise is not much more than a guess.
3. Does the target function f depend primarily on the top principal components, or are the small fluctuations in the bottom principal components key in determining the value of f? If the latter, then PCA will not help the machine learning task. In practice, it is difficult to determine whether this is true (without snooping ☺). A validation method can help determine whether to use PCA-dimension-reduction or not. Usually, throwing away the lowest principal components does not throw away significant information related to the target function, and what little it does throw away is made up for in the reduced generalization error bar because of the lower dimension.

Clearly, PCA will not work for our data in Figure 9.3. However, we are not dead yet. We have an ace up our sleeve, namely the all-powerful nonlinear transform. Looking at the data in Figure 9.3 suggests that the angular coordinate is important. So, lets consider a transform to the nonlinear feature space defined by polar coordinates.

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \xrightarrow{\Phi} \begin{bmatrix} r \\ \theta \end{bmatrix} = \begin{bmatrix} \sqrt{x_1^2 + x_2^2} \\ \tan^{-1}(\frac{x_2}{x_1}) \end{bmatrix}$$

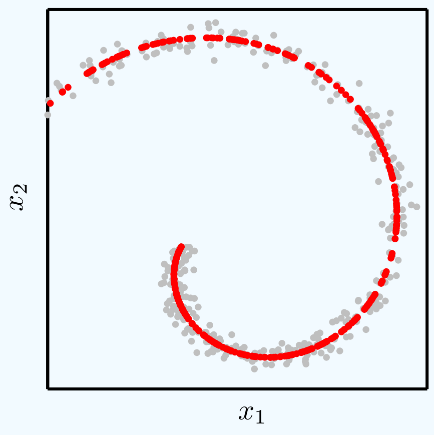
The data using polar-coordinates is shown in Figure 9.4(a). In this space, the data clearly lie on a linear subspace, appropriate for PCA. The top-1 PCA reconstructed data (in the nonlinear feature space) is shown in Figure 9.4(b).



(a) Transformed data in $\mathcal{Z}$ space (b) Reconstruction in $\mathcal{Z}$ space

Figure 9.4: PCA in a nonlinear feature space. (a) The transformed data in the $\mathcal{Z}$ space are approximately on a linear manifold. (b) Shows nonlinear reconstruction of the data in the nonlinear feature space.

We can obtain the reconstructed data in the original $\mathcal{X}$ space by transforming the red reconstructed points in Figure 9.4(b) back to $\mathcal{X}$ space, as shown below.



Exercise 9.11

Using the feature transform $\Phi : \left[\begin{smallmatrix} x_1 \\ x_2 \end{smallmatrix}\right] \mapsto \left[\begin{smallmatrix} x_1 \\ x_2 \\ x_1+x_2 \end{smallmatrix}\right]$, you have run top-1 PCA on your data $\mathbf{z}_1, \ldots, \mathbf{z}_n$ in $\mathcal{Z}$ space to obtain $\mathrm{V}_1 = \left[\begin{smallmatrix} 0 \\ 0 \\ 1 \end{smallmatrix}\right]$ and $\bar{\mathbf{z}} = \mathbf{0}$.

For the test point $\mathbf{x} = \left[\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right]$, compute $\mathbf{z}, \hat{\mathbf{z}}, \hat{\mathbf{x}}$.

($\mathbf{z}$ is the test point in $\mathcal{Z}$ space; $\hat{\mathbf{z}}$ is the reconstructed test point in $\mathcal{Z}$ space using top-1 PCA; $\hat{\mathbf{x}}$ is the reconstructed test point in $\mathcal{X}$ space.)

Exercise 9.11 illustrates that you may not always be able to obtain the reconstructed data in your original $\mathcal{X}$ space. For our spiral example, we can obtain the reconstructed data in the $\mathcal{X}$ space because the polar feature transform is *invertible*; invertibility of Φ is essential to reconstruction in $\mathcal{X}$ space. In general, you may want to use a feature transform that is not invertible, for example the 2nd-order polynomial transform. In this case, you will not be able to reconstruct your data in the $\mathcal{X}$ space, but that need not prevent you from using PCA with the nonlinear feature transform, if your goal is prediction. You can transform to your $\mathcal{Z}$ space, run PCA in the $\mathcal{Z}$ space, discard the bottom principal components (dimension reduction), and run your learning algorithm using the top principal components. Finally, to classify a new test point, you first transform it, and then classify in the $\mathcal{Z}$ space. The entire work-flow is summarized in the following algorithm.

PCA with Nonlinear Feature Transform:

```
Inputs: The data $\mathrm{X}, \mathbf{y}$; $k \geq 1$; and, transform $\Phi$.
  1: Transform the data: $\mathbf{z}_n = \Phi(\mathbf{x}_n)$.
  2: Center the data: $\mathbf{z}_n \leftarrow \mathbf{z}_n - \bar{\mathbf{z}}$ where $\bar{\mathbf{z}} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{z}_n$.
  3: Obtain the centered data matrix Z whose rows are $\mathbf{z}_n$.
  4: Compute the SVD of Z: $[\mathrm{U}, \Gamma, \mathrm{V}] = \mathsf{svd}(\mathrm{Z})$.
  5: Let $\mathrm{V}_k = [\mathbf{v}_1, \ldots, \mathbf{v}_k]$ be the first $k$ columns of V.
  6: Construct the top-k PCA-feature matrix $\mathrm{Z}_k = \mathrm{Z}\mathrm{V}_k$.
  7: Use the data $(\mathrm{Z}_k, \mathbf{y})$ to learn a final hypothesis $\tilde{g}$.
  8: The final hypothesis is
         $g(\mathbf{x}) = \tilde{g}(\mathrm{V}_k^{\mathrm{T}}(\Phi(\mathbf{x}) - \bar{\mathbf{z}}))$
```

There are other approaches to nonlinear dimension reduction. Kernel-PCA is a way to combine PCA with the nonlinear feature transform without visiting the $\mathcal{Z}$ space. Kernel-PCA uses a kernel in the $\mathcal{X}$ space in much the same way that kernels were used to combine the maximum-margin linear separator with the nonlinear transform to get the Kernel-Support Vector Machine. Two other popular approaches are the Neural-Network auto-encoder (which we discussed in the context of Deep Learning with Neural Networks in Section 7.6) and nonlinear principal curves and surfaces which are nonlinear analogues to the PCA-generated linear principal surfaces. With respect to reconstructing the data onto lower dimensional manifolds, there are also nonparametric techniques like the Laplacian Eigenmap or the Locally Linear Embedding (LLE). The problems explore some of these techniques in greater depth.

9.3 Hints And Invariances

Hints are tidbits of information about the target function that you know ahead of time (before you look at any data). You know these tidbits because you know something about the learning problem. Why do we need hints? If we had