20 Kernel Principal Component Analysis

Bernhard Schölkopf, Alexander J. Smola, Klaus-Robert Müller
GMD FIRST
Rudower Chaussee 5, 12489 Berlin, Germany
bs,smola,klaus@first.gmd.de
http://first.gmd.de/~bs,smola,klaus

The idea of implicitly mapping the data into a high-dimensional feature space has been a very fruitful one in the context of SV machines. Indeed, it is this feature which distinguishes them from the Generalized Portrait algorithm which has been known for a long time (Vapnik and Lerner, 1963; Vapnik and Chervonenkis, 1974), and which makes them applicable to complex real-world problems which are not linearly separable. Thus, it was natural to ask the question whether the same idea could prove fruitful in other domains of learning.

The present chapter proposes a new method for performing a nonlinear form of Principal Component Analysis. By the use of Mercer kernels, one can efficiently compute principal components in high-dimensional feature spaces, related to input space by some nonlinear map; for instance the space of all possible 5-pixel products in 16×16 images. We give the derivation of the method and present first experimental results on polynomial feature extraction for pattern recognition.[1]

20.1 Introduction

Principal Component Analysis (PCA) is a powerful technique for extracting structure from possibly high-dimensional data sets. It is readily performed by solving an eigenvalue problem, or by using iterative algorithms which estimate principal components. For reviews of the existing literature, see Jolliffe (1986); Diamantaras and Kung (1996); some of the classical papers are due to Pearson

1. This chapter is an extended version of a chapter from (Schölkopf, 1997), and an article published in *Neural Computation*, Vol. 10, Issue 5, pp. 1299 – 1319, 1998, The MIT Press (Schölkopf et al., 1998e) (First version: (Schölkopf et al., 1996b)).

(1901); Hotelling (1933); Karhunen (1946). PCA is an orthogonal transformation of the coordinate system in which we describe our data. The new coordinate values by which we represent the data are called *principal components*. It is often the case that a small number of principal components is sufficient to account for most of the structure in the data. These are sometimes called *factors* or *latent variables* of the data.

The present work studies PCA in the case where we are not interested in principal components in input space, but rather in principal components of variables, or *features*, which are nonlinearly related to the input variables. Among these are for instance variables obtained by taking arbitrary higher-order correlations between input variables. In the case of image analysis, this amounts to finding principal components in the space of products of input pixels.

To this end, we are computing dot products in feature space by means of kernel functions in input space (cf. chapter 1). Given *any* algorithm which can be expressed solely in terms of dot products, i.e. without explicit usage of the variables themselves, this kernel method enables us to construct different nonlinear versions of it. Even though this general fact was known (Burges, private communication), the machine learning community has made little use of it, the exception being Vapnik's Support Vector machines. In this chapter, we give an example of applying this method in the domain of unsupervised learning, to obtain a nonlinear form of PCA.

In the next section, we will first review the standard PCA algorithm. In order to be able to generalize it to the nonlinear case, we formulate it in a way which uses exclusively dot products. Using kernel representations of dot products (chapter 1), section 20.3 presents a kernel-based algorithm for nonlinear PCA and explains some of the differences to previous generalizations of PCA. First experimental results on kernel-based feature extraction for pattern recognition are given in section 20.4. We conclude with a discussion (section 20.5). Some technical material that is not essential for the main thread of the argument has been relegated to the Appendix.

20.2 Principal Component Analysis in Feature Spaces

Covariance Matrix

Given a set of centered observations $\mathbf{x}_k \in \mathbb{R}^N$, $k = 1, \ldots, M$, $\sum_{k=1}^{M} \mathbf{x}_k = 0$, PCA diagonalizes the covariance matrix[2]

$$C = \frac{1}{M} \sum_{j=1}^{M} \mathbf{x}_j \mathbf{x}_j^\top . \tag{20.1}$$

2. More precisely, the covariance matrix is defined as the expectation of $\mathbf{x}\mathbf{x}^\top$; for convenience, we shall use the same term to refer to the estimate (20.1) of the covariance matrix from a finite sample.

To do this, one has to solve the eigenvalue equation

$$\lambda \mathbf{v} = C\mathbf{v} \tag{20.2}$$

for eigenvalues $\lambda \geq 0$ and eigenvectors $\mathbf{v} \in \mathbb{R}^N \backslash \{0\}$. As

$$\lambda \mathbf{v} = C\mathbf{v} = \frac{1}{M} \sum_{j=1}^{M} (\mathbf{x}_j \cdot \mathbf{v}) \mathbf{x}_j, \tag{20.3}$$

all solutions $\mathbf{v}$ with $\lambda \neq 0$ must lie in the span of $\mathbf{x}_1 \ldots \mathbf{x}_M$, hence (20.2) in that case is equivalent to

$$\lambda (\mathbf{x}_k \cdot \mathbf{v}) = (\mathbf{x}_k \cdot C\mathbf{v}) \text{ for all } k = 1, \ldots, M. \tag{20.4}$$

In the remainder of this section, we describe the same computation in another dot product space F, which is related to the input space by a possibly nonlinear map

$$\Phi : \mathbb{R}^N \to F, \quad \mathbf{x} \mapsto \mathbf{X}. \tag{20.5}$$

Feature Space

Note that the *feature space* F could have an arbitrarily large, possibly infinite, dimensionality. Here and in the following, upper case characters are used for elements of F, while lower case characters denote elements of $\mathbb{R}^N$.

Again, we assume that we are dealing with centered data, $\sum_{k=1}^{M} \Phi(\mathbf{x}_k) = 0$ — we shall return to this point later. In F, the covariance matrix takes the form

$$\bar{C} = \frac{1}{M} \sum_{j=1}^{M} \Phi(\mathbf{x}_j) \Phi(\mathbf{x}_j)^\top. \tag{20.6}$$

Note that if F is infinite-dimensional, we think of $\Phi(\mathbf{x}_j)\Phi(\mathbf{x}_j)^\top$ as a linear operator on F, mapping

$$\mathbf{X} \mapsto \Phi(\mathbf{x}_j)(\Phi(\mathbf{x}_j) \cdot \mathbf{X}). \tag{20.7}$$

We now have to find eigenvalues $\lambda \geq 0$ and eigenvectors $\mathbf{V} \in F \backslash \{0\}$ satisfying

$$\lambda \mathbf{V} = \bar{C}\mathbf{V}. \tag{20.8}$$

Again, all solutions $\mathbf{V}$ with $\lambda \neq 0$ lie in the span of $\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_M)$. For us, this has two useful consequences: first, we may instead consider the set of equations

$$\lambda (\Phi(\mathbf{x}_k) \cdot \mathbf{V}) = (\Phi(\mathbf{x}_k) \cdot \bar{C}\mathbf{V}) \text{ for all } k = 1, \ldots, M, \tag{20.9}$$

and second, there exist coefficients α_i $(i = 1, \ldots, M)$ such that

$$\mathbf{V} = \sum_{i=1}^{M} \alpha_i \Phi(\mathbf{x}_i). \tag{20.10}$$

Combining (20.9) and (20.10), we get

$$\lambda \sum_{i=1}^{M} \alpha_i (\Phi(\mathbf{x}_k) \cdot \Phi(\mathbf{x}_i)) = \frac{1}{M} \sum_{i=1}^{M} \alpha_i \left(\Phi(\mathbf{x}_k) \cdot \sum_{j=1}^{M} \Phi(\mathbf{x}_j)(\Phi(\mathbf{x}_j) \cdot \Phi(\mathbf{x}_i)) \right) \tag{20.11}$$

for all $k = 1, \ldots, M$. Defining an $M \times M$ matrix K by

$$K_{ij} := (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)), \tag{20.12}$$

this reads

$$M\lambda K\boldsymbol{\alpha} = K^2\boldsymbol{\alpha}, \tag{20.13}$$

where $\boldsymbol{\alpha}$ denotes the column vector with entries $\alpha_1, \ldots, \alpha_M$. To find solutions of (20.13), we solve the eigenvalue problem

Eigenvalue Problem in F

$$M\lambda\boldsymbol{\alpha} = K\boldsymbol{\alpha} \tag{20.14}$$

for nonzero eigenvalues. In the Appendix, we show that this gives us all solutions of (20.13) which are of interest for us.

Let $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_M$ denote the eigenvalues of K (i.e. the solutions $M\lambda$ of (20.14)), and $\boldsymbol{\alpha}^1, \ldots, \boldsymbol{\alpha}^M$ the corresponding complete set of eigenvectors, with λ_p being the first nonzero eigenvalue (assuming that Φ is not identically 0). We normalize $\boldsymbol{\alpha}^p, \ldots, \boldsymbol{\alpha}^M$ by requiring that the corresponding vectors in F be normalized, i.e.

$$(\mathbf{V}^k \cdot \mathbf{V}^k) = 1 \text{ for all } k = p, \ldots, M. \tag{20.15}$$

By virtue of (20.10) and (20.14), this translates into a normalization condition for $\boldsymbol{\alpha}^p, \ldots, \boldsymbol{\alpha}^M$:

$$\begin{aligned} 1 &= \sum_{i,j=1}^{M} \alpha_i^k \alpha_j^k (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)) = \sum_{i,j=1}^{M} \alpha_i^k \alpha_j^k K_{ij} \\ &= (\boldsymbol{\alpha}^k \cdot K\boldsymbol{\alpha}^k) = \lambda_k (\boldsymbol{\alpha}^k \cdot \boldsymbol{\alpha}^k) \end{aligned} \tag{20.16}$$

For the purpose of principal component extraction, we need to compute projections onto the eigenvectors $\mathbf{V}^k$ in F ($k = p, \ldots, M$). Let $\mathbf{x}$ be a test point, with an image $\Phi(\mathbf{x})$ in F, then

$$(\mathbf{V}^k \cdot \Phi(\mathbf{x})) = \sum_{i=1}^{M} \alpha_i^k (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x})) \tag{20.17}$$

may be called its nonlinear principal components corresponding to Φ.

In summary, the following steps were necessary to compute the principal components: first, compute the matrix K, second, compute its eigenvectors and normalize them in F; third, compute projections of a test point onto the eigenvectors.[3]

For the sake of simplicity, we have above made the assumption that the observations are centered. This is easy to achieve in input space, but more difficult in

3. Note that in our derivation we could have used the known result (e.g. Kirby and Sirovich, 1990) that PCA can be carried out on the dot product matrix $(\mathbf{x}_i \cdot \mathbf{x}_j)_{ij}$ instead of (20.1), however, for the sake of clarity and extendability (in the Appendix, we shall consider the question how to center the data in F), we gave a detailed derivation.

F, as we cannot explicitly compute the mean of the mapped observations in F. There is, however, a way to do it, and this leads to slightly modified equations for kernel-based PCA (see Appendix).

To conclude this section, note that Φ can be an arbitrary nonlinear map into the possibly high-dimensional space F, for instance, the space of all d-th order monomials in the entries of an input vector. In that case, we need to compute dot products of input vectors mapped by Φ, with a possibly prohibitive computational cost. The solution to this problem, however, is to use kernel functions (1.20) — we *exclusively* need to compute dot products between mapped patterns (in (20.12) and (20.17)); *we never need the mapped patterns explicitly.* Therefore, we can use the kernels described in chapter 1. The particular kernel used then implicitly determines the space F of all possible features. The proposed algorithm, on the other hand, is a mechanism for *selecting* features in F.

20.3 Kernel Principal Component Analysis

The Algorithm. To perform kernel-based PCA (figure 20.1), henceforth referred to as *kernel PCA*, the following steps have to be carried out: first, we compute the matrix $K_{ij} = (k(\mathbf{x}_i, \mathbf{x}_j))_{ij}$. Next, we solve (20.14) by diagonalizing K, and normalize the eigenvector expansion coefficients α^n by requiring $\lambda_n(\boldsymbol{\alpha}^n \cdot \boldsymbol{\alpha}^n) = 1$. To extract the principal components (corresponding to the kernel k) of a test point $\mathbf{x}$, we then compute projections onto the eigenvectors by (cf. Eq. (20.17), figure 20.2),

Feature Extraction

$$(\mathbf{V}^n \cdot \Phi(\mathbf{x})) = \sum_{i=1}^{M} \alpha_i^n k(\mathbf{x}_i, \mathbf{x}). \tag{20.18}$$

If we use a kernel satisfying Mercer's conditions (Proposition 1.1), we know that this procedure exactly corresponds to standard PCA in some high-dimensional feature space, except that we do not need to perform expensive computations in that space.

Properties of Kernel-PCA. For Mercer kernels, we know that we are in fact doing a standard PCA in F. Consequently, all mathematical and statistical properties of PCA (cf. Jolliffe (1986); Diamantaras and Kung (1996)) carry over to kernel PCA, with the modifications that they become statements about a set of points $\Phi(\mathbf{x}_i), i = 1, \ldots, M$, in F rather than in $\mathbb{R}^N$. In F, we can thus assert that PCA is the orthogonal basis transformation with the following properties (assuming that the eigenvectors are sorted in descending order of the eigenvalue size):

- the first q ($q \in \{1, \ldots, M\}$) principal components, i.e. projections on eigenvectors, carry more variance than any other q orthogonal directions
- the mean-squared approximation error in representing the observations by the first q principal components is minimal[4]

4. To see this, in the simple case where the data $\mathbf{z}_1, \ldots, \mathbf{z}_\ell$ are centered, we consider an

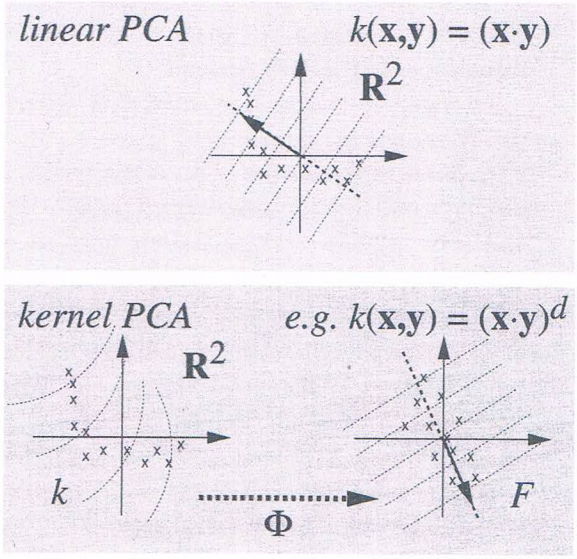

Figure 20.1 The basic idea of kernel PCA. In some high-dimensional feature space F (bottom right), we are performing linear PCA, just as a PCA in input space (top). Since F is nonlinearly related to input space (via Φ), the contour lines of constant projections onto the principal eigenvector (drawn as an arrow) become *nonlinear* in input space. Note that we cannot draw a preimage of the eigenvector in input space, as it may not even exist. Crucial to kernel PCA is the fact that there is no need to perform the map into F: all necessary computations are carried out by the use of a kernel function k in input space (here: $\mathbb{R}^2$).

- the principal components are uncorrelated
- the first q principal components have maximal mutual information with respect to the inputs (this holds under Gaussianity assumptions, and thus depends on the particular kernel chosen and on the data)

---

orthogonal basis transformation W, and use the notation P_q for the projector on the first q canonical basis vectors $\{\mathbf{e}_1, \ldots, \mathbf{e}_q\}$. Then the mean squared reconstruction error using q vectors is

$$\begin{aligned} & \tfrac{1}{\ell}\sum_i \|\mathbf{z}_i - W^\top P_q W \mathbf{z}_i\|^2 = \tfrac{1}{\ell}\sum_i \|W\mathbf{z}_i - P_q W\mathbf{z}_i\|^2 = \tfrac{1}{\ell}\sum_i \sum_{j>q} (W\mathbf{z}_i \cdot \mathbf{e}_j)^2 \\ = \; & \tfrac{1}{\ell}\sum_i \sum_{j>q} (\mathbf{z}_i \cdot W^\top \mathbf{e}_j)^2 = \tfrac{1}{\ell}\sum_i \sum_{j>q} (W^\top \mathbf{e}_j \cdot \mathbf{z}_i)(\mathbf{z}_i \cdot W^\top \mathbf{e}_j) = \sum_{j>q} (W^\top \mathbf{e}_j \cdot C W^\top \mathbf{e}_j). \end{aligned}$$

It can easily be seen that the values of this quadratic form (which gives the variances in the directions $W^\top \mathbf{e}_j$) are minimal if the $W^\top \mathbf{e}_j$ are chosen as its (orthogonal) eigenvectors with smallest eigenvalues.

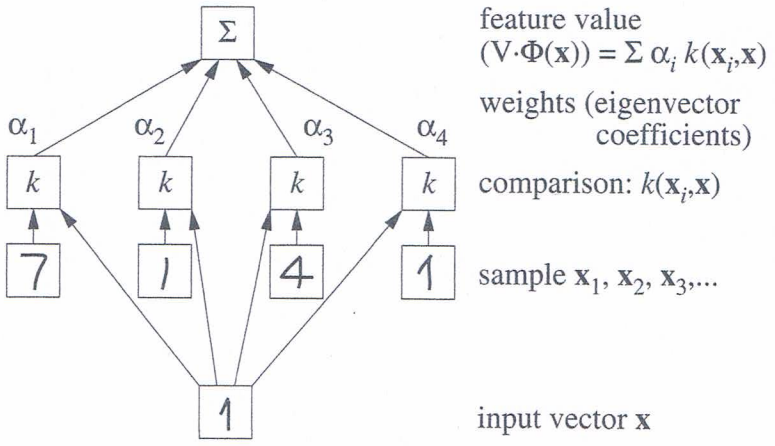

Figure 20.2 Feature extractor constructed by kernel PCA (cf. (20.18)). In the first layer, the input vector is compared to the sample via a kernel function, chosen a priori (e.g. polynomial, Gaussian, or sigmoid). The outputs are then linearly combined using weights which are found by solving an eigenvector problem. As shown in the text, the depicted network's function can be thought of as the projection onto an eigenvector of a covariance matrix in a high-dimensional feature space. As a function on input space, it is nonlinear.

To translate these properties of PCA in F into statements about the data in input space, they need to be investigated for specific choices of a kernels.

Invariance of Polynomial Kernels

We conclude this section with a characterization of kernel PCA with polynomial kernels. In section 1.3, it was explained how using polynomial kernels $(\mathbf{x} \cdot \mathbf{y})^d$ corresponds to mapping into a feature space whose dimensions are spanned by all possible d-th order monomials in input coordinates. The different dimensions are scaled with the square root of the number of ordered products of the respective d pixels (e.g. $\sqrt{2}$ in (1.22)). These scaling factors precisely ensure invariance of kernel PCA under the group of all orthogonal transformations (rotations and mirroring operations). This is a desirable property: it ensures that the features extracted do not depend on which orthonormal coordinate system we use for representing our input data.

Theorem 20.1 (Invariance of Polynomial Kernels)
Up to a scaling factor, kernel PCA with $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^d$ is the only PCA in a space of all monomials of degree d which is invariant under orthogonal transformations of input space.

This means that even if we *could* compute all monomials of degree p for the data at hand and perform PCA on the monomials, with the additional requirement of not implying any preferred directions, we would obtain multiples of the results generated by kernel PCA.

Cum grano salis, the theorem applies to all methods which are based on Mercer kernels. In the context of SV machines, it states that the estimated function is the same if the SV optimization is performed on data which is described in a rotated coordinate system.

The proof is given in the appendix.

Computational Complexity. A fifth order polynomial kernel on a 256-dimensional input space yields a 10^{10}-dimensional feature space. For two reasons, kernel PCA can deal with this huge dimensionality. First, as pointed out in Sect. 20.2 we do not need to look for eigenvectors in the full space F, but just in the subspace spanned by the images of our observations $\mathbf{x}_k$ in F. Second, we do not need to compute dot products explicitly between vectors in F (which can be impossible in practice, even if the vectors live in a lower-dimensional subspace), as we are using kernel functions. Kernel PCA thus is computationally comparable to a linear PCA on ℓ observations with an $\ell \times \ell$ dot product matrix. If k is easy to compute, as for polynomial kernels, e.g., the computational complexity is hardly changed by the fact that we need to evaluate kernel functions rather than just dot products. Furthermore, in the case where we need to use a large number ℓ of observations, we may want to work with an algorithm for computing only the largest eigenvalues, as for instance the power method with deflation (for a discussion, see Diamantaras and Kung (1996)). In addition, we can consider using an estimate of the matrix K, computed from a subset of $M < \ell$ examples, while still extracting principal components from all ℓ examples (this approach was chosen in some of our experiments described below). The situation can be different for principal component extraction. There, we have to evaluate the kernel function M times for each extracted principal component (20.18), rather than just evaluating one dot product as for a linear PCA. Of course, if the dimensionality of F is 10^{10}, this is still vastly faster than linear principal component extraction in F. Still, in some cases, e.g. if we were to extract principal components as a preprocessing step for classification, we might want to speed up things. This can be carried out by the reduced set technique (Burges, 1996; Burges and Schölkopf, 1997) used in the context of Support Vector machines. In the present setting, we approximate each eigenvector

Reduced Set

$$\mathbf{V} = \sum_{i=1}^{\ell} \alpha_i \Phi(\mathbf{x}_i) \tag{20.19}$$

(Eq. (20.10)) by another vector

$$\tilde{\mathbf{V}} = \sum_{j=1}^{m} \beta_j \Phi(\mathbf{z}_j), \tag{20.20}$$

where $m < \ell$ is chosen a priori according to the desired speedup, and $\mathbf{z}_j \in \mathbb{R}^N$, $j = 1, \ldots, m$.

This is done by minimizing the squared difference

$$\rho = \|\mathbf{V} - \tilde{\mathbf{V}}\|^2. \tag{20.21}$$

This can be carried out without explicitly dealing with the possibly high-dimensional space F. Since

$$\rho = \|\mathbf{V}\|^2 + \sum_{i,j=1}^{m} \beta_i \beta_j k(\mathbf{z}_i, \mathbf{z}_j) - 2 \sum_{i=1}^{\ell} \sum_{j=1}^{m} \alpha_i \beta_j k(\mathbf{x}_i, \mathbf{z}_j), \tag{20.22}$$

the gradient of ρ with respect to the β_j and the $\mathbf{z}_j$ is readily expressed in terms of the kernel function, thus ρ can be minimized by standard gradient methods. For the task of handwritten character recognition, this technique led to a speedup by a factor of 50 at almost no loss in accuracy (Burges and Schölkopf, 1997).

Finally, we add that although kernel principal component extraction is computationally more expensive than its linear counterpart, this additional investment can pay back afterwards. In experiments on classification based on the extracted principal components, we found when we trained on nonlinear features, it was sufficient to use a linear Support Vector machine to construct the decision boundary. Linear Support Vector machines, however, are much faster in classification speed than nonlinear ones. This is due to the fact that for $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})$, the Support Vector decision function (1.32) can be expressed with a single weight vector $\mathbf{w} = \sum_{i=1}^{\ell} y_i \alpha_i \mathbf{x}_i$ as $f(\mathbf{x}) = \operatorname{sgn}((\mathbf{x} \cdot \mathbf{w}) + b)$. Thus the final stage of classification can be done extremely fast; the speed of the principal component extraction phase, on the other hand, and thus the accuracy-speed trade-off of the whole classifier, can be controlled by the number of components which we extract, or by the number m (cf. Eq. (20.20)).

Interpretability and Variable Selection. In PCA, it is sometimes desirable to be able to select specific axes which span the subspace into which one projects in doing principal component extraction. This way, it may for instance be possible to choose variables which are more accessible to interpretation. In the nonlinear case, there is an additional problem: some elements of F do not have preimages in input space. To make this plausible, note that the linear span of the training examples mapped into feature space can have dimensionality up to M (the number of examples). If this exceeds the dimensionality of input space, it is rather unlikely that each vector of the form (20.10) has a preimage (cf. Schölkopf et al., 1998c). To get interpretability, we thus need to find directions in input space (i.e. input variables) whose images under Φ span the PCA subspace in F. This can be done with an approach akin to the one described above: we could parametrize our set of desired input variables and run the minimization of (20.22) only over those parameters. The parameters can be e.g. group parameters which determine the amount of translation, say, starting from a set of images.

Dimensionality Reduction, Feature Extraction, and Reconstruction. Unlike linear PCA, the proposed method allows the extraction of a number of principal components which *can* exceed the input dimensionality. Suppose that the

number of observations M exceeds the input dimensionality N. Linear PCA, even when it is based on the $M \times M$ dot product matrix, can find at most N nonzero eigenvalues — they are identical to the nonzero eigenvalues of the $N \times N$ covariance matrix. In contrast, kernel PCA can find up to M nonzero eigenvalues — a fact that illustrates that it is impossible to perform kernel PCA directly on an $N \times N$ covariance matrix. Even more features could be extracted by using several kernels. Being just a basis transformation, standard PCA allows the reconstruction of the original patterns $\mathbf{x}_i, i = 1, \ldots, \ell$, from a complete set of extracted principal components $(\mathbf{x}_i \cdot \mathbf{v}_j), j = 1, \ldots, \ell$, by expansion in the eigenvector basis. Even from an incomplete set of components, good reconstruction is often possible. In kernel PCA, this is more difficult: we can reconstruct the image of a pattern in F from its nonlinear components; however, if we only have an approximate reconstruction, there is no guarantee that we can find an exact preimage of the reconstruction in input space. In that case, we would have to resort to an approximation method (cf. (20.22)). First results obtained by using this approach are reported in (Schölkopf et al., 1998c). Alternatively, we could use a suitable regression method for estimating the reconstruction mapping from the kernel-based principal components to the inputs.

Comparison to Other Methods for Nonlinear PCA. Starting from some of the properties characterizing PCA (see above), it is possible to develop a number of possible generalizations of linear PCA to the nonlinear case. Alternatively, one may choose an iterative algorithm which adaptively estimates principal components, and make some of its parts nonlinear to extract nonlinear features. Rather than giving a full review of this field here, we briefly describe four approaches, and refer the reader to Diamantaras and Kung (1996) for more details.

Generalizations of PCA

Hebbian Networks. Initiated by the pioneering work of Oja (1982), a number of unsupervised neural-network type algorithms computing principal components have been proposed (e.g. Sanger, 1989). Compared to the standard approach of diagonalizing the covariance matrix, they have advantages for instance in cases where the data are nonstationary. Nonlinear variants of these algorithms are obtained by adding nonlinear activation functions. The algorithms then extract features that the authors have referred to as nonlinear principal components. These approaches, however, do not have the geometrical interpretation of kernel PCA as a standard PCA in a feature space nonlinearly related to input space, and it is thus more difficult to understand what exactly they are extracting. For a discussion of some approaches, see Karhunen and Joutsensalo (1995).

Autoassociative Multi-Layer Perceptrons. Consider a linear perceptron with one hidden layer, which is smaller than the input. If we train it to reproduce the input values as outputs (i.e. use it in autoassociative mode), then the hidden unit activations form a lower-dimensional representation of the data, closely related to PCA (see for instance Diamantaras and Kung (1996)). To generalize to a nonlinear setting, one uses nonlinear activation functions and

additional layers.[5] While this of course can be considered a form of nonlinear PCA, it should be stressed that the resulting network training consists in solving a hard nonlinear optimization problem, with the possibility to get trapped in local minima, and thus with a dependence of the outcome on the starting point of the training. Moreover, in neural network implementations there is often a risk of getting overfitting. Another drawback of neural approaches to nonlinear PCA is that the number of components to be extracted has to be specified in advance. As an aside, note that hyperbolic tangent kernels can be used to extract neural network type nonlinear features using kernel PCA (figure 20.7). The principal components of a test point $\mathbf{x}$ in that case take the form (figure 20.2) $\sum_i \alpha_i^n \tanh(\kappa(\mathbf{x}_i, \mathbf{x}) + \Theta)$.

Principal Curves. An approach with a clear geometric interpretation in input space is the method of principal curves (Hastie and Stuetzle, 1989), which iteratively estimates a curve (or surface) capturing the structure of the data. The data are projected onto (i.e. mapped to the closest point on) a curve, and the algorithm tries to find a curve with the property that each point on the curve is the average of all data points projecting onto it. It can be shown that the only straight lines satisfying the latter are principal components, so principal curves are indeed a generalization of the latter. To compute principal curves, a nonlinear optimization problem has to be solved. The dimensionality of the surface, and thus the number of features to extract, is specified in advance. Some authors (e.g. Ritter et al., 1990) have discussed parallels between the Principal Curve algorithm and self-organizing feature maps (Kohonen, 1982) for dimensionality reduction.

Kernel PCA. Kernel PCA is a nonlinear generalization of PCA in the sense that (a) it is performing PCA in feature spaces of arbitrarily large (possibly infinite) dimensionality, and (b) if we use the kernel $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})$, we recover the original PCA algorithm. Compared to the above approaches, kernel PCA has the main advantage that no nonlinear optimization is involved — it is essentially linear algebra, as simple as standard PCA. In addition, we need not specify the number of components that we want to extract in advance. Compared to neural approaches, kernel PCA could be disadvantageous if we need to process a very large number of observations, as this results in a large matrix K. Compared to principal curves, kernel PCA is so far harder to interpret in input space; however, at least for polynomial kernels, it has a very clear interpretation in terms of higher-order features.

---

5. Simply using nonlinear activation functions in the hidden layer would not suffice: already the linear activation functions lead to the best approximation of the data (given the number of hidden nodes), so for the nonlinearities to have an effect on the components, the architecture needs to be changed (see e.g. Diamantaras and Kung (1996)).

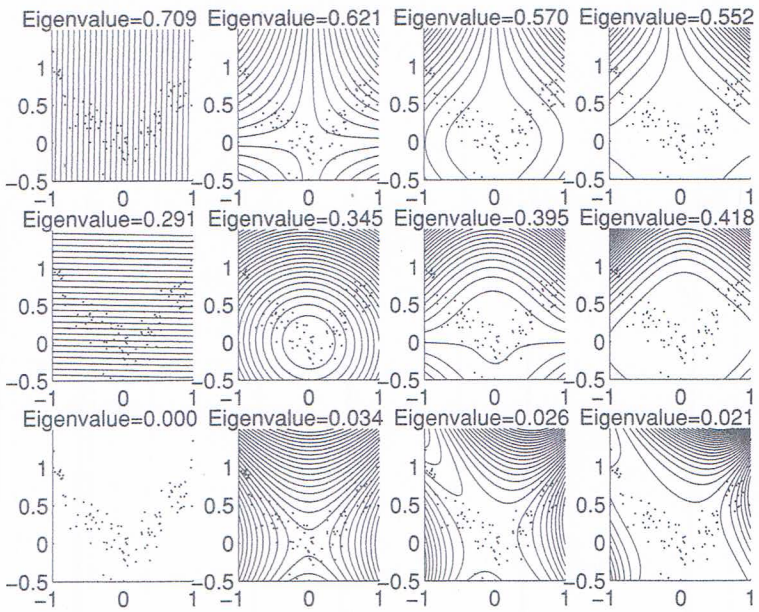

Figure 20.3 Two-dimensional toy example, with data generated in the following way: x-values have uniform distribution in $[-1, 1]$, y-values are generated from $y_i = x_i^2 + v$, were v is normal noise with standard deviation 0.2. From left to right, the polynomial degree in the kernel (1.21) increases from 1 to 4; from top to bottom, the first 3 eigenvectors are shown, in order of decreasing eigenvalue size. The figures contain lines of constant principal component value (contour lines); in the linear case, these are orthogonal to the eigenvectors. We did not draw the eigenvectors, as in the general case, they live in a higher-dimensional feature space.

20.4 Feature Extraction Experiments

In this section, we present a set of experiments where we used kernel PCA (in the form taking into account centering in F, as described in the Appendix) to extract principal components. First, we shall take a look at a simple toy example; following that, we describe real-world experiments where we assess the utility of the extracted principal components by classification tasks.

Toy Examples. To provide some insight into how PCA in F behaves in input space, we show a set of experiments with an artificial 2-D data set, using polynomial kernels (cf. (1.21)) of degree 1 through 4 (see figure 20.3). Linear PCA (on the left) only leads to 2 nonzero eigenvalues, as the input dimensionality is 2. In contrast, nonlinear PCA allows the extraction of further components. In the figure, note that nonlinear PCA produces contour lines of constant feature value which reflect the structure in the data better than in linear PCA. In all cases, the first principal

component varies monotonically along the parabola which underlies the data. In the nonlinear cases, also the second and the third components show behaviour which is similar for different polynomial degrees. The third component, which comes with small eigenvalues (rescaled to sum to 1), seems to pick up the variance caused by the noise, as can be nicely seen in the case of degree 2. Dropping this component would thus amount to noise reduction.

In figure 20.3, it can be observed that for larger polynomial degrees, the principal component extraction functions become increasingly flat around the origin. Thus, different data points not too far from the origin would only differ slightly in the value of their principal components. To understand this, consider the following example: suppose we have two data points

$$\mathbf{x}_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad \mathbf{x}_2 = \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \tag{20.23}$$

and a kernel $k(\mathbf{x}, \mathbf{y}) := (\mathbf{x} \cdot \mathbf{y})^2$. Then the differences between the entries of $\mathbf{x}_1$ and $\mathbf{x}_2$ get scaled up by the kernel, namely $k(\mathbf{x}_1, \mathbf{x}_1) = 1$, but $k(\mathbf{x}_2, \mathbf{x}_2) = 16$. We can compensate for this by rescaling the individual entries of each vector $\mathbf{x}_i$ by

$$(\mathbf{x}_i)_k \mapsto \operatorname{sign}\left((\mathbf{x}_i)_k\right) \cdot |(\mathbf{x}_i)_k|^{\frac{1}{2}}. \tag{20.24}$$

Indeed, figure 20.4, taken from Schölkopf et al. (1997a), shows that when the data are preprocessed according to (20.24) (where higher degrees are treated correspondingly), the first principal component extractors do hardly depend on the degree anymore, as long as it is larger than 1. If necessary, we can thus use (20.24) to preprocess our data. Note, however, that the above scaling problem is irrelevant for the character and object databases to be considered below: there, most entries of the patterns are ± 1.

Further toy examples, using radial basis function kernels (1.31) and neural network type sigmoid kernels (1.30), are shown in figures 20.5 – 20.8. Matlab code for carrying out kernel PCA can be obtained from http://svm.first.gmd.de.

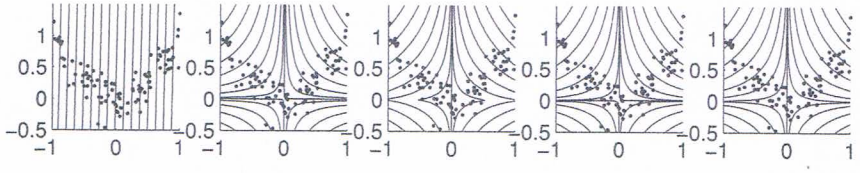

Figure 20.4 PCA with kernel (1.21), degrees $d = 1, \ldots, 5$. 100 points $((x_i)_1, (x_i)_2)$ were generated from $(x_i)_2 = (x_i)_1^2 + \upsilon$ (where υ is Gaussian noise with standard deviation 0.2); all $(x_i)_j$ were rescaled according to $(x_i)_j \mapsto \operatorname{sgn}((x_i)_j) \cdot |(x_i)_j|^{1/d}$. Displayed are contour lines of constant value of the first principal component. Nonlinear kernels ($d > 1$) extract features which nicely increase along the direction of main variance in the data; linear PCA ($d = 1$) does its best in that respect, too, but it is limited to straight directions.

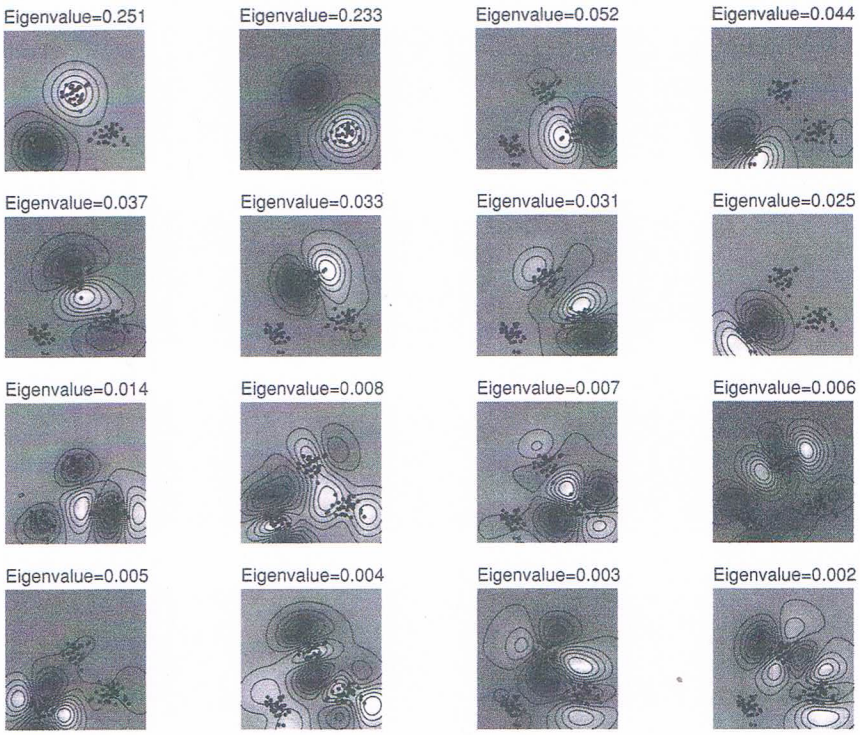

Figure 20.5 Two-dimensional toy example with three data clusters (Gaussians with standard deviation 0.1, depicted region: $[-1,1] \times [-0.5,1]$): first 16 nonlinear principal components extracted with $k(\mathbf{x},\mathbf{y}) = \exp\left(-\|\mathbf{x}-\mathbf{y}\|^2/0.1\right)$. Note that the first 2 principal component (top left), with the largest eigenvalues, nicely separate the three clusters. The components 3 – 5 split up the clusters into halves. Similarly, the components 6 – 8 split them again, in a way orthogonal to the above splits. The higher components are more difficult to describe. They look for finer structure in the data set, identifying higher-order moments.

Object Recognition. In this set of experiments, we used the MPI chair database with 89 training views of 25 different chair models (figure 20.9, Blanz et al. (1996); Schölkopf (1997)). We computed the matrix K from all 2225 training examples, and used polynomial kernel PCA to extract nonlinear principal components from the training and test set. To assess the utility of the components, we trained a soft margin hyperplane classifier on the classification task. This is a special case of Support Vector machines, using the standard dot product as a kernel function. Table 20.1 summarizes our findings: in all cases, nonlinear components as extracted by polynomial kernels (Eq. (1.21) with $d > 1$) led to classification accuracies superior to standard PCA. Specifically, the nonlinear components afforded top test performances between 2% and 4% error; in the linear case we obtained 17%.

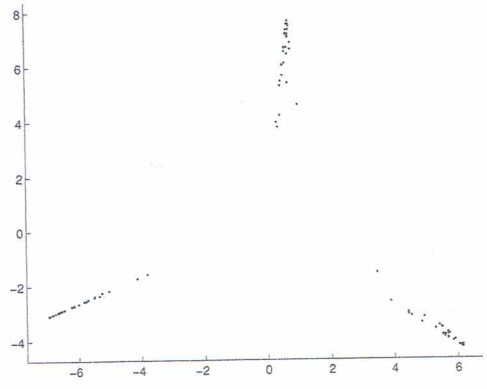

Figure 20.6 A plot of the data representation given by the first two principal components of figure 20.5. The clusters of figure 20.5 end up roughly on separated lines (the left, right, and top region corresponds to the clusters left, top, and right, respectively). Note that already the first component (the horizontal axis) separates the clusters — this cannot be done using linear PCA.

# of components	Test Error Rate for degree						
	1	2	3	4	5	6	7
64	23.0	21.0	17.6	16.8	16.5	16.7	16.6
128	17.6	9.9	7.9	7.1	6.2	6.0	5.8
256	16.8	6.0	4.4	3.8	3.4	3.2	3.3
512	n.a.	4.4	3.6	3.9	2.8	2.8	2.6
1024	n.a.	4.1	3.0	2.8	2.6	2.6	2.4
2048	n.a.	4.1	2.9	2.6	2.5	2.4	2.2

Table 20.1 Test error rates on the MPI chair database for linear Support Vector machines trained on nonlinear principal components extracted by PCA with polynomial kernel (1.21), for degrees 1 through 7. In the case of degree 1, we are doing standard PCA, with the number of nonzero eigenvalues being at most the dimensionality of the space, 256; thus, we can extract at most 256 principal components. The performance for the nonlinear cases (degree > 1) is significantly better than for the linear case, illustrating the utility of the extracted nonlinear components for classification.

Character Recognition. To validate the above results on a widely used pattern recognition benchmark database, we repeated the same experiments on the US postal service database of handwritten digits (figure 20.10, e.g. Simard et al. (1993); LeCun et al. (1989)). This database contains 9298 examples of dimensionality 256; 2007 of them make up the test set. For computational reasons, we decided to use a subset of 3000 training examples for the matrix K. Table 20.2 illustrates two advantages of using nonlinear kernels: first, performance of a linear classifier trained on nonlinear principal components is better than for the same number of linear components; second, the performance for nonlinear components can be further improved by using more components than possible in the linear case. The

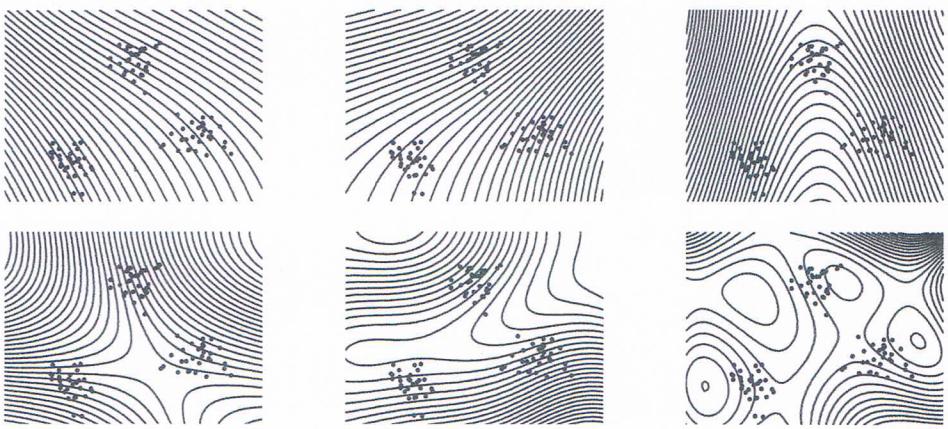

Figure 20.7 Two-dimensional toy example with three data clusters (Gaussians with standard deviation 0.1, depicted region: $[-1,1] \times [-0.5,1]$): first 6 nonlinear principal components extracted with $k(\mathbf{x},\mathbf{y}) = \tanh(2(\mathbf{x}\cdot\mathbf{y})-1)$ (the gain and threshold values were chosen according to the values used in SV machines, cf. Schölkopf et al. (1995)). Note that the first 2 principal components are sufficient to separate the three clusters, and the third and fourth component simultaneously split all clusters into halves.

latter is related to the fact that of course there are many more higher-order features than there are pixels in an image. Regarding the first point, note that extracting a certain number of features in a 10^{10}-dimensional space constitutes a much higher reduction of dimensionality than extracting the same number of features in 256-dimensional input space.

For all numbers of features, the optimal degree of kernels to use is around 4, which is compatible with Support Vector machine results on the same data set (Schölkopf et al., 1995). Moreover, with only one exception, the nonlinear features are superior to their linear counterparts. The resulting error rate for the best of our classifiers (4.0%) is competitive with convolutional 5-layer neural networks (5.0% were reported by LeCun et al. (1989)) and nonlinear Support Vector classifiers (4.0%, Schölkopf et al. (1995)); it is much better than linear classifiers operating directly on the image data (a linear Support Vector machine achieves 8.9%; Schölkopf et al. (1995)). Our results were obtained without using any prior knowledge about symmetries of the problem at hand, explaining why the performance is inferior to Virtual Support Vector classifiers (3.2%, Schölkopf et al. (1996a)), and Tangent Distance Nearest Neighbour classifiers (2.6%, Simard et al. (1993)). We believe that adding e.g. local translation invariance, be it by generating "virtual" translated examples (cf. Schölkopf et al., 1996a) or by choosing a suitable kernel incorporating locality (e.g. as the ones in (Schölkopf et al., 1998d), which led to an error rate of 3.0%), could further improve the results.

USPS Benchmark

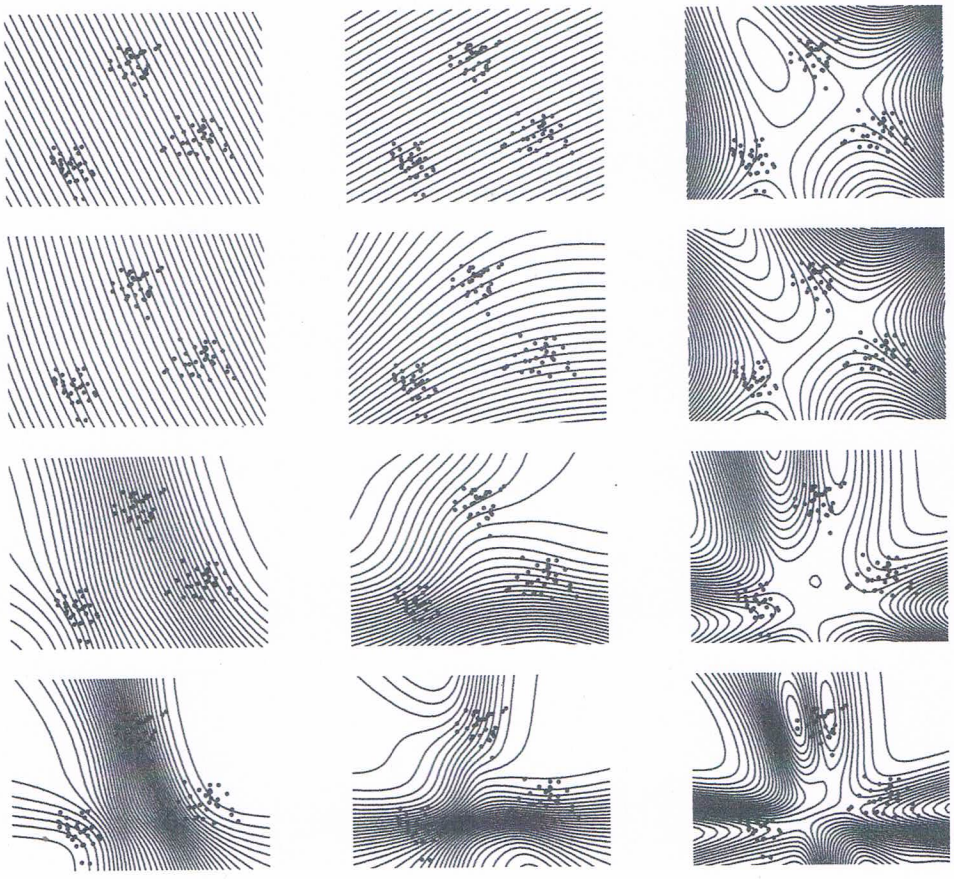

Figure 20.8 A smooth transition from linear PCA to nonlinear PCA is obtained by using hyperbolic tangent kernels $k(\mathbf{x}, \mathbf{y}) = \tanh(\kappa(\mathbf{x} \cdot \mathbf{y}) + 1)$ with varying gain κ: from top to bottom, $\kappa = 0.1, 1, 5, 10$ (data as in the previous figures). For $\kappa = 0.1$, the first two features look like linear PCA features. For large κ, the nonlinear region of the tanh function becomes effective. In that case, kernel PCA can exploit this nonlinearity to allocate the highest feature gradients to regions where there are data points, as can be seen nicely in the case $\kappa = 10$.

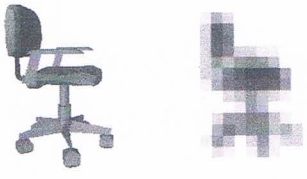

Figure 20.9 *Left*: rendered view of a 3-D model from the MPI chair database; *right*: 16×16 downsampled image, as used in the experiments.

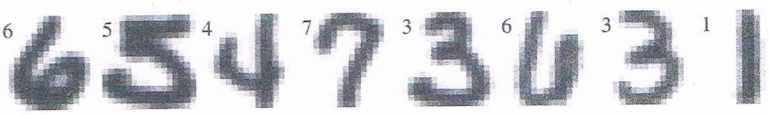

Figure 20.10 The first 8 patterns from the US Postal Service database of handwritten digits, with class labels.

# of components	Test Error Rate for degree						
	1	2	3	4	5	6	7
32	9.6	8.8	8.1	8.5	9.1	9.3	10.8
64	8.8	7.3	6.8	6.7	6.7	7.2	7.5
128	8.6	5.8	5.9	6.1	5.8	6.0	6.8
256	8.7	5.5	5.3	5.2	5.2	5.4	5.4
512	n.a.	4.9	4.6	4.4	5.1	4.6	4.9
1024	n.a.	4.9	4.3	4.4	4.6	4.8	4.6
2048	n.a.	4.9	4.2	4.1	4.0	4.3	4.4

Table 20.2 Test error rates on the USPS handwritten digit database for linear Support Vector machines trained on nonlinear principal components extracted by PCA with kernel (1.21), for degrees 1 through 7. In the case of degree 1, we are doing standard PCA, with the number of nonzero eigenvalues being at most the dimensionality of the space, 256. Clearly, nonlinear principal components afford test error rates which are superior to the linear case (degree 1).

20.5 Discussion

Feature Extraction for Classification. This chapter was devoted to the presentation of a new technique for nonlinear PCA. To develop this technique, we made use of the Mercer kernel method so far only used in supervised learning. Kernel PCA constitutes a mere first step towards exploiting this technique for a large class of algorithms.

In experiments comparing the utility of kernel PCA features for pattern recognition using a linear classifier, we found two advantages of nonlinear kernels: first, nonlinear principal components afforded better recognition rates than corresponding numbers of linear principal components; and second, the performance for nonlinear components can be further improved by using more components than possible in the linear case. We have not yet compared kernel PCA to other techniques for nonlinear feature extraction and dimensionality reduction. We can, however, compare results to other feature extraction methods which have been used in the past by researchers working on the USPS classification problem (cf. section 20.4). Our system of kernel PCA feature extraction plus linear support vector machine for in-

stance performed better than LeNet1 (LeCun et al., 1989). Even though the latter result has been obtained a number of years ago, it should be stressed that LeNet1 provides an architecture which contains a great deal of prior information about the handwritten character classification problem. It uses shared weights to improve transformation invariance, and a hierarchy of feature detectors resembling parts of the human visual system. These feature detectors were for instance used by Bottou and Vapnik (1992) as a preprocessing stage in their experiments in local learning. Note that, in addition, our features were extracted without taking into account that we want to do classification. Clearly, in supervised learning, where we are given a set of labelled observations $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)$, it would seem advisable to make use of the labels not only during the training of the final classifier, but already in the stage of feature extraction.

To conclude this paragraph on feature extraction for classification, we note that a similar approach could be taken in the case of regression estimation, to generalize PCA regression (e.g. Jolliffe, 1986) to the nonlinear case.

Feature Space and the Curse of Dimensionality. We are doing PCA in 10^{10}-dimensional feature spaces, yet getting results in finite time which are comparable to state-of-the-art techniques. In fact, of course, we are not working in the full feature space, but just in a comparably small linear subspace of it, whose dimension equals at most the number of observations. The method automatically chooses this subspace and provides a means of taking advantage of the lower dimensionality — an approach which consisted in explicitly mapping into feature space and then performing PCA would have severe difficulties at this point: even if PCA was done based on an $M \times M$ dot product matrix (M being the sample size) whose diagonalization is tractable, it would still be necessary to evaluate dot products in a 10^{10}-dimensional feature space to compute the *entries* of the matrix in the first place. Kernel-based methods avoid this problem — they do not explicitly compute all dimensions of F (loosely speaking, all possible features), but only work in a relevant subspace of F.

Note, moreover, that we did not get overfitting problems when training the linear SV classifier on the extracted features. The basic idea behind this two-step approach is very similar in spirit to nonlinear SV machines: one maps into a very complex space to be able to approximate a large class of possible decision functions, and then uses a low VC-dimension classifier in that space to control generalization.

Regularization. As kernel PCA belongs to the domain of unsupervised learning, VC-theory methods are not directly applicable to discuss its regularization and generalization properties. It is, however, possible to draw some simple analogies to the standard SV reasoning. The feature extractors (20.18) are linear functions in the feature space F whose regularization properties, as in the SV case, can be characterized by the length of their weight vector. When applied to the training data, the k-th feature extractor generates a set of outputs with variance λ_k. Dividing each weight vector $\boldsymbol{\alpha}^k$ by $\sqrt{\lambda_k}$, we obtain a set of nonlinear feature extractors with unit variance output and the property that out of all such extractors which can be

written in the form (20.18), the first k kernel PCA extractors are those with the shortest weight vectors (subject to the condition that they are orthogonal in F). Therefore, kernel PCA can be considered as a method for extracting potentially interesting functions (with unit variance on the data) that have low capacity. The complexity measure used here is identical to the one used in Gaussian processes (Williams, 1998), i.e. it could be interpreted as a Bayesian prior on the space of functions by setting $p(f) \propto \exp(-\frac{1}{2}\|Pf\|^2)$ where P is the regularization operator corresponding to k (see Smola and Schölkopf (1998b) for details). In this view, the first extractor (cf. (20.18)) $f(\mathbf{x}) = \sum_{i=1}^{M} \alpha_i k(\mathbf{x}_i, \mathbf{x})$ is given by

$$f = \underset{\mathrm{Var}(f)=1}{\operatorname{argmax}} \exp\left(-\frac{1}{2}\|Pf\|^2\right), \tag{20.25}$$

where $\mathrm{Var}(f)$ denotes the (estimate of the) variance of $f(\mathbf{x})$ for $\mathbf{x}$ drawn from the underlying distribution.

Conclusion. Compared to other techniques for nonlinear feature extraction, kernel PCA has the advantages that it does not require nonlinear optimization, but only the solution of an eigenvalue problem, and by the possibility to use different kernels, it comprises a fairly general class of nonlinearities that can be used. Clearly, the last point has yet to be evaluated in practice, however, for the support vector machine, the utility of different kernels has already been established. Different kernels (polynomial, sigmoid, Gaussian) led to fine classification performances (Schölkopf et al., 1995). The general question of how to select the ideal kernel for a given task (i.e. the appropriate feature space), however, is an open problem.

We conclude this chapter with a twofold outlook. The scene has been set for using the kernel method to construct a wide variety of rather general and still feasible nonlinear variants of classical algorithms. It is beyond the scope of the present work to explore all the possibilities, including many distance-based algorithms, in detail. In (Schölkopf et al., 1996b), we have proposed and begun to work out several possibilities, for instance kernel-based independent component analysis and nonlinear forms of k-means clustering. The latter has meanwhile been further developed and implemented by Graepel and Obermayer (1998). Other domains where researchers have recently started to investigate the use of Mercer kernels include Gaussian Processes (Williams, 1998).

Linear PCA is being used in numerous technical and scientific applications, including noise reduction, density estimation, image indexing and retrieval systems, and the analysis of natural image statistics. Kernel PCA can be applied to all domains where traditional PCA has so far been used for feature extraction, and where a nonlinear extension would make sense.

Acknowledgements

AS and BS were supported by grants from the Studienstiftung des deutschen Volkes, and thank V. Vapnik for introducing them to kernel representations of dot products during joint work on support vector machines. AS was supported by a grant of the

DFG (JA 379/71). This work profited from discussions with V. Blanz, L. Bottou, C. Burges, H. Bülthoff, P. Haffner, Y. Le Cun, S. Mika, N. Murata, P. Simard, S. Solla, V. Vapnik, and T. Vetter. We are grateful to V. Blanz, C. Burges, and S. Solla for reading a preliminary version of the manuscript, and to L. Bottou, C. Burges, and C. Cortes for parts of the SV training code.

Appendix

The Eigenvalue Problem in the Space of Expansion Coefficients

Being symmetric, K has an orthonormal basis of eigenvectors $(\boldsymbol{\beta}^i)_i$ with corresponding eigenvalues μ_i, thus for all i, we have $K\boldsymbol{\beta}^i = \mu_i\boldsymbol{\beta}^i$ $(i = 1, \ldots, M)$. To understand the relation between Eq. (20.13) and Eq. (20.14), we proceed as follows: first suppose $\lambda, \boldsymbol{\alpha}$ satisfy (20.13). We may expand $\boldsymbol{\alpha}$ in K's eigenvector basis as $\boldsymbol{\alpha} = \sum_{i=1}^{M} a_i\boldsymbol{\beta}^i$. Eq. (20.13) then reads

$$M\lambda \sum_i a_i\mu_i\boldsymbol{\beta}^i = \sum_i a_i\mu_i^2\boldsymbol{\beta}^i, \tag{20.26}$$

or, equivalently, for all $i = 1, \ldots, M$, $M\lambda a_i\mu_i = a_i\mu_i^2$. This in turn means that for all $i = 1, \ldots, M$,

$$M\lambda = \mu_i \quad or \quad a_i = 0 \quad or \quad \mu_i = 0. \tag{20.27}$$

Note that the above are not exclusive *or*-s. We next assume that $\lambda, \boldsymbol{\alpha}$ satisfy (20.14), to carry out a similar derivation. In that case, we find that Eq. (20.14) is equivalent to

$$M\lambda \sum_i a_i\boldsymbol{\beta}^i = \sum_i a_i\mu_i\boldsymbol{\beta}^i, \tag{20.28}$$

i.e. for all $i = 1, \ldots, M$,

$$M\lambda = \mu_i \quad or \quad a_i = 0. \tag{20.29}$$

Comparing (20.27) and (20.29), we see that all solutions of the latter satisfy the former. However, they do not give its full set of solutions: given a solution of (20.14), we may always add multiples of eigenvectors of K with eigenvalue 0 and still satisfy (20.13), with the same eigenvalue.[6] This means that there exist solutions of Eq. (20.13) which belong to different eigenvalues yet are not orthogonal in the space of the $\boldsymbol{\alpha}^k$. It does, however, not mean that the eigenvectors of $\bar{C}$ in F are not orthogonal. Indeed, note that if $\boldsymbol{\alpha}$ is an eigenvector of K with eigenvalue 0, then the corresponding vector $\sum_i \alpha_i\Phi(\mathbf{x}_i)$ is orthogonal to *all* vectors in the span of the $\Phi(\mathbf{x}_j)$ in F, since $(\Phi(\mathbf{x}_j) \cdot \sum_i \alpha_i\Phi(\mathbf{x}_i)) = (K\boldsymbol{\alpha})_j = 0$ for all j, which means that

6. This observation could be used to change the vectors $\boldsymbol{\alpha}$ of the solution, e.g. to make them maximally sparse, without changing the solution.

$\sum_i \alpha_i \Phi(\mathbf{x}_i) = 0$. Thus, the above difference between the solutions of (20.13) and (20.14) is irrelevant, since we are interested in vectors in F rather than vectors in the space of the expansion coefficients of (20.10). We thus only need to diagonalize K to find all relevant solutions of (20.13).

Centering in High-Dimensional Space

Given any Φ and any set of observations $\mathbf{x}_1, \ldots, \mathbf{x}_M$, the points

$$\tilde{\Phi}(\mathbf{x}_i) := \Phi(\mathbf{x}_i) - \frac{1}{M} \sum_{i=1}^{M} \Phi(\mathbf{x}_i) \tag{20.30}$$

are centered. Thus, the assumptions of section 20.2 now hold, and we go on to define covariance matrix and $\tilde{K}_{ij} = (\tilde{\Phi}(\mathbf{x}_i) \cdot \tilde{\Phi}(\mathbf{x}_j))$ in F. We arrive at our already familiar eigenvalue problem

$$\tilde{\lambda}\tilde{\boldsymbol{\alpha}} = \tilde{K}\tilde{\boldsymbol{\alpha}}, \tag{20.31}$$

with $\tilde{\boldsymbol{\alpha}}$ being the expansion coefficients of an eigenvector (in F) in terms of the points (20.30), $\tilde{\mathbf{V}} = \sum_{i=1}^{M} \tilde{\alpha}_i \tilde{\Phi}(\mathbf{x}_i)$. As we do not have the centered data (20.30), we cannot compute $\tilde{K}$ directly; however, we can express it in terms of its non-centered counterpart K. In the following, we shall use $K_{ij} = (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j))$, and the notations $1_{ij} = 1$ for all i, j, $(1_M)_{ij} := 1/M$, to compute $\tilde{K}_{ij} = (\tilde{\Phi}(\mathbf{x}_i) \cdot \tilde{\Phi}(\mathbf{x}_j))$:

$$\begin{aligned} \tilde{K}_{ij} &= \left((\Phi(\mathbf{x}_i) - \frac{1}{M} \sum_{m=1}^{M} \Phi(\mathbf{x}_m)) \cdot (\Phi(\mathbf{x}_j) - \frac{1}{M} \sum_{n=1}^{M} \Phi(\mathbf{x}_n)) \right) \\ &= K_{ij} - \frac{1}{M} \sum_{m=1}^{M} 1_{im} K_{mj} - \frac{1}{M} \sum_{n=1}^{M} K_{in} 1_{nj} + \frac{1}{M^2} \sum_{m,n=1}^{M} 1_{im} K_{mn} 1_{nj} \\ &= (K - 1_M K - K 1_M + 1_M K 1_M)_{ij}. \end{aligned} \tag{20.32}$$

We thus can compute $\tilde{K}$ from K, and then solve the eigenvalue problem (20.31). As in (20.16), the solutions $\tilde{\boldsymbol{\alpha}}^k$ are normalized by normalizing the corresponding vectors $\tilde{\mathbf{V}}^k$ in F, which translates into $\tilde{\lambda}_k(\tilde{\boldsymbol{\alpha}}^k \cdot \tilde{\boldsymbol{\alpha}}^k) = 1$. For feature extraction, we compute projections of centered Φ-images of test patterns $\mathbf{t}$ onto the eigenvectors of the covariance matrix of the centered points,

$$(\tilde{\mathbf{V}}^k \cdot \tilde{\Phi}(\mathbf{t})) = \sum_{i=1}^{M} \tilde{\alpha}_i^k (\tilde{\Phi}(\mathbf{x}_i) \cdot \tilde{\Phi}(\mathbf{t})). \tag{20.33}$$

Consider a set of test points $\mathbf{t}_1, \ldots, \mathbf{t}_L$, and define two $L \times M$ matrices by $K_{ij}^{test} = (\Phi(\mathbf{t}_i) \cdot \Phi(\mathbf{x}_j))$ and $\tilde{K}_{ij}^{test} = \left((\Phi(\mathbf{t}_i) - \frac{1}{M} \sum_{m=1}^{M} \Phi(\mathbf{x}_m)) \cdot (\Phi(\mathbf{x}_j) - \frac{1}{M} \sum_{n=1}^{M} \Phi(\mathbf{x}_n)) \right)$. As in (20.32), we express $\tilde{K}^{test}$ in terms of K^{test}, and arrive at

$$\tilde{K}^{test} = K^{test} - 1'_M K - K^{test} 1_M + 1'_M K 1_M, \tag{20.34}$$

where $1'_M$ is the $L \times M$ matrix with all entries equal to $1/M$.

Proof of Theorem 20.1

The proof requires some basic notions from group theory. Denote $\mathrm{O}(N)$ the orthogonal group on $\mathbb{R}^N$, i.e. the group of $N \times N$ matrices with $O^\top O = 1$; with the additional requirement $\det O = 1$, we obtain the special orthogonal group $\mathrm{SO}(N)$. A representation ρ of $\mathrm{SO}(N)$ is a map that preserves the group structure, i.e. $T(O_1 O_2) = T(O_1)T(O_2)$ for all $O_1, O_2 \in \mathrm{SO}(N)$.

Proof ($\Longrightarrow$) Due to the definition of $k(\mathbf{x}, \mathbf{y}) := (\mathbf{x} \cdot \mathbf{y})^d$, it follows immediately that $k(O\mathbf{x}, O\mathbf{y}) = (\mathbf{x} \cdot O^\top O\mathbf{y})^d = (\mathbf{x} \cdot \mathbf{y})^d$ for any $O \in \mathrm{O}(N)$.

($\Longleftarrow$) Define $P(d, N)$ to be the (feature) space given by the evaluation of all possible monomials of order p on $\mathbb{R}^N$, furnished with a Euclidean dot product.[7] Then the map into feature space, $\Phi : \mathbb{R}^N \to P(d, N), \mathbf{x} \to \Phi(\mathbf{x})$, induces a representation ρ of $\mathrm{SO}(N)$ on $P(d, N)$ via $\Phi(O\mathbf{x}) = \rho(O)\Phi(\mathbf{x})$. This is a consequence of (Vilenkin, 1968, chapter IX.2). We have

$$(\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})) = (\mathbf{x} \cdot \mathbf{y})^d = (O\mathbf{x} \cdot O\mathbf{y})^d = (\rho(O)\Phi(\mathbf{x}) \cdot \rho(O)\Phi(\mathbf{y})). \tag{20.35}$$

What is more, ρ is an orthogonal representation, i.e. $\rho(O)^\top \rho(O) = 1$ for all $O \in \mathrm{SO}(N)$. This follows from $(\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})) = (\rho(O)\Phi(\mathbf{x}) \cdot \rho(O)\Phi(\mathbf{y}))$ and $\operatorname{span} \Phi(\mathbb{R}^N) = P(d, N)$.

We now prove that any positive diagonal matrix D acting on $P(d, N)$, satisfying the invariance condition

$$(D^{\frac{1}{2}}\Phi(\mathbf{x}) \cdot D^{\frac{1}{2}}\Phi(\mathbf{y})) = (D^{\frac{1}{2}}\rho(O)\Phi(\mathbf{x}) \cdot D^{\frac{1}{2}}\rho(O)\Phi(\mathbf{y})) \tag{20.36}$$

for all $O \in \mathrm{SO}(N)$, is necessarily a multiple of the unit matrix. If that were not true, then $k_D(\mathbf{x}, \mathbf{y}) := (D^{\frac{1}{2}}\Phi(\mathbf{x}) \cdot D^{\frac{1}{2}}\Phi(\mathbf{y})) = k_D(O\mathbf{x}, O\mathbf{y})$ would be a different kernel invariant under $\mathrm{SO}(N)$. Again, as $\operatorname{span} \Phi(\mathbb{R}^N) = P(d, N)$, we may rewrite (20.36) as

$$D = \rho(O)^\top D\rho(O), \text{ i.e. } D\rho(O) = \rho(O)D. \tag{20.37}$$

In componentwise notation (in $P(d, N)$), this reads

$$D_i \rho(O)_{ij} = D_j \rho(O)_{ij}. \tag{20.38}$$

Therefore, we can show that $D_i = D_j$ for all i, j by showing that there exist sufficiently many nonzero $\rho(O)_{ij}$. To this end, consider a rotation $\tilde{O}$ mapping $\mathbf{x}_1 := (1, 0, \ldots, 0)$ into $\mathbf{x}_2 := \frac{1}{\sqrt{N}}(1, \ldots, 1)$. Clearly, $\Phi(\mathbf{x}_1) = (1, 0, \ldots, 0) \in P(d, N)$, whereas $\Phi(\mathbf{x}_2) = \Phi(\tilde{O}\mathbf{x}_1) = \rho(\tilde{O})\Phi(\mathbf{x}_1)$ contains only nonzero entries. Hence also

7. It is straightforward to see that the polynomial kernel corresponds to a dot product in the space of all monomials: simply compute $(\mathbf{x} \cdot \mathbf{y})^d = \sum_{j_1, \ldots, j_d = 1}^N x_{j_1} \cdot \ldots \cdot x_{j_d} \cdot y_{j_1} \cdot \ldots \cdot y_{j_d} = (C_d(\mathbf{x}) \cdot C_d(\mathbf{y}))$, where C_d maps $\mathbf{x}$ to the vector $C_d(\mathbf{x})$ whose entries are all possible d-th degree ordered products of the entries of $\mathbf{x}$ (Schölkopf, 1997). Identifying the entries which just differ by the ordering of the coordinates, we obtain pre-factors of the form $\sqrt{\frac{d!}{p_1! p_2! \ldots p_N!}} x_1^{p_1} x_2^{p_2} \ldots x_N^{p_N}$, with $\sum_i p_i = p$ (Smola et al., 1998a).

the first row of $\rho(\tilde{O})$ contains only nonzero entries. By (20.38), we conclude that $D_1 = D_i$ for all i, and therefore $D = \lambda 1$.

This completes the argument concerning the invariance of the polynomial kernel. The transfer to the invariance of kernel PCA, i.e. to the invariance of kernel PCA feature extraction for all test and training sets, is straightforward. ■

Note that the above statement does not hold if we allow D to be an arbitrary matrix of full rank. In particular, due to Schur's lemma (Hamermesh, 1962) one can show that the number of different subspaces that can be scaled separately equals the number of irreducible representations contained in ρ.

Kernels

The remainder of the chapter provides some further material on kernels, partly reprinted from (Schölkopf et al., 1996b).

Kernels Corresponding to Bilinear Forms in Another Space. Mercer's theorem of functional analysis gives conditions under which we can construct the mapping Φ from the eigenfunction decomposition of k (cf. (1.25)) such that $k(\mathbf{x}, \mathbf{y}) = (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}))$.
In fact, k does not have to be the kernel of a positive *definite* operator: even if a finite number of eigenvalues λ_i in (1.25) is negative, the expansion (1.25) is still valid. In that case, k corresponds to a Lorentz scalar product in a space with indefinite signature. We can then no longer interpret our method as PCA in some feature space; however, it could still be viewed as a nonlinear factor analysis.[8] For instance, we have used the thin plate spline kernel $k(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|^2 \ln \|\mathbf{x} - \mathbf{y}\|$, which is not positive definite (it is conditionally positive definite of order 2, cf. e.g. Smola et al. (1998c)).

Kernels Constructed from Mappings. We stated above that once we have a suitable kernel, we need not worry anymore about exactly which map Φ the kernel corresponds to. For the purpose of constructing kernels, however, it can well be useful to compute the kernels from mappings into some dot product space F, $\Phi : \mathbb{R}^N \to F$. Ideally, we would like to choose Φ such that we can obtain an expression for $(\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}))$ which can be computed efficiently. Presently, we shall consider mappings into function spaces,

$$\mathbf{x} \mapsto f_{\mathbf{x}}, \tag{20.39}$$

with $f_{\mathbf{x}}$ being a complex-valued function on some measure space. We furthermore

8. The fact that we can use indefinite operators distinguishes this approach from the usage of kernels in the support vector machine: in the latter, the definiteness is necessary for the convex programming.

assume that these spaces are equipped with a dot product

$$(f_{\mathbf{x}} \cdot f_{\mathbf{y}}) = \int f_{\mathbf{x}}(u)\, f_{\mathbf{y}}(u)\; du. \tag{20.40}$$

We can then define kernels of the type

$$k(\mathbf{x}, \mathbf{y}) := (f_{\mathbf{x}} \cdot f_{\mathbf{y}})^d. \tag{20.41}$$

(These kernels can also be used if our observations are already given as functions, as is usually the case for the variant of PCA which is referred to as the Karhunen-Loève-Transformation; see Karhunen (1946)) As an example, suppose the input patterns $\mathbf{x}_i$ are $q \times q$ images. Then we can map them to two-dimensional image intensity distributions $f_{\mathbf{x}_i}$ (e.g. splines on $[0,1]^2$). The corresponding kernel will then approximately equal the original dot product between the images represented as pixel vectors, which can be seen by considering the finite sum approximation to the integral,

$$q^2 \int_0^1 \int_0^1 f_{\mathbf{x}}(u) f_{\mathbf{y}}(u)\, d^2u \approx \sum_{i=1}^{q} \sum_{j=1}^{q} f_{\mathbf{x}}\left(\frac{i-\frac{1}{2}}{q}, \frac{j-\frac{1}{2}}{q}\right) f_{\mathbf{y}}\left(\frac{i-\frac{1}{2}}{q}, \frac{j-\frac{1}{2}}{q}\right). \tag{20.42}$$

Combining Kernels. If k and k' satisfy Mercer's conditions, then so will $k + k'$ and, for $\lambda > 0$, λk. In other words, the admissible kernels form a cone in the space of all integral operators. Clearly, $k + k'$ corresponds to mapping into the direct sum of the respective spaces into which k and k' map. Of course, we could also explicitly do the principal component extraction twice, for both kernels, and decide ourselves on the respective numbers of components to extract. In this case, we would not obtain combinations of the two feature types.
Combining polynomial kernels, we can thus construct admissible kernels by series expansions: given a function f with a uniformly convergent power series expansion $f(x) = \sum_i a_i x_i$, then $k_f(\mathbf{x}, \mathbf{y}) := f((\mathbf{x} \cdot \mathbf{y}))$ is a Mercer kernel. In fact, the latter can be generalized to expansions in terms of Mercer kernels k, i.e. $f(k(\mathbf{x}, \mathbf{y}))$, other than just the usual dot product. This is due to the fact that by an argument similar to the one proving that $(\mathbf{x} \cdot \mathbf{y})^d$ is a Mercer kernel ($d \in \mathbb{N}$), also $k(\mathbf{x}, \mathbf{y})^d$ is a Mercer kernel, since it can be written as $(\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}))^d$ (Smola et al., 1998c).
Another way of combining kernels is to use different kernels for different parts of the input vectors. In the simplest case, we can define $k(\mathbf{x}, \mathbf{y}) := k_1(\mathbf{x}_1, \mathbf{y}_1) + k_2(\mathbf{x}_2, \mathbf{y}_2)$, where $\mathbf{x} = \mathbf{x}_1 \oplus \mathbf{x}_2$ and $\mathbf{y} = \mathbf{y}_1 \oplus \mathbf{y}_2$, and k_1, k_2 are Mercer kernels on the two lower-dimensional spaces comprising only parts of the input. This method can be useful if the different parts of the input have different meanings and should be dealt with differently.

Iterating Kernels. Given a kernel k, we can construct iterated kernels (e.g. Courant and Hilbert, 1953) by

$$k^{(2)}(\mathbf{x}, \mathbf{y}) := \int k(\mathbf{x}, \mathbf{z}) k(\mathbf{z}, \mathbf{y})\; d\mathbf{z}. \tag{20.43}$$

In fact, $k^{(2)}$ will be positive even if k is not, as can be seen from

$$\begin{aligned} \int k^{(2)}(\mathbf{x}, \mathbf{y}) f(\mathbf{x}) f(\mathbf{y})\; d\mathbf{x} d\mathbf{y} &= \int \int k(\mathbf{x}, \mathbf{z}) k(\mathbf{z}, \mathbf{y}) f(\mathbf{x}) f(\mathbf{y})\; d\mathbf{z} d\mathbf{x} d\mathbf{y} \\ &= \int \left(\int k(\mathbf{x}, \mathbf{z}) f(\mathbf{x})\; d\mathbf{x} \right)^2 d\mathbf{z}. \end{aligned} \tag{20.44}$$

This gives us a method for constructing admissible kernels.

Multi-Layer Support Vector Machines

By first extracting nonlinear principal components according to (20.18), and then training a Support Vector machine, we can construct Support Vector type machines with additional layers. The number of components extracted then determines the size of of the first hidden layer. Combining (20.18) with the Support Vector decision function (1.32), we thus get machines of the type

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{\ell} \psi_i k_2(\vec{g}(\mathbf{x}_i) \cdot \vec{g}(\mathbf{x})) + b \right) \tag{20.45}$$

with

$$\vec{g}(\mathbf{x})_j = \left(\mathbf{V}^j \cdot \Phi(\mathbf{x})\right) = \sum_{k=1}^{M} \alpha_k^j k_1(\mathbf{x}_k, \mathbf{x}). \tag{20.46}$$

Here, the expansion coefficients ψ_i are computed by a standard Support Vector Machine. Note that different kernel functions k_1 and k_2 can be used for the different layers. Also note that this could provide an efficient means of building multivariate Support Vector Machines, i.e. q machines mapping $\mathbb{R}^N \to \mathbb{R}^q$, where $q \in \mathbb{N}$. All these machines may share the first preprocessing layer which includes the numerically expensive steps and then use a simple kernel for the second layer. Similar considerations apply for multi-class classification, where often a set of binary classifiers (which could share some preprocessing) is constructed.