Max-Planck-Institut
für biologische Kybernetik
Arbeitsgruppe Bülthoff
Spemannstraße 38 • 72076 Tübingen • Germany

Technical Report No. 44 December 1996

Nonlinear Component Analysis as a Kernel Eigenvalue Problem

Bernhard Schölkopf, Alexander Smola, and Klaus–Robert Müller

Abstract

We describe a new method for performing a nonlinear form of Principal Component Analysis. By the use of integral operator kernel functions, we 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, along with a discussion of other techniques which can be made nonlinear with the kernel approach; and present first experimental results on nonlinear feature extraction for pattern recognition.

AS and KRM are with GMD First (Forschungszentrum Informationstechnik), Rudower Chaussee 5, 12489 Berlin. AS and BS were supported by grants from the Studienstiftung des deutschen Volkes. BS thanks the GMD First for hospitality during two visits. AS and BS thank V. Vapnik for introducing them to kernel representations of dot products during joint work on Support Vector machines. This work profited from discussions with V. Blanz, L. Bottou, C. Burges, H. Bülthoff, K. Gegenfurtner, P. Haffner, 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.

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) and Diamataras & Kung (1996). PCA is an orthogonal transformation of the coordinate system in which we describe our data. The new coordinate values by which we represent out 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 the *factors* or *latent variables* of the data.

The present work generalizes PCA to 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 higher–order correlations between input variables. In the case of image analysis, this would amount to finding principal components in the space of products of input pixels.

To this end, we are using the method of expressing dot products in feature space in terms of kernel functions in input space. 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 (Aizerman, Braverman, & Rozonoer, 1964; Boser, Guyon, & Vapnik, 1992). Even though this general fact was known (Burges, 1996), the machine learning community has made little use of it, the exception being Support Vector machines (Vapnik, 1995).

In this paper, we give some examples of nonlinear methods constructed by this approach. For one example, the case of a nonlinear form of principal component analysis, we shall give details and experimental results (Sections 2 – 6); for some other cases, we shall briefly sketch the algorithms (Sec. 7).

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 shall then formulate it in a way which uses exclusively dot products. In Sec. 3, we shall discuss the kernel method for computing dot products in feature spaces. Together, these two sections form the basis for Sec. 4, which presents the proposed kernel–based algorithm for nonlinear PCA. Following that, Sec. 5 will discuss some differences between kernel–based PCA and other generalizations of PCA. In Sec. 6, we shall give some first experimental results on kernel–based feature extraction for pattern recognition. After a discussion of other applications of the kernel method (Sec. 7), we conclude with a discussion (Sec. 8). Finally, some technical material which is not essential for the main thread of the argument has been relegated into the appendix.

2 PCA in Feature Spaces

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

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

To do this, one has to solve the Eigenvalue equation

$$\lambda \mathbf{v} = C\mathbf{v} \tag{2}$$

for Eigenvalues $\lambda \geq 0$ and $\mathbf{v} \in \mathbf{R}^N \backslash \{0\}$. As $C\mathbf{v} = \frac{1}{M}\sum_{j=1}^M (\mathbf{x}_j \cdot \mathbf{v})\mathbf{x}_j$, all solutions $\mathbf{v}$ must lie in the span of $\mathbf{x}_1 \ldots \mathbf{x}_M$, hence (2) is equivalent to

$$\lambda(\mathbf{x}_k \cdot \mathbf{v}) = (\mathbf{x}_k \cdot C\mathbf{v}) \text{ for all } k = 1, \ldots, M. \tag{3}$$

The remainder of this section is devoted to a straightforward translation to a nonlinear scenario, in order to prepare the ground for the method proposed in the present paper. We shall now describe this computation in another dot product space F, which is related to the input space by a possibly nonlinear map

$$\begin{aligned} \Phi : \mathbf{R}^N &\to F, \\ \mathbf{x} &\mapsto \mathbf{X}. \end{aligned} \tag{4}$$

Note that F, which we will refer to as the *feature space*, 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 $\mathbf{R}^N$.

Again, we make the assumption that we are dealing with centered data, i.e. $\sum_{k=1}^M \Phi(\mathbf{x}_k) = 0$ — we shall return to this point later. Using the covariance matrix in F,

$$\bar{C} = \frac{1}{M}\sum_{j=1}^{M} \Phi(\mathbf{x}_j)\Phi(\mathbf{x}_j)^\top , \tag{5}$$

[1] 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 maximum likelihood *estimate* (1) of the covariance matrix from a finite sample.

(if F is infinite–dimensional, we think of $\Phi(\mathbf{x}_j)\Phi(\mathbf{x}_j)^\top$ as the linear operator which maps $\mathbf{X} \in F$ to $\Phi(\mathbf{x}_j)(\Phi(\mathbf{x}_j)\cdot\mathbf{X})$) we now have to find Eigenvalues $\lambda \geq 0$ and Eigenvectors $\mathbf{V} \in F\backslash\{0\}$ satisfying

$$\lambda \mathbf{V} = \bar{C}\mathbf{V} \tag{6}$$

By the same argument as above, the solutions $\mathbf{V}$ lie in the span of $\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_M)$. For us, this has two useful consequences: first, we can consider the equivalent equation

$$\lambda(\Phi(\mathbf{x}_k)\cdot\mathbf{V}) = (\Phi(\mathbf{x}_k)\cdot\bar{C}\mathbf{V}) \text{ for all } k = 1, \ldots, M, \tag{7}$$

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

$$\mathbf{V} = \sum_{i=1}^{M} \alpha_i \Phi(\mathbf{x}_i). \tag{8}$$

Combining (7) and (8), we get

$$\lambda \sum_{i=1}^{M} \alpha_i (\Phi(\mathbf{x}_k)\cdot\Phi(\mathbf{x}_i)) =$$

$$\frac{1}{M}\sum_{i=1}^{M} \alpha_i (\Phi(\mathbf{x}_k)\cdot \sum_{j=1}^{M}\Phi(\mathbf{x}_j))(\Phi(\mathbf{x}_j)\cdot\Phi(\mathbf{x}_i))$$
$$\text{for all } k = 1, \ldots, M. \tag{9}$$

Defining an $M \times M$ matrix K by

$$K_{ij} := (\Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j)), \tag{10}$$

this reads

$$M\lambda K\boldsymbol{\alpha} = K^2\boldsymbol{\alpha}, \tag{11}$$

where $\boldsymbol{\alpha}$ denotes the column vector with entries $\alpha_1, \ldots, \alpha_M$. As K is symmetric, it has a set of Eigenvectors which spans the whole space, thus

$$M\lambda\boldsymbol{\alpha} = K\boldsymbol{\alpha} \tag{12}$$

gives us all solutions $\boldsymbol{\alpha}$ of Eq. (11). Note that K is positive semidefinite, which can be seen by noticing that it equals

$$(\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_M))^\top \cdot (\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_M)), \tag{13}$$

which implies that for all $\mathbf{X} \in F$,

$$(\mathbf{X}\cdot K\mathbf{X}) = \|(\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_M))\mathbf{X}\|^2 \geq 0. \tag{14}$$

Consequently, K's Eigenvalues will be nonnegative, and will exactly give the solutions $M\lambda$ of Eq. (11). We therefore only need to diagonalize K. Let $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_M$ denote the Eigenvalues, and $\boldsymbol{\alpha}^1, \ldots, \boldsymbol{\alpha}^M$ the corresponding complete set of Eigenvectors, with λ_p being the first nonzero Eigenvalue.[2] We normalize $\boldsymbol{\alpha}^p, \ldots, \boldsymbol{\alpha}^M$ by requiring that the corresponding vectors in F be normalized, i.e.

$$(\mathbf{V}^k \cdot \mathbf{V}^k) = 1 \text{ for all } k = p, \ldots, M. \tag{15}$$

By virtue of (8) and (12), this translates into a normalization condition for $\boldsymbol{\alpha}^p, \ldots, \boldsymbol{\alpha}^M$:

$$\begin{aligned} 1 &= \sum_{i,j=1}^{M} \alpha_i^k \alpha_j^k (\Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j)) \\ &= \sum_{i,j=1}^{M} \alpha_i^k \alpha_j^k K_{ij} \\ &= (\boldsymbol{\alpha}^k \cdot K\boldsymbol{\alpha}^k) \\ &= \lambda_k(\boldsymbol{\alpha}^k\cdot\boldsymbol{\alpha}^k) \end{aligned} \tag{16}$$

For the purpose of principal component extraction, we need to compute projections on 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

$$(\mathbf{V}^k \cdot \Phi(\mathbf{x})) = \sum_{i=1}^{M} \alpha_i^k (\Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x})) \tag{17}$$

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 dot product matrix K defined by (10);[3] second, compute its Eigenvectors and normalize them in F; third, compute projections of a test point onto the Eigenvectors by (17).

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 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 A).

Before we proceed to the next section, which more closely investigates the role of the map Φ, the following observation is essential. The mapping Φ used in the matrix computation can be an arbitrary nonlinear map into the possibly high–dimensional space F. e.g. the space of all nth order monomials in the entries of an input vector.

[2] If we require that Φ should not map all observations to zero, then such a p will always exist.

[3] Note that in our derivation we could have used the known result (e.g. Kirby & Sirovich, 1990) that PCA can be carried out on the dot product matrix $(\mathbf{x}_i\cdot\mathbf{x}_j)_{ij}$ instead of (1), however, for the sake of clarity and extendability (in Appendix A, we shall consider the case where the data must be centered in F), we gave a detailed derivation.

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, which will be described in the following section, builds on the fact that we *exclusively* need to compute dot products between mapped patterns (in (10) and (17)); *we never need the mapped patterns explicitly.*

3 Computing Dot Products in Feature Space

In order to compute dot products of the form $(\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}))$, we use kernel representations of the form

$$k(\mathbf{x}, \mathbf{y}) = (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})), \tag{18}$$

which allow us to compute the value of the dot product in F without having to carry out the map Φ. This method was used by Boser, Guyon, & Vapnik (1992) to extend the "Generalized Portrait" hyperplane classifier of Vapnik & Chervonenkis (1974) to nonlinear Support Vector machines. To this end, they substitute a priori chosen kernel functions for all occurances of dot products. This way, the powerful results of Vapnik & Chervonenkis (1974) for the Generalized Portrait carry over to the nonlinear case. Aizerman, Braverman & Rozonoer (1964) call F the "linearization space", and use it in the context of the potential function classification method to express the dot product between elements of F in terms of elements of the input space. If F is high–dimensional, we would like to be able to find a closed form expression for k which can be efficiently computed. Aizerman et al. (1964) consider the possibility of choosing k a priori, without being directly concerned with the corresponding mapping Φ into F. A specific choice of k might then correspond to a dot product between patterns mapped with a suitable Φ. A particularly useful example, which is a direct generalization of a result proved by Poggio (1975, Lemma 2.1) in the context of polynomial approximation, is

$$(\mathbf{x} \cdot \mathbf{y})^d = (C_d(\mathbf{x}), C_d(\mathbf{y})), \tag{19}$$

where C_d maps $\mathbf{x}$ to the vector $C_d(\mathbf{x})$ whose entries are all possible n-th degree ordered products of the entries of $\mathbf{x}$. For instance (Vapnik, 1995), if $\mathbf{x} = (x_1, x_2)$, then $C_2(\mathbf{x}) = (x_1^2, x_2^2, x_1x_2, x_2x_1)$, or, yielding the same value of the dot product,

$$c_2(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2). \tag{20}$$

For this example, it is easy to verify that

$$\begin{aligned} &\left((x_1, x_2)(y_1, y_2)^\top\right)^2 \\ &= (x_1^2, x_2^2, \sqrt{2}x_1x_2)(y_1^2, y_2^2, \sqrt{2}y_1y_2)^\top \\ &= c_2(\mathbf{x})c_2(\mathbf{y})^\top. \end{aligned} \tag{21}$$

In general, the function

$$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^d \tag{22}$$

corresponds to a dot product in the space of d-th order monomials of the input coordinates. If $\mathbf{x}$ represents an image with the entries being pixel values, we can thus easily work in the space spanned by products of any d pixels — provided that we are able to do our work solely in terms of dot products, without any explicit usage of a mapped pattern $c_d(\mathbf{x})$. The latter lives in a possibly very high–dimensional space: even though we will identify terms like x_1x_2 and x_2x_1 into one coordinate of F as in (20), the dimensionality of F, the image of $\mathbf{R}^N$ under c_d, still is $\frac{(N+p-1)!}{p!(N-1)!}$ and thus grows like N^p. For instance, 16×16 input images and a polynomial degree $d = 5$ yield a dimensionality of 10^{10}. Thus, using kernels of the form (22) is our only way to take into account higher–order statistics without a combinatorial explosion of time complexity.

The general question which function k corresponds to a dot product in some space F has been discussed by Boser, Guyon, & Vapnik (1992) and Vapnik (1995): Mercer's theorem of functional analysis states that if k is a continuous kernel of a positive integral operator, we can construct a mapping into a space where k acts as a dot product (for details, see Appendix B.

The application of (18) to our problem is straightforward: we simply substitute an a priori chosen kernel function $k(\mathbf{x}, \mathbf{y})$ for all occurances of $(\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}))$. This was the reason why we had to formulate the problem in Sec. 2 in a way which only makes use of the values of dot products in F. The choice of k then *implicitly* determines the mapping Φ and the feature space F.

In Appendix B, we give some examples of kernels other than (22) which may be used.

4 Kernel PCA

4.1 The Algorithm

To perform kernel–based PCA (Fig. 1), from now on referred to as *kernel PCA*, the following steps have to be carried out: first, we compute the dot product matrix (cf. Eq. (10))

$$K_{ij} = (k(\mathbf{x}_i, \mathbf{x}_j))_{ij}. \tag{23}$$

Next, we solve (12) by diagonalizing K, and normalize the Eigenvector expansion coefficients α^n by requiring Eq. (16),

$$1 = \lambda_n(\boldsymbol{\alpha}^n \cdot \boldsymbol{\alpha}^n).$$

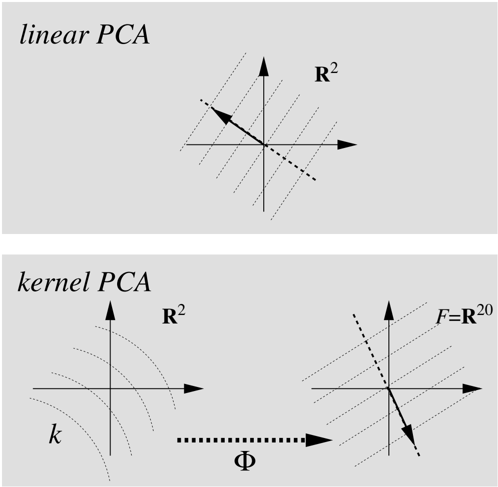

Figure 1: The basic idea of kernel PCA. In some high–dimensional feature space F (bottom right), we are performing linear PCA, just as a PCA in input space (top). Since F is nonlinearly related to input space (via Φ), the contour lines of constant projections onto the principal Eigenvector (drawn as an arrow) become *nonlinear* in input space. Note that we cannot draw a pre–image of the Eigenvector in input space, as it may not even exist. Crucial to kernel PCA is the fact that we do not actually perform the map into F, but instead perform all necessary computations by the use of a kernel function k in input space (here: $\mathbf{R}^2$).

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. (17)),

$$(k\mathrm{PC})_n(\mathbf{x}) = (\mathbf{V}^n \cdot \Phi(\mathbf{x})) = \sum_{i=1}^{M} \alpha_i^n k(\mathbf{x}_i, \mathbf{x}). \quad (24)$$

If we use a kernel as described in Sec. 3, 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.

4.2 Properties of (Kernel–) PCA

If we use a kernel which satisfies the conditions given in Sec. 3, we know that we are in fact doing a standard PCA in F. Consequently, all mathematical and statistical properties of PCA (see for instance Jolliffe, 1986) carry over to kernel–based 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 $\mathbf{R}^N$. In F, we can thus assert that PCA is the orthogonal basis transformation with the following properties (assuming that the Eigenvectors are sorted in ascending 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
- the principal components are uncorrelated
- the representation entropy is minimized
- the first q principal components have maximal mutual information with respect to the inputs

For more details, see Diamantaras & Kung (1996).

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. We shall not go into detail on that matter, but rather proceed in our discussion of kernel PCA.

4.3 Dimensionality Reduction and Feature Extraction

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[4] — a fact that illustrates that it is impossible to perform kernel PCA based on an $N \times N$ covariance matrix.

4.4 Computational Complexity

As mentioned in Sec. 3, a fifth order polynomial kernel on a 256–dimensional input space yields a 10^{10}–dimensional space. It would seem that looking for principal components in his space should pose intractable computational problems. However, as we have explained above, this is not the case. First, as pointed out in Sect. 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, as we know that in our case this can be done directly in the input space, using kernel

[4] If we use one kernel — of course, we could extract features with several kernels, to get even more.

functions. The proposed kernel principal component analysis thus is computationally comparable to a linear PCA on ℓ observations with an $\ell \times \ell$ dot product matrix. The overall computational complexity is not changed by the following additional cost: we need to evaluate kernel functions rather than just dot products. If k is easy to compute, as for polynomial kernels, e.g., this is negligible.

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 & Kung, 1996). In addition, we can consider using an estimate of the dot product matrix 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 is different for principal component extraction. There, we have to evaluate the kernel function ℓ times for each extracted principal component (24), rather than just evaluating one dot product as for a linear PCA. In some cases, e.g. if we were to extract principal components as a preprocessing step for classification, this is a disadvantage. However, in that case, we can speed up the extraction by a method analoguous to a technique proposed by Burges (1996) in the context of Support Vector machines. Specifically, we can try to approximate each Eigenvector $\mathbf{V} = \sum_{i=1}^{\ell} \alpha_i \Phi(\mathbf{x}_i)$ (Eq. (8)) by another vector

$$\tilde{\mathbf{V}} = \sum_{j=1}^{m} \beta_j \Phi(\mathbf{z}_j), \tag{25}$$

where $m < \ell$ is chosen a priori according to the desired speed–up, and $\mathbf{z}_j \in \mathbf{R}^N, j = 1, \cdots, m$. This is done by minimizing the squared difference

$$\rho = \|\mathbf{V} - \tilde{\mathbf{V}}\|^2. \tag{26}$$

The crucial point is that this also can be done without explicitly dealing with the possibly high–dimensional space F. As

$$\rho = \|\mathbf{V}\| + \|\tilde{\mathbf{V}}\| - 2\sum_{i=1}^{\ell}\sum_{j=1}^{m} \alpha_i \beta_j k(\mathbf{x}_i, \mathbf{z}_j), \tag{27}$$

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. In the case $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^2$ it is possible to get an exact expansion with at most N vectors $\mathbf{z}_j$ (N being the dimension of the input space) by solving an Eigenvalue problem (Burges, 1996). For the task of handwritten character recognition, this technique led to a speed–up by a factor of 50 at almost no loss in accuracy, yielding a state of the art classifier (Burges & Schölkopf, 1996).

Finally, we add that although the 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 that in the nonlinear case, 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. (The latter are slower than comparable Neural Networks (Burges & Schölkopf, 1996)). This is due to the fact that for $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})$, the Support Vector decision function (Boser, Guyon, & Vapnik, 1992)

$$f(\mathbf{x}) = \operatorname{sgn}(\sum_{i=1}^{\ell} \lambda_i k(\mathbf{x}, \mathbf{x}_i) + b) \tag{28}$$

can be expressed with a single weight vector $\mathbf{w} = \sum_{i=1}^{\ell} \lambda_i \mathbf{x}_i$ as

$$f(\mathbf{x}) = \operatorname{sgn}((\mathbf{x} \cdot \mathbf{w}) + b). \tag{29}$$

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 tradeoff of the whole classifier, can be controlled by the number of components which we extract, or by the above reduced set parameter m.

4.5 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, the problem is slightly different: to get interpretability, we want 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 in Sec. 4.4: we could parametrize our set of desired input variables and run the minimization of (26) 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.

4.6 Reconstruction

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.

In kernel PCA, this is no longer possible, the reason being that it may happen that a vector $\mathbf{V}$ in F does not have a pre–image in $\mathbf{R}^N$. We can, however, find a vector $\mathbf{z}$ in $\mathbf{R}^N$ which maps to a vector that optimally approximates $\mathbf{V}$. To this end, we can proceed as in Sec. 4.4 and write the distance in F in terms of the kernel function. Again, we can minimize it by gradient descent.

Alternatively, we can use a suitable regression method for estimating the reconstruction mapping from the kernel–based principal components to the inputs. Once this is estimated from the data, we can use canonical basis vectors as inputs to estimate the approximate pre–images in $\mathbf{R}^N$ of the Eigenvectors in F.

4.7 Multi–Layer Support Vector machines

By first extracting nonlinear principal components according to (24), and then training a Support Vector machine (Vapnik, 1995), 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 (24) with the Support Vector decision function (Vapnik, 1995), we thus get machines of the type

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{\ell} \psi_i K_2(\vec{g}(\mathbf{x}_i), \vec{g}(\mathbf{x})) + b\right) \qquad (30)$$

with

$$\vec{g}(\mathbf{x})_j = \left(\mathbf{V}^j, \Phi(\mathbf{x})\right) = \sum_{k=1}^{M} \alpha_k^j K_1(\mathbf{x}_k, \mathbf{x}). \qquad (31)$$

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 can provide an efficient means of building multivariate Support Vector Machines, i.e. q machines mapping $\mathbf{R}^N \to \mathbf{R}^q$, where $q \in \mathbf{N}$. 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.

5 Comparison to Other Methods for Nonlinear PCA

Starting from some of the properties characterizing PCA (Sec. 4.2), 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 just three approaches, and refer the reader to Diamantaras & Kung (1996) for more details.

Hebbian Networks. Initiated by the pioneering work of Oja (1982), a number of unsupervised neural–network type algorithms computing principal components have been proposed. Compared to the standard approach of diagonalizing the covariance matrix, they have advantages for instance in cases where the data are nonstationary. It is fairly straightforward to construct nonlinear variants of these algorithms by adding nonlinear activation functions. The algorithms then extract features that the authors have referred to as nonlinear principal components. Compared to kernel PCA, however, these approaches are lacking the geometrical interpretation as a standard PCA in a feature space nonlinearly related to input space, and it is difficult to understand what exactly they are extracting.

Autoassociative Multi–Layer Perceptrons. Consider a linear 2–layer perceptron with a hidden layer which is smaller than the input. If we train it to reproduce the input values as outputs (i.e. use it in an autoassociative mode), then the hidden unit activations form a lower–dimensional representation of the data, closely related to PCA (see for instance Diamantaras & 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

[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 & Kung, 1996).

training. Moreover, in neural network implementations there is often a risk of getting overfitting. Another problem with neural approaches to nonlinear PCA is that the number of components to be extracted has to be specified in advance. This is also the case for the following method, which, however, has the advantage that there is a clear geometrical picture of what kind of nonlinear features are being extracted.

Principal Curves. An approach with a geometric interpretation in input space is the method of principal curves (Hastie & Stuetzle, 1989). This method iteratively estimates a curve (or surface) which captures 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, again a nonlinear optimization problem has to be solved.

Kernel PCA. Kernel PCA is a nonlinear generalization of PCA in the sense that if we use the kernel $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})$, we recover original PCA. To get nonlinear forms of PCA, we simpy choose a nonlinear kernel. Moreover, kernel PCA is a generalization of PCA is the respect that it is performing PCA in feature spaces of arbitrarily large (possibly infinite) dimension.

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 (cf. our considerations in Sec. 4.4). 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.

6 Experiments

In this section, we present a set of experiments where we used kernel PCA to extract principal components. First, we shall take a look at a simple toy example; following that, we describe real–world experiments where we tried to assess the utility of the extracted principal components by classification tasks.

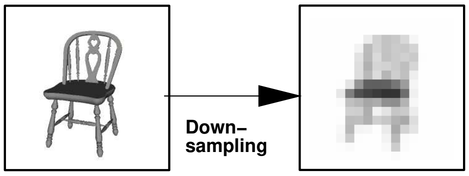

Figure 3: An example image from the MPI chair database. Left: original view of the chair, right: downsampled to 16×16. This is what the views in the database look like.

6.1 Toy Examples

To provide some intuition on how the principal component analysis in F behaves in input space, we show a set of experiments with an artificial 2–D data set, using polynomial kernels (cf. Eq.(22)) of degree 1 through 4 (see Fig. 2).

6.2 Object Recognition

In this set of experiments, we used the MPI database of images of realistically rendered 3–D chair models (Blanz et al., 1996).[6] It contains downsampled (16×16) snapshots of 25 chairs, taken from the upper half of the viewing sphere; for an example image, see Fig. 3. The training set consists of 89 regularly spaced views of each chair, the test set contains 100 random views each. Blanz et al. compared three different classifiers on this task, with a Support Vector classifier outperforming both an oriented filter approach and a 2–layer perceptron.

In the experiment, we computed the dot product 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 *linear* Support Vector classifier (Vapnik & Chervonenkis, 1979; Cortes & Vapnik, 1995) on the classification task.[7] Table 1 summarizes our findings: in all cases, nonlinear components as extracted by polynomial kernels (cf. Eq. (22, with $d > 1$) led to significantly better classification accuracy on the test set than standard PCA. Specifically, the nonlinear components afforded top test performances between 2% and 4% error, whereas in the linear case we obtained 17%. In many cases, the resulting system even outperformed *nonlinear*

[6]The dataset is available from ftp://ftp.mpik-tueb.mpg.de/pub/chair_dataset/.

[7]Thanks to L. Bottou, C. Burges, C. Cortes for parts of the code used in the SV optimization.

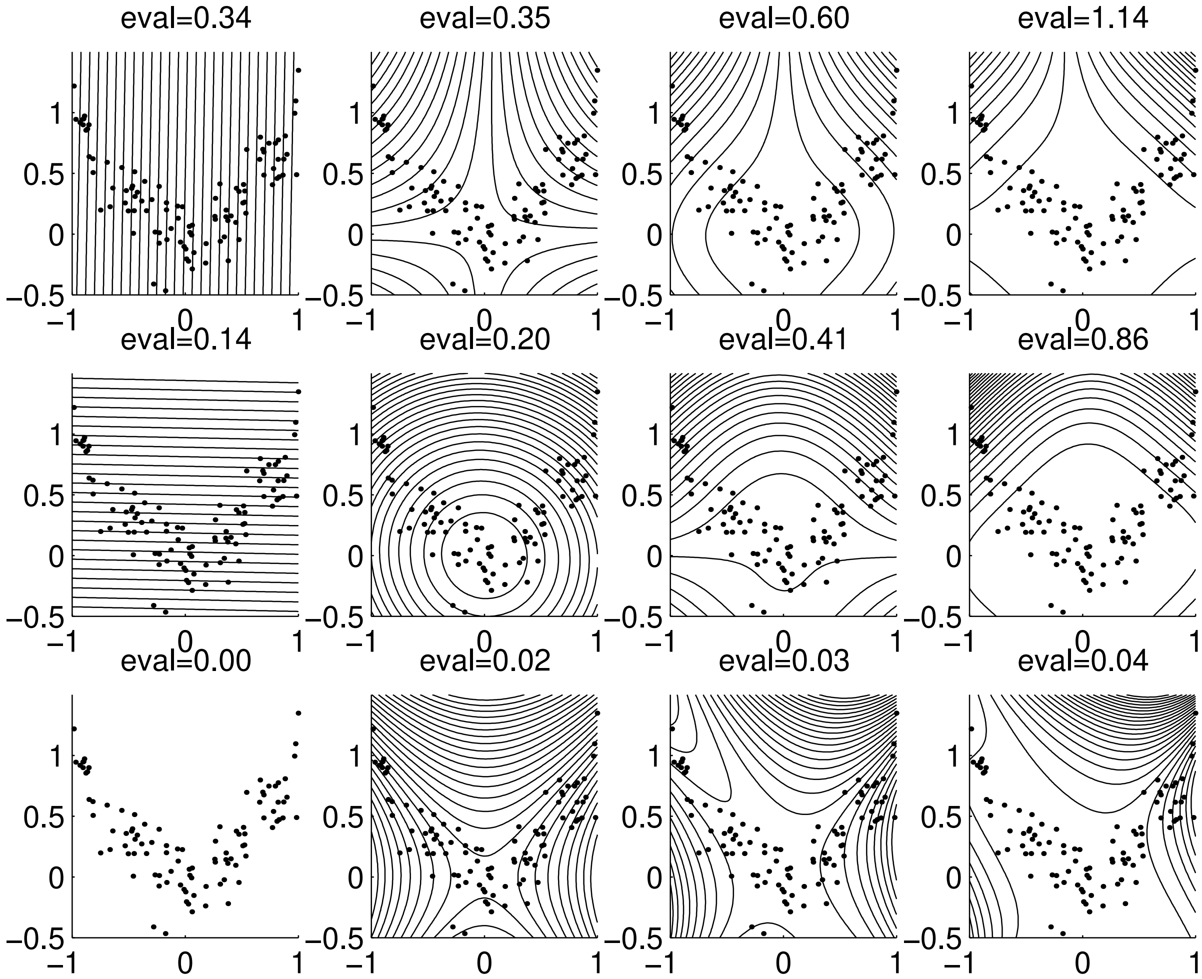

Figure 2: Two–dimensional toy examples, with data generated in the following way: x–values have uniform distribution in $[-1, 1]$, y–values are generated from $y_i = x_i^2 + \xi$, were ξ is normal noise with standard deviation 0.2. From left to right, the polynomial degree in the kernel (22) increases 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 space. Note that linear PCA only leads to 2 nonzero Eigenvalues, as the input dimensionality is 2. In contrast, nonlinear PCA uses the third component to pick up the variance caused by the noise, as can be seen in the case of degree 2.

# 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 1: Test error rates on the MPI chair database for linear Support Vector machines trained on nonlinear principal components extracted by PCA with kernel (22), 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.

Support Vector machines, which on this data set achieved 8% (Blanz et al. 1996).

6.3 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 (USPS) database of handwritten digits collected from mail envelopes in Buffalo.[8] This database contains 9300 examples of dimensionality 256; 2000 of them make up the test set. For computational reasons, we decided to use a subset of 3000 training examples for the dot product matrix. Polynomial kernel PCA followed by a linear Support Vector classifier leads to performance superior to PCA (see Table 2). 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, Burges, & Vapnik, 1995); it is far superior to linear classifiers operating directly on the image data (a linear Support Vector machine achieves 8.9%; Schölkopf, Burges, & Vapnik, 1995). We should add that our results were obtained without using any prior knowledge about symmetries of the problem at hand. This explains why the performance is inferior to Virtual Support Vector classifiers (3.2%, Schölkopf, Burges, & Vapnik, 1996), and Tangent Distance Nearest Neighbour classifiers (2.6%, Simard, LeCun, & Denker, 1993). We believe that adding e.g. local translation invariance, be it by generating "virtual" translated examples or by choosing a suitable kernel, could further improve the results.

Table 2 nicely illustrates two advantages of using nonlinear kernels: first, performance for 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.[9]

We conclude this section with a comment on the approach taken. 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. This was not done in our experiments, nevertheless good results were obtained. We hope that taking into account the labels could further improve performance.

Finally, we note that a similar approach can be taken in the case of regression estimation. We conjecture that also in that case, nonlinear principal components will be more useful for some problems than standard PCA regression (e.g. Joliffe, 1986).

7 Nonlinear Variants of Other Algorithms

As pointed out in Sec. 1, we can use the kernel method to construct nonlinear variants of any algorithm, as long as it can be cast in terms of dot products. Clearly, it is beyond the scope of the present paper to explore all the possibilities in detail. Instead, we shall just give a few examples, and point out some possibilities for future work.

7.1 Projection Pursuit, Independent Components, and Higher Order Moments

In order to do Projection Pursuit (Friedman, 1987) and Independent component Analysis (ICA) (Jutten & Herault, 1991, Bell & Sejnowski, 1995), we are, loosely speaking, looking for directions in the dataset such that the projections onto these are maximally "non–Gaussian". The rationale behind this approach is the following: assume that the probability distribution underlying the data factorizes in some set of variables. Then its projections onto directions *different* from the above variables will tend to look more Gaussian, as the basis transformation introduces mixtures. Conversely, if we start from mixtures, we thus should strive to find axis tranformations to make them as non–Gaussian as possible. To find non–Gaussian directions, we need to take into account higher–order statistics of the data.

Kernel PCA provides a convenient way of doing this: using e.g. polynomial kernels of degree d (22), we are taking into account d–th order statistics. In order to get a decomposition of our observations into independent components, we thus need to find directions in input space which correspond as closely as possible to directions in F which lead to extremal values of some higher–order moments, specified by a nonlinear kernel (cf. Sec. 4.6).

In this sense, already linear forms of ICA require the use of nonlinearities. We can, however, go one step further and construct tools for *nonlinear* ICA and *nonlinear* Projection Pursuit. To this end, we

[8] Thanks to AT&T and Bell Laboratories for the possibility of using this database.

[9] If the linear classifier trained on the components had not been an SV machine, this might look different — SV machines are known to possess high generalization ability for high–dimensional data, due to their built–in capacity control (Vapnik, 1995).

# 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 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 (22), 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. Clearly, nonlinear principal components afford test error rates which are superior to the linear case (degree 1).

describe a way to compute higher–order moments in feature space.

The same procedure used for computing the covariance matrix $\bar{C}$ also applies for moments of order higher than 2. Tensors belonging to moments of order p may be written as[10]

$$C^p = \frac{1}{M}\sum_{i=1}^{M} \underbrace{\Phi(\mathbf{x}_i)\otimes \ldots \otimes \Phi(\mathbf{x}_i)}_{p \text{ times}} \qquad (32)$$

where $\otimes$ denotes the outer product. Analogous to Sec. 2, C^p has to be rewritten in terms of dot products. In the subspace of F spanned by the images $\Phi(\mathbf{x}_i)$, the tensor C^p is determined by its projections onto the eigenvectors $\mathbf{V}_i$ of $\bar{C}$,

$$\begin{aligned} P_{j_1\ldots j_p} &:= C^p\cdot\left(V^{j_1}\otimes\ldots\otimes V^{j_p}\right) & (33)\\ &= \frac{1}{M}\sum_{i=1}^{M}\prod_{k=1}^{p}\left(\Phi(\mathbf{x}_i), \mathbf{V}^{j_k}\right). & (34)\end{aligned}$$

Here, we have used the inner product (33) between two tensors. Using the expansion (8)

$$\mathbf{V}^{j_k} = \sum_{m=1}^{M}\alpha_m^{j_k}\Phi(\mathbf{x}_m),$$

we get

$$\left(\Phi(\mathbf{x}_i), \mathbf{V}^{j_k}\right) = \sum_m K_{im}\alpha_m^{j_k}$$

thus

$$P_{j_1\ldots j_p} = \frac{1}{M}\sum_{i=1}^{M}\prod_{k=1}^{p}\sum_{m=1}^{M}K_{im}\alpha_m^{j_k} \qquad (35)$$

[10]This requires the existence of these moments, which need not always be the case, as they are the Taylor expansion of the characteristic function of the probability density function in feature space.

It may be advisable to sum only over the M' eigenvectors corresponding to the biggest eigenvalues. Otherwise, C^p is of size M^p, with M being the number of samples used for computing the dot product matrix. This can be computationally infeasible for large p.

7.2 Kernel–k–Means Clustering

The kernel approach is applicable to clustering techniques, as long as both the clustering algorithm and the clustering result's usage can be cast in terms of dot products. As an example, we consider k–means, but the same reasoning holds for other algorithms (for an overview of clustering techniques, see Buhmann, 1995).

Let $M_{i\nu}$ be the cluster assignment variables, i.e. $M_{i\nu} = 1$ if $\mathbf{x}_i$ belongs to cluster ν, 0 otherwise. We are trying to find k centers (or means) $\mathbf{m}_\nu$ such that each observation in the training set is close to at least one of the centers. Clearly, the centers should lie in the span of $\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_M)$: assume this was not the case for $\mathbf{m}_\alpha$, say. Then we could project $\mathbf{m}_\alpha$ to the above span, and by Pythagoras reduce the distances to all observations. We therefore expand them as

$$\mathbf{m}_\nu = \sum_{j=1}^{M}\gamma_{\nu j}\Phi(\mathbf{x}_j), \qquad (36)$$

and note that the squared distance between $\mathbf{m}_\nu$ and a mapped pattern $\Phi(\mathbf{x})$ can be expressed as

$$\begin{aligned} &\|\Phi(\mathbf{x}) - \sum_{j=1}^{M}\gamma_{\nu j}\Phi(\mathbf{x}_j)\|^2 \\ = \quad & k(\mathbf{x},\mathbf{x}) - 2\sum_{j=1}^{M}\gamma_{\nu j}k(\mathbf{x},\mathbf{x}_j)\end{aligned}$$

$$+\sum_{i,j=1}^{M}\gamma_{\nu i}\gamma_{\nu j}k(\mathbf{x}_i,\mathbf{x}_j). \quad (37)$$

We initialize the means to the first training patterns, i.e. $\gamma_{ij} = \delta_{ij}$. Then Kernel–$k$–means proceeds as follows: each new data point $\mathbf{x}_{t+1}, t \geq k$, is assigned to the closest mean $\mathbf{m}_\alpha$, i.e.

$$M_{t+1,\alpha} = \begin{cases} 1 & \text{if for all } \nu \neq \alpha \\ & \|\Phi(\mathbf{x}_{t+1}) - \mathbf{m}_\alpha\|^2 \\ & < \|\Phi(\mathbf{x}_{t+1}) - \mathbf{m}_\nu\|^2 \\ 0 & \text{otherwise,} \end{cases} \quad (38)$$

or, in terms of the kernel function,

$$M_{t+1,\alpha} = \begin{cases} 1 & \text{if} \quad \sum_{i,j=1}^{M}\gamma_{\alpha i}\gamma_{\alpha j}k(\mathbf{x}_i,\mathbf{x}_j) \\ & \quad - \quad 2\sum_{j=1}^{M}\gamma_{\alpha j}k(\mathbf{x}_{t+1},\mathbf{x}_j) \\ & \quad < \quad \sum_{i,j=1}^{M}\gamma_{\nu i}\gamma_{\nu j}k(\mathbf{x}_i,\mathbf{x}_j) \\ & \quad - \quad 2\sum_{j=1}^{M}\gamma_{\nu j}k(\mathbf{x}_{t+1},\mathbf{x}_j) \\ & \quad \text{for all } \nu \neq \alpha \\ 0 & \quad \text{otherwise.} \end{cases} \quad (39)$$

The expansion coefficients of the closest mean vector are then adjusted according to

$$\mathbf{m}_\alpha^{t+1} = \mathbf{m}_\alpha^t + \zeta\ (\Phi(\mathbf{x}_{t+1}) - \mathbf{m}_\alpha^t), \quad (40)$$

where

$$\zeta := \frac{M_{t+1,\alpha}}{\sum_{i=1}^{t+1} M_{i\alpha}}. \quad (41)$$

Substituting (36) back into (40)

$$\sum_{j=1}^{M}\gamma_{\alpha j}^{t+1}\Phi(\mathbf{x}_j) = \sum_{j=1}^{M}\gamma_{\alpha j}^{t}(1-\zeta)\Phi(\mathbf{x}_j) + \zeta\Phi(\mathbf{x}_{t+1}) \quad (42)$$

and sorting the terms for different $\Phi(\mathbf{x}_j)$ yields an update equation for $\gamma_{\alpha i}^{t+1}$,

$$\gamma_{\alpha j}^{t+1} = \begin{cases} \gamma_{\alpha j}^{t}(1-\zeta) & \text{for } j \neq t+1 \\ \zeta & \text{for } j = t+1 \end{cases} \quad (43)$$

Once the clustering procedure has converged for an initial data set, we want to be able to determine the distance of a test point to the cluster centers in F. To this end, we again use Eq. (37).

7.3 Classification, Image Indexing and Retrieval

Clearly, distance–based algorithms like k–NN can be easily recast in the nonlinear kernel framework. In addition, and more interesting, it would be desirable to develop nonlinear forms of discriminant analysis based on kernels. A related approach, using an explicit map into a higher–dimensional space instead of the kernel method, was proposed by Hastie, Tibshirani, & Buja (1994).

PCA has been successfully used for face recognition (Turk & Pentland, 1991) and face representation (Vetter & Poggio, 1995). Features with good approximative quality, however, are not always good descriptive features, as irrelevant information for the solution of a given problem might be encoded in the data, too. Swets and Weng (1996) show a solution to this problem by using PCA as a preprocessing step only. Discriminant Analysis is applied in a second step to compute linear combinations that are useful for extracting meaningful features. PCA, however, has the shortcoming of only being able to take into account second order correlations between pixels. Nonlinear Component Analysis provides a drop–in replacement which also takes into account higher order correlations. We propose this as a method for increasing the precision of image indexing and retrieval systems. It combines the flexibility of nonlinear feature extractors with the simplicity of a principal axis projection.

8 Discussion

Constructing Nonlinear Algorithms. This paper was devoted to the exposition of a new technique for nonlinear principal component analysis. To develop this technique, we made use of a kernel method which so far only had been used in supervised learning (Vapnik, 1995). We think that the present work will not be the last application of the kernel method in constructing a rather general and still feasible nonlinear variant of a classical algorithm. Indeed, we have mentioned and, in part, outlined a number of techniques which could be made nonlinear with essentially the same method. Some of them are currently under investigation, in particular the development of ICA and clustering techniques.

Feature Space and the Curse of Dimensionality. Some readers may be puzzled by the fact that 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. The solution is that in fact, 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. An important aspect of the proposed method is that this subspace is chosen automatically, without even knowing the mapping into feature space.

Still, working in a space whose dimension equals the number of observations can pose difficulties. To deal with these, one can either use only a subset of the extracted features, or use some other form of capacity control or regularization. In our case, this was done by using a Support Vector machine which automatically controls capacity in constructing the decision surface for classification.

Empirical Issues. The main goal of the present paper was to present an idea for nonlinear PCA; however we have not compared the extracted features 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. Our system of kernel PCA feature extraction plus linear Support Vector machine for instance 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 contains 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 as preprocessing in their experiments in local learning (1992).

Even though this result is promising, a full experimental evaluation of kernel PCA and other kernel–based methods remains to be carried out.

Main Points of Kernel PCA. Our experiments suggest that compared to linear PCA, kernel PCA does extract features which are more useful for classification purposes. In addition, kernel PCA has the a priori advantage that it gives us the possibility to extract more principal components than linear PCA — which is not too astonishing in the case of polynomial kernels, as there are many more higher–order features than there are pixels in an image. Indeed, it is astonishing that classification works so well already with a number of principal components which (even though it is large compared to the number of linear principal components) is very small compared to the dimensionality of feature space. In fact, our results show that to get the same classification performance as in the linear case, we need fewer nonlinear principal components. The main drawback of kernel PCA compared to linear PCA is that up to date, we do not have a simple method for reconstructing patterns from their principal components. There exist, however, some ideas, discussed in Sec. 4.6.

Compared to other techniques for nonlinear feature extraction, kernel PCA has the advantages that (1) it does not require nonlinear optimization, but just the solution of an Eigenvalue problem, and (2) 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 practise, however, for the Support Vector machine, the utility of different kernels has already been established (Schölkopf, Burges, & Vapnik, 1995).

Possible Applications. Linear PCA is being used in numerous technical and scientific applications. As some further examples not discussed in the present paper so far, we mention noise reduction, density estimation, and the analysis of natural image statistics. Kernel PCA can be applied to all domains where traditional PCA has been used for feature extraction before, with little extra computational effort.

A Centering in High–Dimensional Space

In Sec. 2, we made the assumption that our mapped data is centered in F, i.e.

$$\sum_{n=1}^{M} \Phi(\mathbf{x}_n) = 0. \tag{44}$$

We shall now drop this assumption. First note that given any Φ and any set of observations $\mathbf{x}_1, \ldots, \mathbf{x}_M$, the points

$$\tilde{\Phi}(\mathbf{x}_i) := \Phi(\mathbf{x}_i) - \frac{1}{M}\sum_{i=1}^{M} \Phi(\mathbf{x}_i) \tag{45}$$

will be centered. Thus, the assumptions of Sec. 2 now hold, and we go on to define covariance matrix and dot product matrix $\tilde{K} = \tilde{\Phi}(\mathbf{x}_i)^\top \tilde{\Phi}(\mathbf{x}_j)$ in F. We arrive at our already familiar Eigenvalue problem

$$\tilde{\lambda}\tilde{\boldsymbol{\alpha}} = \tilde{K}\tilde{\boldsymbol{\alpha}}, \tag{46}$$

with $\tilde{\boldsymbol{\alpha}}$ being the expansion coefficients of an Eigenvector (in F) in terms of the points (45),

$$\tilde{\mathbf{V}} = \sum_{i=1}^{M} \tilde{\alpha}_i \tilde{\Phi}(\mathbf{x}_i). \tag{47}$$

As we do not have the centered data (45), 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 $K(\mathbf{x}_i, \mathbf{x}_j) =$

$\Phi(\mathbf{x}_i)^\top\Phi(\mathbf{x}_j)$, in addition, we shall make use of the notation $1_{ij} = 1$ for all i, j.

$$\begin{aligned}
\tilde{K}_{ij} &= \tilde{\Phi}(\mathbf{x}_i)^\top\tilde{\Phi}(\mathbf{x}_j) \quad (48)\\
&= (\Phi(\mathbf{x}_i) - \sum_{m=1}^{M}\Phi(\mathbf{x}_m))^\top(\Phi(\mathbf{x}_j) - \sum_{n=1}^{M}\Phi(\mathbf{x}_n))^\top\\
&= \Phi(\mathbf{x}_i)^\top\Phi(\mathbf{x}_j) - \frac{1}{M}\sum_{m=1}^{M}\Phi(\mathbf{x}_m)^\top\Phi(\mathbf{x}_j)\\
&\quad -\frac{1}{M}\sum_{n=1}^{M}\Phi(\mathbf{x}_i)^\top\Phi(\mathbf{x}_n)\\
&\quad +\frac{1}{M^2}\sum_{m,n=1}^{M}\Phi(\mathbf{x}_m)^\top\Phi(\mathbf{x}_n)\\
&= K_{ij} - \frac{1}{M}\sum_{m=1}^{M}1_{im}K_{mj}\\
&\quad -\sum_{n=1}^{M}K_{in}1_{nj} + \frac{1}{M^2}\sum_{m,n=1}^{M}1_{im}K_{mn}1_{nj}
\end{aligned}$$

Using the matrix $(1_M)_{ij} := 1/M$, we get the more compact expression

$$\tilde{K}_{ij} = K - 1_M K - K 1_M + 1_M K 1_M. \quad (49)$$

We thus can compute $\tilde{K}$ from K, and then solve the Eigenvalue problem (46). As in (16), the solutions $\tilde{\boldsymbol{\alpha}}^k$ are normalized by normalizing the corresponding vectors $\tilde{\mathbf{V}}^k$ in F, which translates into

$$\tilde{\lambda}_k(\tilde{\boldsymbol{\alpha}}^k \cdot \tilde{\boldsymbol{\alpha}}^k) = 1. \quad (50)$$

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,

$$\begin{aligned}
(\tilde{\mathbf{V}}^k \cdot \Phi(\mathbf{t})) &= \sum_{i=1}^{M}\tilde{\alpha}_i^k(\tilde{\Phi}(\mathbf{x}_k)\cdot\tilde{\Phi}(\mathbf{t}))\\
&= \sum_{i=1}^{M}\tilde{\alpha}_i^k\tilde{K}(\mathbf{x}_k, \mathbf{t}). \quad (51)
\end{aligned}$$

Consider a set of test points $\mathbf{t}_1, \ldots, \mathbf{t}_L$, and define two $L \times M$ test dot product matrices by

$$K_{ij}^{test} = (\Phi(\mathbf{t}_i)\cdot\Phi(\mathbf{x}_j)) \quad (52)$$

and

$$\tilde{K}_{ij}^{test} = ((\Phi(\mathbf{t}_i) - \frac{1}{M}\sum_{m=1}^{M}\Phi(\mathbf{x}_m))\cdot(\Phi(\mathbf{x}_j) - \frac{1}{M}\sum_{n=1}^{M}\Phi(\mathbf{x}_n))). \quad (53)$$

Similar to (49), we can then again express $\tilde{K}^{test}$ in terms of K^{test}, and arrive at

$$\tilde{K}^{test} = K^{test} - 1'_M K - K^{test}1_M + 1'_M K 1_M, \quad (54)$$

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

B Addenda on Kernels

B.1 Polynomial Kernels and Higher Order Correlations

Consider the mappings corresponding to kernels of the form (22): suppose the monomials $x_{i_1}x_{i_2}\ldots x_{i_d}$ are written such that $i_1 \leq i_2 \leq \ldots \leq i_d$. Then the coefficients (as the $\sqrt{2}$ in Eq. (20)), arising from the fact that different combinations of indices occur with different frequencies, are largest for $i_1 < i_2 < \ldots < i_d$ (let us assume here that the input dimensionality is not smaller than the polynomial degree d): in that case, we have a coefficient of $\sqrt{d!}$. If $i_1 = i_2$, say, the coefficient will be $\sqrt{(d-1)!}$. In general, if n of the x_i are equal, then the coefficient in the corresponding component of Φ is $\sqrt{(d-n+1)!}$. Thus, the terms belonging to the d-th order correlations will be weighted with an extra factor $\sqrt{d!}$ compared to the terms x_i^d, and compared to the terms where only $d-1$ different components occur, they are still weighted stronger by $\sqrt{d}$. Consequently, kernel PCA with polynomial kernels will tend to pick up variance in the d-th order correlations mainly.

B.2 Kernels Corresponding to Dot Products in Another Space

Mercer's theorem of functional analysis (e.g. Courant & Hilbert, 1953) gives the conditions under which we can construct the mapping Φ from the Eigenfunction decomposition of k. Namely, if k is the continuous kernel of an integral operator $\mathcal{K}$,

$$\begin{aligned}
\mathcal{K} : L_2 &\to L_2\\
f &\mapsto \mathcal{K}f\\
(\mathcal{K}f)(y) &= \int k(x,y)f(x)\,dx \quad (55)
\end{aligned}$$

which is positive definite, i.e.

$$\int f(x)k(x,y)f(y)\,dx\,dy > 0 \text{ if } f \neq 0, \quad (56)$$

then k can be expanded into a series

$$k(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{\infty}\lambda_i\phi_i(\mathbf{x})\phi_i(\mathbf{y}) \quad (57)$$

with positive coefficients λ_i, and

$$(\phi_i\cdot\phi_j)_{L_2} = \delta_{ij} \quad (58)$$

for $i, j \in \mathbf{N}$. (In other words, the compact operator $\mathcal{K}$ has an Eigenvector decomposition with nonnegative Eigenvalues.) Using (58), it is then easy to see that

$$\Phi(\mathbf{x}) := \sum_{i=1}^{\infty}\sqrt{\lambda_i}\phi_i(\mathbf{x}) \quad (59)$$

is a map into a space where k acts as the Euclidean dot product,[11] i.e.

$$(\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})) = k(\mathbf{x}, \mathbf{y}). \qquad (60)$$

In fact, for the latter to hold, k does not have to be the kernel of a positive *definite* operator: even if some of the Eigenvalues λ_i are zero, the sum (59) maps into the space in which k corresponds to the dot product.

In practise, we are free to try to use also symmetric kernels of indefinite operators. In that case, the matrix K can still be diagonalized and we can extract nonlinear feature values, with the one modification that we need to modify our normalization condition (16) in order to deal with possible negative Eigenvalues. K then induces a mapping to a Riemannian space with indefinite metric. This implies that we no longer can require vectors to have positive length but unit–length only. Hence, normalization is done by multiplying by $\frac{1}{\sqrt{|\lambda_i|}}$. This criterion is much less strict than requiring K to be a positive semidefinite Hilbert–Schmidt kernel. In fact, many symmetric forms may induce spaces with indefinite signature.[12] In this case, we can no longer interpret our method as PCA in some feature space; however, it could still be viewed as a nonlinear factor analysis.[13]

In the following sections, we shall give some examples of kernels that can be used for kernel PCA. Our treatment will be concise; further elaborations and experimental evaluations will be dealt with in future investigations.

B.3 Kernels Chosen A Priori

As examples of kernels which can be used, Boser, Guyon, & Vapnik (1992) give polynomial kernels of the form[14]

$$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + 1)^d \qquad (61)$$

radial basis functions

$$k(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2\,\sigma^2}\right), \qquad (62)$$

and Neural Network type kernels[15]

$$k(\mathbf{x}, \mathbf{y}) = \tanh((\mathbf{x} \cdot \mathbf{y}) + b). \qquad (63)$$

Interestingly, these different types of kernels allow the construction of Polynomial Classifiers, Radial Basis Function Classifiers and Neural Networks with the Support Vector algorithm which exhibit very similar accuracy. In addition, they all construct their decision functions from an almost identical subset of a small number of training patterns, the Support Vectors (Schölkopf, Burges, & Vapnik, 1995).

Besides the above stated kernels, there is a variety of other kernels which can be used in order to tailor the type of nonlinearity used to the problem at hand, as those given in the following section.

B.4 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, $\Phi : \mathbf{R}^N \to F$. 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,

$$\mathbf{x} \mapsto f_{\mathbf{x}}, \qquad (64)$$

with $f_{\mathbf{x}}$ being a complex–valued function on some measure space. We furthermore assume that these spaces are equipped with a dot product

$$(f_{\mathbf{x}} \cdot f_{\mathbf{y}}) = \int f_{\mathbf{x}}(u)\; f_{\mathbf{y}}(u)\;\; du. \qquad (65)$$

We can then define kernels of the type

$$k(\mathbf{x}, \mathbf{y}) := (f_{\mathbf{x}} \cdot f_{\mathbf{y}})^d. \qquad (66)$$

(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,

$$q^2 \int_0^1 \int_0^1 f_{\mathbf{x}}(u) f_{\mathbf{y}}(u)\; d^2u \qquad (67)$$

[11] It is not the only one — e.g., sign reversal in some of the summands gives other maps corresponding to the same kernel.

[12] suggested by Y. LeCun

[13] 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 optimization procedure.

[14] Note that this kernel generates not only homogeneous polynomials, but all polynomials of degree up to d. To assign variable weight to the different degrees of the monomials which appear if we compute (22) or (61), we can use $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + c)^d$ with different values of c. The choice of c should depend on the range of the input variables.

[15] The two last kernels allow the construction of neural network type feature extraction systems.

$$\approx \sum_{i=1}^{q}\sum_{j=1}^{q} f_{\mathbf{x}}\left(\frac{i-\frac{1}{2}}{q}, \frac{j-\frac{1}{2}}{q}\right) f_{\mathbf{y}}\left(\frac{i-\frac{1}{2}}{q}, \frac{j-\frac{1}{2}}{q}\right). \tag{68}$$

B.5 Scaling with Kernels

We consider two forms of scaling, in images and in input space, respectively. The former can be achieved with Kernels of the form (68). This is useful if we for instance would like to process image patterns which were taken at different resolution levels q. The second form of scaling, in input space, can be useful if different input variables strongly vary in their range. Given a kernel k which is a function of $(\mathbf{x}\cdot\mathbf{y})$, and a diagonal matrix D with nonnegative diagonal elements $d_1, \ldots, d_N$, we can construct another kernel

$$k_D(\mathbf{x}, \mathbf{y}) := k(\mathbf{x} \cdot D\mathbf{y}) \tag{69}$$

which effectively scales direction i of input space by $\sqrt{d_i}$.[16]

B.6 Local Kernels

Locality in our context means that the principal component extraction should take into account only neighbourhoods. Depending on whether we consider neighbourhoods in input space or in another space, say the image space, where the input vectors correspond to 2–d functions (the functions in (68)), locality can assume different meanings. In input space, locality consists of basing our component extraction for a point $\mathbf{x}$ on other points in an appropriately chosen neighbourhood of $\mathbf{x}$. This additional degree of freedom can greatly improve statistical estimates which are computed from a limited amount of data (Bottou & Vapnik, 1992).

In image space, locality is not a statistical concept, but a tool to incorporate domain knowledge. Suppose we want to take into account correlations of d pixels, but only if these pixels are close to each other.

To get locality in image space, we could modify (67), (66) by adding a suitable smoothing kernel $c(u, v)$ to get

$$k(\mathbf{x}, \mathbf{y}) = \int f_{\mathbf{x}}(u)\, c(u, v)\, f_{\mathbf{y}}(v)\ \ d^2u\, d^2v. \tag{70}$$

Rather than raising this expression to the d-th power as in (66), which would introduce taking products which are far distant in the image, we can split the domain of integration into r small regions, thus getting

$$k(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{r} k_i(\mathbf{x}, \mathbf{y}), \tag{71}$$

and then introduce *local* d-th order correlations by using

$$k(\mathbf{x}, \mathbf{y}) = \sum_{i} k_i(\mathbf{x}, \mathbf{y})^d. \tag{72}$$

Of course, the corresponding kernels can also be written in terms of the original image vectors, splitting them into subvectors before raising the subvector–kernels to the d-th power.[17]

B.7 Constructing Kernels from other Kernels

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.

Iterating Kernels. Given a kernel k, we can construct iterated kernels (e.g. Riesz & Nagy, 1955) by

$$k^{(2)}(\mathbf{x}, \mathbf{y}) := \int k(\mathbf{x}, \mathbf{z}) k(\mathbf{z}, \mathbf{y})\ d\mathbf{z}. \tag{73}$$

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

$$\begin{aligned} & \int k^{(2)}(\mathbf{x}, \mathbf{y}) f(\mathbf{x}) f(\mathbf{y})\ d\mathbf{x}d\mathbf{y} \\ = & \int\int k(\mathbf{x}, \mathbf{z}) k(\mathbf{z}, \mathbf{y}) f(\mathbf{x}) f(\mathbf{y})\ d\mathbf{z}d\mathbf{x}d\mathbf{y} \\ = & \int \left(\int k(\mathbf{x}, \mathbf{z}) f(\mathbf{x})\ d\mathbf{x}\right)^2 d\mathbf{z}. \end{aligned} \tag{74}$$

This gives us a method for constructing admissible kernels.

[16] Together with C. Burges, the first two authors are currently studying the use of kernels of the form $k(\mathbf{x}, \mathbf{y}) = (A\mathbf{x}, A\mathbf{y})^d$, with A being an $L \times N$ matrix, in Support Vector machines. This includes the above kernels as special cases.

[17] thanks to P. Simard for this suggestion

References

[1] M. A. Aizerman, E. M. Braverman, and L. I. Rozonoér. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

[2] A. Bell and T. Sejnowski. An information–maximization approach to blind separation and blind deconvolution. *Neural Computation*, 7:1129 – 1159, 1995.

[3] V. Blanz, B. Schölkopf, H. Bülthoff, C. Burges, V. Vapnik, and T. Vetter. Comparison of view–based object recognition algorithms using realistic 3d models. In C. von der Malsburg, W. von Seelen, J. C. Vorbrüggen, and B. Sendhoff, editors, *Artificial Neural Networks — ICANN'96*, pages 251 – 256, Berlin, 1996. Springer Lecture Notes in Computer Science, Vol. 1112.

[4] B. E. Boser, I. M. Guyon, and V .Vapnik. A training algorithm for optimal margin classifiers. In *Fifth Annual Workshop on Computational Learning Theory*, Pittsburgh, 1992. ACM.

[5] L. Bottou and V. N. Vapnik. Local learning algorithms. *Neural Computation*, 4(6):888–900, 1992.

[6] J. M. Buhmann. Data clustering and learning. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 278–281. MIT Press, 1995.

[7] C. J. C. Burges. Private communication. 1996.

[8] C. J. C. Burges. Simplified support vector decision rules. In *13th International Conference on Machine Learning*, 1996.

[9] C. J. C. Burges and B. Schölkopf. Improving the accuracy and speed of support vector learning machines. In *Advances in Neural Information Processing Systems 9*, 1996. forthcoming.

[10] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273 – 297, 1995.

[11] R. Courant and D. Hilbert. *Methods of Mathematical Physics*, volume 1. Interscience Publishers, Inc, New York, 1953.

[12] Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. J. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541 – 551, 1989.

[13] K. I. Diamantaras and S. Y. Kung. *Principal Component Neural Networks*. Wiley, New York, 1996.

[14] J. H. Friedman. Exploratory projection pursuit. *Journal of the American Statistical Association*, 82:249–266, 1987.

[15] T. Hastie and W. Stuetzle. Principal curves. *JASA*, 84:502 – 516, 1989.

[16] T. Hastie, R. Tibshirani, and A. Buja. Flexible discriminant analysis. *JASA*, 89:1255 – 1270, 1994.

[17] I. T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, New York, New York, 1986.

[18] C. Jutten and J. Herault. Blind separation of sources, part I: an adaptive alorithm based on neuromimetic architecture. *Signal Processing*, 24:1 – 10, 1991.

[19] K. Karhunen. Zur Spektraltheorie stochastischer Prozesse. *Ann. Acad. Sci. Fenn.*, 34, 1946.

[20] M. Kirby and L. Sirovich. Application of the Karhunen–Loève procedure for the characterization of human faces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-12(1):103–108, January 1990.

[21] E. Oja. A simplified neuron model as a principal component analyzer. *J. Math. Biology*, 15:267 – 273, 1982.

[22] T. Poggio. On optimal nonlinear associative recall. *Biological Cybernetics*, 19:201–209, 1975.

[23] F. Riesz and B. Sz. Nagy. *Functional Analysis*. Frederick Ungar Publishing Co., 1955.

[24] B. Schölkopf, C. Burges, and V. Vapnik. Extracting support data for a given task. In U. M. Fayyad and R. Uthurusamy, editors, *Proceedings, First International Conference on Knowledge Discovery & Data Mining*, Menlo Park, CA, 1995. AAAI Press.

[25] B. Schölkopf, C. Burges, and V. Vapnik. Incorporating invariances in support vector learning machines. In C. von der Malsburg, W. von Seelen, J. C. Vorbrüggen, and B. Sendhoff, editors, *Artificial Neural Networks — ICANN'96*, pages 47 – 52, Berlin, 1996. Springer Lecture Notes in Computer Science, Vol. 1112.

[26] P. Simard, Y. LeCun, and J. Denker. Efficient pattern recognition using a new transformation distance. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, San Mateo, CA, 1993. Morgan Kaufmann.

[27] D. L. Swets and J. Weng. Using discriminant Eigenfeatures for image retrieval. *IEEE Trans. on PAMI*, 18:831 – 836, 1996.

[28] M. Turk and A. Pentland. Eigenfaces for recognition. *J. Cognitive Neuroscience*, 3(1):71–86, 1991.

[29] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, New York, 1995.

[30] V. Vapnik and A. Chervonenkis. *Theory of Pattern Recognition [in Russian]*. Nauka, Moscow, 1974. (German Translation: W. Wapnik & A. Tscherwonenkis, *Theorie der Zeichenerkennung*, Akademie–Verlag, Berlin, 1979).

[31] T. Vetter and T. Poggio. Image synthesis from a single example image. In *Proceedings of the European Conference on Computer Vision*, 1996.