The final version of this article is published in *Neural Computation*,
Vol. 10, Issue 5, pp. 1299 – 1319, 1998, The MIT Press.
(First version: Tech. Rep. 44, Max-Planck-Institut für biologische Kybernetik, 1996)

Nonlinear Component Analysis as a Kernel Eigenvalue Problem

Bernhard Schölkopf
Max-Planck-Institut für biologische Kybernetik
Spemannstr. 38, 72076 Tübingen, Germany

Alexander Smola, Klaus-Robert Müller
GMD First (Forschungszentrum Informationstechnik)
Rudower Chaussee 5, 12489 Berlin, Germany

Abstract. *A new method for performing a nonlinear form of Principal Component Analysis is proposed. By the use of integral operator kernel functions, 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 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 Diamantaras & 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 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. 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, private communication), the machine learning community has made little use of it, the exception being Support Vector machines (Vapnik, 1995). In this paper, 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. In Sec. 3, we 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. First experimental results on kernel-based feature extraction for pattern recognition are given in Sec. 5. We conclude with a discussion (Sec. 6) and an appendix containing some technical material that is not essential for the main thread of the argument.

2 PCA in Feature Spaces

Given a set of 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 $\mathbf{v}$ with $\lambda \neq 0$ must lie in the span of $\mathbf{x}_1 \ldots \mathbf{x}_M$, hence (2) in that case is equivalent to

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

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

$$\Phi : \mathbf{R}^N \to F, \quad \mathbf{x} \mapsto \mathbf{X}. \tag{4}$$

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 assume 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 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

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

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

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

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 $\alpha_1, \ldots, \alpha_M$. To find solutions of (11), we solve the eigenvalue problem

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

for nonzero eigenvalues. A justification of this procedure is given in Appendix A.

Let $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_M$ denote the eigenvalues of K (i.e. the solutions $M\lambda$ of (12)), and $\boldsymbol{\alpha}^1, \ldots, \boldsymbol{\alpha}^M$ the corresponding complete set of eigenvectors, with λ_p being the first nonzero eigenvalue (assuming $\Phi \not\equiv 0$). We normalize $\boldsymbol{\alpha}^p, \ldots, \boldsymbol{\alpha}^M$ by requiring that the corresponding vectors in F be normalized, i.e.

$$(\mathbf{V}^k \cdot \mathbf{V}^k) = 1 \text{ for all } k = p, \ldots, M. \tag{13}$$

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

$$\begin{aligned} 1 &= \sum_{i,j=1}^{M} \alpha_i^k \alpha_j^k (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)) = \sum_{i,j=1}^{M} \alpha_i^k \alpha_j^k K_{ij} \\ &= (\boldsymbol{\alpha}^k \cdot K\boldsymbol{\alpha}^k) = \lambda_k (\boldsymbol{\alpha}^k \cdot \boldsymbol{\alpha}^k) \end{aligned} \tag{14}$$

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

$$(\mathbf{V}^k \cdot \Phi(\mathbf{x})) = \sum_{i=1}^{M} \alpha_i^k (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x})) \tag{15}$$

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.[2]

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 harder in F, as we cannot explicitely compute the mean of the $\Phi(\mathbf{x}_i)$ in F. There is, however, a way to do it, and this leads to slightly modified equations for kernel-based PCA (see Appendix B).

Before we proceed to the next section, which more closely investigates the role of the map Φ, the following observation is essential: Φ can be an arbitrary nonlinear map into the possibly high-dimensional space F, for example, 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, 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 (15)); *we never need the mapped patterns explicitely.*

3 Computing Dot Products in Feature Spaces

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

$$k(\mathbf{x}, \mathbf{y}) = (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})), \tag{16}$$

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 k for all occurences of dot products, obtaining decision functions

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{\ell} \nu_i k(\mathbf{x}, \mathbf{x}_i) + b\right). \tag{17}$$

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

[2] 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 B, we shall consider the question how to center the data in F), we gave a detailed derivation.

proved by Poggio (1975, Lemma 2.1) in the context of polynomial approximation, is

$$(\mathbf{x}\cdot\mathbf{y})^d = \left(\sum_{j=1}^{N} x_j \cdot y_j\right)^d = \sum_{j_1,\ldots,j_d=1}^{N} x_{j_1}\cdot\ldots\cdot x_{j_d}\cdot y_{j_1}\cdot\ldots\cdot y_{j_d} = (C_d(\mathbf{x})\cdot C_d(\mathbf{y})), \quad (18)$$

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 $\mathbf{x}$. 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,

$$\Phi_2(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}\, x_1x_2). \quad (19)$$

For this example, it is easy to verify that $(\mathbf{x}\cdot\mathbf{y})^2 = (x_1^2, x_2^2, \sqrt{2}\, x_1x_2)(y_1^2, y_2^2, \sqrt{2}\, y_1y_2)^\top = (\Phi_2(\mathbf{x})\cdot\Phi_2(\mathbf{y}))$. In general, the function

$$k(\mathbf{x},\mathbf{y}) = (\mathbf{x}\cdot\mathbf{y})^d \quad (20)$$

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 $\Phi_d(\mathbf{x})$. 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 (19), the dimensionality of F still is $\frac{(N+d-1)!}{d!(N-1)!}$ and thus grows like N^d. For instance, 16×16 pixel input images and a polynomial degree $d = 5$ yield a dimensionality of 10^{10}. Thus, using kernels of the form (20) is our only way to take into account higher-order statistics without a combinatorial explosion of time and memory complexity.

The general question which function k does correspond 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 implies that if k is a continuous kernel of a positive integral operator, there exists a mapping into a space where k acts as a dot product (for details, see Appendix C). Besides (20), radial basis functions

$$k(\mathbf{x},\mathbf{y}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\,\sigma^2}\right) \quad (21)$$

and sigmoid kernels

$$k(\mathbf{x},\mathbf{y}) = \tanh(\kappa(\mathbf{x}\cdot\mathbf{y}) + \Theta). \quad (22)$$

have been used in Support Vector machines. These different 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).

The application of (16) to our problem is straightforward: we simply substitute an a priori chosen kernel function $k(\mathbf{x},\mathbf{y})$ for all occurences of $(\Phi(\mathbf{x})\cdot\Phi(\mathbf{y}))$. The choice of k then *implicitely* determines the mapping Φ and the feature space F.

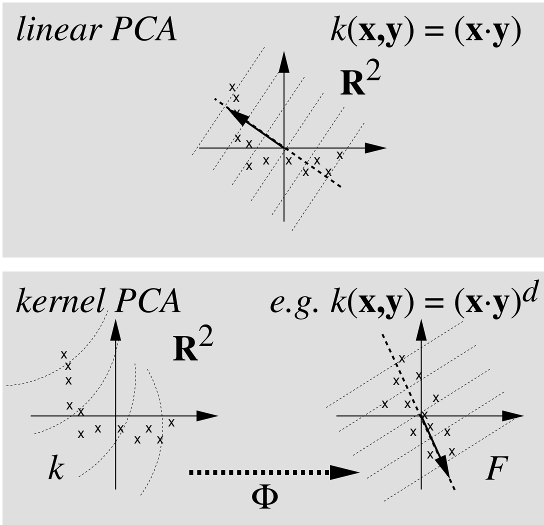

Figure 1: The basic idea of kernel PCA. In some high-dimensional feature space F (bottom right), we are performing linear PCA, just as a PCA in input space (top). Since F is nonlinearly related to input space (via Φ), the contour lines of constant projections onto the principal eigenvector (drawn as an arrow) become *nonlinear* in input space. Note that we cannot draw a 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: $\mathbf{R}^2$).

4 Kernel PCA

The Algorithm. To perform kernel-based PCA (Fig. 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 (12) 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. (15), Fig. 2),

$$(\mathbf{V}^n \cdot \Phi(\mathbf{x})) = \sum_{i=1}^{M} \alpha_i^n k(\mathbf{x}_i, \mathbf{x}). \tag{23}$$

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. Note that in practice, our algorithm is not equivalent to the form of nonlinear PCA that can be obtained by explicitely mapping into the feature space F: even though the rank of the matrix K is always limited by the sample size, we may not even be able to compute this matrix, if the dimensionality is prohibitively high. In that case, using kernels is imperative.

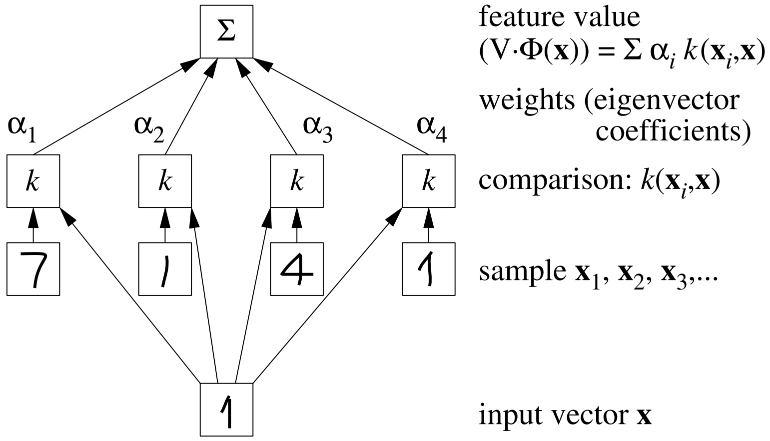

Figure 2: Feature extraction architecture in kernel PCA (cf. (23)). In the first layer, the input vector is compared to the sample via a kernel function, chosen a priori (e.g. polynomial, Gaussian, or sigmoid). The outputs are then linearly combined using weights which are found by solving an eigenvector problem.

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; Diamantaras & Kung, 1996) carry over to kernel-based PCA, with the modifications that they become statements concerning F rather than $\mathbf{R}^N$. In F, we can thus assert that PCA is the orthogonal basis transformation with the following properties (assuming that the eigenvectors are sorted in descending order of the eigenvalue size): (1) the first q ($q \in \{1, \ldots, M\}$) principal components, i.e. projections on eigenvectors, carry more variance than any other q orthogonal directions, (2) the mean-squared approximation error in representing the observations by the first q principal components is minimal, (3) the principal components are uncorrelated, and (4) the first q principal components have maximal mutual information with respect to the inputs (this holds under Gaussianity assumptions, and thus depends on the data and the chosen kernel).

We conclude this section by noting one general property of kernel PCA *in input space*: for kernels which depend only on dot products or distances in input space (as all the examples that we have given so far do), kernel PCA has the property of unitary invariance, following directly from the fact that both the eigenvalue problem and the feature extraction only depend on kernel values. This ensures that the features extracted do not depend on which orthonormal coordinate system we use for representing our input data.

Computational Complexity. As mentioned in Sec. 3, 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. 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 explicitely 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 & 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 (23), 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 done by a technique proposed by Burges (1996) in the context of Support Vector machines. In the present setting, we approximate each eigenvector $\mathbf{V} = \sum_{i=1}^{\ell} \alpha_i \Phi(\mathbf{x}_i)$ (Eq. (8)) by another vector $\tilde{\mathbf{V}} = \sum_{j=1}^{m} \beta_j \Phi(\mathbf{z}_j)$, where $m < \ell$ is chosen a priori according to the desired speedup, and $\mathbf{z}_j \in \mathbf{R}^N, j = 1, \ldots, m$. This is done by minimizing the squared difference $\rho = \|\mathbf{V} - \tilde{\mathbf{V}}\|^2$. The crucial point is that this also can be done without explicitely dealing with the possibly high-dimensional space F. As

$$\rho = \|\mathbf{V}\|^2 + \sum_{i,j=1}^{m} \beta_i \beta_j k(\mathbf{z}_i, \mathbf{z}_j) - 2 \sum_{i=1}^{\ell} \sum_{j=1}^{m} \alpha_i \beta_j k(\mathbf{x}_i, \mathbf{z}_j), \tag{24}$$

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 gradient descent.

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 (17) can be expressed with a single weight vector $\mathbf{w} = \sum_{i=1}^{\ell} \nu_i \mathbf{x}_i$ as $f(\mathbf{x}) = \mathrm{sgn}((\mathbf{x} \cdot \mathbf{w}) + b)$. Thus the final stage of classification can be done extremely fast.

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 directions in 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 (8) has a preimage. 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 (24) 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. (24)). Alternatively, we could use a suitable regression method for estimating the reconstruction mapping from the kernel-based principal components to the inputs.

5 Experiments

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. Eq.(20)) of degree 1 through 4 (see Fig. 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. Further toy examples, using radial basis function kernels (21) and neural network type sigmoid kernels (22), are shown in figures 4 and 5.

Character Recognition. In this experiment, we extracted nonlinear principal components from a handwritten character database, using kernel PCA in the form given in

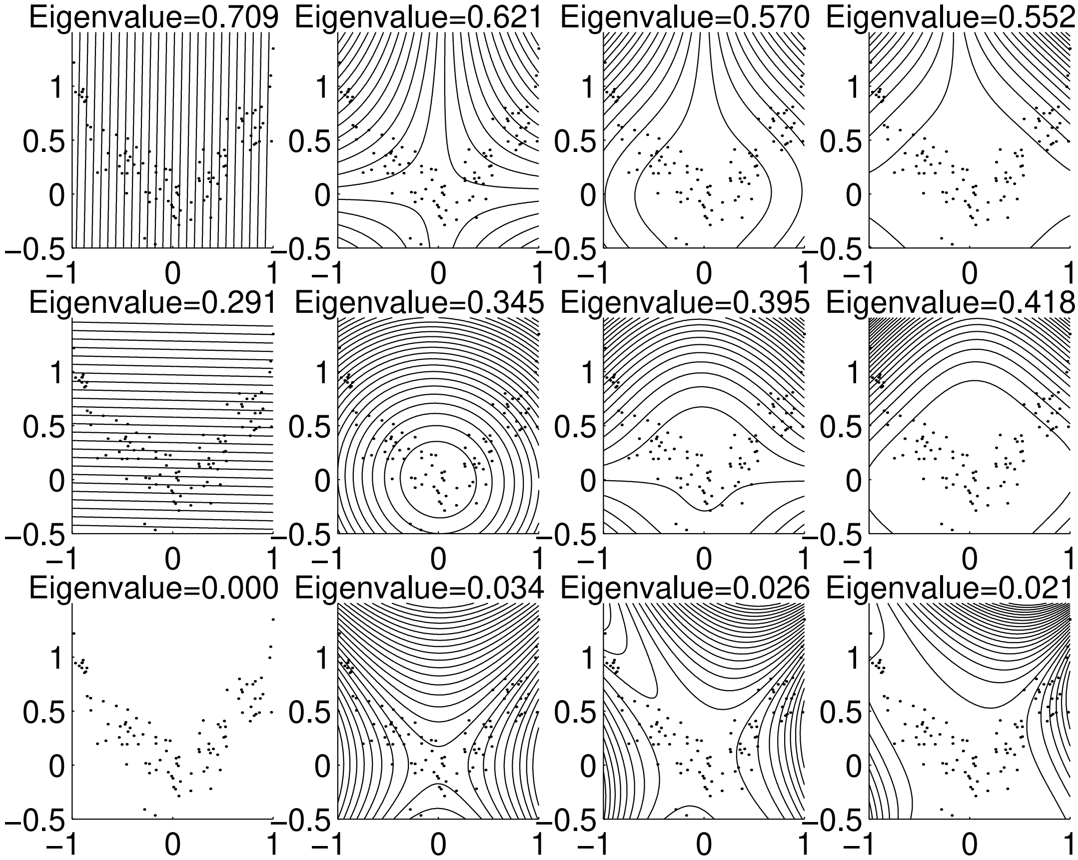

Figure 3: Two-dimensional toy example, with data generated in the following way: x-values have uniform distribution in $[-1, 1]$, y-values are generated from $y_i = x_i^2 + \xi$, were ξ is normal noise with standard deviation 0.2. From left to right, the polynomial degree in the kernel (20) increases 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.

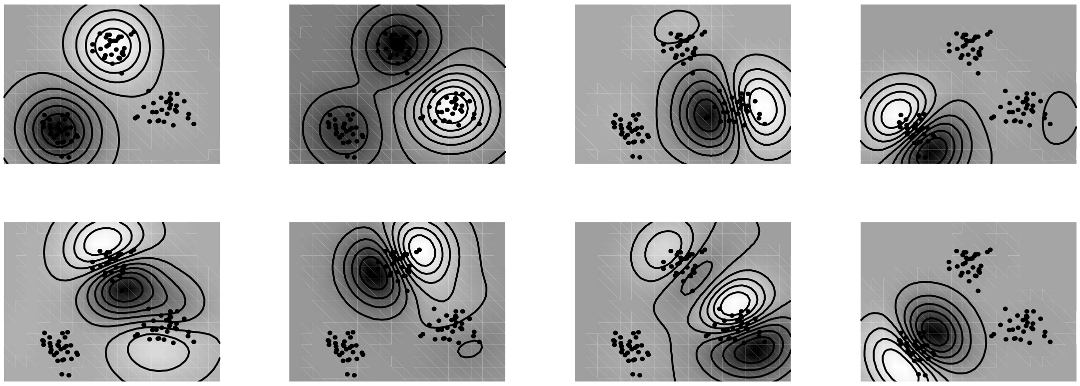

Figure 4: Two-dimensional toy example with three data clusters (Gaussians with standard deviation 0.1, depicted region: $[-1,1] \times [-0.5,1]$): first 8 nonlinear principal components extracted with $k(\mathbf{x},\mathbf{y}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{0.1}\right)$. Note that the first 2 principal component (top left) nicely separate the three clusters. The components 3 – 5 split up the clusters into halfs. Similarly, the components 6 – 8 split them again, in a way orthogonal to the above splits. Thus, the first 8 components divide the data into 12 regions. The matlab code used for generating this figure can be obtained from http://svm.first.gmd.de.

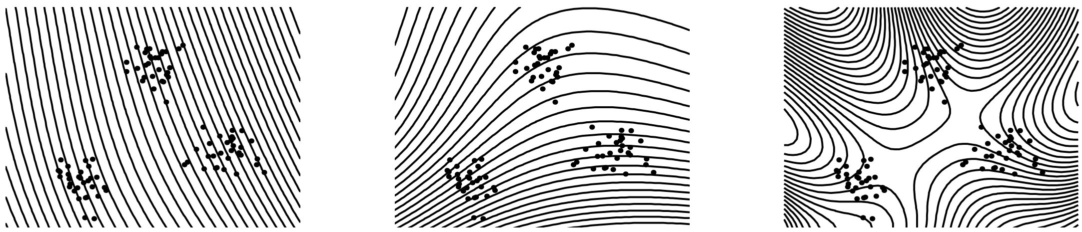

Figure 5: Two-dimensional toy example with three data clusters (Gaussians with standard deviation 0.1, depicted region: $[-1,1] \times [-0.5,1]$): first 3 nonlinear principal components extracted with $k(\mathbf{x},\mathbf{y}) = \tanh\left(2(\mathbf{x}\cdot\mathbf{y})+1\right)$. Note that the first 2 principal component (top left) are sufficient to separate the three clusters, and the third component splits the clusters into halves.

Appendix B. We chose the US postal service (USPS) database of handwritten digits collected from mail envelopes in Buffalo.[3] 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. To assess the utility of the components, we trained a soft margin hyperplane classifier (Vapnik & Chervonenkis, 1979; Cortes & Vapnik, 1995) on the classification task.[4] This is a

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

[4]Thanks to L. Bottou, C. Burges, C. Cortes for parts of the training code.

# 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 1: Test error rates on the USPS handwritten digit database for linear Support Vector machines trained on nonlinear principal components extracted by PCA with kernel (20), 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).

special case of Support Vector machines, using the standard dot product as a kernel function. It simply tries to separate the training data by a hyperplane with large margin.

Table 1 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 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, Burges, & Vapnik, 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, Burges & Vapnik, 1995); it is much better than linear classifiers operating directly on the image data (a linear Support Vector machine achieves 8.9%; Schölkopf, Burges, & Vapnik, 1995). These encouraging results have been reproduced on an object recognition task (Schölkopf, Smola & Müller, 1996).

6 Discussion

Feature Extraction for Classification. This paper was devoted to the presentation of a new technique for nonlinear PCA. To develop this technique, we made use of a kernel method so far only used in supervised learning (Vapnik, 1995). 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. Sec. 5). 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 uses shared weights to improve transformation invariance, and a hierarchy of feature detectors resembling parts of the human visual system. 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 $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_\ell, y_\ell)$, 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.

Finally, we note that a similar approach can be taken in the case of regression estimation.

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 explicitely 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 explicitely compute all dimensions of F (loosely speaking, all possible features), but only work in a relevant subspace of F.

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 five 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 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. 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.

Autoassociative Multi-Layer Perceptrons. Consider a linear 3-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 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 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 (Fig. 5). The principal components of a test point $\mathbf{x}$ in that case take the form (Fig. 2) $\sum_i \alpha_i^n \tanh \cdot (\kappa(\mathbf{x}_i, \mathbf{x}) + \Theta)$.

Principal Curves. An approach with a clear geometric interpretation in input space is the method of principal curves (Hastie & Stuetzle, 1989), which iteratively estimates a curve (or surface) capturing the structure of the data. The data are 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.

Locally Linear PCA. In cases where a linear PCA fails because the dependences in the data vary nonlinearly with the region in input space, it can be fruitful to use an approach where linear PCA is applied locally (e.g. Bregler & Omohundro, 1994). Possibly, also kernel PCA could be improved by taking locality into account.

Kernel PCA. Kernel PCA is a nonlinear generalization of PCA in the sense that (1) it is performing PCA in feature spaces of arbitrarily large (possibly infinite) dimensionality, and (2) if we use the kernel $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})$, we recover standard PCA. Compared

[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 to comprise more layers (see e.g. Diamantaras & Kung, 1996).

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.

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, Burges, & Vapnik, 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 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 paper to explore all the possibilities, including many distance-based algorithms, in detail. Some of them are currently being investigated, for instance nonlinear forms of k-means clustering and kernel-based independent component analysis (Schölkopf, Smola, & Müller, 1996).

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.

Acknowledgement. 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, 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.

A 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 $K\boldsymbol{\beta}^i = \mu_i \boldsymbol{\beta}^i$ $(i = 1, \dots, M)$. To understand the relation between Eq. (11) and Eq. (12), we proceed as follows: first suppose $\lambda, \boldsymbol{\alpha}$ satisfy (11). We may expand $\boldsymbol{\alpha}$ in K's eigenvector basis as $\boldsymbol{\alpha} = \sum_{i=1}^{M} a_i \boldsymbol{\beta}^i$. Eq. (11) then reads

$M\lambda \sum_i a_i \mu_i \boldsymbol{\beta}^i = \sum_i a_i \mu_i^2 \boldsymbol{\beta}^i$, 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 $i = 1, \ldots, M$,

$$M\lambda = \mu_i \quad or \quad a_i = 0 \quad or \quad \mu_i = 0. \tag{25}$$

Note that the above are not exclusive *or*-s. We next assume that $\lambda, \boldsymbol{\alpha}$ satisfy (12), to carry out a similar derivation. In that case, we find that Eq. (12) is equivalent to $M\lambda \sum_i a_i \boldsymbol{\beta}^i = \sum_i a_i \mu_i \boldsymbol{\beta}^i$, i.e. for all $i = 1, \ldots, M$,

$$M\lambda = \mu_i \quad or \quad a_i = 0. \tag{26}$$

Comparing (25) and (26), 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 (12), we may always add multiples of eigenvectors of K with eigenvalue 0 and still satisfy (11), with the same eigenvalue. This means that there exist solutions of Eq. (11) 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 $\sum_i \alpha_i \Phi(\mathbf{x}_i) = 0$. Thus, the above difference between the solutions of (11) and (12) is irrelevant, since we are interested in vectors in F rather than vectors in the space of the expansion coefficients of (8). We thus only need to diagonalize K to find all relevant solutions of (11).

B Centering in High-Dimensional Space

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

$$\tilde{\Phi}(\mathbf{x}_i) := \Phi(\mathbf{x}_i) - \frac{1}{M}\sum_{i=1}^{M} \Phi(\mathbf{x}_i) \tag{27}$$

are centered. Thus, the assumptions of Sec. 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

$$\tilde{\lambda}\tilde{\boldsymbol{\alpha}} = \tilde{K}\tilde{\boldsymbol{\alpha}}, \tag{28}$$

with $\tilde{\boldsymbol{\alpha}}$ being the expansion coefficients of an eigenvector (in F) in terms of the points (27), $\tilde{\mathbf{V}} = \sum_{i=1}^{M} \tilde{\alpha}_i \tilde{\Phi}(\mathbf{x}_i)$. As we do not have the centered data (27), 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 $\tilde{K}_{ij} = (\tilde{\Phi}(\mathbf{x}_i) \cdot \tilde{\Phi}(\mathbf{x}_j))$:

$$\begin{aligned}
\tilde{K}_{ij} &= \left((\Phi(\mathbf{x}_i) - \frac{1}{M}\sum_{m=1}^{M} \Phi(\mathbf{x}_m)) \cdot (\Phi(\mathbf{x}_j) - \frac{1}{M}\sum_{n=1}^{M} \Phi(\mathbf{x}_n)) \right) \\
&= K_{ij} - \frac{1}{M}\sum_{m=1}^{M} 1_{im} K_{mj} - \frac{1}{M}\sum_{n=1}^{M} K_{in} 1_{nj} + \frac{1}{M^2}\sum_{m,n=1}^{M} 1_{im} K_{mn} 1_{nj} \\
&= (K - 1_M K - K 1_M + 1_M K 1_M)_{ij}.
\end{aligned} \tag{29}$$

We thus can compute $\tilde{K}$ from K, and then solve the eigenvalue problem (28). As in (14), 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,

$$(\tilde{\mathbf{V}}^k \cdot \tilde{\Phi}(\mathbf{t})) = \sum_{i=1}^{M} \tilde{\alpha}_i^k (\tilde{\Phi}(\mathbf{x}_i) \cdot \tilde{\Phi}(\mathbf{t})). \tag{30}$$

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 $\tilde{K}_{ij}^{test} = \Big((\Phi(\mathbf{t}_i) - \frac{1}{M}\sum_{m=1}^{M}\Phi(\mathbf{x}_m)) \cdot (\Phi(\mathbf{x}_j) - \frac{1}{M}\sum_{n=1}^{M}\Phi(\mathbf{x}_n))\Big)$. As in (29), we express $\tilde{K}^{test}$ in terms of K^{test}, and arrive at $\tilde{K}^{test} = K^{test} - 1'_M K - K^{test} 1_M + 1'_M K 1_M$, where $1'_M$ is the $L \times M$ matrix with all entries equal to $1/M$.

C Mercer Kernels

Mercer's theorem of functional analysis (e.g. Courant & Hilbert, 1953) gives 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 $\mathcal{K} : L^2 \to L^2$, $(\mathcal{K}f)(\mathbf{y}) = \int k(\mathbf{x}, \mathbf{y}) f(\mathbf{x})\, d\mathbf{x}$, which is positive, i.e.

$$\int f(\mathbf{x}) k(\mathbf{x}, \mathbf{y}) f(\mathbf{y})\, d\mathbf{x}\, d\mathbf{y} \geq 0 \quad \text{for all} \quad f \in L^2, \tag{31}$$

then k can be expanded into a uniformly convergent series

$$k(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{\infty} \lambda_i \phi_i(\mathbf{x}) \phi_i(\mathbf{y}) \tag{32}$$

with $\lambda_i \geq 0$. In this case,

$$\Phi : \mathbf{x} \mapsto (\sqrt{\lambda_1}\psi_1(\mathbf{x}), \sqrt{\lambda_2}\psi_2(\mathbf{x}), \ldots) \tag{33}$$

is a map into F such that k acts as the given dot product, i.e. $(\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})) = k(\mathbf{x}, \mathbf{y})$.

Although formulated originally for the case where the integral operator acts on functions f from $L^2([a, b])$, Mercer's theorem also holds if f is defined on a space of arbitrary dimensionality, provided that it is compact (e.g. Dunford & Schwartz, 1963).

References

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

B. E. Boser, I .M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152, Pittsburgh, PA, 1992. ACM Press.

C. Bregler and M. Omohundro. Surface learning with applications to lipreading. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Precessing Systems 6*, San Mateo, CA, 1994. Morgan Kaufmann Publishers.

C. J. C. Burges. Simplified support vector decision rules. In L. Saitta, editor, *Proc. 13th Intl. Conf. on Machine Learning*, San Mateo, CA, 1996. Morgan Kaufmann.

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

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

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.

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

N. Dunford and J. T. Schwartz. *Linear Operators Part II: Spectral Theory, Self Adjoint Operators in Hilbert Space.* John Wiley & Sons, New York, 1963.

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

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

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*, 12(1):103–108, 1990.

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

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

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

B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. Technical Report 44, Max-Planck-Institut für biologische Kybernetik, 1996.

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

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).