Input Space vs. Feature Space in Kernel-Based Methods

Bernhard Schölkopf[*,⋆], Sebastian Mika[*], Chris J. C. Burges[†],

Philipp Knirsch[‡], Klaus-Robert Müller[*], Gunnar Rätsch[*], Alex J. Smola[*,⋆]

[*] GMD FIRST, Rudower Chaussee 5, 12489 Berlin, Germany

[⋆] Engineering Department, Australian National University, Canberra 0200, Australia

[†] Bell Laboratories, 101 Crawfords Corner Rd., Holmdel NJ, USA

[‡] Max-Planck-Institut für biologische Kybernetik, Spemannstr. 38, 72076 Tübingen, Germany

{bs, mika, smola, raetsch, klaus}@first.gmd.de, burges@lucent.com, phil@kyb.tuebingen.mpg.de

Abstract. *This paper collects some ideas targeted at advancing our understanding of the feature spaces associated with Support Vector (SV) kernel functions. We first discuss the geometry of feature space. In particular, we review what is known about the shape of the image of input space under the feature space map, and how this influences the capacity of SV methods. Following this, we describe how the metric governing the intrinsic geometry of the mapped surface can be computed in terms of the kernel, using the example of the class of inhomogeneous polynomial kernels, which are often used in SV pattern recognition.*

We then discuss the connection between feature space and input space by dealing with the question of how one can, given some vector in feature space, find a pre-image (exact or approximate) in input space. We describe algorithms to tackle this issue, and show their utility in two applications of kernel methods. First, we use it to reduce the computational complexity of SV decision functions; second, we combine it with the Kernel PCA algorithm, thereby constructing a nonlinear statistical denoising technique which is shown to perform well on real-world data.

1 Introduction

Reproducing kernels are functions $k : \mathcal{X}^2 \to \mathbb{R}$ which for all pattern sets

$$\{\mathbf{x}_1, \ldots, \mathbf{x}_\ell\} \subset \mathcal{X} \tag{1}$$

give rise to positive matrices $K_{ij} := k(\mathbf{x}_i, \mathbf{x}_j)$ (Saitoh, 1988). Here, $\mathcal{X}$ is some compact set in which the data lives, typically (but not necessarily) a subset of $\mathbb{R}^N$. In the SV community, reproducing kernels are often referred to as *Mercer kernels* (section 2.2 will show why). They provide an elegant way of dealing with nonlinear algorithms by reducing them to linear ones in some feature space F nonlinearly related to input space: Using k instead of a dot product in $\mathbb{R}^N$ corresponds to mapping the data into a possibly high-dimensional dot product space F by a (usually nonlinear) map $\Phi : \mathbb{R}^N \to F$, and taking the dot product there, i.e. (Boser et al., 1992)

$$k(\mathbf{x}, \mathbf{y}) = (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})). \tag{2}$$

By virtue of this property, we shall call Φ a *feature map associated with k*. Any linear algorithm which can be carried out in terms of dot products can be made nonlinear by substituting an a priori chosen kernel. Examples of such algorithms include the potential function method, SV Machines, and kernel PCA (Aizerman et al., 1964; Vapnik, 1995; Schölkopf et al., 1998d). The price that one has to pay for this elegance, however, is that the solutions are only obtained as expansions in terms of input patterns mapped into feature space. For instance, the normal vector of an SV hyperplane is expanded in terms of Support Vectors, just as the kernel PCA feature extractors are expressed in terms of training examples,

$$\Psi = \sum_{i=1}^{\ell} \alpha_i \Phi(\mathbf{x}_i). \tag{3}$$

When evaluating an SV decision function or a kernel PCA feature extractor, this is normally not a problem: due to (2), multiplying Ψ with some mapped test point $\Phi(\mathbf{x})$ transforms (3) into a kernel expansion which can be evaluated even if Ψ lives in an infinite-dimensional space. In some cases, however, there are reasons mandating a more comprehensive understanding of what exactly is the connection between patterns in input space and elements of feature space, given as expansions such as (3). This field being far from understood, the current paper attempts to gather some ideas elucidating the problem, and simultaneously proposes some algorithms for situations where the above connection is important. These are the problem of denoising by kernel PCA, and the problem of speeding up Support Vector decision functions.

The remainder of this article is organized as follows. Section 2 discusses different ways of understanding the mapping from input space into feature space. Following that, we briefly review two feature space algorithms, SV machines and kernel PCA (section 3). The focus of interest of the paper, the way back from feature space to input space, is described in section 4, and, in the more general form of constructing sparse approximations of feature space expansions, in section 5. The algorithms proposed in these two sections are experimentally evaluated in section 6 and discussed in section 7.

2 From Input Space to Feature Space

In this section we show how the feature spaces in question are defined by choice of a suitable kernel function. Insight into the structure of feature space can then be gained by considering their relation to reproducing kernel Hilbert spaces, by how they can be approximated by an empirical map, by their extrinsic geometry, which leads to useful new capacity results, and by their intrinsic geometry, which can be computed solely in terms of the kernel.[1]

[1] Those readers who are chiefly interested in applications and algorithms might want to consider skipping over the present section.

2.1 The Mercer Kernel Map

We start by stating the version of Mercer's theorem given in (Williamson et al., 1998). We assume $(\mathcal{X}, \mu)$ to be a finite measure space, i.e. $\mu(\mathcal{X}) < \infty$. By "almost all" we mean except for sets of measure zero.

Theorem 1 (Mercer) *Suppose $k \in L_\infty(\mathcal{X}^2)$ is a symmetric kernel such that the integral operator $T_k : L_2(\mathcal{X}) \to L_2(\mathcal{X})$,*

$$(T_k f)(\mathbf{x}) := \int_{\mathcal{X}} k(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) d\mu(\mathbf{y}) \tag{4}$$

is positive. Let $\psi_j \in L_2(\mathcal{X})$ be the normalized eigenfunctions of T_k associated with the eigenvalues $\lambda_j > 0$, sorted in nonincreasing order, and let $\overline{\psi_j}$ denote their complex conjugates. Then

1. *$(\lambda_j(T))_j \in l_1$,*

2. *$\psi_j \in L_\infty(\mathcal{X})$ and $\sup_j \|\psi_j\|_{L_\infty} < \infty$,*

3. *$k(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{N_F} \lambda_j \overline{\psi_j(\mathbf{x})} \psi_j(\mathbf{y})$ holds for almost all $(\mathbf{x}, \mathbf{y})$. Either $N_F \in \mathbb{N}$, or $N_F = \infty$; in the latter case, the series converges absolutely and uniformly for almost all $(\mathbf{x}, \mathbf{y})$.*

From statement 3 it follows that $k(\mathbf{x}, \mathbf{y})$ corresponds to a dot product in $l_2^{N_F}$, i.e. $k(\mathbf{x}, \mathbf{y}) = (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}))$ with

$$\begin{aligned} \Phi : \mathcal{X} &\to l_2^{N_F} \\ \mathbf{x} &\mapsto (\sqrt{\lambda_j} \psi_j(\mathbf{x}))_j, \quad j = 1, \ldots, N_F, \end{aligned} \tag{5}$$

for almost all $\mathbf{x} \in \mathcal{X}$.

In fact, the *uniform* convergence of the series implies that given $\epsilon > 0$, there exists an $n \in \mathbb{N}$ such that even if the range of Φ is infinite-dimensional, k can be approximated within accuracy ϵ as a dot product in $\mathbb{R}^n$, between images of

$$\Phi^n : \mathbf{x} \mapsto (\sqrt{\lambda_1} \psi_1(\mathbf{x}), \ldots, \sqrt{\lambda_n} \psi_n(\mathbf{x})).$$

2.2 The Reproducing Kernel Map

We can also think of the feature space as a *reproducing kernel Hilbert space* (RKHS). To see this (Aronszajn, 1950; Saitoh, 1988; Wahba, 1990; Schölkopf, 1997; Girosi, 1998), recall that a RKHS is a Hilbert space of functions f on some set $\mathcal{X}$ such that all evaluation functionals $f \mapsto f(\mathbf{y})$ $(\mathbf{y} \in \mathcal{X})$ are continuous. In that case, by the Riesz representation theorem, for each $\mathbf{y} \in \mathcal{X}$ there exists a unique function of $\mathbf{x}$, call it $k(\mathbf{x}, \mathbf{y})$, such that

$$f(\mathbf{y}) = \langle f, k(., \mathbf{y}) \rangle \tag{6}$$

(here, $k(., \mathbf{y})$ is the function on $\mathcal{X}$ obtained by fixing the second argument of k to $\mathbf{y}$, and $\langle ., . \rangle$ is the dot product of the RKHS. In contrast, we use $(. \cdot .)$ to denote the canonical (Euclidean) dot product). In view of this property, k is called a *reproducing kernel*.

Note that by (6), $\langle f, k(., \mathbf{y}) \rangle = 0$ for all $\mathbf{y}$ implies that f is identically zero. Hence the set of functions $\{k(., \mathbf{y}) : \mathbf{y} \in \mathcal{X}\}$ spans the whole RKHS. The dot product on the RKHS thus only needs to be defined on $\{k(., \mathbf{y}) : \mathbf{y} \in \mathcal{X}\}$ and can then be extended to the whole RKHS by linearity and continuity. From (6), it follows that in particular

$$\langle k(., \mathbf{x}), k(., \mathbf{y}) \rangle = k(\mathbf{y}, \mathbf{x}) \tag{7}$$

for all $\mathbf{x}, \mathbf{y} \in \mathcal{X}$ (this implies that k is symmetric). Note that this means that any reproducing kernel k corresponds to a dot product in another space.

Let us now consider a Mercer kernel, i.e. one which satisfies the condition of Theorem 1, and construct a dot product such that k becomes a reproducing kernel for the Hilbert space H containing the functions

$$f(\mathbf{x}) = \sum_{i=1}^{\infty} a_i k(\mathbf{x}, \mathbf{x}_i) = \sum_{i=1}^{\infty} a_i \sum_{j=1}^{N_F} \lambda_j \psi_j(\mathbf{x}) \psi_j(\mathbf{x}_i). \tag{8}$$

By linearity, we have

$$\langle f, k(., \mathbf{y}) \rangle = \sum_{i=1}^{\infty} a_i \sum_{j,n=1}^{N_F} \lambda_j \psi_j(\mathbf{x}_i) \langle \psi_j, \psi_n \rangle \lambda_n \psi_n(\mathbf{y}). \tag{9}$$

Since k is a Mercer kernel, the ψ_i ($i = 1, \ldots, N_F$) can be chosen to be orthogonal with respect to the dot product in $L^2(\mathcal{X})$. Hence it is straightforward to construct a dot product $\langle ., . \rangle$ such that

$$\langle \psi_j, \psi_n \rangle = \delta_{jn} / \lambda_j \tag{10}$$

(using the Kronecker symbol δ_{jn}), in which case (9) reduces to the reproducing kernel property (6) (using (8)).

Therefore, (7) provides us with a feature map $\tilde{\Phi}$ associated with k:

Proposition 2 *For any Mercer kernel k, there exists a RKHS H such that for*

$$\tilde{\Phi} : \mathbb{R}^N \to H, \;\; \mathbf{x} \mapsto k(., \mathbf{x}), \tag{11}$$

we have

$$\langle \tilde{\Phi}(\mathbf{x}), \tilde{\Phi}(\mathbf{y}) \rangle = k(\mathbf{x}, \mathbf{y}). \tag{12}$$

2.3 The Empirical Kernel Map

While giving an interesting alternative theoretical viewpoint, the map $\tilde{\Phi}$ does not appear all that useful at first sight. In practise, it would seem pointless to first map the inputs into functions, i.e. into infinite-dimensional objects. However, for a given dataset, it is possible to approximate $\tilde{\Phi}$ from (11) by only evaluating it on these points (Mattera (1998); Mika (1998); Tsuda (1999), cf. also Hastie and Tibshirani (1990)):

Definition 3 *For a given set* $\{\mathbf{z}_1, \ldots, \mathbf{z}_m\}$*, we call*

$$\begin{aligned} \Phi_\ell : \mathbb{R}^N \to \mathbb{R}^\ell, \;\; \mathbf{x} \;\; &\mapsto \;\; k(.,\mathbf{x})|_{\{\mathbf{z}_1,\ldots,\mathbf{z}_\ell\}} \\ &= \;\; (k(\mathbf{z}_1,\mathbf{x}), \ldots, k(\mathbf{z}_\ell,\mathbf{x})). \end{aligned} \tag{13}$$

the empirical kernel map w.r.t. $\{\mathbf{z}_1, \ldots, \mathbf{z}_m\}$.

Example. Consider first the case where k is a Mercer kernel, and $\{\mathbf{z}_1, \ldots, \mathbf{z}_m\} = \{\mathbf{x}_1, \ldots, \mathbf{x}_\ell\}$, i.e. we evaluate $k(.,\mathbf{x})$ on the training patterns. If we carry out a linear algorithm in feature space, then everything will take place in the linear span of the mapped training patterns. Therefore, we can represent the $k(.,\mathbf{x})$ of (11) as $\Phi_\ell(\mathbf{x})$ without losing information. However, the dot product to use in that representation is not simply the canonical dot product in $\mathbb{R}^\ell$, since the $\Phi(\mathbf{x}_i)$ will usually not form an orthonormal system. To turn Φ_ℓ into a feature map associated with k, we need to endow $\mathbb{R}^\ell$ with a dot product $\langle .,. \rangle$ such that

$$k(\mathbf{x},\mathbf{y}) = \langle \Phi_\ell(\mathbf{x}), \Phi_\ell(\mathbf{y}) \rangle. \tag{14}$$

To this end, we use the ansatz $\langle .,. \rangle = (., A.)$, with A being a positive matrix.[2] Enforcing (14) on the training patterns, this yields the self-consistency condition (cf. Schölkopf et al., 1998c; Smola and Schölkopf, 1998a)

$$K = KAK. \tag{15}$$

Here, we have used K to denote the $\ell \times \ell$ *kernel Gram matrix*

$$K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j). \tag{16}$$

The condition (15) can be satisfied for instance by the pseudo-inverse $A = K^{-1}$.

Equivalently, we could have incorporated this rescaling operation, which corresponds to a kernel PCA whitening (Schölkopf et al., 1998d,c; Tsuda, 1999), directly into the map, by modifying (13) to

$$\Phi_\ell^w : \mathbf{x} \mapsto K^{-1/2}(k(\mathbf{x}_1,\mathbf{x}), \ldots, k(\mathbf{x}_\ell,\mathbf{x})). \tag{17}$$

[2] Note that every dot product can be written in this form. Moreover, we do not require definiteness of A, as the null space can be projected out, leading to a lower-dimensional feature space.

This simply amounts to dividing the eigenvector basis vectors of K by $\sqrt{\lambda_i}$, where the λ_i are the eigenvalues of K.[3] It parallels the rescaling of the eigenfunctions of the integral operator belonging to the kernel, given by (10).

For data sets where the number of examples is smaller than their dimensionality, it can actually be computationally attractive to carry out Φ_ℓ^w explicitly rather than using kernels in whatever subsequent algorithm (SVMs, or kernel PCA, say) one wants to use. As an aside, note that in the case of kernel PCA (to be described in section 3), one does not even need to worry about the whitening step: Using the canonical dot product in $\mathbb{R}^\ell$ (rather than $\langle .,. \rangle$) will simply lead to diagonalizing K^2 instead of K, which yields the same eigenvectors with squared eigenvalues. This was pointed out by Mattera (1998), and independently noticed by Mika (1998).

We end this section with two notes which illustrate why the use of (13) need not be restricted to the special case we just discussed.

More general kernels. When using non-symmetric kernels k in (13), together with the canonical dot product, one will effectively work with a matrix $K^\top K$, with general K. Note that each positive semidefinite matrix can be written as $K^\top K$.

If we wanted to carry out the whitening step, it would have to be using $(K^\top K)^{-1/4}$ (cf. footnote 3 concerning potential singularities).

Different evaluation sets. Mika (1998) has performed experiments to speed up kernel PCA by choosing $\{\mathbf{z}_1, \ldots, \mathbf{z}_m\}$ as a subset of $\{\mathbf{x}_1, \ldots, \mathbf{x}_\ell\}$.

2.4 The Capacity of the Kernel Map

Vapnik (1995, 1998) gives a bound on the capacity, measured by the VC-dimension h, of optimal margin classifiers. It takes the form

$$h \leq R^2 \Lambda^2 + 1, \tag{18}$$

[3]It is understood that if K is singular, we use the pseudo-inverse of $K^{1/2}$.

where Λ is an upper bound constraining the length of the weight vector of the hyperplane in canonical form, and R is the radius of the smallest sphere containing the data in the space where the hyperplane is constructed. The smaller this sphere is, the smaller is also the capacity, with beneficial effects on the generalization error bounds.

If the data is distributed in a reasonably isotropic way, which is often the case in input space, then (18) can be fairly precise.[4] If, however, the distribution of the data is such that it does not fill the sphere, then (18) is wasteful. The argument in the remainder of the section, which is summarized from (Williamson et al., 1998, 1999), shows that using a kernel typically entails that the data in fact lies in some box with rapidly decaying sidelengths, which can be much smaller than the above sphere.

From statement 2 of Theorem 1, there exists some constant $C_k > 0$ depending on the kernel k such that

$$|\psi_j(\mathbf{x})| \leq C_k \text{ for all } j \in \mathbb{N} \text{ and almost all } \mathbf{x} \in \mathcal{X}. \tag{19}$$

Therefore, $\Phi(\mathcal{X})$ is essentially contained in an axis parallel parallelepiped in l_2 with side lengths $2C_k\sqrt{\lambda_j}$ (cf. (5)).

To see how this effectively restricts the class of functions we are using, we first note that everything in F is done in terms of dot products. Therefore, we can compensate any invertible linear transformation of the data in F by the corresponding inverse adjoint transformation on the set of admissible weight vectors in F, i.e. for any invertible operator A on l_2, we have

$$\langle A^*\mathbf{w}, A^{-1}\Phi(\mathbf{x})\rangle = \langle \mathbf{w}, \Phi(\mathbf{x})\rangle \quad \text{for all } \mathbf{x} \in \mathcal{X}. \tag{20}$$

Hence, we may construct a diagonal scaling operator A^{-1} which inflates the sides of the above parallelepiped as much as possible, while ensuring that it still lives in a sphere of the original radius R in l_2 (figure 1). This will not change the R^2 factor on the right hand side of (18), but it buys us something regard-

[4]In terms of entropy numbers, this is due to the tightness of the rate given by a famous theorem of Maurey (e.g. Carl and Stephani, 1990).

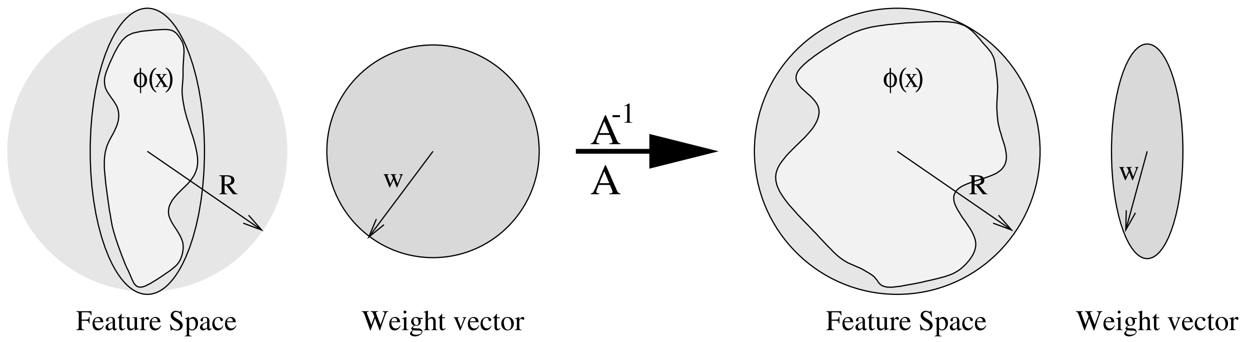

Figure 1: Since everything is done in terms of dot products, scaling up the data by A can be compensated by scaling the weight vectors with A^{-1}. By choosing A such that the data is still contained in a ball of the same radius R, we effectively reduce our function class (parameterized by the weight vector), which leads to better generalization bounds which depend on the kernel inducing the map Φ.

ing the second factor: one can show that the function class essentially behaves as if it was finite-dimensional, with a cut-off determined by the rate of decay of T_k's eigenvalues.

The reasoning that leads to the improved bounds is somewhat intricate and cannot presently be explained in full detail. In a nutshell, the idea is to compute the capacity (measured in terms of covering numbers of the SV function class, evaluated on an ℓ-sample) via the entropy numbers of a suitable linear operator.[5] Using $\mathbf{X} := (\mathbf{x}_1, \ldots, \mathbf{x}_\ell)$, consider first the operator

$$\begin{aligned} S_{\Phi(\mathbf{X})} : l_2 &\to l_\infty^\ell \\ \mathbf{w} &\mapsto \Big((\langle \mathbf{w}, \Phi(\mathbf{x}_1) \rangle), \ldots, (\langle \mathbf{w}, \Phi(\mathbf{x}_\ell) \rangle) \Big). \end{aligned} \tag{21}$$

For our purpose, the entropy numbers of $S_{\Phi(\mathbf{X})}$ applied to a sphere of radius Λ are crucial. These can be

[5] Consider two normed spaces E and F. For $n \in \mathbb{N}$, the *nth entropy number* of a set $M \subset E$ is defined as $\epsilon_n(M) := \inf\{\epsilon > 0$: there exists an ϵ-cover for M in E containing n or fewer points$\}$. (Recall that the covering number $\mathcal{N}(\epsilon, M)$, being essentially its functional inverse, measures how many balls of radius ϵ one needs to cover M.) The entropy numbers $\epsilon_n(T)$ of an *operator* $T : E \to F$ are defined as the entropy numbers of the image of the unit ball under T. Note that $\epsilon_1(T) = \|T\|$; intuitively, the higher entropy number allow a finer characterization of the complexity of the image of T (e.g. Carl and Stephani, 1990; Williamson et al., 1998). Note that entropy numbers have some nice properties that covering numbers are lacking. For instance, scaling a subset of a normed vector space by some factor simply scales its entropy numbers by the same factor.

computed as the entropy numbers of the operator $\Lambda A S_{A^{-1}\Phi(\mathbf{X})}$. Using factorization properties of entropy numbers, these can be upper bounded taking into account the above scaling operator A in a rather precise way. The faster the eigenvalues of the integral operator T_k associated with a given kernel decay, the smaller the entropy numbers (and hence the capacity) of the corresponding feature space algorithm function class, and the stronger the generalization error bounds that one can prove.

As an example, we now consider how the entropy numbers $\epsilon_n(A\colon l_2 \to l_2)$ depend asymptotically on the eigenvalues of T_k.

Proposition 4 (Exponential-Polynomial decay (Williamson et al., 1998)) *Suppose k is a Mercer kernel with $\ln(\lambda_j) \sim -j^p$ for some $p > 0$. Then $\ln \epsilon_n^{-1}(A\colon l_2 \to l_2) = O(\ln^{\frac{p}{p+1}} n)$*

An example of such a kernel (for $p = 2$) is the Gaussian $k(x, y) = e^{-|x-y|^2}$.

Entropy numbers are a promising tool for studying the capacity of feature space methods. This is due to the fact that in the linear case, which is what we are interested in for feature space algorithms, they can be studied using powerful methods of functional analysis (e.g. Carl and Stephani, 1990).

2.5 The Metric of the Kernel Map

Another way to gain insight into the structure of feature space is to consider the intrinsic shape of the manifold to which one's data is mapped. It is important here to distinguish between the feature space F and the surface in that space to which points in input space $\mathbb{R}^N$ actually map, which we will call S. In general S will be an N dimensional submanifold embedded in F. For simplicity, we assume here that S is sufficiently smooth that structures such as a Riemannian metric can be defined on it. Here we will follow the analysis of Burges (1999), to which the reader is referred for more details, although the application to the class of inhomogeneous polynomial kernels is new.

We first note that all intrinsic geometrical properties of S can be derived once we know the Riemannian metric induced by the embedding of S in F. The Riemannian metric can be defined by a symmetric metric tensor $g_{\mu\nu}$. Interestingly, we do not need to know the explicit mapping Φ to construct $g_{\mu\nu}$; it can be written solely in terms of the kernel. To see this consider the line element:

$$\begin{aligned} ds^2 &= g_{ab} d\Phi^a(\mathbf{x}) d\Phi^b(\mathbf{x}) \\ &= g_{\mu\nu} dx^\mu dx^\nu, \end{aligned} \tag{22}$$

where indices a, b correspond to the vector space F and μ, ν to input space $\mathbb{R}^N$. Now letting $d\mathbf{x}$ represent a small but finite displacement in $\mathbb{R}^N$, we have

$$\begin{aligned} ds^2 &= \|\Phi(\mathbf{x} + d\mathbf{x}) - \Phi(\mathbf{x})\|^2 \\ &= k(\mathbf{x} + d\mathbf{x}, \mathbf{x} + d\mathbf{x}) - 2k(\mathbf{x}, \mathbf{x} + d\mathbf{x}) + k(\mathbf{x}, \mathbf{x}) \\ &= \big((1/2)\partial_{x_\mu}\partial_{x_\nu} k(\mathbf{x}, \mathbf{x}) - \partial_{y_\mu}\partial_{y_\nu} k(\mathbf{x}, \mathbf{y})\big)_{\mathbf{y}=\mathbf{x}} dx^\mu dx^\nu \\ &\equiv g_{\mu\nu} dx^\mu dx^\nu \end{aligned}$$

Thus we can read off the components of the metric tensor:

$$g_{\mu\nu} = (1/2)\partial_{x_\mu}\partial_{x_\nu} k(\mathbf{x}, \mathbf{x}) - \{\partial_{y_\mu}\partial_{y_\nu} k(\mathbf{x}, \mathbf{y})\}_{\mathbf{y}=\mathbf{x}} \tag{23}$$

For the class of kernels which are functions only of dot products between points in $\mathbb{R}^N$, both covariant and contravariant components of the metric take a simple form:

$$g_{\mu\nu} = \delta_{\mu\nu} k'(\|\mathbf{x}\|^2) + x_\mu x_\nu k''(\|\mathbf{x}\|^2) \tag{24}$$

$$g^{\mu\nu} = \frac{\delta_{\mu\nu}}{k'} - x_\mu x_\nu \frac{k''/k'}{k' + \|\mathbf{x}\|^2 k''} \tag{25}$$

where the prime denotes derivative with respect to the argument $\|\mathbf{x}\|^2$. To illustrate, let us compute the intrinsic geometrical properties of the surface S corresponding to the class of inhomogeneous polynomial Mercer kernels:

$$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + c)^p, \tag{26}$$

where c is a constant. Properties relating to the intrinsic curvature of a surface are completely captured by the Riemann curvature tensor

$$R_{\nu\alpha\beta}{}^{\mu} = \partial_\alpha \Gamma^\mu_{\beta\nu} - \partial_\beta \Gamma^\mu_{\alpha\nu} + \Gamma^\rho_{\alpha\nu}\Gamma^\mu_{\beta\rho} - \Gamma^\rho_{\beta\nu}\Gamma^\mu_{\alpha\rho} \tag{27}$$

where the Christoffel symbols of the second kind are defined by:

$$\Gamma^\alpha_{\beta\gamma} \equiv g^{\alpha\mu}\Gamma_{\beta\gamma\mu} = \frac{1}{2} g^{\alpha\mu}(\partial_\beta g_{\gamma\mu} - \partial_\mu g_{\beta\gamma} + \partial_\gamma g_{\mu\beta}) \tag{28}$$

Thus we find that for this class of kernels, the Riemann curvature, for arbitrary input space dimension N and polynomial order p, is given by

$$\begin{aligned} R_{\nu\alpha\beta}{}^{\mu} &= \frac{-2(p-1)}{(\|\mathbf{x}\|^2+c)^2}(x_\alpha x_\nu \delta_{\mu\beta} - x_\beta x_\nu \delta_{\mu\alpha}) + \frac{(p-1)}{(\|\mathbf{x}\|^2+c)}(\delta_{\nu\alpha}\delta_{\mu\beta} - \delta_{\nu\beta}\delta_{\mu\alpha}) \\ &- \frac{p-1}{(\|\mathbf{x}\|^2+c)^2(\|\mathbf{x}\|^2+c/p)}(x_\beta x_\nu \delta_{\mu\alpha} - x_\alpha x_\nu \delta_{\mu\beta} + x_\beta x_\mu \delta_{\nu\alpha} - x_\alpha x_\mu \delta_{\nu\beta}) \\ &+ \frac{(p-1)^2}{(\|\mathbf{x}\|^2+c)^2}\frac{c/p}{\|\mathbf{x}\|^2+c/p}(x_\alpha x_\nu \delta_{\beta\mu} - x_\beta x_\nu \delta_{\alpha\mu}) \end{aligned} \tag{29}$$

It's interesting to compare this result with that for the homogeneous case ($c = 0$) (Burges, 1999):

$$\begin{aligned} R_{\mu\nu\rho}{}^{\sigma}(c=0) &= \frac{(p-1)}{\|\mathbf{x}\|^2}(\delta_{\rho\nu}\delta_{\sigma\mu} - \delta_{\rho\mu}\delta_{\sigma\nu}) \\ &- \frac{(p-1)}{\|\mathbf{x}\|^4}(x_\nu x_\rho \delta_{\mu\sigma} - x_\mu x_\rho \delta_{\sigma\nu} + x_\sigma x_\mu \delta_{\rho\nu} - x_\sigma x_\nu \delta_{\rho\mu}) \end{aligned} \tag{30}$$

This analysis shows that adding the constant c to the kernel results in striking differences in the geometries. Both curvatures vanish for $p = 1$, as expected. However for $N = 2$, $R_{\mu\nu\rho}{}^{\sigma}(c = 0)$ vanishes for all powers p, whereas $R_{\mu\nu\rho}{}^{\sigma}(c \neq 0)$ does not vanish for any $p > 1$. Furthermore, all surfaces for homogeneous kernels with non-vanishing curvature (i.e. those with $N > 2$ and $p > 1$) have a singularity at $\|\mathbf{x}\|^2 = 0$, whereas none of the corresponding surfaces for inhomogeneous kernels do, provided $c > 0$.

Thus beyond providing insight into the geometrical structure of the surfaces in F generated by choice of kernel, the geometrical analysis also gives more concrete results. For example, one can expect that the density on the surface S will become ill behaved for data $\mathbf{x}$ whose norm is small, for homogeneous polynomial kernels, but not for the inhomogeneous case. Similar problems may arise in the pattern recognition case if one's data lies near the singularity. These considerations can be extended to compute the intrinsic volume element on S, which may be used to compute the density $p(\mathbf{x})$ on S, and to give some simple necessary tests that must be satisfied by any kernel if it is to be a Mercer kernel: the reader is referred to Burges (1999) for details.

3 Feature Space Algorithms

We next describe the two algorithms used in this paper: SV machines, and kernel PCA. The SV algorithm shall not be described in detail, let us only briefly fix the notation.

Support Vector (SV) classifiers (Vapnik, 1995) construct a maximum margin hyperplane in F. In input space, this corresponds to a nonlinear decision boundary of the form

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{\ell} \gamma_i k(\mathbf{x}, \mathbf{x}_i) + b\right), \tag{31}$$

where the $\mathbf{x}_i$ are the training examples. Those with $\gamma_i \neq 0$ are called *Support Vectors*; in many applications, most of the γ_i, which are found by solving a quadratic program, turn out to be 0. Excellent classification accuracies in both OCR and object recognition have been obtained using SV machines (Schölkopf, 1997; Burges, 1998). A generalization to the case of regression estimation, leading to similar function expansion, exists (Vapnik, 1995; Smola and Schölkopf, 1998b).

Kernel Principal Component Analysis (Schölkopf et al., 1998d) carries out a linear PCA in the feature space F. The extracted features take the nonlinear form

$$f_k(\mathbf{x}) = \sum_{i=1}^{\ell} \alpha_i^k k(\mathbf{x}_i, \mathbf{x}), \tag{32}$$

where, up to a normalization, the α_i^k are the components of the k-th eigenvector of the matrix $(k(\mathbf{x}_i, \mathbf{x}_j))_{ij}$.

This can be understood as follows. We wish to find eigenvectors $\mathbf{V}$ and eigenvalues λ of the covariance matrix C in the feature space,[6] where

$$C := \sum_{i=1}^{\ell} \Phi(\mathbf{x}_i)\Phi(\mathbf{x}_i)^\top. \tag{33}$$

In the case when F is very high dimensional this will be impossible to compute directly. To be still able to solve this problem, one uses Mercer kernels. To this end, we need to derive a formulation which only uses Φ in dot products. We then replace any occurrence of $(\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}))$ by $k(\mathbf{x}, \mathbf{y})$. This way we avoid dealing with the mapped data explicitly, which may be intractable in terms of memory and computational cost.

To find a formulation for PCA which uses only dot products, we first substitute the covariance matrix (33) into the eigenequation $C\mathbf{V} = \lambda\mathbf{V}$. Note that all solutions to this equation with $\lambda \neq 0$ must lie in the span of Φ-images of the training data. Thus, we may consider the equivalent system

$$\lambda(\Phi(\mathbf{x}_k) \cdot \mathbf{V}) = (\Phi(\mathbf{x}_k) \cdot C\mathbf{V}) \text{ for all } k = 1, \dots, \ell, \tag{34}$$

and expand the solution $\mathbf{V}$ as

$$\mathbf{V} = \sum_{i=1}^{\ell} \alpha_i \Phi(\mathbf{x}_i). \tag{35}$$

Substituting (33) and (35) into (34), and defining a $\ell \times \ell$ matrix K by $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ we arrive at a problem which is cast in terms of dot products: solve

$$\ell\lambda\boldsymbol{\alpha} = K\boldsymbol{\alpha}, \tag{36}$$

where $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_\ell)^\top$ (for details on the last step see Schölkopf et al. (1998d)). Normalizing the solution $\mathbf{V}_k$ in F translates into $\lambda_k(\boldsymbol{\alpha}^k \cdot \boldsymbol{\alpha}^k) = 1$. To extract features, we compute the projection of the Φ-image of a test point $\mathbf{x}$ onto the k-th eigenvector in the feature space by

$$\beta_k := (\mathbf{V}^k \cdot \Phi(\mathbf{x})) = \sum_{i=1}^{\ell} \alpha_i^k k(\mathbf{x}_i, \mathbf{x}). \tag{37}$$

Usually, this will be much cheaper than taking the dot product in the feature space explicitly.

[6] Here we assume that the mapped data is centered, too. In general this will not be true, but all computations can easily be reformulated to perform an explicit centering in F (Schölkopf et al., 1998d).

To conclude the brief summary of kernel PCA, we state a characterization which involves the same regularizer (the length of the weight vector) as the one used in SV machines.

Proposition 5 *For all k, the k-th kernel PCA feature extractor, scaled by $1/\lambda_k$, is optimal among all feature extractors of the form*

$$f(\mathbf{x}) = \sum_i \alpha_i k(\mathbf{x}_i, \mathbf{x})$$

in the sense that it has the shortest weight vector

$$\|\mathbf{V}\|^2 = \sum_i \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j)$$

subject to the conditions that

(1) it is orthogonal to the first $k-1$ kernel PCA feature extractors (in feature space)

(2) applied to the training set $\mathbf{x}_1, \ldots, \mathbf{x}_\ell$, it leads to a unit variance set of outputs.

Both SV machines and kernel PCA utilize Mercer kernels to generalize a linear algorithm to a nonlinear setting; moreover, both use the same regularizer, albeit in different domains of learning — supervised vs. unsupervised. Nevertheless, feature extraction experiments using kernel PCA have shown that a linear hyperplane classifier trained on the extracted features performs as well as a nonlinear SV machine trained directly on the inputs.

4 From Feature Space to Input Space

Unlike section 2, which described how to get from input space into feature space, we now study the way back. There has been a fair amount of work on aspects of this problem in the context of developing so-called reduced set methods (e.g. Burges, 1996; Burges and Schölkopf, 1997; Schölkopf et al., 1998a; Frieß and Harrison, 1998; Osuna and Girosi, 1999). For pedagogical reasons, we shall postpone reduced set methods to section 5, as they focus on a problem that is already more complex than the one we would like to start with.

4.1 The Pre-Image Problem

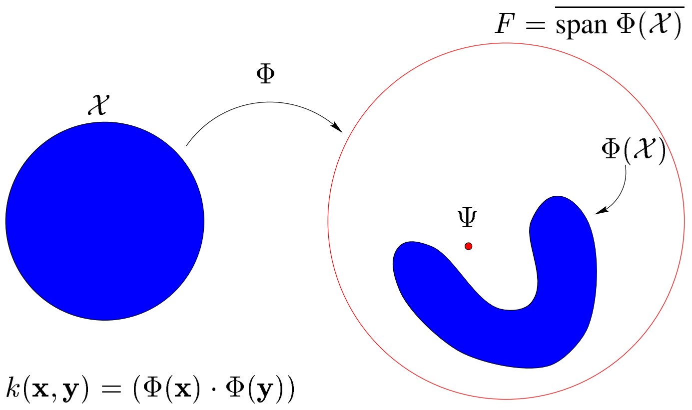

Figure 2: The pre-image problem. Not each point in the span of the mapped input data is necessarily the image of some input pattern. Therefore, not each point that can be written as an expansion in terms of mapped input patterns (e.g. a kernel PCA eigenvector, or a SVM hyperplane normal vector), can necessarily be expressed as the image of a single input pattern.

As stated in the introduction, feature space algorithms express their solutions as expansions in terms of mapped input points (3). However, since the map Φ into the feature space F is nonlinear, we cannot generally assert that each such expansion will have a pre-image under Φ, i.e. a point $\mathbf{z} \in \mathbb{R}^N$ such that $\Phi(\mathbf{z}) = \Psi$ (figure 2). If the pre-image existed, it would be easy to compute, as shown by the following result (Schölkopf, 1997):

Proposition 6 *Consider a feature space expansion* $\Psi = \sum_{j=1}^{\ell} \alpha_j \Phi(\mathbf{x}_j)$. *If there exists a* $\mathbf{z} \in \mathbb{R}^N$ *such that*

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

and if k *is an invertible function* f_k *of* $(\mathbf{x} \cdot \mathbf{y})$, *then we can compute* $\mathbf{z}$ *as*

$$\mathbf{z} = \sum_{i=1}^{N} f_k^{-1} \left(\sum_{j=1}^{\ell} \alpha_j k(\mathbf{x}_j, \mathbf{e}_i) \right) \mathbf{e}_i,$$

where $\{\mathbf{e}_1, \ldots, \mathbf{e}_N\}$ *is any orthonormal basis of input space.*

Proof We expand $\mathbf{z}$ as

$$\begin{aligned}\mathbf{z} &= \sum_{i=1}^{N}(\mathbf{z}\cdot\mathbf{e}_i)\mathbf{e}_i \\ &= \sum_{i=1}^{N} f_k^{-1}(k(\mathbf{z},\mathbf{e}_i))\mathbf{e}_i \\ &= \sum_{i=1}^{N} f_k^{-1}\left(\sum_{j=1}^{\ell}\alpha_j k(\mathbf{x}_j,\mathbf{e}_i)\right)\mathbf{e}_i. \end{aligned} \tag{38}$$

∎

Several remarks on this proposition should be given. First, examples of kernels which are invertible functions of $(\mathbf{x}\cdot\mathbf{y})$ are polynomial kernels

$$k(\mathbf{x},\mathbf{y}) = (\mathbf{x}\cdot\mathbf{y}+c)^d, \quad c \geq 0,\ d \text{ odd}, \tag{39}$$

and sigmoid kernels

$$k(\mathbf{x},\mathbf{y}) = \sigma(\kappa\cdot(\mathbf{x}\cdot\mathbf{y})+\Theta), \quad \kappa,\Theta\in\mathbb{R}. \tag{40}$$

A similar result holds for RBF kernels (using the polarization identity) — all we need is a kernel which allows the reconstruction of $(\mathbf{x}\cdot\mathbf{y})$ from k, evaluated on some input points which we are allowed to choose (for details, cf. Schölkopf, 1997).

The crucial assumption, clearly, is the *existence* of the pre-image. Unfortunately, there are many situations where there are no pre-images. To illustrate this, we consider the feature map $\tilde{\Phi}$ (11). Clearly, only points in feature space which can be written as $k(.,\mathbf{x})$ do have a pre-image under this map. To characterize this set of points in a specific example, consider Gaussian kernels

$$k(\mathbf{x},\mathbf{y}) = e^{-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\,\sigma^2}}. \tag{41}$$

In this case, $\tilde{\Phi}$ maps each input into a Gaussian sitting on that point. However, it is known (Micchelli, 1986) that no Gaussian can be written as a linear combination of Gaussians centered at other points. Therefore, in the Gaussian case, none of the expansions (3), excluding trivial cases with only one term, has an exact pre-image.

The problem that we had initially set out to solve has turned out to be insolvable in the general case. Let us try to ask for less. Rather than trying to find exact pre-images, we now consider approximate ones. We call $\mathbf{z} \in \mathbb{R}^N$ an *approximate pre-image* for Ψ if

$$\rho(\mathbf{z}) = \|\Psi - \Phi(\mathbf{z})\|^2 \tag{42}$$

is small.[7]

Are there vectors Ψ for which good approximate pre-images exist? As described in section 3, kernel PCA is nothing but PCA in F. Therefore, for $n = 1, 2, \dots, p$, it provides projections

$$P_n\Phi(\mathbf{x}) = \sum_{k=1}^{n} (\Phi(\mathbf{x}) \cdot \mathbf{V}^k)\mathbf{V}^k \tag{43}$$

with the following optimal approximation property (e.g. Schölkopf, 1997) Here, we assume that the $\mathbf{V}^k$ are sorted according to nonincreasing eigenvalues λ_k, with λ_p being the smallest nonzero eigenvalue.

Proposition 7 *P_n is the n-dimensional projection minimizing*

$$\sum_{i=1}^{\ell} \|P_n\Phi(\mathbf{x}_i) - \Phi(\mathbf{x}_i)\|^2. \tag{44}$$

Therefore, $P_n\Phi(\mathbf{x})$ can be expected to *have* a good approximate pre-image: trivially, already $\mathbf{x}$ *is* a good approximate pre-image. As we shall see in experiments, however, even better pre-images can be found, which makes some interesting applications possible (Schölkopf et al., 1998b; Mika et al., 1999):

[7]Just how small it needs to be in order to be considered a satisfactory approximation will depend on the problem at hand. Therefore, we have refrained from giving a formal definition.

Denoising. Given a noisy $\mathbf{x}$, map it into $\Phi(\mathbf{x})$, discard higher components to obtain $P_n\Phi(\mathbf{x})$, and then compute a pre-image $\mathbf{z}$. Here, the hope is that the main structure in the data set is captured in the first n directions, and the remaining components mainly pick up the noise — in this sense, $\mathbf{z}$ can be thought of as a denoised version of $\mathbf{x}$.

Compression. Given the eigenvectors $\boldsymbol{\alpha}^k$ and a small number of features β_k of $\Phi(\mathbf{x})$, but not $\mathbf{x}$, compute a pre-image as an approximate reconstruction of $\mathbf{x}$. This is useful if n is smaller than the dimensionality of the input data.

Interpretation. Visualize a nonlinear feature extractor $\mathbf{V}^k$ by computing a pre-image.

In this paper, we focus on the first point. In the next section, we shall develop a method for minimizing (42), which we will later, in the experimental section, apply to the case where $\Psi = P_n\Phi(\mathbf{x})$.

4.2 An Algorithm for Approximate Pre-Images

The present section (Schölkopf et al., 1998a) gives an analysis for the case of the Gaussian kernel, which has proven to perform very well in applications (Schölkopf et al., 1997), and proposes an iteration procedure for computing pre-images of kernel expansions.

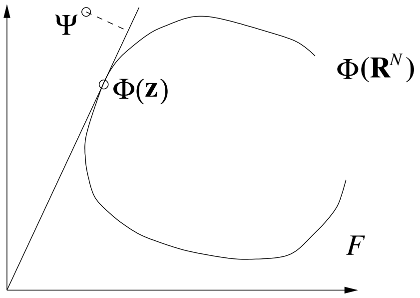

Figure 3: Given a vector $\Psi \in F$, we try to approximate it by a multiple of a vector $\Phi(\mathbf{z})$ in the image of input space ($\mathbb{R}^N$) under the nonlinear map Φ by finding $\mathbf{z}$ such that the projection distance of Φ onto span($\Phi(\mathbf{z})$) is minimized.

We start by considering a problem slightly more general than the pre-image problem: we are seeking to approximate $\Psi = \sum_{i=1}^{N_y} \gamma_i \Phi(\mathbf{y}_i)$ by $\Psi' = \beta\Phi(\mathbf{z})$. First observe that rather than minimizing

$$\|\Psi - \Psi'\|^2 = \sum_{i,j=1}^{N_y} \gamma_i\gamma_j k(\mathbf{y}_i, \mathbf{y}_j) + \beta^2 k(\mathbf{z}, \mathbf{z}) - 2\sum_{i=1}^{N_y} \gamma_i \beta k(\mathbf{y}_i, \mathbf{z}), \tag{45}$$

we can minimize the distance between Ψ and the orthogonal projection of Ψ onto span $(\Phi(\mathbf{z}))$ (figure 3),

$$\left\| \frac{(\Psi \cdot \Phi(\mathbf{z}))}{(\Phi(\mathbf{z}) \cdot \Phi(\mathbf{z}))} \Phi(\mathbf{z}) - \Psi \right\|^2 = \|\Psi\|^2 - \frac{(\Psi \cdot \Phi(\mathbf{z}))^2}{(\Phi(\mathbf{z}) \cdot \Phi(\mathbf{z}))}. \tag{46}$$

To this end, we maximize

$$\frac{(\Psi \cdot \Phi(\mathbf{z}))^2}{(\Phi(\mathbf{z}) \cdot \Phi(\mathbf{z}))}, \tag{47}$$

which can be expressed in terms of the kernel. The maximization of (47) over $\mathbf{z}$ is preferable to the one of (45) over $\mathbf{z}$ and β, since it comprises a lower-dimensional problem, and since $\mathbf{z}$ and β have different scaling behavior. Once the maximum of (47) is found, it is extended to the minimum of (45) by setting (cf. (46)) $\beta = (\Psi \cdot \Phi(\mathbf{z}))/(\Phi(\mathbf{z}) \cdot \Phi(\mathbf{z}))$. The function (47) can either be minimized using standard techniques (as Burges (1996)), or, for particular choices of kernels, using fixed-point iteration methods, as shown presently.

For kernels which satisfy $k(\mathbf{z}, \mathbf{z}) = 1$ for all $\mathbf{z} \in \mathbb{R}^N$ (e.g. Gaussian kernels), (47) reduces to

$$(\Psi \cdot \Phi(\mathbf{z}))^2. \tag{48}$$

For the extremum, we have $0 = \nabla_{\mathbf{z}}(\Psi \cdot \Phi(\mathbf{z}))^2 = 2(\Psi \cdot \Phi(\mathbf{z}))\nabla_{\mathbf{z}}(\Psi \cdot \Phi(\mathbf{z}))$. To evaluate the gradient in terms of k, we substitute (54) to get the sufficient condition $0 = \sum_{i=1}^{N_y} \gamma_i \nabla_{\mathbf{z}} k(\mathbf{y}_i, \mathbf{z})$. For $k(\mathbf{x}_i, \mathbf{z}) = k(\|\mathbf{x}_i - \mathbf{z}\|^2)$ (e.g. Gaussians, or $(\|\mathbf{x}_i - \mathbf{z}\|^2 + 1)^c$ for $c = -1, -1/2$), we obtain $0 = \sum_{i=1}^{N_y} \gamma_i k'(\|\mathbf{y}_i - \mathbf{z}\|^2)(\mathbf{y}_i - \mathbf{z})$, leading to

$$\mathbf{z} = \frac{\sum_{i=1}^{N_y} \gamma_i k'(\|\mathbf{y}_i - \mathbf{z}\|^2)\mathbf{y}_i}{\sum_{i=1}^{N_y} \gamma_i k'(\|\mathbf{y}_i - \mathbf{z}\|^2)}. \tag{49}$$

For the Gaussian kernel $k(\mathbf{x}_i, \mathbf{z}) = \exp(-\|\mathbf{x}_i - \mathbf{z}\|^2/(2\sigma^2))$ we thus arrive at

$$\mathbf{z} = \frac{\sum_{i=1}^{N_y} \gamma_i \exp(-\|\mathbf{y}_i - \mathbf{z}\|^2/(2\sigma^2))\mathbf{y}_i}{\sum_{i=1}^{N_y} \gamma_i \exp(-\|\mathbf{y}_i - \mathbf{z}\|^2/(2\sigma^2))}, \tag{50}$$

and devise an iteration

$$\mathbf{z}_{n+1} = \frac{\sum_{i=1}^{N_y} \gamma_i \exp(-\|\mathbf{y}_i - \mathbf{z}_n\|^2/(2\sigma^2))\mathbf{y}_i}{\sum_{i=1}^{N_y} \gamma_i \exp(-\|\mathbf{y}_i - \mathbf{z}_n\|^2/(2\sigma^2))}. \tag{51}$$

The denominator equals $(\Psi \cdot \Phi(\mathbf{z}_n))$ and thus is nonzero in a neighbourhood of the extremum of (48), unless the extremum itself is zero. The latter only occurs if the projection of Ψ on the linear span of $\Phi(\mathbb{R}^N)$ is zero, in which case it is pointless to try to approximate Ψ. Numerical instabilities related to $(\Psi \cdot \Phi(\mathbf{z}))$ being small can thus be approached by restarting the iteration with different starting values.

Interestingly, (51) can be interpreted in the context of clustering (e.g. Buhmann, 1995). It determines the center of a single Gaussian cluster, trying to capture as many of the $\mathbf{y}_i$ with positive γ_i as possible, and simultaneously avoids those $\mathbf{y}_i$ with negative γ_i. For SV classifiers, the sign of the γ_i equals the label of the pattern $\mathbf{y}_i$. It is this sign which distinguishes (51) from plain clustering or parametric density estimation. The occurrence of negative signs is related to the fact that we are not trying to estimate a parametric density but the *difference* between two densities (modulo normalization constants).

To see this, we define the sets $pos = \{i : \gamma_i > 0\}$ and $neg = \{i : \gamma_i < 0\}$, and the shorthands $p_{pos}(\mathbf{z}) = \sum_{pos} \gamma_i \exp(-\|\mathbf{y}_i - \mathbf{z}\|^2/(2\sigma^2))$ and $p_{neg}(\mathbf{z}) = \sum_{neg} |\gamma_i| \exp(-\|\mathbf{y}_i - \mathbf{z}\|^2/(2\sigma^2))$. The target (48) then reads $(p_{pos}(\mathbf{z}) - p_{neg}(\mathbf{z}))^2$, i.e. we are trying to find a point $\mathbf{z}$ where the difference between the (unnormalized) "probabilities" for the two classes is maximal, and estimate the approximation to (54) by a Gaussian centered at $\mathbf{z}$. Moreover, note that we can rewrite (50) as

$$\mathbf{z} = \frac{p_{pos}(\mathbf{z})}{p_{pos}(\mathbf{z}) - p_{neg}(\mathbf{z})}\mathbf{y}_{pos} + \frac{p_{neg}(\mathbf{z})}{p_{neg}(\mathbf{z}) - p_{pos}(\mathbf{z})}\mathbf{y}_{neg}, \tag{52}$$

where

$$\mathbf{y}_{pos/neg} = \frac{\sum_{pos/neg} \gamma_i \exp(-\|\mathbf{y}_i - \mathbf{z}\|^2/(2\sigma^2))\mathbf{y}_i}{\sum_{pos/neg} \gamma_i \exp(-\|\mathbf{y}_i - \mathbf{z}\|^2/(2\sigma^2))}. \tag{53}$$

5 Reduced Set (RS) Methods

5.1 The Problem

We now move on to a slightly more general problem, first studied by Burges (1996), where we are no longer only looking for single pre-images, but expansions. It will turn out that one can also use the method developed in the last section to design an algorithm for the more general case.

Assume we are given a vector $\Psi \in F$, expanded in images of input patterns $\mathbf{y}_i \in \mathbb{R}^N$,

$$\Psi = \sum_{i=1}^{N_y} \gamma_i \Phi(\mathbf{y}_i), \tag{54}$$

with $\gamma_i \in \mathbb{R}, \mathbf{y}_i \in \mathbb{R}^N$. Rather than looking for a single pre-image, we now try to approximate it by a *reduced set* expansion (Burges, 1996)

$$\Psi' = \sum_{i=1}^{N_z} \beta_i \Phi(\mathbf{z}_i), \tag{55}$$

with $N_z < N_y, \beta_i \in \mathbb{R}, \mathbf{z}_i \in \mathbb{R}^N$. To this end, one can minimize (Burges, 1996)

$$\|\Psi - \Psi'\|^2 = \sum_{i,j=1}^{N_y} \gamma_i \gamma_j k(\mathbf{y}_i, \mathbf{y}_j) + \sum_{i,j=1}^{N_z} \beta_i \beta_j k(\mathbf{z}_i, \mathbf{z}_j) - 2 \sum_{i=1}^{N_y} \sum_{j=1}^{N_z} \gamma_i \beta_j k(\mathbf{y}_i, \mathbf{z}_j). \tag{56}$$

The crucial point is that even if Φ is not given explicitly, (56) can be computed (and minimized) in terms of the kernel.

In the NIST benchmark of 60000 handwritten digits, SV machines are more accurate than any other single classifier (Schölkopf, 1997); however, they are inferior to neural nets in run-time classification speed (Burges and Schölkopf, 1997). In applications where the latter is an issue, it is thus desirable to come up with methods to speed up things by making the SV expansion more sparse, i.e. replacing (54) by (55).

5.2 Finding the Coefficients

Evidently, the RS problem consists of two parts. One has to determine the RS vectors $\mathbf{z}_i$, and one has to compute the expansion coefficients β_i. We start with the latter; partly, as it is easier, partly, as it is common to different RS methods.

Proposition 8 *The optimal coefficients* $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_m)$ *for approximating* $\Psi = \sum_{i=1}^{\ell} \alpha_i \Phi(\mathbf{x}_i)$ *by* $\sum_{i=1}^{m} \beta_i \Phi(\mathbf{z}_i)$ *(for linearly independent* $\Phi(\mathbf{z}_1), \ldots, \Phi(\mathbf{z}_m)$*) in the 2-norm are given by*

$$\boldsymbol{\beta} = (K^z)^{-1} K^{zx} \boldsymbol{\alpha}. \tag{57}$$

Here, $K_{ij}^z := (\Phi(\mathbf{z}_i) \cdot \Phi(\mathbf{z}_j))$ *and* $K_{ij}^{zx} := (\Phi(\mathbf{z}_i) \cdot \Phi(\mathbf{x}_j))$.

Note that if the $\Phi(\mathbf{z}_i)$ are linearly independent, as they should be if we want to use them for approximation, then K^z has full rank. Otherwise, one can use the Pseudoinverse, or select the solution which has the largest number of zero components.

Proof (Schölkopf et al., 1998a) We evaluate the derivative of the distance in F,

$$\frac{\partial}{\partial \beta_k} \|\Psi - \sum_{i=1}^{m} \beta_i \Phi(\mathbf{z}_i)\|^2 = -2\Phi(\mathbf{z}_k)(\Psi - \sum_{i=1}^{m} \beta_i \Phi(\mathbf{z}_i)), \tag{58}$$

and set it to 0. Substituting $\Psi = \sum_{i=1}^{\ell} \alpha_i \Phi(\mathbf{x}_i)$, we obtain (using $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_\ell)$)

$$K^{zx} \boldsymbol{\alpha} = K^z \boldsymbol{\beta}, \tag{59}$$

hence

$$\boldsymbol{\beta} = (K^z)^{-1} K^{zx} \boldsymbol{\alpha}. \tag{60}$$

∎

No RS algorithm using the 2-norm optimality criterion can circumvent this result. For instance, suppose we are given an algorithm that computes the β_i and $\mathbf{z}_i$ simultaneously and comes up with a solution. Then we can always use the proposition to recompute the optimal coefficients to get a solution which is at least as good as the original one. Different algorithms can, however, differ in the way they determine the vectors $\mathbf{z}_i$ in the first place. The one dealt with in the next section simply *selects* subsets of the $\mathbf{x}_i$, while the one in section 5.4 uses vectors different from the original $\mathbf{x}_i$.

5.3 Reduced Set Selection

The idea for the first algorithm arises from the observation that the null space of the Gram matrix $K_{ij} = (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j))$ precisely tells us how many vectors can be removed from an expansion while committing zero approximation error (assuming we correctly adjust the coefficients), i.e. how sparse we can make an SV expansion, say, without changing it the least (Schölkopf, 1997; Frieß and Harrison, 1998). (Here, the Gram matrix can either be computed only from those examples which have a nonzero β_j, or from a larger set which we want to use for the expansion.) Interestingly, it will turn out that this problem is closely related to kernel PCA.

Let us start with the simplest case. Assume there exists an eigenvector $\boldsymbol{\alpha} \neq 0$ of K with eigenvalue 0, i.e. $K\boldsymbol{\alpha} = 0$. Using (2), this reads

$$\sum_{j=1}^{\ell} (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j))\alpha_j = 0 \quad \text{for all } i = 1, \dots, \ell, \tag{61}$$

hence

$$\sum_{j=1}^{\ell} \alpha_j \Phi(\mathbf{x}_j) = 0. \tag{62}$$

This means that the $\Phi(\mathbf{x}_j)$ are linearly dependent, and therefore any of the $\Phi(\mathbf{x}_j)$ which comes with a nonzero α_j can be expressed in terms of the others. Hence, we may use the eigenvectors with eigenvalue 0 to eliminate certain terms from any expansion in the $\Phi(\mathbf{x}_j)$.

What happens if we do not have nonzero eigenvalues, such as in the case of Gaussian kernels (Micchelli, 1986)? Intuitively, we would still believe that even though the above is no longer precisely true, it should give a good approximation. However, the crucial difference is that in order to get the best possible approximation, we now need to take into account the coefficients of the expansion $\Psi = \sum_{j=1}^{\ell} \beta_j \Phi(\mathbf{x}_j)$: if we commit an error by removing $\Phi(\mathbf{x}_n)$, say, then this error will depend on β_n, too. How do we then select the optimal n?

Clearly, we would like to find coefficients γ_j minimizing the error we commit by replacing $\beta_n \Phi(\mathbf{x}_n)$ with $\sum_{j \neq n} \gamma_j \Phi(\mathbf{x}_j)$,

$$\rho(\boldsymbol{\gamma}, n) = \left\| \beta_n \Phi(\mathbf{x}_n) - \sum_{j \neq n} \gamma_j \Phi(\mathbf{x}_j) \right\|^2. \tag{63}$$

To establish a connection to kernel PCA, we make a change of variables. First, define $\eta_j = 1$ for $j = n$, $\eta_j = -\gamma_j / \beta_n$ for $j \neq n$. Hence (63) equals $|\beta_n|^2 \| \sum_{j=1}^{\ell} \eta_j \Phi(\mathbf{x}_j) \|^2$. Normalizing $\boldsymbol{\eta}$ to obtain $\boldsymbol{\alpha} := \boldsymbol{\eta} / \|\boldsymbol{\eta}\|$, hence $\alpha_n = 1/\|\boldsymbol{\eta}\|$, this leads to the problem of minimizing

$$\rho(\boldsymbol{\alpha}, n) = \left| \frac{\beta_n}{\alpha_n} \right|^2 (\boldsymbol{\alpha} \cdot K\boldsymbol{\alpha}), \tag{64}$$

over $\|\boldsymbol{\alpha}\| = 1$ (note that ρ is invariant when rescaling $\boldsymbol{\alpha}$). A straightforward calculation shows that we can recover the approximation coefficients for $\beta_n \Phi(\mathbf{x}_n)$, i.e. the values to add to the β_j ($j \neq n$) for leaving out $\beta_n \Phi(\mathbf{x}_n)$, as $\gamma_j = -\frac{\beta_n \alpha_j}{\alpha_n}, \quad j \neq n$.

Rather than minimizing the nonlinear function (64), we now devise a computationally attractive approximate solution. It is motivated by the observation that $(\boldsymbol{\alpha} \cdot K\boldsymbol{\alpha})$ alone is minimized for the eigenvector with minimal eigenvalue, consistent with the special case discussed above (cf. (62)). In that case,

$(\boldsymbol{\alpha} \cdot K\boldsymbol{\alpha}) = \lambda_{min}$. More generally, if $\boldsymbol{\alpha}_i$ is any normalized eigenvector of K, with eigenvalue λ_i, then

$$\rho(i,n) = \left|\frac{\beta_n}{\alpha_n^i}\right|^2 \lambda_i. \tag{65}$$

This can be minimized in $O(\ell^3)$ operations by performing kernel PCA and scanning through the matrix $(\rho(i,n))_{in}$. The complexity can be reduced to $O(m\ell^2)$ by only considering the smallest m eigenvalues, with $m < \ell$ chosen a priori. Hence, we can eliminate $\Phi(\mathbf{x}_n)$, with n chosen in a principled yet efficient way.

Setting all computational considerations aside, the *optimal* greedy solution to the above selection problem, equivalent to (64), can also be obtained by using Proposition 8: compute the optimal solution for all possible patterns that one could leave out (i.e. use subsets of $\{\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_\ell)\}$, of size $\ell - 1$, as $\{\mathbf{z}_1, \ldots, \mathbf{z}_m\}$, and evaluate (58) in each case.

The same applies to subsets of any size. If we have the resources to exhaustively scan through all subsets of size m ($1 \leq m \leq \ell - 1$), then Proposition 8 provides the *optimal* way of selecting the best expansion of a given size. Better expansions can only be obtained if we drop the restriction that the approximation is written in terms of the original patterns, as done in the next section.

No matter how we end up choosing n, we approximate Ψ by

$$\begin{aligned} \Psi &= \sum_{j \neq n} \beta_j \Phi(\mathbf{x}_j) + \beta_n \Phi(\mathbf{x}_n) \\ &\approx \sum_{j \neq n} \left(\beta_j - \frac{\beta_n \alpha_j}{\alpha_n}\right) \Phi(\mathbf{x}_j). \end{aligned} \tag{66}$$

The whole scheme can be iterated until the expansion of Ψ is sufficiently sparse. If one wants to avoid having to find the smallest eigenvalue of K at each step anew, then approximate schemes using heuristics can be conceived.

In our experiments to be reported below, we did compute all eigenvectors at each step, using the Gram matrix computed from the SVs and then selected n according to (65).

We conclude this section with two notes on the utility of this procedure. First, for SV machines, where $\beta_j \in [-C, C]$ for some positive value of the regularization constant C, there is reason to believe that the

SV expansion can be made sparser by removing the constraint on β_j (note that (66) does not care about the constraint).[8] Second, the number of eigenvalues which are zero, or at least small, depends on the problem at hand and the kernel used. For instance, Gaussian kernel Gram matrices do not have zero eigenvalues unless some of the patterns are duplicates (Micchelli, 1986). Nevertheless, good approximations are possible, since the eigenvalues of Gaussian kernels decay rapidly (e.g. Williamson et al., 1998).

5.4 Reduced Set Construction

Suppose we want to approximate a vector

$$\Psi_1 = \sum_{i=1}^{\ell} \alpha_i \Phi(\mathbf{x}_i) \tag{67}$$

by an expansion of the type (55) with $N_z > 1$. To this end, we iterate the procedure of section 4.2 by $\Psi_{m+1} := \Psi_m - \beta_m \Phi(\mathbf{z}_m)$. Here, $\mathbf{z}_m$ denotes the $\mathbf{z}$ found for Ψ_m, obtained by iterating (51). To apply (51), Ψ_m needs to be utilized in its representation in terms of mapped input images,

$$\Psi_m = \sum_{i=1}^{\ell} \alpha_i \Phi(\mathbf{x}_i) - \sum_{i=1}^{m-1} \beta_i \Phi(\mathbf{z}_i), \tag{68}$$

i.e. we need to set $N_y = \ell + m - 1$, $(\gamma_1, \ldots, \gamma_{N_y}) = (\alpha_1, \ldots, \alpha_\ell, -\beta_1, \ldots, -\beta_{m-1})$, and $(\mathbf{y}_1, \ldots, \mathbf{y}_{N_y}) = (\mathbf{x}_1, \ldots, \mathbf{x}_\ell, \mathbf{z}_1, \ldots, \mathbf{z}_{m-1})$.

The coefficient $\beta_m = \beta(\mathbf{z}_m)$ could be computed as $(\Psi \cdot \Phi(\mathbf{z}))/(\Phi(\mathbf{z}) \cdot \Phi(\mathbf{z}))$. However, if the vectors $\Phi(\mathbf{z}_1), \ldots, \Phi(\mathbf{z}_m)$ are not orthogonal in F, then the best approximation of Ψ_1 in their span is *not* obtained by computing orthogonal projections onto each direction. Instead, we need to compute the optimal coefficients $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_m)$ anew in each step. Using Proposition 8, this leads to [9]

$$\boldsymbol{\beta} = (K^z)^{-1} K^{zx} \boldsymbol{\alpha}. \tag{69}$$

[8] Imagine a case where a certain pattern appears twice in the training set, and the SV expansion has to utilize both of the copies only because the upper bound constraint limits the coefficient of each of them to C.

[9] If the discrepancy Ψ_{m+1} has not yet reached zero, then K^z will be invertible.

The iteration is stopped after N_z steps, either specified in advance, or by monitoring when $\|\Psi_{m+1}\|$ (i.e. $\|\Psi_1 - \sum_{i=1}^{m} \beta_i \Phi(\mathbf{z}_i)\|$) falls below a specified threshold. The solution vector takes the form (55).

We conclude this section by noting that in many cases, such as multiclass SV machines, or multiple kernel PCA feature extractors, we may actually want to approximate several vectors simultaneously. This leads to more complex equations, given in (Schölkopf et al., 1998a).

6 Experiments

To see how the proposed methods work in practice we ran several toy and real-world experiments. In section 6.1, we give denoising results for the approach of finding approximate pre-images presented in section 4.1. In section 6.2 we present some experiments for the reduced set methods presented in section 5.3 and section 5.4.

6.1 Kernel PCA Denoising

6.1.1 Toy Examples

All experiments reported were carried out with Gaussian kernels (41), minimizing (42) with the iteration scheme given by (51). However, similar results were obtained with polynomial kernels. Matlab code for computing kernel PCA is available on the web from http://svm.first.gmd.de.

We generated an artificial data set from three point sources at (-0.5,-0.1), (0,0.7), (0.5,0.1) (100 points each) with Gaussian noise ($\sigma = 0.1$), and performed kernel PCA on it (figure 4). Using the resulting eigenvectors, we extracted nonlinear principal components from a set of test points generated from the same model, and reconstructed the points from varying numbers of principal components. Figure 5 shows that discarding higher-order components leads to removal of the noise — the points move towards their respective sources.

In a second experiment (table 1), we generated a data set from eleven Gaussians in $\mathbb{R}^{10}$ with zero mean

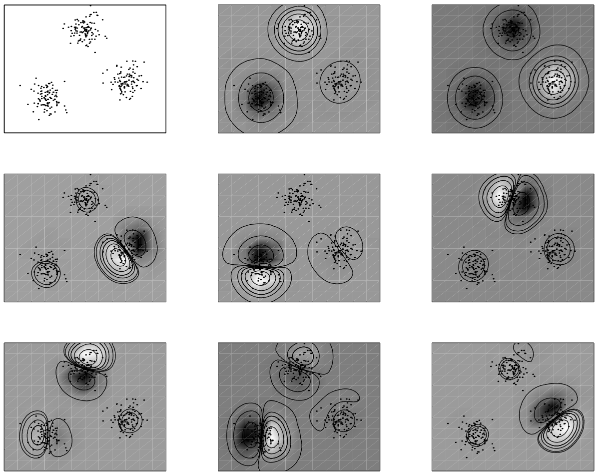

Figure 4: Kernel PCA toy example (see text): lines of constant feature value for the first 8 nonlinear principal components extracted with $k(\mathbf{x}, \mathbf{y}) = \exp\left(-\|\mathbf{x} - \mathbf{y}\|^2/0.1\right)$. The first 2 principal components (top middle/right) separate the three clusters. Components 3–5 split the clusters. Components 6–8 split them again, orthogonal to the above splits.

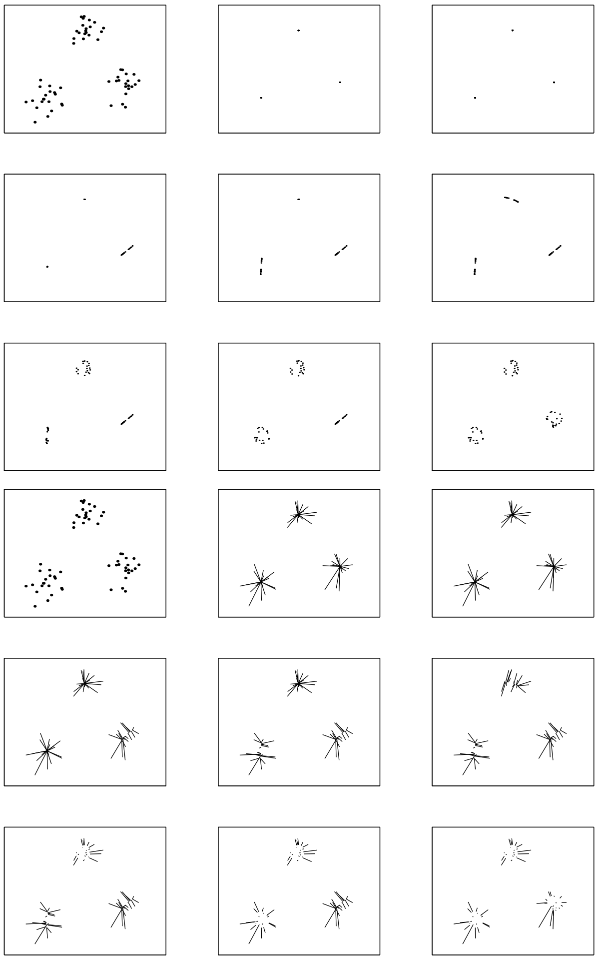

Figure 5: Kernel PCA denoising by reconstruction from projections onto the eigenvectors of figure 4. We generated 20 new points from each Gaussian, represented them in feature space by their first $n = 1, 2, \ldots, 8$ nonlinear principal components, and computed approximate pre-images, shown in the upper 9 pictures (top left: original data, top middle: $n = 1$, top right: $n = 2$, etc.). Note that by discarding higher order principal components (i.e. using a small n), we remove the noise inherent in the nonzero variance σ^2 of the Gaussians. The lower 9 pictures show how the original points "move" in the denoising. Unlike the corresponding case in linear PCA, where where we obtain lines (see figure 6), in kernel PCA clusters shrink to points.

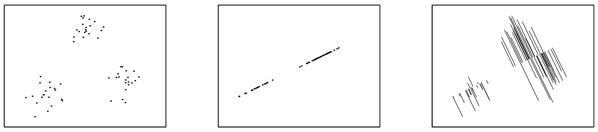

Figure 6: Reconstructions and point movements for linear PCA, based on the first principal component.

and variance σ^2 in each component, by selecting from each source 100 points as a training set and 33 points for a test set (centers of the Gaussians randomly chosen in $[-1, 1]^{10}$). Then we applied kernel PCA to the training set and computed the projections β_k of the points in the test set. With these, we carried out denoising, yielding an approximate pre-image in $\mathbb{R}^{10}$ for each test point. This procedure was repeated for different numbers of components in reconstruction, and for different values of σ (also used in the kernel). We compared the results provided by our algorithm to those of linear PCA via the mean squared distance of all denoised test points to their corresponding center. Table 1 shows the *ratio* of these values; here and below, ratios larger than one indicate that kernel PCA performed better than linear PCA. For almost every choice of n and σ, kernel PCA did better. Note that using all 10 components, linear PCA is just a basis transformation and hence cannot denoise. The extreme superiority of kernel PCA for small σ is due to the fact that all test points are in this case located close to the eleven spots in input space, and linear PCA has to cover them with less than ten directions. Kernel PCA moves each point to the correct source even when using only a small number of components.

σ	n=1	2	3	4	5	6	7	8	9
0.05	2058.42	1238.36	846.14	565.41	309.64	170.36	125.97	104.40	92.23
0.1	10.22	31.32	21.51	29.24	27.66	23.53	29.64	40.07	63.41
0.2	0.99	1.12	1.18	1.50	2.11	2.73	3.72	5.09	6.32
0.4	1.07	1.26	1.44	1.64	1.91	2.08	2.22	2.34	2.47
0.8	1.23	1.39	1.54	1.70	1.80	1.96	2.10	2.25	2.39

Table 1: Denoising Gaussians in $\mathbb{R}^{10}$ (see text). Performance ratios larger than one indicate how much better kernel PCA did, compared to linear PCA, for different choices of the Gaussians' standard deviation σ, and different numbers of components used in reconstruction.

To get some intuitive understanding in a low-dimensional case, figure 7 depicts the results of denoising a half circle and a square in the plane, using kernel PCA, a *nonlinear autoencoder*, *principal curves*, and linear

PCA. The principal curves algorithm (Hastie and Stuetzle, 1989) iteratively estimates a curve capturing the structure of the data. The data are projected to the closest point on a curve which the algorithm tries to construct such that each point 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 a generalization of the latter. The algorithm uses a smoothing parameter which is annealed during the iteration. In the nonlinear autoencoder algorithm, a 'bottleneck' 5-layer network is trained to reproduce the input values as outputs (i.e. it is used in autoassociative mode). The hidden unit activations in the third layer form a lower-dimensional representation of the data, closely related to PCA (see for instance Diamantaras and Kung (1996)). Training is done by conjugate gradient descent. In all algorithms, parameter values were selected such that the best possible denoising result was obtained. The figure shows that on the closed square problem, kernel PCA does (subjectively) best, followed by principal curves and the nonlinear autoencoder; linear PCA fails completely. However, note that all algorithms except for kernel PCA actually provide an explicit one-dimensional parameterization of the data, whereas kernel PCA only provides us with a means of mapping points to their denoised versions (in this case, we used four kernel PCA features, and hence obtain a four-dimensional parameterization).

6.1.2 Handwritten Digit Denoising

To test our approach on real-world data, we also applied the algorithm to the USPS database of handwritten digits (e.g. LeCun et al., 1989; Schölkopf, 1997) of 7291 training patterns and 2007 test patterns (size 16×16). For each of the ten digits, we randomly chose 300 examples from the training set and 50 examples from the test set. We used the method of section 4.2, with $k(\mathbf{x}, \mathbf{y}) = \exp(-\|\mathbf{x} - \mathbf{y}\|^2/(0.5 \cdot 16^2))$. The width 0.5 equals twice the average of the data's variance in each dimension. In figure 8, we give two possible depictions of the eigenvectors found by kernel PCA, compared to those found by linear PCA for the USPS set. The second row shows the approximate pre-images of the eigenvectors $\mathbf{V}^k$, $k = 2^0, \ldots, 2^8$, found by our iterative algorithm. In the third row each image is computed as follows: Pixel i is the projection of the Φ-image of the i-th canonical basis vector in input space onto the corresponding eigenvector in features

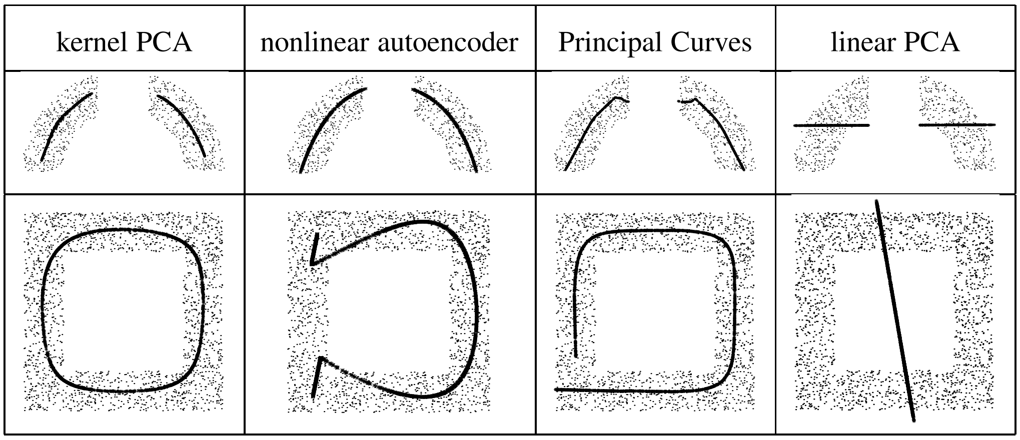

Figure 7: Denoising in 2-d (see text). Depicted are the data set (small points) and its denoised version (big points, joining up to solid lines). For linear PCA, we used one component for reconstruction, as using two components, reconstruction is perfect and thus does not denoise. Note that all algorithms except for our approach have problems in capturing the circular structure in the bottom example (taken from Mika et al., 1999).

space (upper left $\Phi(\mathbf{e}_1) \cdot \mathbf{V}^k$, lower right $\Phi(\mathbf{e}_{256}) \cdot \mathbf{V}^k$). In the linear case, both methods would simply yield the eigenvectors of linear PCA depicted in the first row; in this sense, they may be considered as generalized eigenvectors in input space. We see that the first eigenvectors are almost identical (except for arbitrary signs). However, we also see that eigenvectors in linear PCA start to focus on high-frequency structures already at smaller eigenvalue size. To understand this, note that in linear PCA we only have a maximum number of 256 eigenvectors, contrary to kernel PCA which gives us the number of training examples (here 3000) possible eigenvectors.

This also explains some of the results we found when working with the USPS set (figures 9 and 10). In these experiments, linear and kernel PCA were trained with the original data. To the test set, we added

(i) additive Gaussian noise with zero mean and standard deviation $\sigma = 0.5$, or

(ii) 'speckle' noise, where each pixel is flipped to black or white with probability $p = 0.2$.

For the noisy test sets, we computed the projections onto the first n linear and nonlinear components, and carried out reconstruction for each case. The results were compared by taking the mean squared distance of

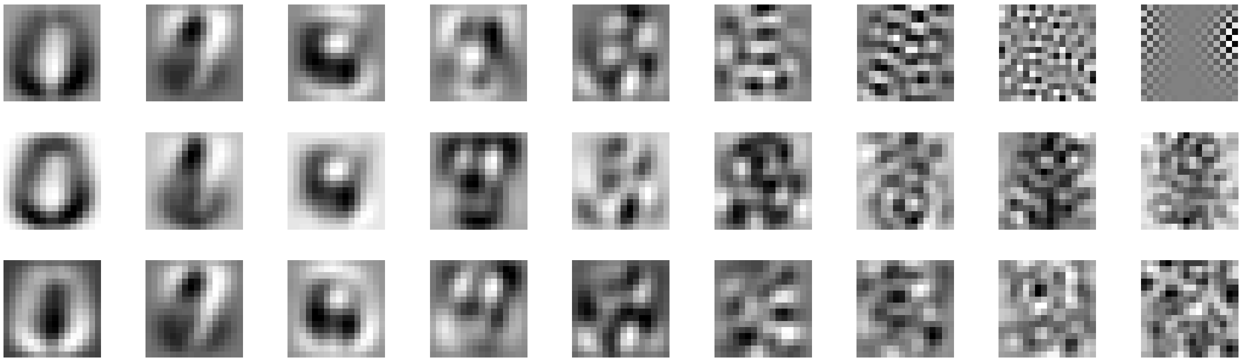

Figure 8: Visualization of eigenvectors (see text). Depicted are the $2^0, \ldots, 2^8$-th eigenvector (from left to right). First row: linear PCA, second and third row: different visualizations for kernel PCA.

each reconstructed digit of the noisy test set to its original counterpart.

For the optimal number of components in linear and kernel PCA, our approach did better by a factor of 1.6 for the Gaussian noise, and 1.2 for the 'speckle' noise (the optimal number of components were 32 in linear PCA, and 512 and 256 in kernel PCA, respectively). Taking identical numbers of components in both algorithms, kernel PCA becomes up to 8 times better than linear PCA. However, note that kernel PCA comes with a higher computational complexity.

6.2 Speeding up Support Vector Decision Rules

As in section 6.1, we used the USPS handwritten digit database. We approximated the SV expansions (31) of ten binary classifiers, each trained to separate one digit from the rest. We used the Gaussian kernel $k(\mathbf{x}, \mathbf{y}) = \exp(-\|\mathbf{x} - \mathbf{y}\|^2/(0.5 \cdot 16^2))$, and the approximation techniques described in sections 5.3 and 5.4.

The original SV system had on average 254 SVs per classifier. Table 2 shows the classification error results for approximation using the reduced set selection technique described in section 5.3, while table 3 gives results for using the reduced set construction method described in 5.4, for varying numbers of RS vectors. Shown in each line is the number of misclassified digits for each single classifier and the error of the combined 10-class machine.

For the method of section 5.3, RSS-n means, that for each binary classifier, we removed support vectors until only n were left; for the method of section 5.4, that we approximated each decision function using n vectors (i.e. n steps using the described iteration procedure). For large numbers n, i.e. small reductions of the decision functions complexity, the accuracy of the original system can be approached closely with both techniques. In table 2 we see that removal of up to 40% of the support vectors leaves the error practically unchanged. Reducing the numbers of support vectors further can lead to large performance losses.

The second method, which is computationally and conceptually more complex, performs better in this situation as it is able to utilize vectors different from the original support patterns in the expansion. To get a speedup by a factor of 10, we have to use a system with 25 RS vectors (RSC-25). For the method in table 3, the classification accuracy only drops moderately from 4.4% to 5.1%, which is still competitive with convolutional neural networks on that data base (LeCun et al., 1989). Moreover, we can further improve this result by adding the second phase of the traditional RS algorithm, where a global gradient descent is performed in the space of all $(\mathbf{z}_i, \beta_i)$ (Burges, 1996; Burges and Schölkopf, 1997) (computationally more expensive than the first phase by about two orders of magnitude): this led to an error rate of 4.7%. For the considered kernel, this is almost identical to the traditional RS method, which yielded 5.0% (for polynomial kernels, the latter method led to 4.3% at the same speedup (Burges, 1996)). Note that the traditional RS method restarts the second phase many times to make sure that the global minimum of the cost function is actually found, which makes it plausible that the final results are similar.

Finally, figure 11 shows the RS-20 vectors of the 10 binary classifiers. As an aside, note that unlike the approach of Burges (1996), our algorithm produces images which do look meaningful (i.e. digit-like).

7 Discussion

Algorithms utilizing Mercer kernels construct their solutions as expansions $\Psi = \sum_{i=1}^{\ell} \alpha_i \Phi(\mathbf{x}_i)$ in terms of mapped input patterns. However, the map Φ is often unknown, or too complex to provide any intuition about the solution Ψ. This has motivated our efforts to reduce the complexity of the expansion, summarized in this paper.

As an extreme case, we have first studied how to approximate Ψ by a single $\Phi(\mathbf{z})$ (i.e. how to find an approximate pre-image of Ψ), and proposed a fixed-point iteration algorithm to perform the task.

In situations where no good approximate pre-image exists, one can still reduce the complexity of $\sum_{i=1}^{\ell} \alpha_i \Phi(\mathbf{x}_i)$ by expressing it as a sparser expansion $\sum_{i=1}^{m} \beta_i \Phi(\mathbf{z}_i)$ ($m < \ell$). We have proposed methods for computing the optimal coefficients β_i and for coming up with suitable patterns $\mathbf{z}_i$ either by selecting among the $\mathbf{x}_i$ or by iterating the above pre-image algorithm.

Both types of approximations are of theoretical interest for feature space methods; however, they also lead to practical applications. In this paper, we have considered two such applications, namely, the problem of statistical de-noising via kernel PCA reconstruction, and the problem of speeding up SV decision rules. We address them in turn.

Kernel PCA de-noising. In de-noising experiments on real-world data, we obtained results significantly better than using linear PCA. Our interpretation of this finding is as follows. Linear PCA can extract at most N components, where N is the dimensionality of the data. Being a basis transform, all N components together fully describe the data. If the data are noisy, this implies that a certain fraction of the components will be devoted to the extraction of noise. Kernel PCA, on the other hand, allows the extraction of up to ℓ features, where ℓ is the number of training examples. Accordingly, kernel PCA can provide a larger number of features carrying information about the structure in the data (in our experiments, we had $\ell > N$). In addition, if the structure to be extracted is nonlinear, then linear PCA must necessarily fail, as we have illustrated with toy examples.

Open questions and problems include the choice of a suitable kernel for a given noise reduction problem, possibly in conjunction with the regularization properties of the kernel (e.g. Williamson et al., 1998), the application of the approach to compression, and the comparison (and connection) to alternative nonlinear denoising methods (cf. Mallat and Zhang, 1993).

Speeding up SV machines. We have shown experimentally that our approximation algorithms can be used to speed up a Gaussian RBF SV machine. Note that in the Gaussian RBF case, the approximation can never be as good as the original, since the kernel matrix $K_{ij} = (k(\mathbf{y}_i, \mathbf{y}_j))$ has full rank (Micchelli, 1986).

As in (Burges and Schölkopf, 1997), good results are obtained even though the objective function does not decrease to zero (in our experiments, it was reduced by a factor of 2 to 20 in the first phase, depending on how many RS vectors were computed; the global gradient descent yielded another factor 2 – 3). We conjecture that this is due to the following: in classification, we are not interested in $\|\Psi - \Psi'\|$, but in $\int |\text{sgn}(\sum_{i=1}^{N_y} \gamma_i k(\mathbf{x}, \mathbf{y}_i) + b) - \text{sgn}(\sum_{j=1}^{N_z} \beta_i k(\mathbf{x}, \mathbf{z}_i) + \tilde{b})| dP(\mathbf{x})$, where P is the underlying probability distribution of the patterns (cf. Blackmore et al., 1997). This is consistent with the fact that the performance of a RS SV classifier can be improved by re-computing an optimal threshold $\tilde{b}$.

The previous RS method (Burges, 1996; Burges and Schölkopf, 1997) can be used for any SV kernel; the new one is limited to $k(\mathbf{x}, \mathbf{y}) = k(\|\mathbf{x} - \mathbf{y}\|^2)$. However, it is fast, and it led to interpretable RS images and an interesting connection between clustering and approximation in feature spaces. It appears intriguing to pursue the question whether this connection could be exploited to form more general types of approximations of SV and kernel PCA expansions by making use of Gaussians of variable widths.

All proposed methods are applicable to arbitrary feature space algorithms. For instance, we could also speed up SV regression machines or kernel PCA feature extractors. Moreover, we expect further possibilities to open up in the future, as Mercer kernel methods are applied in an increasing number of learning and signal processing problems.

Acknowledgments Part of this work was supported by the DFG (# Ja 379/71 and 379/51) and the ARC.

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.

N. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68: 337 – 404, 1950.

K. L. Blackmore, R. C. Williamson, and I. M. Y. Mareels. Decision region approximation by polynomials or neural networks. *IEEE Transactions in Information Theory*, 43:903 – 907, 1997.

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, July 1992. ACM Press.

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.

C. J. C. Burges. Simplified support vector decision rules. In L. Saitta, editor, *Proceedings, 13th Intl. Conf. on Machine Learning*, pages 71–77, San Mateo, CA, 1996. Morgan Kaufmann.

C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):1–47, 1998.

C. J. C. Burges. Geometry and invariance in kernel based methods. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 89 – 116, Cambridge, MA, 1999. MIT Press.

C. J. C. Burges and B. Schölkopf. Improving the accuracy and speed of support vector learning machines. In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 375–381, Cambridge, MA, 1997. MIT Press.

B. Carl and I. Stephani. *Entropy, compactness, and the approximation of operators*. Cambridge University Press, Cambridge, UK, 1990.

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

T. Frieß and R. F. Harrison. Linear programming support vector machines for pattern classification and regression estimation; and the SR algorithm: improving speed and tightness of VC bounds in SV algorithms. Research Report 706, University of Sheffield, Dept. Automatic Control and Systems Engineering, 1998.

F. Girosi. An equivalence between sparse approximation and support vector machines. *Neural Computation*, 10(6):1455–1480, 1998.

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

T. J. Hastie and R. J. Tibshirani. *Generalized Additive Models*, volume 43 of *Monographs on Statistics and Applied Probability*. Chapman & Hall, London, 1990.

Y. LeCun, 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.

S. Mallat and Z. Zhang. Matching Pursuit in a time-frequency dictionary. *IEEE Transactions on Signal Processing*, 41:3397–3415, 1993.

D. Mattera. personal communication. 1998.

C. A. Micchelli. Interpolation of scattered data: distance matrices and conditionally positive definite functions. *Constructive Approximation*, 2:11–22, 1986.

S. Mika. Nichtlineare Signalverarbeitung in Feature-Räumen. Diplomarbeit, Technische Universität Berlin, 1998.

S. Mika, B. Schölkopf, A. Smola, K.-R. Müller, M. Scholz, and G. Rätsch. Kernel PCA and de-noising in feature spaces. In *Advances in Neural Information Processing Systems 11*, 1999. To appear.

E. Osuna and F. Girosi. Reducing run-time complexity in support vector machines. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 271 – 283. MIT Press, Cambridge, MA, 1999.

S. Saitoh. *Theory of Reproducing Kernels and its Applications*. Longman Scientific & Technical, Harlow, England, 1988.

B. Schölkopf. *Support Vector Learning*. R. Oldenbourg Verlag, Munich, 1997.

B. Schölkopf, P. Knirsch, A. Smola, and C. Burges. Fast approximation of support vector kernel expansions, and an interpretation of clustering as approximation in feature spaces. In P. Levi, M. Schanz, R.-J. Ahlers, and F. May, editors, *Mustererkennung 1998 — 20. DAGM-Symposium*, Informatik aktuell, pages 124 – 132, Berlin, 1998a. Springer.

B. Schölkopf, S. Mika, A. Smola, G. Rätsch, and K.-R. Müller. Kernel PCA pattern reconstruction *via* approximate pre-images. In L. Niklasson, M. Bodén, and T. Ziemke, editors, *Proceedings of the 8th International Conference on Artificial Neural Networks*, Perspectives in Neural Computing, pages 147 – 152, Berlin, 1998b. Springer Verlag.

B. Schölkopf, P. Simard, A. Smola, and V. Vapnik. Prior knowledge in support vector kernels. In M. Jordan, M. Kearns, and S. Solla, editors, *Advances in Neural Information Processing Systems 10*, pages 640 – 646, Cambridge, MA, 1998c. MIT Press.

B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299 – 1319, 1998d.

B. Schölkopf, K. Sung, C. Burges, F. Girosi, P. Niyogi, T. Poggio, and V. Vapnik. Comparing support vector machines with gaussian kernels to radial basis function classifiers. *IEEE Trans. Sign. Processing*, 45: 2758 – 2765, 1997.

A. Smola and B. Schölkopf. From regularization operators to support vector kernels. In M. Jordan, M. Kearns, and S. Solla, editors, *Advances in Neural Information Processing Systems 10*, pages 343 – 349, Cambridge, MA, 1998a. MIT Press.

A. Smola and B. Schölkopf. A tutorial on support vector regression. http://svm.first.gmd.de, 1998b.

K. Tsuda. Support vector classifier with asymmetric kernel function. In *Proc. ESANN*. D Facto, 1999.

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

V. Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998.

G. Wahba. *Spline Models for Observational Data*, volume 59 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. SIAM, Philadelphia, 1990.

R. Williamson, A. Smola, and B. Schölkopf. Entropy numbers, operators and support vector kernels. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 127 – 144. MIT Press, Cambridge, MA, 1999.

R. C. Williamson, A. J. Smola, and B. Schölkopf. Generalization performance of regularization networks and support vector machines via entropy numbers of compact operators. Technical Report NC-TR-98-019, Royal Holloway College, University of London, UK, 1998.

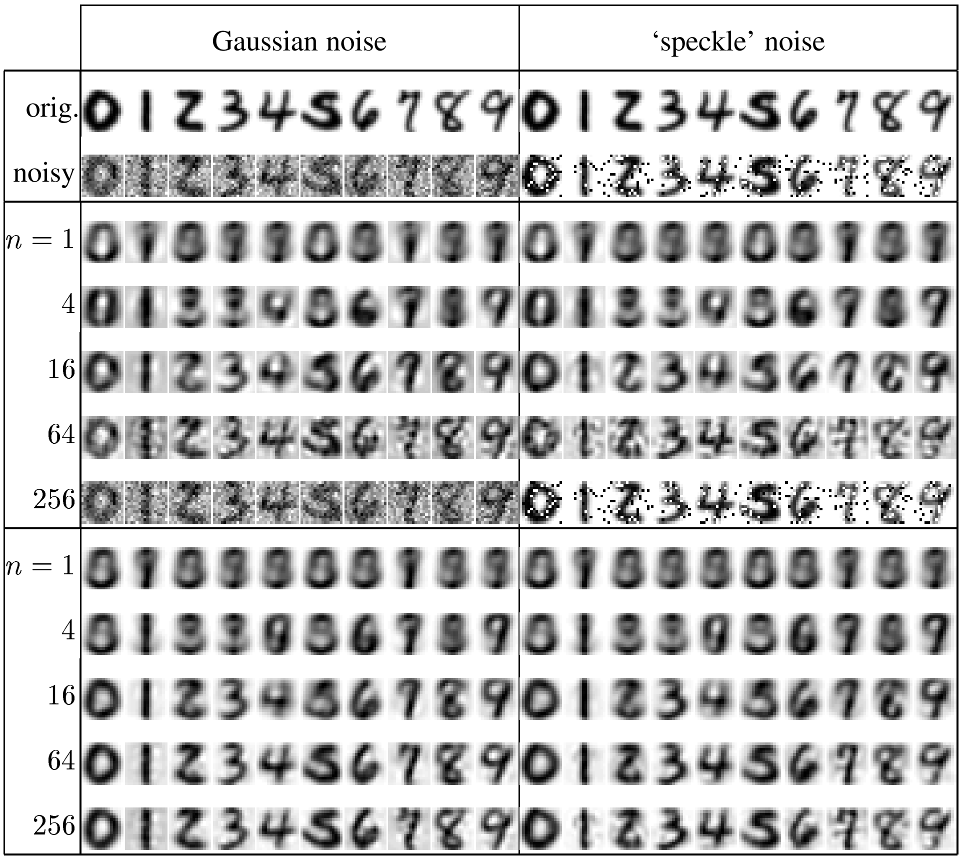

Figure 9: De-noising of USPS data (see text). The left half shows: *top:* the first occurrence of each digit in the test set, *second row:* the upper digit with additive Gaussian noise ($\sigma = 0.5$), *following five rows:* the reconstruction for linear PCA using $n = 1, 4, 16, 64, 256$ components, and, *last five rows:* the results of our approach using the same number of components. In the right half we show the same but for 'speckle' noise with probability $p = 0.2$.

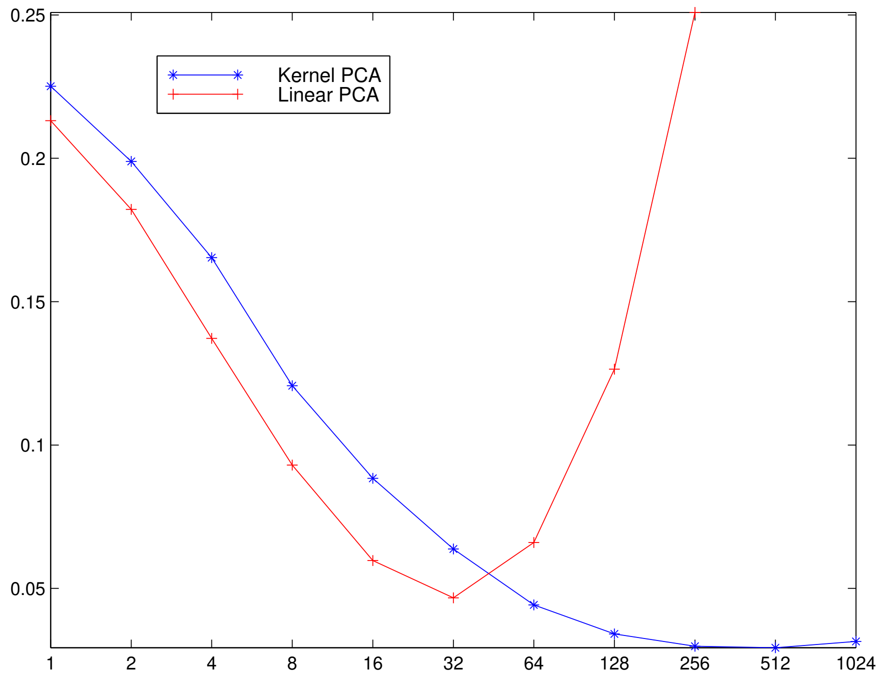

Figure 10: Mean squared error of denoised images vs. # of features used, for kernel PCA and linear PCA. Kernel PCA exploits nonlinearities and has the potential to utilize more features to code structure rather than noise. Therefore, it outperforms linear PCA denoising if a sufficiently large number of features is used.

Table 2: Numbers of test errors for each binary recognizer, and test error rates for 10-class classification using the RS selection method from section 5.3. Top: numbers of SVs for the original SV RBF-system. Bottom, first row: original SV system, with 254 SVs on average; following rows: systems with varying numbers of RS vectors (RSS-n stands for n RS vectors selected) per binary recognizer, selected from the original SVs as described in section 5.3, separately for each recognizer.

digit	0	1	2	3	4	5	6	7	8	9	ave.
#SVs	219	92	316	309	288	340	213	206	304	250	254

digit	0	1	2	3	4	5	6	7	8	9	10-class
SV-254	16	13	30	17	32	22	11	12	26	17	4.4%
RSS-10	474	101	581	288	549	851	536	328	601	334	61.7%
RSS-15	328	17	338	236	674	983	313	357	298	409	53.5%
RSS-20	216	28	297	260	330	668	302	524	340	560	45.4%
RSS-25	89	21	292	186	351	715	165	168	439	534	42.1%
RSS-50	44	12	91	79	288	353	67	90	207	172	20.2%
RSS-75	20	13	50	41	84	134	11	32	79	39	8.2%
RSS-100	15	13	38	25	48	46	11	16	43	24	5.9%
RSS-125	17	13	40	17	34	32	10	15	36	17	5.2%
RSS-150	17	13	33	27	33	27	10	14	32	16	4.6%
RSS-175	16	13	30	23	36	20	11	14	23	15	4.3%
RSS-200	16	13	28	18	33	19	11	12	24	17	4.4%
RSS-225	16	13	31	18	34	21	11	12	25	17	4.5%
RSS-250	16	13	31	17	32	22	11	12	23	17	4.6%

Table 3: Number of test errors for each binary recognizer, and test error rates for 10-class classification using the RS construction method of section 5.4. First row: original SV system, with 254 SVs on average (see also table 2); following rows: systems with varying numbers of RS vectors (RSC-n stands for n vectors constructed) per binary recognizer (optimal threshold re-computed on the training set), computed by iterating one-term approximations, separately for each recognizer. Last two rows: with a subsequent global gradient descent, the results can be further improved (see text).

digit	0	1	2	3	4	5	6	7	8	9	10-class
SV-254	16	13	30	17	32	22	11	12	26	17	4.4%
RSC-10	26	13	45	49	35	54	22	24	39	24	7.1%
RSC-15	17	16	43	50	49	37	14	18	45	35	6.4%
RSC-20	27	11	38	30	35	43	12	16	30	25	5.6%
RSC-25	21	12	38	32	31	22	12	18	33	28	5.1%
RSC-50	18	10	33	28	32	23	12	15	35	27	5.0%
RSC-100	14	13	26	22	30	26	11	14	28	23	4.8%
RSC-150	13	14	28	32	27	24	12	14	29	26	4.7%
RSC-200	14	13	28	28	29	24	10	15	26	26	4.9%
RSC-250	12	13	26	26	32	25	11	14	26	24	4.6%
RSC2-25	14	14	31	22	30	23	11	14	26	17	4.7%
RSC2^{old}-25	16	13	32	19	31	26	11	15	25	18	5.0%

Figure 11: Complete display of reduced set vectors constructed by the iterative approach of section 5.4 for $n = 20$, with coefficients (Top: recognizer of digit 0,…, bottom: digit 9). Note that positive coefficients (roughly) correspond to positive examples in the classification problem.