Support Vector Methods in Learning and Feature Extraction

Bernhard Schölkopf[§,†], Alex Smola[§,†], Klaus-Robert Müller[§],
Chris Burges[‡], Vladimir Vapnik[⋆]
[§]GMD FIRST, Rudower Chaussee 5, 12489 Berlin, Germany
[†]Australian National University, Engineering Dept., Canberra ACT 0200
[‡]Bell Laboratories, 101 Crawfords Corner Road, Holmdel NJ 07733, USA
[⋆]AT&T Research, 100 Schultz Dr., Red Bank, NJ 07701, USA
bs,smola,klaus@first.gmd.de, burges@bell-labs.com, vlad@research.att.com

ABSTRACT

The last years have witnessed an increasing interest in Support Vector (SV) machines, which use Mercer kernels for efficiently performing computations in high-dimensional spaces. In pattern recognition, the SV algorithm constructs nonlinear decision functions by training a classifier to perform a linear separation in some high-dimensional space which is nonlinearly related to input space. Recently, we have developed a technique for Nonlinear Principal Component Analysis (Kernel PCA) based on the same types of kernels. This way, we can for instance efficiently extract polynomial features of arbitrary order by computing projections onto principal components in the space of all products of n pixels of images.

We explain the idea of Mercer kernels and associated feature spaces, and describe connections to the theory of reproducing kernels and to regularization theory, followed by an overview of the above algorithms employing these kernels.

1. Introduction

For the case of two-class pattern recognition, the task of *learning from examples* can be formulated in the following way: we are given a set of functions

$$\{f_\alpha : \alpha \in \Lambda\}, \quad f_\alpha : \mathbf{R}^N \to \{\pm 1\} \tag{1}$$

and a set of *examples*, i.e. pairs of *patterns* $\mathbf{x}_i$ and *labels* y_i,

$$\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)\} \subset \mathbf{R}^N \times \{\pm 1\}, \tag{2}$$

each one of them generated from an unknown probability distribution $P(\mathbf{x}, y)$ containing the underlying dependency. We want to *learn* a function f_{α^*} minimizing the average error committed on independent examples randomly drawn from the same distribution, called the *risk*

$$R(\alpha) = \int \frac{1}{2}|f_\alpha(\mathbf{x}) - y| \, dP(\mathbf{x}, y). \tag{3}$$

The problem is that R is unknown, since P is unknown. Therefore an *induction principle* for risk minimization is necessary. The straightforward approach to minimize the *empirical risk*

$$R_{emp}(\alpha) = \frac{1}{\ell}\sum_{i=1}^{\ell} \frac{1}{2}|f_\alpha(\mathbf{x}_i) - y_i| \tag{4}$$

does not guarantee a small actual risk, if the number ℓ of training examples is limited. Therefore, novel statistical techniques have been developed during the last 30 years. The *Structural Risk Minimization* principle [23] is based on the fact that for the above learning problem, for any $\alpha \in \Lambda$ and $\ell > h$, with a probability of at least $1-\eta$, the bound

$$R(\alpha) \le R_{emp}(\alpha) + \phi\left(\frac{h}{\ell}, \frac{\log(\eta)}{\ell}\right) \tag{5}$$

holds, where the *confidence term* ϕ is defined as

$$\phi\left(\frac{h}{\ell}, \frac{\log(\eta)}{\ell}\right) = \sqrt{\frac{h\left(\log\frac{2\ell}{h} + 1\right) - \log(\eta/4)}{\ell}}. \tag{6}$$

The parameter h is called the *VC(Vapnik-Chervonenkis)-dimension* of a set of functions. To control h, one introduces a structure of nested subsets $S_n := \{f_\alpha : \alpha \in \Lambda_n\}$ of $\{f_\alpha : \alpha \in \Lambda\}$. For a given set of observations $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)$, the Structural Risk Minimization principle chooses the function $f_{\alpha_\ell^n}$ in the subset $\{f_\alpha : \alpha \in \Lambda_n\}$ for which the guaranteed risk bound (the right hand side of (5)) is minimal.

The above remarks show that for learning, the proper choice of a set of functions which the learning machine can implement is crucial. It should allow a small training error yet still have small capacity. For a problem at hand, our ability to choose such a set of functions critically depends on the *representation* of the data. A striking example is the problem of backing up a truck with a trailer to a given position [7]. This is a complicated classification problem (steering wheel left or right) when expressed in cartesian coordinates; in polar coordinates, however, it becomes linearly separable.

More formally speaking, we are free to take advantage of the fact that by preprocessing our data with a fixed map g, and constructing a function $f = f^* \circ g$, the problem is reduced to learning f^*, and we need not worry anymore about potentially overly complex functions f. By a suitably a priori chosen g, we are often able to select a learning machine (i.e. a set of functions that f^* is chosen from) with a comparably small VC-dimension. The map g is referred to as performing *preprocessing* or *feature extraction*.

In this paper, we shall briefly give examples of algorithms performing the tasks of pattern recognition and feature extraction, respectively. In Sec. 3, we describe the *Support Vector algorithm*, which approximately performs Structural Risk Minimization; and in Sec. 4, we present a nonlinear feature extraction algorithm called *Kernel PCA*. In our exposition, both algorithms merely serve to illustrate a method for dealing with nonlinearities which has a potential far exceeding these two applications. This method will be explained in the next section.

2. Feature Spaces

Suppose we are given patterns $\mathbf{x} \in \mathbf{R}^N$ where most information is contained in the d-th order products (monomials) of entries x_j of $\mathbf{x}$, $x_{j_1} \cdot \ldots \cdot x_{j_d}$, where $j_1, \ldots, j_d \in \{1, \ldots, N\}$. In that case, we might prefer to *extract* these product features first, and work in the feature space F of all products of d entries. This approach fails for realistically sized problems: for N-dimensional input patterns, there exist $(N+d-1)!/(d!(N-1)!)$ different monomials. Already 16×16 pixel input images (e.g. in optical character recognition) and a monomial degree $d = 5$ yield a dimensionality of 10^{10}.

In certain cases described below, there exists, however, a way of *computing dot products* in these high-dimensional feature spaces without explicitly mapping into them: by means of nonlinear kernels in input space $\mathbf{R}^N$. Thus, if the subsequent processing can be carried out using dot products exclusively, we are able to deal with the high dimensionality. In order to compute dot products of the form $(\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}))$, we employ kernel representations of the form $k(\mathbf{x}, \mathbf{y}) = (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}))$. This method was used to extend the *Generalized Portrait* hyperplane classifier to nonlinear Support Vector machines [1, 25, 2]. If F is high-dimensional, we would like to be able to find a closed form expression for k which can be efficiently computed.

What does k look like for the case of polynomial features? We start by giving an example [24] for $N = d = 2$. For the map

$$C_2 : (x_1, x_2) \mapsto (x_1^2, x_2^2, x_1x_2, x_2x_1), \qquad (7)$$

dot products in F take the form

$$(C_2(\mathbf{x}) \cdot C_2(\mathbf{y})) = x_1^2y_1^2 + x_2^2y_2^2 + 2x_1x_2y_1y_2 = (\mathbf{x} \cdot \mathbf{y})^2,$$

i.e. the desired kernel k is simply the square of the dot product in input space. In [2] it was noted that the same works for arbitrary $N, d \in \mathbf{N}$:

Proposition 2.1 *Define C_d to map $\mathbf{x} \in \mathbf{R}^N$ to the vector $C_d(\mathbf{x})$ whose entries are all possible d-th degree ordered products of the entries of $\mathbf{x}$. Then the corresponding kernel computing the dot product of vectors mapped by C_d is*

$$k(\mathbf{x}, \mathbf{y}) = (C_d(\mathbf{x}) \cdot C_d(\mathbf{y})) = (\mathbf{x} \cdot \mathbf{y})^d. \qquad (8)$$

Proof. We directly compute $(C_d(\mathbf{x}) \cdot C_d(\mathbf{y})) = \sum_{j_1,\ldots,j_d=1}^N x_{j_1} \cdot \ldots \cdot x_{j_d} \cdot y_{j_1} \cdot \ldots \cdot y_{j_d} = \left(\sum_{j=1}^N x_j \cdot y_j\right)^d = (\mathbf{x} \cdot \mathbf{y})^d$. □

Instead of ordered products, we can use unordered ones to obtain a map Φ_d which yields the same value of the dot product. To this end, we have to compensate for the multiple occurence of certain monomials in C_d by scaling the respective monomial entries of Φ_d with the square roots of their numbers of occurence.

If $\mathbf{x}$ represents an image with the entries being pixel values, we can use the kernel $(\mathbf{x} \cdot \mathbf{y})^d$ to 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 $\Phi_d(\mathbf{x})$. Using kernels of the form (8), we take into account higher-order statistics without the combinatorial explosion of time and memory complexity which goes along already with moderately high N and d.

Rather than constructing k to compute the dot product for a given Φ, we may ask the question which function k, chosen a priori, does correspond to a dot product in some space F [2, 24]. To construct a map Φ induced by a kernel k, i.e. a map Φ such that k computes the dot product in the space that Φ maps to, Mercer's theorem of functional analysis is used [6]. It states that if k is a continuous symmetric kernel of a positive integral operator K on $L^2(C)$ (C being a compact subset of $\mathbf{R}^N$), it can be expanded in a uniformly convergent series in terms of Eigenfunctions ψ_j and positive Eigenvalues λ_j, $(N_F \leq \infty)$

$$k(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{N_F} \lambda_j \psi_j(\mathbf{x}) \psi_j(\mathbf{y}). \qquad (9)$$

From (9), it is straightforward to construct a map Φ, mapping into a potentially infinite-dimensional

l^2 space, which does the job. For instance, we may use

$$\Phi : \mathbf{x} \mapsto (\sqrt{\lambda_1}\psi_1(\mathbf{x}), \sqrt{\lambda_2}\psi_2(\mathbf{x}), \ldots). \tag{10}$$

We thus have the following result:

Proposition 2.2 *If k is a continuous symmetric kernel of a positive integral operator, one can construct a mapping Φ into a space where k acts as a dot product,*

$$k(\mathbf{x}, \mathbf{y}) = (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})). \tag{11}$$

Besides (8), we can for instance use Gaussian radial basis function kernels [1, 17]

$$k(\mathbf{x}, \mathbf{y}) = \exp\left(-\|\mathbf{x} - \mathbf{y}\|^2/(2\,\sigma^2)\right) \tag{12}$$

and, for certain values of κ and Θ, sigmoid kernels

$$k(\mathbf{x}, \mathbf{y}) = \tanh(\kappa(\mathbf{x} \cdot \mathbf{y}) + \Theta). \tag{13}$$

We conclude this section by describing the connection to the theory of *reproducing kernel Hilbert spaces* (RKHS). To this end, consider the map

$$\begin{array}{rcl} \tilde{\Phi} : \mathbf{R}^N & \longrightarrow & \mathcal{H} \\ \mathbf{x} & \mapsto & k(\mathbf{x}, .). \end{array} \tag{14}$$

Can we endow $\mathcal{H}$ with a dot product $\langle ., . \rangle$ such that $\langle \tilde{\Phi}(\mathbf{x}), \tilde{\Phi}(\mathbf{y}) \rangle = k(\mathbf{x}, \mathbf{y})$, i.e. such that $\mathcal{H}$ is an alternative representation of the feature space that we are working in by using k? Clearly, this dot product would have to satisfy

$$\langle k(\mathbf{x}, .), k(\mathbf{y}, .) \rangle = k(\mathbf{x}, \mathbf{y}), \tag{15}$$

which amounts to saying that k is a *reproducing kernel* for $\mathcal{H}$.

For a Mercer kernel (9), such a dot product does exist. Since k is symmetric, the ψ_i $(i = 1, \ldots, N_F)$ can be chosen to be orthogonal with respect to the dot product in $L^2(C)$, and hence we may construct $\langle ., . \rangle$ such that

$$\langle \sqrt{\lambda_j}\psi_j, \sqrt{\lambda_n}\psi_n \rangle = \delta_{jn}, \tag{16}$$

using the Kronecker δ_{jn}. Substituting (9) into (15) then proves the desired equality.

$\mathcal{H}$, the closure of the space of all functions

$$f(\mathbf{x}) = \sum_{i=1}^{\infty} a_i k(\mathbf{x}, \mathbf{x}_i) = \sum_{i=1}^{\infty} a_i \sum_{j=1}^{N_F} \lambda_j \psi_j(\mathbf{x}) \psi_j(\mathbf{x}_i), \tag{17}$$

with the dot product $\langle ., . \rangle$, is called an RKHS [26, 27, 8, 14].

What is the connection between F and $\mathcal{H}$? Let us write $\langle ., . \rangle$ as a dot product of coordinate vectors, by expressing $f \in \mathcal{H}$ in the basis $(\sqrt{\lambda_n}\psi_n)_{n=1,\ldots,N_F}$,

$$f(\mathbf{x}) = \sum_{n=1}^{N_F} \alpha_n \sqrt{\lambda_n}\psi_n(\mathbf{x}). \tag{18}$$

Using the dot product in F, this can be written as $f(\mathbf{x}) = (\boldsymbol{\alpha} \cdot \Phi(\mathbf{x}))$. To obtain the α_n, we compute, using (16) and (17),

$$\alpha_n = \langle f, \sqrt{\lambda_n}\psi_n \rangle = \sqrt{\lambda_n} \sum_{i=1}^{\infty} a_i \psi_n(\mathbf{x}_i). \tag{19}$$

Comparing (18) and (10), we see that F has the structure of a RKHS in the sense that for f given by (18), and $g(\mathbf{x}) = (\boldsymbol{\beta} \cdot \Phi(\mathbf{x}))$, we have

$$(\boldsymbol{\alpha} \cdot \boldsymbol{\beta}) = \langle f, g \rangle. \tag{20}$$

Therefore, we can alternatively think of the feature space as an RKHS of functions (17) where only functions of the form (14) have a pre-image in input space.

3. Support Vector Machines

Given a dot product space $\mathbf{Z}$ (e.g. the input space $\mathbf{R}^N$, or a feature space F), a hyperplane $\{\mathbf{z} \in \mathbf{Z} : (\mathbf{w} \cdot \mathbf{z}) + b = 0\}$, and a set of examples $(\mathbf{z}_1, y_1), \ldots, (\mathbf{z}_\ell, y_\ell) \in \mathbf{Z}$, disjoint from the hyperplane, we are looking for parameters $(\mathbf{w}, b)$ to separate the data, i.e.

$$y_i((\mathbf{w} \cdot \mathbf{z}_i) + b) \geq \delta, \quad i = 1, \ldots, \ell \tag{21}$$

for some $\delta > 0$. First note that we can always rescale $(\mathbf{w}, b)$ such that

$$\min_{i=1,\ldots,\ell} |(\mathbf{w} \cdot \mathbf{z}_i) + b| = 1, \tag{22}$$

i.e. such that the point closest to the hyperplane has a distance of $1/\|\mathbf{w}\|$.[1] Then, (21) becomes

$$y_i((\mathbf{w} \cdot \mathbf{z}_i) + b) \geq 1, \quad i = 1, \ldots, \ell. \tag{23}$$

In the case of pattern recognition, the SV algorithm is based on two facts: first, the complexity of the classifier can be kept low by minimizing $\|\mathbf{w}\|$ (amounting to maximizing the margin of separation) subject to the condition of separating the data (23); and second, this minimization can be carried out as a quadratic program based solely on values of dot products [24]. Hence one may employ the feature space methods of the previous section to construct nonlinear decision functions.

In practice, a separating hyperplane often does not exist. To allow for the possibility of examples violating (23), one introduces slack variables [5]

$$\xi_i \geq 0, \quad i = 1, \ldots, \ell, \tag{24}$$

to get

$$y_i((\mathbf{w} \cdot \mathbf{z}_i) + b) \geq 1 - \xi_i, \quad i = 1, \ldots, \ell. \tag{25}$$

[1] Strictly speaking, we should use training *and* test patterns in (22). For separable problems, using just the training set is a reasonable approximation.

The SV approach to minimizing the guaranteed risk bound (5) consists of the following: minimize

$$\tau(\mathbf{w}, \xi) = \frac{\lambda}{2}(\mathbf{w} \cdot \mathbf{w}) + \sum_{i=1}^{\ell} \xi_i \tag{26}$$

subject to the constraints (24) and (25). The first term is minimized to control the second term of the bound (5); the second term, on the other hand, is an upper bound on the number of misclassifications on the training set, i.e. the empirical risk.[2]

Introducing Lagrange multipliers α_i, and using the Kuhn-Tucker theorem of optimization theory, the solution can be shown to have an expansion

$$\mathbf{w} = \sum_{i=1}^{\ell} y_i \alpha_i \mathbf{z}_i, \tag{27}$$

with nonzero coefficients α_i only where the corresponding example $(\mathbf{z}_i, y_i)$ precisely meets the constraint (25). These $\mathbf{z}_i$ are called *Support Vectors*. All other training examples are irrelevant: their constraint (25) is satisfied automatically (with $\xi_i = 0$), and they do not appear in the expansion (27). The coefficients α_i are found by maximizing

$$W(\alpha) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i \alpha_j y_i y_j (\mathbf{z}_i \cdot \mathbf{z}_j) \tag{28}$$

subject to

$$0 \leq \alpha_i \leq \frac{1}{\lambda}, \quad i = 1, \ldots, \ell, \text{ and } \sum_{i=1}^{\ell} \alpha_i y_i = 0. \tag{29}$$

The hyperplane decision function can thus be written as

$$f(\mathbf{z}) = \operatorname{sgn}\left(\sum_{i=1}^{\ell} y_i \alpha_i \cdot (\mathbf{z} \cdot \mathbf{z}_i) + b \right). \tag{30}$$

To allow for much more general decision surfaces, one substitutes a suitable kernel function k for the dot product, leading to decision functions of the form

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

and a quadratic program with target function

$$W(\alpha) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j). \tag{32}$$

Empirically, different SV machines have been found to use largely the same SVs $\mathbf{x}_i$; e.g. most of the centers of an SV machine with Gaussian kernel (12) [17] coincide with the first-layer weights of SV classifiers with tanh kernel (13) (in which case the trained SV machine looks like a neural network) [13]. Moreover, experimental results have shown that in digit and object recognition tasks, SV machines are competitive with state-of-the-art techniques [14], especially when enhanced by methods for incorporating prior knowledge about the problem at hand [15]. Other areas where SV machines have been successfully applied include time series prediction [11] and text categorization [10].

From a *computational* point of view, the formulation as a quadratic programming problem with a positive matrix (cf. (32)) is crucial, as it allows the risk minimization problem to be solved efficiently.[3] From a *statistical* point of view, it is crucial that the kernel method allows to reduce a large class of learning machines to separating hyperplanes in some space. For those, an upper bound on the VC-dimension can be given ([24], cf. [18, 4] for a caveat), which is taken into account in training the classifier. This bound does not depend on the dimensionality of the feature space, but on the separation margin of the classes. This is how the SV machine handles the "curse of dimensionality." Along similar lines, analyses of generalization performance in terms of separation margins and fat shattering dimension are relevant to SV machines [12].

Additionally, the connection to regularization theory provides insight. In [20], a regularization framework is described which contains the SV algorithm as a special case. For kernel-based function expansions, it is shown that given a regularization operator P mapping the functions of the learning machine into some dot product space $\mathcal{D}$, the problem of minimizing the regularized risk

$$R_{reg}[f] = R_{emp}[f] + \frac{\lambda}{2}\|Pf\|^2, \tag{33}$$

(with a regularization parameter $\lambda \geq 0$) can be written as a constrained optimization problem. For particular choices of the cost function, it further reduces to a SV type quadratic programming problem. The latter thus is not specific to SV machines, but is common to a much wider class of approaches. What gets lost in this case, however, is the fact that the solution can usually be expressed in terms of a small number of SVs (cf. also [8]). This specific feature of SV machines is due to the fact that the type of regularization and the class of functions which are considered as admissible solutions are intimately related [9, 19, 21]: the SV algorithm is

[2] The computational simplicity of the problem stays the same in regression tasks, even with more general cost functions [22].

[3] This refers to the training of the machine, but not to its application on test examples. In the latter case, the computational complexity can be larger than for neural nets. Methods for reducing it have successfully been applied in character recognition [3].

equivalent to minimizing the regularized risk on the set of functions

$$f(\mathbf{x}) = \sum_i \alpha_i k(\mathbf{x}_i, \mathbf{x}) + b, \tag{34}$$

provided that k and P are interrelated by

$$k(\mathbf{x}_i, \mathbf{x}_j) = ((Pk)(\mathbf{x}_i, .) \cdot (Pk)(\mathbf{x}_j, .)). \tag{35}$$

To this end, k is chosen as Green's function of P^*P, for in that case, the right hand side of (35) equals $(k(\mathbf{x}_i, .) \cdot (P^*Pk)(\mathbf{x}_j, .)) = (k(\mathbf{x}_i, .) \cdot \delta_{\mathbf{x}_j}(.)) = k(\mathbf{x}_i, \mathbf{x}_j)$.

For instance, an RBF kernel thus corresponds to regularization with a functional containing a specific differential operator.

In the context of SV machines, often the question arises as to which kernel should be chosen for a particular learning task. In view of the above, the answer comprises two parts. First, the kernel determines the class of functions (34) that the solution is taken from; second, via (35), the kernel determines the type of regularization that is used.

4. Kernel PCA

Principal Component Analysis (PCA) is a basis transformation to diagonalize an estimate of the covariance matrix of the data $\mathbf{x}_k$, $k = 1, \ldots, \ell$, $\mathbf{x}_k \in \mathbf{R}^N$, $\sum_{k=1}^{\ell} \mathbf{x}_k = 0$, defined as $C = \frac{1}{\ell}\sum_{j=1}^{\ell} \mathbf{x}_j \mathbf{x}_j^\top$. The new coordinates in the Eigenvector basis, i.e. the orthogonal projections onto the Eigenvectors, are called *principal components*. We have generalized this setting to a nonlinear one, using kernels and associated feature spaces [16].

Assume for the moment that our data mapped into feature space, $\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_\ell)$, is centered, i.e. $\sum_{k=1}^{\ell} \Phi(\mathbf{x}_k) = 0$. To do PCA for the covariance matrix

$$\bar{C} = \frac{1}{\ell}\sum_{j=1}^{\ell} \Phi(\mathbf{x}_j)\Phi(\mathbf{x}_j)^\top, \tag{36}$$

we have to find Eigenvalues $\lambda \geq 0$ and Eigenvectors $\mathbf{V} \in F\backslash\{0\}$ satisfying $\lambda\mathbf{V} = \bar{C}\mathbf{V}$. Substituting (36), we note that all solutions $\mathbf{V}$ with $\lambda \neq 0$ lie in the span of $\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_\ell)$. This implies that we may consider the equivalent system

$$\lambda(\Phi(\mathbf{x}_k) \cdot \mathbf{V}) = (\Phi(\mathbf{x}_k) \cdot \bar{C}\mathbf{V}) \text{ for all } k = 1, \ldots, \ell, \tag{37}$$

and that there exist coefficients $\alpha_1, \ldots, \alpha_\ell$ such that

$$\mathbf{V} = \sum_{i=1}^{\ell} \alpha_i \Phi(\mathbf{x}_i). \tag{38}$$

Substituting (36) and (38) into (37), and defining an $\ell \times \ell$ matrix K by

$$K_{ij} := (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)) = (k(\mathbf{x}_i, \mathbf{x}_j)), \tag{39}$$

we arrive at a problem which is cast in terms of dot products: solve

$$\ell\lambda\boldsymbol{\alpha} = K\boldsymbol{\alpha} \tag{40}$$

for nonzero Eigenvalues λ, and coefficient Eigenvectors $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_\ell)^\top$. We normalize the solutions $\boldsymbol{\alpha}^k$ by requiring that the corresponding vectors in F be normalized, i.e. $(\mathbf{V}^k \cdot \mathbf{V}^k) = 1$, which translates into $\lambda_k(\boldsymbol{\alpha}^k \cdot \boldsymbol{\alpha}^k) = 1$. For principal component extraction, we compute projections of the image of a test point $\Phi(\mathbf{x})$ onto the Eigenvectors $\mathbf{V}^k$ in F according to

$$(\mathbf{V}^k \cdot \Phi(\mathbf{x})) = \sum_{i=1}^{\ell} \alpha_i^k (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x})) = \sum_{i=1}^{\ell} \alpha_i^k k(\mathbf{x}_i, \mathbf{x}). \tag{41}$$

Note that for feature extraction, we thus have to evaluate ℓ kernel functions in input space rather than a dot product in a 10^{10}–dimensional space, say. Moreover, Kernel PCA can be carried out for all kernels described in Sec. 2, no matter whether we know the corresponding map Φ or not. The nonlinearity is taken into account implicitly when computing the matrix elements of K and when computing the projections (41), the remainder of the algorithm is simple linear algebra.

For the general case, we have to drop the assumption that the $\Phi(\mathbf{x}_i)$ are centered in F. Instead, we have to go through the above algebra using $\tilde{\Phi}(\mathbf{x}_i) := \Phi(\mathbf{x}_i) - (1/\ell)\sum_{i=1}^{\ell} \Phi(\mathbf{x}_i)$ (for details, see [16]).

In experiments comparing the utility of kernel PCA features for pattern recognition using a linear classifier, we found two advantages of nonlinear kernel PCA: 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. In that case, the performance is competitive with the best nonlinear SV machines, which in turn beat Neural Networks like LeNet1 (for more benchmark results, see [24]). A simple toy example of kernel PCA is shown in Fig. 1.

SV machines and kernel PCA have been the first two applications of the powerful idea of Mercer kernels in machine learning technology. They share this crucial ingredient, yet they are based on different learning paradigms — supervised, and unsupervised, respectively. Nevertheless, an interesting parallel has recently been discovered: if one constructs transformation invariant SV machines by requiring local invariance with respect to some Lie group of transformations $\mathcal{L}_t$, one arrives at the result [15] that this can be achieved by a preprocessing matrix $B = C^{-\frac{1}{2}}$, where C is the *tangent*

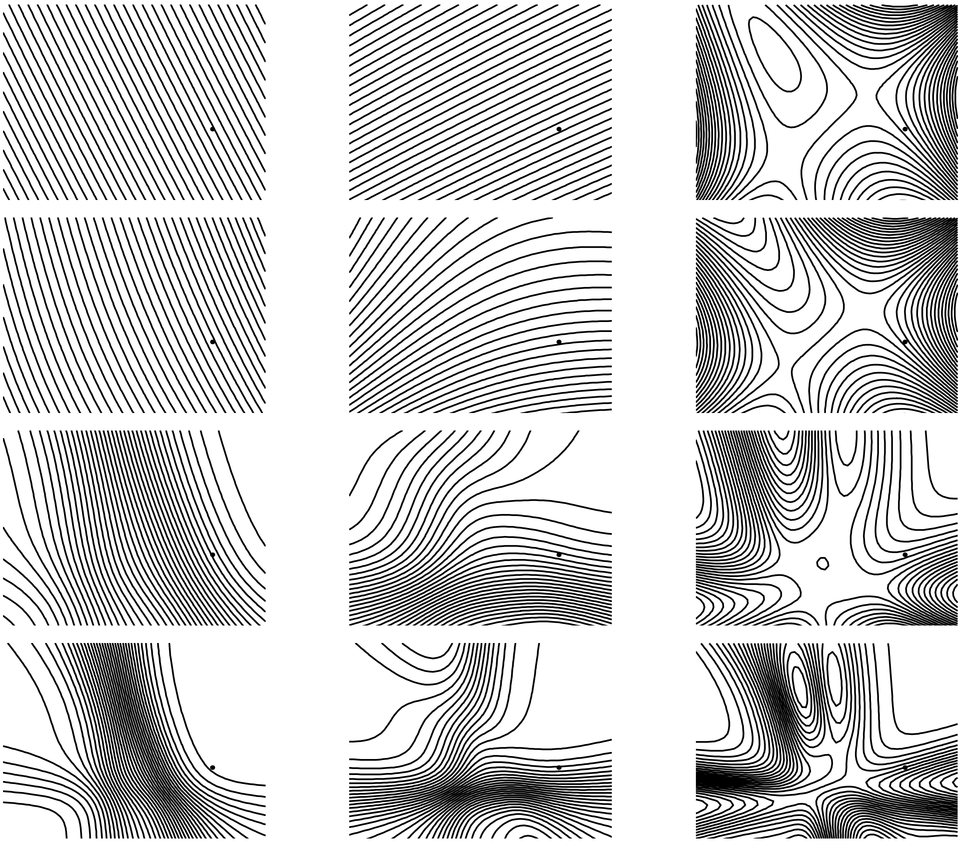

Fig. 1: Kernel PCA toy example (from [14]); three clusters (Gaussians with standard deviation 0.1, depicted region: $[-1,1]\times[-0.5,1]$). A smooth transition from linear PCA to nonlinear PCA is obtained by using hyperbolic tangent kernels $k(\mathbf{x},\mathbf{y}) = \tanh(\kappa(\mathbf{x}\cdot\mathbf{y})+1)$ with varying gain κ: from top to bottom, $\kappa = 0.1, 1, 5, 10$. For $\kappa = 0.1$, the first two features look like linear PCA features. For large κ, the nonlinear region of the tanh function becomes effective. In that case, kernel PCA can exploit this nonlinearity to allocate the highest feature gradients to regions where there are data points, as can be seen nicely in the case $\kappa = 10$.

covariance matrix

$$C := \frac{1}{\ell}\sum_{j=1}^{\ell}\left(\frac{\partial}{\partial t}\Big|_{t=0}\mathcal{L}_t\mathbf{x}_j\right)\left(\frac{\partial}{\partial t}\Big|_{t=0}\mathcal{L}_t\mathbf{x}_j\right)^{\top}. \quad (42)$$

To interpret this, note that C is a sample estimate of the covariance matrix of the random vector $s\cdot\frac{\partial}{\partial t}|_{t=0}\mathcal{L}_t\mathbf{x}$, $s\in\{\pm 1\}$ being a random sign. Using B, a given pattern $\mathbf{x}$ is thus first transformed by projecting it onto the Eigenvectors of C. The resulting feature vector is then rescaled by dividing by the square roots of C's Eigenvalues. In other words, the directions of main transformation variance are scaled back. So far, these ideas have only been tested in the linear case. For nonlinear kernels, an analysis similar to the one for kernel PCA yields a tangent covariance matrix C in F.

5. Conclusion

We believe that Support Vector machines and Kernel Principal Component Analysis are only the first examples of a series of potential applications of Mercer-kernel-based methods in learning theory. *Any* algorithm which can be formulated solely in terms of dot products can be made nonlinear by carrying it out in feature spaces induced by Mercer kernels. However, already the above two fields are large enough to render an exhaustive discussion in this article infeasible. To illustrate how nonlinear feature spaces can beneficially be used in complex learning tasks, we have summarized some aspects of SV learning and Kernel PCA, and we hope that the reader may find it worthwhile to consider employing kernel methods in their own work.

Acknowledgments. This work was supported by a grant of the DFG JA 379/71 and by the Australian Research Council. More information on SV methods can be obtained via `http://svm.first.gmd.de`.

References

[1] 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.

[2] 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.

[3] 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.

[4] C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Knowledge Discovery and Data Mining*, 1998. Submitted: draft available at http://svm.research.bell-labs.com/SVMdoc.html.

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

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

[7] S. Geva, J. Sitte, and G. Willshire. A one neuron truck backer–upper. In *International Joint Conference on Neural Networks*, pages 850–856, Baltimore, ML, June 1992. IEEE.

[8] F. Girosi. An equivalence between sparse approximation and support vector machines. A.I. Memo No. 1606, MIT, 1997.

[9] F. Girosi, M. Jones, and T. Poggio. Priors, stabilizers and basis functions: From regularization to radial, tensor and additive splines. A.I. Memo No. 1430, MIT, 1993.

[10] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. Technical Report 23, LS VIII, University of Dortmund, 1997.

[11] K.-R. Müller, A. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Predicting time series with support vector machines. In W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud, editors, *ICANN'97*, pages 999 – 1004, Berlin, 1997. Springer Lecture Notes in Computer Science, Vol. 1327.

[12] R. E. Schapire, Y. Freund, P. Bartlett, and W.S. Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. In *Machine Learning: Proceedings of the Fourteenth International Conference*, 1997.

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

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

[15] 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*, Cambridge, MA, 1998. MIT Press. In press.

[16] B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10, 1998. In press.

[17] 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.

[18] J. Shawe-Taylor, P. Bartlett, R. Williamson, and M. Anthony. A framework for structural risk minimization. In *COLT*, 1996.

[19] 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*, Cambridge, MA, 1998. MIT Press. In press.

[20] A. Smola and B. Schölkopf. On a kernel-based method for pattern recognition, regression, approximation and operator inversion. *Algorithmica*, 1998. In press.

[21] A. Smola, B. Schölkopf, and K.-R. Müller. The connection between regularization operators and support vector kernels. *Neural Networks*, 1998. In press.

[22] A. Smola, B. Schölkopf, and K.-R. Müller. General cost functions for support vector regression. In *Proc. of the Ninth Australian Conf. on Neural Networks*, 1998. In press.

[23] V. Vapnik. *Estimation of Dependences Based on Empirical Data [in Russian]*. Nauka, Moscow, 1979. (English translation: Springer Verlag, New York, 1982).

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

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

[26] G. Wahba. Convergence rates of certain approximate solutions to Fredholm integral equations of the first kind. *Journal of Approximation Theory*, 7:167 – 185, 1973.

[27] G. Wahba. Support vector machines, reproducing kernel hilbert spaces and the randomized GACV. Technical Report 984, Department of Statistics, University of Wisconsin, Madison, 1997. NIPS 97 Workshop on Support Vector Machines.