An Introduction to Kernel-Based Learning Algorithms

Klaus-Robert Müller, Sebastian Mika, Gunnar Rätsch, Koji Tsuda, Bernhard Schölkopf

Abstract— **This review provides an introduction to Support Vector Machines, Kernel Fisher Discriminant analysis and Kernel PCA, as examples for successful kernel based learning methods. We first give a short background about VC theory and kernel feature spaces and then proceed to kernel based learning in supervised and unsupervised scenarios including practical and algorithmic considerations. We illustrate the usefulness of kernel algorithms by finally discussing applications such as OCR and DNA analysis.**

Keywords— **Kernel methods, Support Vector Machines, Fisher's discriminant, Mathematical Programming Machines, PCA, Kernel PCA, single-class classification, Boosting, Mercer Kernels.**

I. Introduction

IN the last years, a number of powerful kernel-based learning machines, e.g. Support Vector Machines (SVMs) [1], [2], [3], [4], [5], [6], Kernel Fisher Discriminant (KFD) [7], [8], [9], [10] and Kernel Principal Component Analysis (KPCA) [11], [12], [13], have been proposed. These approaches have shown practical relevance not only for classification and regression problems but also, more recently, in unsupervised learning [11], [12], [13], [14], [15]. Successful applications of kernel based algorithms have been reported for various fields, for instance in the context of optical pattern and object recognition [16], [17], [18], [19], [20], text categorization [21], [22], [23], time-series prediction [24], [25], [15], gene expression profile analysis [26], [27], DNA and protein analysis [28], [29], [30] and many more.[1]

The present review introduces the main ideas of kernel algorithms, and reports applications from OCR (optical character recognition) and DNA analysis. We do not attempt a full treatment of all available literature, rather, we present a somewhat biased point of view illustrating the main ideas by drawing mainly from the work of the authors and providing – to the best of our knowledge – reference to related work for further reading. We hope that it nevertheless will be useful for the reader. It differs from other reviews, such as the ones of [3], [32], [6], [33], [34], mainly in the choice of the presented material: we place more emphasis on kernel PCA, kernel Fisher discriminants, and on connections to Boosting.

We start by presenting some basic concepts of learning theory in Section II. Then we introduce the idea of *kernel feature spaces* (Section III) and the original *SVM* approach, its implementation and some variants. Subsequently we discuss other *kernel-based methods* for supervised and unsupervised learning in Sections IV and V. Some attention will be devoted to questions of *model selection* (Section VI), i.e. how to properly choose the parameters in SVMs and other kernel-based approaches. Finally, we describe several recent and interesting applications in Section VII and conclude.

TABLE I
Notation conventions used in this paper

i, n	counter and number of patterns
$\mathbf{X}, N$	the input space, $N = \dim(\mathbf{X})$
$\mathbf{x}, y$	a training pattern and the label
$(\mathbf{x} \cdot \mathbf{x}')$	scalar product between $\mathbf{x}$ and $\mathbf{x}'$
$\mathcal{F}$	feature space
Φ	the mapping $\Phi : \mathbf{X} \to \mathcal{F}$
$\mathrm{k}(\cdot, \cdot)$	scalar product in feature space $\mathcal{F}$
F_i	a function class
h	the VC dimension of a function class
d	the degree of a polynomial
$\mathbf{w}$	normal vector of a hyperplane
α_i	Lagrange multiplier/Expansion coefficient for $\mathbf{w}$
ξ_i	the "slack-variable" for pattern $\mathbf{x}_i$
ν	the quantile parameter (determines the number of outliers)
$\|\cdot\|_p$	the ℓ_p–norm, $p \in [1, \infty]$
$\|S\|$	number of elements in a set S
Θ	The Heaviside function: $\Theta(z) = 0$ for $z < 0$, $\Theta(z) = 1$ otherwise
$\mathbb{R}_+$	space of non-negative real numbers

II. Learning to classify – some theoretical background

Let us start with a general notion of the learning problems that we consider in this paper. The task of classification is to find a rule, which, based on external observations, assigns an object to one of several classes. In the simplest case there are only two different classes. One

K.–R. Müller, S. Mika, G. Rätsch, and K. Tsuda are with GMD FIRST, Kekuléstr. 7, 12489 Berlin, Germany, EMail {klaus, mika, raetsch, tsuda}@first.gmd.de; K.–R. Müller is also with University of Potsdam, Neues Palais 10, 14469 Potsdam, Germany; K. Tsuda is also with Electrotechnical Laboratory, 1-1-4, Umezono, Tsukuba, 305-0031, Japan; B. Schölkopf is with Barnhill Technologies, 6709 Waters Av., Savannah, Georgia 31406, USA, EMail bsc@scientist.com.
We thank A. Smola, A. Zien and S. Sonnenburg for valuable discussions. Moreover, we gratefully acknowledge partial support from DFG (JA 379/9-1, MU 987/1-1), EU (IST-1999-14190 – BLISS) and travel grants from DAAD, NSF and EU (Neurocolt II). SM thanks for warm hospitality during his stay at Microsoft Research in Cambridge. Furthermore, GR would like to thank UC Santa Cruz and CRIEPI for warm hospitality. Thanks also to the reviewers for giving valuable comments that improved this paper.

[1] See also Isabelle Guyon's web page `http://www.clopinet.com/isabelle/Projects/SVM/applist.html` on applications of SVMs.

possible formalization of this task is to estimate a function $f : \mathbb{R}^N \rightarrow \{-1, +1\}$, using input-output training data pairs generated i.i.d. according to an unknown probability distribution $P(\mathbf{x}, y)$

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n) \in \mathbb{R}^N \times Y, \quad Y = \{-1, +1\}$$

such that f will correctly classify unseen examples $(\mathbf{x}, y)$. An example is assigned to the class $+1$ if $f(\mathbf{x}) \geq 0$ and to the class -1 otherwise. The test examples are assumed to be generated from the same probability distribution $P(\mathbf{x}, y)$ as the training data. The best function f that one can obtain is the one minimizing the expected error (risk)

$$R[f] = \int l(f(\mathbf{x}), y) dP(\mathbf{x}, y), \tag{1}$$

where l denotes a suitably chosen loss function, e.g. $l(f(\mathbf{x}), y) = \Theta(-yf(\mathbf{x}))$, where $\Theta(z) = 0$ for $z < 0$ and $\Theta(z) = 1$ otherwise (the so-called 0/1-loss). The same framework can be applied for regression problems, where $y \in \mathbb{R}$. Here, the most common loss function is the *squared loss*: $l(f(\mathbf{x}), y) = (f(\mathbf{x}) - y)^2$; see [35], [36] for a discussion of other loss functions.

Unfortunately the risk cannot be minimized directly, since the underlying probability distribution $P(\mathbf{x}, y)$ is unknown. Therefore, we have to try to estimate a function that is *close* to the optimal one based on the available information, i.e. the training sample and properties of the function class F the solution f is chosen from. To this end, we need what is called an induction principle. A particular simple one consists in approximating the minimum of the risk (1) by the minimum of the *empirical risk*

$$R_{emp}[f] = \frac{1}{n} \sum_{i=1}^{n} l(f(\mathbf{x}_i), y_i). \tag{2}$$

It is possible to give conditions on the learning machine which ensure that asymptotically (as $n \rightarrow \infty$), the empirical risk will converge towards the expected risk. However, for small sample sizes large deviations are possible and *overfitting* might occur (see Figure 1). Then a small

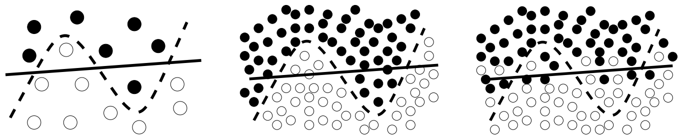


Fig. 1. Illustration of the overfitting dilemma: Given only a small sample (left) either, the solid or the dashed hypothesis might be true, the dashed one being more complex, but also having a smaller training error. Only with a large sample we are able to see which decision reflects the true distribution more closely. If the dashed hypothesis is correct the solid would underfit (middle); if the solid were correct the dashed hypothesis would overfit (right).

generalization error cannot be obtained by simply minimizing the training error (2). One way to avoid the overfitting dilemma is to *restrict* the complexity of the function class F that one chooses the function f from [3]. The intuition, which will be formalized in the following is that a "simple" (e.g. linear) function that explains most of the data is preferable to a complex one (Occam's razor). Typically one introduces a *regularization* term (e.g. [37], [38], [39], [40]) to limit the complexity of the function class F from which the learning machine can choose. This raises the problem of model selection (e.g. [41], [39], [42], [43]), i.e. how to find the optimal complexity of the function (cf. Section VI).

A specific way of controlling the complexity of a function class is given by VC theory and the structural risk minimization (SRM) principle [44], [3], [5]. Here the concept of complexity is captured by the Vapnik-Chervonenkis (VC) dimension h of the function class F that the estimate f is chosen from. Roughly speaking, the VC dimension measures how many (training) points can be shattered (i.e. separated) for all possible labelings using functions of the class. Constructing a nested family of function classes $F_1 \subset \cdots \subset F_k$ with non-decreasing VC dimension the SRM principle proceeds as follows: Let $f_1, \ldots, f_k$ be the solutions of the empirical risk minimization (2) in the function classes F_i. SRM chooses the function class F_i (and the function f_i) such that an upper bound on the generalization error is minimized which can be computed making use of theorems such as the following one (see also Figure 2):

Theorem 1 ([3], [5]) Let h denote the VC dimension of the function class F and let R_{emp} be defined by (2) using the 0/1-loss. For all $\delta > 0$ and $f \in F$ the inequality bounding the risk

$$R[f] \leq R_{emp}[f] + \sqrt{\frac{h\left(\ln \frac{2n}{h} + 1\right) - \ln(\delta/4)}{n}} \tag{3}$$

holds with probability of at least $1 - \delta$ for $n > h$.

Note, this bound is only an example and similar formulations are available for other loss functions [5] and other complexity measures, e.g. entropy numbers [45]. Let us discuss Inequality (3): the goal is to minimize the generalization error $R[f]$, which can be achieved by obtaining a small training error $R_{emp}[f]$ while keeping the function class as small as possible. Two extremes arise for (3): (i) a very small function class (like F_1) yields a vanishing square root term, but a large training error might remain, while (ii) a huge function class (like F_k) may give a vanishing empirical error but a large square root term. The best class is usually in between (cf. Figure 2), as one would like to obtain a function that explains the data quite well *and* to have a small risk in obtaining that function. This is very much in analogy to the bias-variance dilemma scenario described for neural networks (see e.g. [46]).

A. VC-dimension in practice

Unfortunately in practice the bound on the expected error in (3) is often neither easily computable nor very helpful. Typical problems are that the upper bound on the expected test error might be trivial (i.e. larger than one), the VC-dimension of the function class is unknown or it is infinite (in which case one would need an infinite amount of training data). Although there are different, usually tighter bounds, most of them suffer from similar problems. Nevertheless, bounds clearly offer helpful theoretical insights into the nature of learning problems.

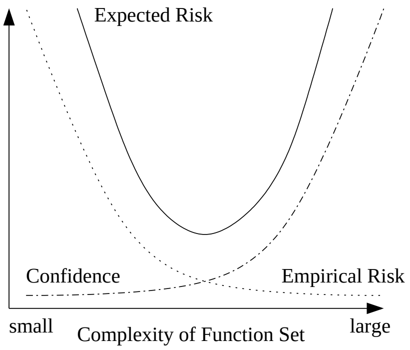


Fig. 2. Schematic illustration of Eq. (3). The dotted line represents the training error (empirical risk), the dash-dotted line the upper bound on the complexity term (confidence). With higher complexity the empirical error decreases but the upper bound on the risk confidence becomes worse. For a certain complexity of the function class the best expected risk (solid line) is obtained. Thus, in practice the goal is to find the best tradeoff between empirical error and complexity.

B. Margins and VC-dimension

Let us for a moment assume that the training sample is separable by a hyperplane (see Figure 3), i.e. we choose functions of the form

$$f(\mathbf{x}) = (\mathbf{w} \cdot \mathbf{x}) + b. \tag{4}$$

It was shown (e.g. [44], [3]) that for the class of hyperplanes the VC-dimension itself can be bounded in terms of another quantity, the *margin* (also Figure 3). The margin is defined as the minimal distance of a sample to the decision surface. The margin in turn can be measured by the length of the weight vector $\mathbf{w}$ in (4): as we assumed that the training sample is separable we can rescale $\mathbf{w}$ and b such that the points closest to the hyperplane satisfy $|(\mathbf{w} \cdot \mathbf{x}_i) + b| = 1$ (i.e. obtain the so-called canonical representation of the hyperplane). Now consider two samples $\mathbf{x}_1$ and $\mathbf{x}_2$ from different classes with $(\mathbf{w} \cdot \mathbf{x}_1) + b = 1$ and $(\mathbf{w} \cdot \mathbf{x}_2) + b = -1$, respectively. Then the margin is given by the distance of these two points, measured perpendicular to the hyperplane, i.e. $\left(\frac{\mathbf{w}}{\|\mathbf{w}\|} \cdot (\mathbf{x}_1 - \mathbf{x}_2)\right) = \frac{2}{\|\mathbf{w}\|}$. The result linking the VC-dimension of the class of separating hyperplanes to the margin or the length of the weight vector $\mathbf{w}$ respectively is given by the following inequalities:

$$h \leq \Lambda^2 R^2 + 1 \quad \text{and} \quad \|\mathbf{w}\|_2 \leq \Lambda \tag{5}$$

where R is the radius of the smallest ball around the data (e.g. [3]). Thus, if we bound the margin of a function class from below, say by $\frac{2}{\Lambda}$, we can control its VC-dimension[2]. Support Vector Machines, which we shall treat more closely in Section IV-A, implement this insight. The choice of linear functions seems to be very limiting (i.e. instead of being likely to overfit we are now more likely to underfit). Fortunately there is a way to have both, linear models *and*

[2]There are some ramifications to this statement, that go beyond the scope of this work. Strictly speaking, VC theory requires the structure to be defined a priori, which has implications for the definition of the class of separating hyperplanes, cf. [47].

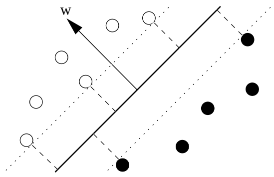


Fig. 3. Linear classifier and margins: A linear classifier is defined by a hyperplane's normal vector $\mathbf{w}$ and an offset b, i.e. the decision boundary is $\{\mathbf{x} | (\mathbf{w} \cdot \mathbf{x}) + b = 0\}$ (thick line). Each of the two halfspaces defined by this hyperplane corresponds to one class, i.e. $f(\mathbf{x}) = \mathrm{sign}((\mathbf{w} \cdot \mathbf{x}) + b)$. The margin of a linear classifier is the minimal distance of any training point to the hyperplane. In this case it is the distance between the dotted lines and the thick line.

a very rich set of nonlinear decision functions, by using the tools that will be discussed in the next section.

III. Nonlinear Algorithms in Kernel Feature Spaces

Algorithms in feature spaces make use of the following idea: via a nonlinear mapping

$$\begin{aligned} \Phi : \mathbb{R}^N &\rightarrow \mathcal{F} \\ \mathbf{x} &\mapsto \Phi(\mathbf{x}) \end{aligned}$$

the data $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^N$ is mapped into a potentially much higher dimensional feature space $\mathcal{F}$. For a given learning problem one now considers the same algorithm in $\mathcal{F}$ instead of $\mathbb{R}^N$, i.e. one works with the sample

$$(\Phi(\mathbf{x}_1), y_1), \ldots, (\Phi(\mathbf{x}_n), y_n) \in \mathcal{F} \times Y.$$

Given this mapped representation a *simple* classification

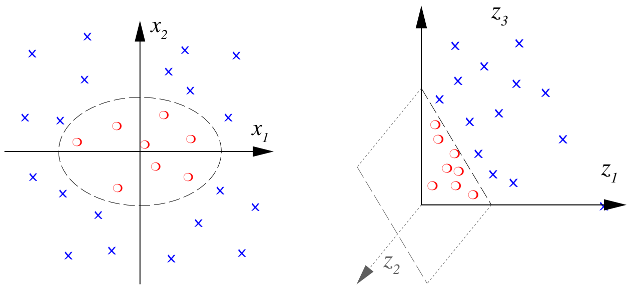


Fig. 4. Two dimensional classification example. Using the second order monomials $x_1^2, \sqrt{2}x_1x_2$ and x_2^2 as features a separation in feature space can be found using a *linear* hyperplane (right). In input space this construction corresponds to a *non-linear* ellipsoidal decision boundary (left) (figure from [48]).

or regression in $\mathcal{F}$ is to be found. This is also implicitly done for (one hidden layer) neural networks, radial basis networks (e.g. [49], [50], [51], [52]) or Boosting algorithms [53] where the input data is mapped to some representation given by the hidden layer, the RBF bumps or the hypotheses space respectively.

The so-called *curse of dimensionality* from statistics says essentially that the difficulty of an estimation problem increases drastically with the dimension N of the space, since – in principle – as a function of N one needs exponentially many patterns to sample the space properly. This well known statement induces some doubts about whether it is a good idea to go to a high dimensional feature space for learning.

However, statistical learning theory tells us that the contrary can be true: learning in $\mathcal{F}$ can be simpler if one uses a low complexity, i.e. *simple* class of decision rules (e.g. linear classifiers). All the variability and richness that one needs to have a powerful function class is then introduced by the mapping Φ. In short: not the dimensionality but the complexity of the function class matters [3]. Intuitively, this idea can be understood from the toy example in Figure 4: in two dimensions a rather complicated *nonlinear* decision surface is necessary to separate the classes, whereas in a feature space of second order monomials (see e.g. [54])

$$\begin{array}{rcl}\Phi : \mathbb{R}^2 & \to & \mathbb{R}^3 \\ (x_1, x_2) & \mapsto & (z_1, z_2, z_3) := (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)\end{array} \quad (6)$$

all one needs for separation is a *linear* hyperplane. In this simple toy example, we can easily control both: the statistical complexity (by using a simple linear hyperplane classifier) and the algorithmic complexity of the learning machine, as the feature space is only three dimensional. However, it becomes rather tricky to control the latter for large real world problems. For instance, consider images of 16×16 pixels as patterns and 5th order monomials as mapping Φ – then one would map to a space that contains all 5^{th} order products of 256 pixels, i.e. to a $\binom{5+256-1}{5} \approx 10^{10}$-dimensional space. So, even if one could control the statistical complexity of this function class, one would still run into intractability problems while executing an algorithm in this space.

Fortunately, for certain feature spaces $\mathcal{F}$ and corresponding mappings Φ there is a highly effective trick for computing scalar products in feature spaces using *kernel functions* [55], [56], [1], [3]. Let us come back to the example from Eq. (6). Here, the computation of a scalar product between two feature space vectors, can be readily reformulated in terms of a kernel function k

$$\begin{array}{rcl}(\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})) & = & (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)(y_1^2, \sqrt{2}\, y_1 y_2, y_2^2)^\top \\ & = & ((x_1, x_2)(y_1, y_2)^\top)^2 \\ & = & (\mathbf{x} \cdot \mathbf{y})^2 \\ & =: & \mathrm{k}(\mathbf{x}, \mathbf{y}).\end{array}$$

This finding generalizes:

- For $\mathbf{x}, \mathbf{y} \in \mathbb{R}^N$, and $d \in \mathbb{N}$ the kernel function

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

computes a scalar product in the space of all products of d vector entries (monomials) of $\mathbf{x}$ and $\mathbf{y}$ [3], [11].

- If $k : \mathcal{C} \times \mathcal{C} \to \mathbb{R}$ is a continuous kernel of a positive integral operator on a Hilbert space $L_2(\mathcal{C})$ on a compact set $\mathcal{C} \subset \mathbb{R}^N$, i.e.

$$\forall f \in L_2(\mathcal{C}) : \quad \int_{\mathcal{C}} \mathrm{k}(\mathbf{x}, \mathbf{y}) f(\mathbf{x}) f(\mathbf{y})\, d\mathbf{x}\, d\mathbf{y} \geq 0,$$

then there exists a space $\mathcal{F}$ and a mapping $\Phi : \mathbb{R}^N \to \mathcal{F}$ such that $\mathrm{k}(\mathbf{x}, \mathbf{y}) = (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}))$ [3]. This can be seen directly from Mercers Theorem [59] saying that any kernel of a positive integral operator can be expanded in its Eigenfunctions ψ_j ($\lambda_j > 0$, $N_{\mathcal{F}} \leq \infty$):

$$\mathrm{k}(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{N_{\mathcal{F}}} \lambda_j \psi_j(\mathbf{x}) \psi_j(\mathbf{y}).$$

In this case

$$\Phi(\mathbf{x}) = (\sqrt{\lambda_1}\psi_1(\mathbf{x}), \sqrt{\lambda_2}\psi_2(\mathbf{x}), \dots)$$

is a possible realization.

- Note furthermore that using a particular SV kernel corresponds to an *implicit* choice of a regularization operator (cf. [39], [57]). For translation invariant kernels, the regularization properties can be expressed conveniently in Fourier space in terms of the frequencies [58], [60]. For example Gaussian kernels (7) correspond to a general smoothness assumption in all k-th order derivatives [58]. Vice versa using this correspondence, kernels matching a certain prior about the frequency content of the data can be constructed that reflect our prior problem knowledge.

Table II lists some of the most widely used kernel functions. More sophisticated kernels (e.g. kernels generating splines or Fourier expansions) can be found in [4], [36], [5], [58], [30], [28], [61].

A. Wrapping up

The interesting point about kernel functions is that the scalar product can be *implicitly* computed in $\mathcal{F}$, *without* explicitly using or even knowing the mapping Φ. So, kernels allow to compute scalar products in spaces, where one could otherwise hardly perform any computations. A direct consequence from this finding is [11]: *every (linear)*

TABLE II

Common kernel functions: Gaussian RBF ($c \in \mathbb{R}$), polynomial ($d \in \mathbb{N}, \theta \in \mathbb{R}$), sigmoidal ($\kappa, \theta \in \mathbb{R}$) and inverse multiquadric ($c \in \mathbb{R}_+$) kernel functions are among the most common ones. While RBF and polynomial are known to fulfill Mercers condition, this is not strictly the case for sigmoidal kernels [33]. Further valid kernels proposed in the context of regularization networks are e.g. multiquadric or spline kernels [39], [57], [58].

Gaussian RBF	$k(\mathbf{x}, \mathbf{y}) = \exp\left(\frac{-\lVert\mathbf{x} - \mathbf{y}\rVert^2}{c}\right)$	(7)
Polynomial	$((\mathbf{x} \cdot \mathbf{y}) + \theta)^d$	
Sigmoidal	$\tanh(\kappa(\mathbf{x} \cdot \mathbf{y}) + \theta)$	
inv. multiquadric	$\frac{1}{\sqrt{\lVert\mathbf{x} - \mathbf{y}\rVert^2 + c^2}}$	

algorithm that only uses scalar products can implicitly be executed in $\mathcal{F}$ by using kernels, i.e. one can very elegantly construct a nonlinear version of a linear algorithm.[3]

In the following sections we follow this philosophy for supervised and unsupervised learning: by (re-) formulating linear, scalar product based algorithms that are *simple* in feature space, one is able to generate powerful nonlinear algorithms, which use rich function classes in input space.

IV. Supervised Learning

We will now briefly outline the algorithms of SVMs and the Kernel Fisher Discriminant (KFD). Furthermore we discuss the Boosting algorithm from the kernel feature space point of view and show a connection to SVMs. Finally, we will point out some extensions of these algorithms proposed recently.

A. Support Vector Machines

Let us recall from Section II that the VC dimension of a linear system, e.g. separating hyperplanes (as computed by a perceptron)

$$y = \operatorname{sign}\left((\mathbf{w}\cdot\mathbf{x}) + b\right)$$

can be upper bounded in terms of the margin (cf. (5)). For separating hyperplane classifiers the conditions for classification without training error are

$$y_i((\mathbf{w}\cdot\mathbf{x}_i) + b) \geq 1, \quad i = 1,\ldots,n.$$

As linear function classes are often not rich enough in practice, we will follow the line of thought of the last section and consider linear classifiers in feature space using dot products. To this end, we substitute $\Phi(\mathbf{x}_i)$ for each training example $\mathbf{x}_i$, i.e. $y = \operatorname{sign}\left((\mathbf{w}\cdot\Phi(\mathbf{x})) + b\right)$. In feature space, the conditions for perfect classification are described as

$$y_i((\mathbf{w}\cdot\Phi(\mathbf{x}_i)) + b) \geq 1, \quad i = 1,\ldots,n. \tag{8}$$

The goal of learning is to find $\mathbf{w} \in \mathcal{F}$ and b such that the expected risk is minimized. However, since we cannot obtain the expected risk itself, we will minimize the bound (3), which consists of the empirical risk and the complexity term. One strategy is to keep the empirical risk zero by constraining $\mathbf{w}$ and b to the perfect separation case, while minimizing the complexity term, which is a monotonically increasing function of the VC dimension h. For a linear classifier in feature space the VC dimension h is bounded according to $h \leq \|\mathbf{w}\|^2 R^2 + 1$ (cf. (5)), where R is the radius of the smallest ball around the training data (e.g. [3]), which is fixed for a given data set. Thus, we can minimize the complexity term by minimizing $\|\mathbf{w}\|^2$. This can be formulated as a quadratic optimization problem

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2 \tag{9}$$

subject to (8). However, if the only possibility to access the feature space is via dot–products computed by the kernel, we can not solve (9) directly since $\mathbf{w}$ lies in that feature space. But it turns out that we can get rid of the explicit usage of $\mathbf{w}$ by forming the dual optimization problem. Introducing Lagrange multipliers $\alpha_i \geq 0$, $i = 1,\ldots,n$, one for each of the constraints in (8), we get the following Lagrangian:

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i (y_i((\mathbf{w}\cdot\Phi(\mathbf{x}_i)) + b) - 1). \tag{10}$$

The task is to minimize (10) with respect to $\mathbf{w}, b$ and to maximize it with respect to α_i. At the optimal point, we have the following saddle point equations:

$$\frac{\partial L}{\partial b} = 0 \quad \text{and} \quad \frac{\partial L}{\partial \mathbf{w}} = 0,$$

which translate into

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \quad \text{and} \quad \mathbf{w} = \sum_{i=1}^{\ell} \alpha_i y_i \Phi(\mathbf{x}_i). \tag{11}$$

From the right equation of (11), we find that $\mathbf{w}$ is contained in the subspace spanned by the $\Phi(\mathbf{x}_i)$. By substituting (11) into (10) and by replacing $(\Phi(x_i)\cdot\Phi(x_j))$ with kernel functions $\mathrm{k}(\mathbf{x}_i, \mathbf{x}_j)$, we get the dual quadratic optimization problem:

$$\begin{array}{ll} \displaystyle\max_{\boldsymbol{\alpha}} & \displaystyle\sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \,\mathrm{k}(\mathbf{x}_i, \mathbf{x}_j) \\ \text{subject to} & \alpha_i \geq 0, \; i = 1,\ldots,n, \\ & \sum_{i=1}^{n} \alpha_i y_i = 0. \end{array}$$

Thus, by solving the dual optimization problem, one obtains the coefficients α_i, $i = 1,\ldots,n$, which one needs to express the $\mathbf{w}$ which solves (9). This leads to the nonlinear decision function

$$\begin{aligned} f(\mathbf{x}) &= \operatorname{sgn}\left(\sum_{i=1}^{n} y_i \alpha_i (\Phi(\mathbf{x})\cdot\Phi(\mathbf{x}_i)) + b\right) \\ &= \operatorname{sgn}\left(\sum_{i=1}^{n} y_i \alpha_i \,\mathrm{k}(\mathbf{x}, \mathbf{x}_i) + b\right). \end{aligned}$$

Note that we have up to now only considered the separable case. This corresponds to an empirical error of zero (cf. Theorem 1). However for noisy data, this might not be the minimum in the expected risk (cf. (3)) and we might face overfitting effects (cf. Fig. 1). Therefore a "good" trade-off between the empirical risk and the complexity term in (3) needs to be found. Using a technique which was first proposed in [62] and later used for SVMs in [2], one introduces slack-variables to relax the hard-margin constraints:

$$y_i((\mathbf{w}\cdot\Phi(\mathbf{x}_i)) + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad i = 1,\ldots,n, \tag{12}$$

[3] Even algorithms that operate on similarity measures k generating positive matrices $\mathrm{k}(\mathbf{x}_i, \mathbf{x}_i)_{ij}$ can be interpreted as linear algorithms in some feature space $\mathcal{F}$ [4].

additionally allowing for some classification errors. The SVM solution can then be found by (a) keeping the upper bound on the VC dimension small and (b) by minimizing an upper bound $\sum_{i=1}^{n} \xi_i$ on the empirical risk,[4] i.e. the number of training errors. Thus, one minimizes

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i.$$

where the regularization constant $C > 0$ determines the trade-off between the empirical error and the complexity term. This leads to the dual problem:

$$\max_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \, \mathrm{k}(\mathbf{x}_i, \mathbf{x}_j) \tag{13}$$

$$\text{subject to} \quad 0 \le \alpha_i \le C, \; i = 1, \ldots, n, \tag{14}$$

$$\textstyle\sum_{i=1}^{n} \alpha_i y_i = 0. \tag{15}$$

From introducing the slack-variables ξ_i, one gets the *box* constraints that limit the size of the Lagrange multipliers: $\alpha_i \le C$, $i = 1, \ldots, n$.

A.1 Sparsity

Most optimization methods are based on the second order optimality conditions, so called Karush-Kuhn-Tucker conditions which state necessary and in some cases sufficient conditions for a set of variables to be optimal for an optimization problem. It comes handy that these conditions are particularly simple for the dual SVM problem (13) [64]:

$$\begin{array}{rcll} \alpha_i = 0 & \Rightarrow & y_i f(\mathbf{x}_i) \ge 1 & \text{and} \quad \xi_i = 0 \\ 0 < \alpha_i < C & \Rightarrow & y_i f(\mathbf{x}_i) = 1 & \text{and} \quad \xi_i = 0 \\ \alpha_i = C & \Rightarrow & y_i f(\mathbf{x}_i) \le 1 & \text{and} \quad \xi_i \ge 0 \end{array} \tag{16}$$

They reveal one of the most important property of SVMs: the solution is sparse in $\boldsymbol{\alpha}$, i.e. many patterns are outside the margin area and the optimal α_i's are zero. Specifically, the KKT conditions show that only such α_i connected to a training pattern $\mathbf{x}_i$, which is either on the margin (i.e. $0 < \alpha_i < C$ and $y_i f(\mathbf{x}_i) = 1$) or inside the margin area (i.e. $\alpha_i = C$ and $y_i f(\mathbf{x}_i) < 1$) are non-zero. Without this sparsity property, SVM learning would hardly be practical for large data sets.

A.2 ν-SVMs

Several modifications have been proposed to the basic SVM algorithm. One particular useful modification are ν-SVMs [65], originally proposed for regression. In the case of pattern recognition, they replace the rather un-intuitive regularization constant C with another constant $\nu \in (0, 1]$ and yield, for appropriate parameter choices, identical solutions. Instead of (13) one solves

$$\begin{array}{ll} \max\limits_{\boldsymbol{\alpha}} & -\dfrac{1}{2}\displaystyle\sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \, \mathrm{k}(\mathbf{x}_i, \mathbf{x}_j) \\ \text{subject to} & 0 \le \alpha_i \le 1/n, \; i = 1, \ldots, n, \\ & \sum_{i=1}^{n} \alpha_i y_i = 0, \\ & \sum_i \alpha_i \ge \nu. \end{array}$$

The advantage is that this new parameter ν has a clearer interpretation than simply "the smaller, the smoother": under some mild assumptions (data i.i.d. from continuous probability distribution [65]) it is asymptotically (i) an upper bound on the number of margin errors[5] and (ii) a lower bound on the number of support vectors.

A.3 Computing the Threshold

The threshold b can be computed by exploiting the fact that for all SVs $\mathbf{x}_i$ with $0 < \alpha_i < C$, the slack variable ξ_i is zero. This follows from the Karush-Kuhn-Tucker (KKT) conditions (cf. (16)). Thus, for any support vector $\mathbf{x}_i$ with $i \in I := \{i : 0 < \alpha_i < C\}$ holds:

$$y_i \left(b + \sum_{j=1}^{n} y_j \alpha_j \, \mathrm{k}(\mathbf{x}_i, \mathbf{x}_j)) \right) = 1.$$

Averaging over these patterns yields a numerically stable solution:

$$b = \frac{1}{|I|} \sum_{i \in I} \left(y_i - \sum_{j=1}^{n} y_j \alpha_j \, \mathrm{k}(\mathbf{x}_i, \mathbf{x}_j) \right).$$

A.4 A geometrical explanation

Here, we will present an illustration of the SVM solution to enhance intuitive understandings. Let us normalize the weight vector to 1 (i.e. $\|\mathbf{w}\|_2 = 1$) and fix the threshold $b = 0$. Then, the set of all $\mathbf{w}$ which separate the training samples is completely described as

$$\mathcal{V} = \{\mathbf{w} | y_i f(\mathbf{x}_i) > 0; i = 1, \cdots, n, \|\mathbf{w}\|_2 = 1\}$$

The set $\mathcal{V}$ is called "version space" [66]. It can be shown that the SVM solution coincides with the Tchebycheff-center of the version space, which is the center of the largest sphere contained in $\mathcal{V}$ (cf. [67]). However, the theoretical optimal point in version space yielding a Bayes-optimal decision boundary is the Bayes point, which is known to be closely approximated by the center of mass [68], [69]. The version space is illustrated as a region on the sphere as shown in Figures 5 and 6. If the version space is shaped as in Figure 5, the SVM solution is near to the optimal point. However, if it has an elongated shape as in Figure 6, the SVM solution is far from the optimal one. To cope with this problem, several researchers [70], [68], [71] proposed a billiard sampling method for approximating the Bayes point. This method can achieve improved results, as shown on several benchmarks in comparison to SVMs.

[4] Other bounds on the empirical error, like $\sum_{i=1}^{n} \xi_i^2$ are also frequently used (e.g. [2], [63]).

[5] A margin error is a point $\mathbf{x}_i$ which is either being misclassified or lying inside the margin area.

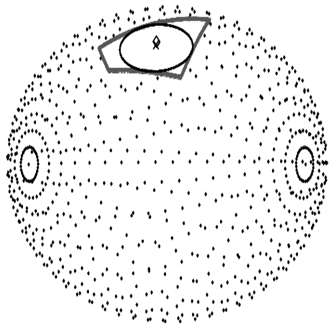


Fig. 5. An example of the version space where the SVM works fine. The center of mass ($\diamond$) is close to the SVM solution ($\times$). Figure taken from [72].

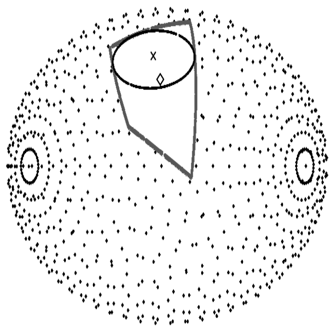


Fig. 6. An example of the version space where SVM works poorly. The version space has an elongated shape and the center of mass ($\diamond$) is far from the SVM solution ($\times$). Figure taken from [72].

A.5 Optimization Techniques for SVMs

To solve the SVM problem one has to solve the (convex) quadratic programming (QP) problem (13) under the constraints (14) and (15) (Eq. (13) can be rewritten as maximizing $-\frac{1}{2}\boldsymbol{\alpha}^\top \hat{K}\boldsymbol{\alpha}+\mathbf{1}^\top\boldsymbol{\alpha}$ where $\hat{K}$ is the positive semidefinite matrix $\hat{K}_{ij} = y_i y_j \,\mathrm{k}(\mathbf{x}_i, \mathbf{x}_j)$ and $\mathbf{1}$ the vector of all ones). As the objective function is convex every (local) maximum is already a global maximum. However, there can be several optimal solutions (in terms of the variables α_i) which might lead to different testing performances.

There exists a huge body of literature on solving quadratic programs and several free or commercial software packages (see e.g. [73], [74], [33] and references therein). However, the problem is that most mathematical programming approaches are either only suitable for small problems or assume that the quadratic term covered by $\hat{K}$ is very sparse, i.e. most elements of this matrix are zero. Unfortunately this is not true for the SVM problem and thus using standard codes with more than a few hundred variables results in enormous training times and more than demanding memory needs. Nevertheless, the structure of the SVM optimization problem allows to derive specially tailored algorithms which allow for fast convergence with small memory requirements even on large problems. Here we will briefly consider three different approaches. References, containing more details and tricks can be found e.g. in [6], [33].

A.5.a Chunking. A key observation in solving large scale SVM problems is the sparsity of the solution $\boldsymbol{\alpha}$. Depending on the problem, many of the optimal α_i will either be zero or on the upper bound C. If one knew beforehand which α_i were zero, the corresponding rows and columns could be removed from the matrix $\hat{K}$ without changing the value of the quadratic form. Further, a point $\boldsymbol{\alpha}$ can only be optimal for (13) if and only if it fulfills the KKT conditions (cf. (16)). In [64] a method called chunking is described, making use of the sparsity and the KKT conditions. At every step chunking solves the problem containing all non–zero α_i plus some of the α_i violating the KKT conditions. The size of this problem varies but is finally equal to the number of non-zero coefficients. While this technique is suitable for fairly large problems it is still limited by the maximal number of support vectors that one can handle and it still requires a quadratic optimizer to solve the sequence of smaller problems. A free implementation can be found e.g. in [75].

A.5.b Decomposition Methods. Those methods are similar in spirit to chunking as they solve a sequence of small QPs as well. But here the size of the subproblems is fixed. They are based on the observations of [76], [77] that a sequence of QPs which at least always contains one sample violating the KKT conditions will eventually converge to the optimal solution. It was suggested to keep the size of the subproblems fixed and to add and remove one sample in each iteration. This allows the training of arbitrary large data sets. In practice, however, the convergence of such an approach is very slow. Practical implementations use sophisticated heuristics to select several patterns to add and remove from the subproblem plus efficient caching methods. They usually achieve fast convergence even on large datasets with up to several thousands of support vectors. A good quality (free) implementation is SVM$_{light}$ [78]. A quadratic optimizer is still required and contained in the package. Alternatively, the package [75] also contains a decomposition variant.

A.5.c Sequential Minimal Optimization (SMO). This method proposed by [79] can be viewed as the most extreme case of decomposition methods. In each iteration it solves a quadratic problem of size two. This can be done analytically and thus no quadratic optimizer is required. Here the main problem is to chose a good pair of variables to optimize in each iteration. The original heuristics presented in [79] are based on the KKT conditions and there has been some work (e.g. [80]) to improve them. The implementation of the SMO approach is straight forward (pseudo code in [79]). While the original work was targeted at an SVM for classification, there are now also approaches which implement variants of SMO for SVM regression (e.g. [36], [33]) and single-class SVMs (cf. below, [14]).

A.5.d Other techniques. Further algorithms have been proposed to solve the SVM problem or a close approximation. For instance, the Kernel-Adatron [81] is derived from the Adatron algorithm by [82] proposed originally in a statistical mechanics setting. It constructs a large margin hyperplane using online learning. Its implementation is very simple. However, its drawback is that is does not allow for training errors, i.e. is only valid for separable data sets. In [83] a slightly more general approach for data mining problems is considered.

A.5.e Codes. A fairly large selection of optimization codes for SVM classification and regression may be found on the web at [84] together with the appropriate references. They range from simple MATLAB implementation to sophisticated C, C++ or FORTRAN programs. Note that most of these implementations are for non-commercial use only.

B. Kernel Fisher Discriminant

The idea of the Kernel Fisher Discriminant (KFD) (e.g. [7], [9], [10]) is to solve the problem of Fisher's linear discriminant [85], [86] in a kernel feature space $\mathcal{F}$, thereby yielding a nonlinear discriminant in the input space. In the

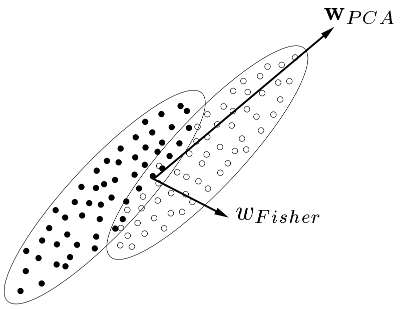


Fig. 7. Illustration of the projections of PCA and Fisher's Discriminant for a toy data set. It is clearly seen that PCA is purely descriptive, whereas the Fisher projection is discriminative.

linear case, Fisher's discriminant aims at finding a linear projections such that the classes are well separated (cf. Figure 7). Separability is measured by two quantities: How far are the projected means apart (should be large) and how big is the variance of the data in this direction (should be small). This can be achieved by maximizing the Rayleigh coefficient

$$J(\mathbf{w}) = \frac{\mathbf{w}^\top S_B \mathbf{w}}{\mathbf{w}^\top S_W \mathbf{w}}, \tag{17}$$

of between and within class variance with respect to $\mathbf{w}$, where

$$S_B = (\boldsymbol{m}_2 - \boldsymbol{m}_1)(\boldsymbol{m}_2 - \boldsymbol{m}_1)^\top$$

and

$$S_W = \sum_{k=1,2} \sum_{i \in \mathcal{I}_k} (\mathbf{x}_i - \boldsymbol{m}_k)(\mathbf{x}_i - \boldsymbol{m}_k)^\top.$$

Here $\boldsymbol{m}_k$ and $\mathcal{I}_k$ denote the sample mean and the index set for class k, respectively. Note, that under the assumption that the class distributions are (identically distributed) Gaussians, Fisher's discriminant is Bayes optimal; it can also be generalized to the multi–class case[6]. To formulate the problem in a kernel feature space $\mathcal{F}$ one can make use of a similar expansion as (11) in SVMs for $\mathbf{w} \in \mathcal{F}$, i.e. one can express $\mathbf{w}$ in terms of mapped training patterns [7]:

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i \Phi(\mathbf{x}_i). \tag{18}$$

Substituting $\Phi(\mathbf{x})$ for all $\mathbf{x}$ in (17) and plugging in (18), the optimization problem for the KFD in the feature space can then be written as [8]:

$$J(\boldsymbol{\alpha}) = \frac{(\boldsymbol{\alpha}^\top \boldsymbol{\mu})^2}{\boldsymbol{\alpha}^\top N \boldsymbol{\alpha}} = \frac{\boldsymbol{\alpha}^\top M \boldsymbol{\alpha}}{\boldsymbol{\alpha}^\top N \boldsymbol{\alpha}}, \tag{19}$$

where $\boldsymbol{\mu}_k = \frac{1}{|\mathcal{I}_k|} K \mathbf{1}_k$, $N = KK^\top - \sum_{k=1,2} |I_k| \boldsymbol{\mu}_k \boldsymbol{\mu}_k^\top$, $\boldsymbol{\mu} = \boldsymbol{\mu}_2 - \boldsymbol{\mu}_1$, $M = \boldsymbol{\mu}\boldsymbol{\mu}^\top$, and $K_{ij} = (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)) = \mathrm{k}(\mathbf{x}_i, \mathbf{x}_j)$. The projection of a test point onto the discriminant is computed by

$$(\mathbf{w} \cdot \Phi(\mathbf{x})) = \sum_{i=1}^{n} \alpha_i \, \mathrm{k}(\mathbf{x}_i, \mathbf{x}).$$

Finally, to use this projections in classification one needs to find a suitable threshold which can either be chosen as the mean of the average projections of the two classes or, e.g., by training a linear SVM on the projections.

As outlined before, the dimension of the feature space is equal to or higher than the number of training samples n which makes regularization necessary. In [7] it was proposed to add a multiple of e.g. the identity or the kernel matrix K to N, penalizing $\|\boldsymbol{\alpha}\|^2$ or $\|\mathbf{w}\|^2$, respectively (see also [87], [88]).

To maximize (19) one could either solve the generalized Eigenproblem $M\boldsymbol{\alpha} = \lambda N \boldsymbol{\alpha}$, selecting the Eigenvector $\boldsymbol{\alpha}$ with maximal Eigenvalue λ, or, equivalently, compute $\boldsymbol{\alpha} \equiv N^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)$. However, as the matrices N and M scale with the number of training samples and the solutions are non–sparse this is only feasible for moderate n. One possible solution is to transform KFD into a convex quadratic programming problem [89] which allows to derive a sparse variant of KFD and a more efficient, sparse–greedy approximation algorithm [90]. Recalling that Fisher's Discriminant tries to minimize the variance of the data along the projection whilst maximizing the distance between the average outputs for each class, the following quadratic program does exactly this:

$$\begin{aligned} \min_{\boldsymbol{\alpha}, b, \boldsymbol{\xi}} \quad & \|\boldsymbol{\xi}\|^2 + C\,\mathrm{P}(\boldsymbol{\alpha}) \\ \text{subject to} \quad & K\boldsymbol{\alpha} + \mathbf{1}b = \mathbf{y} + \boldsymbol{\xi} \\ & \mathbf{1}_k^\top \boldsymbol{\xi} = 0 \text{ for } k = 1, 2 \end{aligned} \tag{20}$$

for $\boldsymbol{\alpha}, \boldsymbol{\xi} \in \mathbb{R}^n$, and $b, C \in \mathbb{R}$. Here P is a regularizer as mentioned before and $(\mathbf{1}_k)_i$ is one for y_i belonging to class k and zero otherwise. It is straightforward to show, that

[6]This can be done with kernel functions as well and has explicitely been carried out e.g. in [9], [10]. However, most further developments for KFD do not easily carry over to the multi–class case, e.g. resulting in integer programming problems.

this program is equivalent to (19) with the same regularizer added to the matrix N [89]. The proof is based on the facts the (i) the matrix M is rank one and (ii) that the solutions $\mathbf{w}$ to (19) are invariant under scaling. Thus one can fix the distance of the means to some arbitrary, positive value, say two, and just minimize the variance. The first constraint, which can be read as $(\mathbf{w}\cdot\mathbf{x}_i)+b=y_i+\xi_i$, $i=1,\dots,n$, pulls the output for each sample to its class–label. The term $\|\boldsymbol{\xi}\|^2$ minimizes the variance of the error committed, while the constraints $\mathbf{1}_k^\top\boldsymbol{\xi}=0$ ensure that the average output for each class is the label, i.e. for ±1 labels the average distance of the projections is two. For $C=0$ one obtains the original Fisher algorithm in feature space.

B.1 Optimization

Besides a more intuitive understanding of the mathematical properties of KFD [89], in particular in relation to SVMs or the Relevance Vector Machine (RVM) [91], the formulation (20) allows to derive more efficient algorithms as well. Choosing a ℓ_1–norm regularizer $\mathrm{P}(\boldsymbol{\alpha})=\|\boldsymbol{\alpha}\|_1$ we obtain sparse solutions (sparse KFD (SKFD))[7]. By going even further and replacing the quadratic penalty on the variables $\boldsymbol{\xi}$ with an ℓ_1–norm as well, we obtain a linear program which can be very efficiently optimized using column generation techniques (e.g. [92]) (Linear Sparse KFD (LSKFD)). An alternative optimization strategy arising from (20) is to iteratively construct a solution to the full problem as proposed in [90]. Starting with an empty solution one adds in each iteration one pattern to the expansion (18). This pattern is chosen such that it (approximately) gives the largest decrease in the objective function (other criteria are possible). When the change in the objective falls below a predefined threshold the iteration is terminated. The obtained solution is sparse and yields competitive results compared to the full solution. The advantages of this approach are the smaller memory requirements and faster training time compared to quadratic programming or the solution of an Eigenproblem.

C. Connection between Boosting and Kernel Methods

We will now show a connection of Boosting to SVMs and KFD. Let us start with a very brief review of Boosting methods, which does not claim to be complete – for more details see e.g. [93], [53], [94], [95], [96], [97]. The first boosting algorithm was proposed by Rob Schapire [98]. This algorithm was able to "boost" the performance of a weak PAC learner [99] such that the resulting algorithm satisfies the strong PAC learning criteria [100].[8] Later, Freund and Schapire found an improved PAC boosting algorithm – called AdaBoost [53] – which repeatedly calls a given "weak learner" (also: base learning algorithm) $\mathcal{L}$ and finally produces a master hypothesis f which is a convex combination of the functions h_j produced by the base learning algorithm, i.e. $f(\mathbf{x})=\sum_{t=1}^{T}\frac{w_t}{\|\mathbf{w}\|_1}h_t(\mathbf{x})$ and $w_t\ge0$, $t=1,\dots,T$. The given weak learner $\mathcal{L}$ is used with different distributions $p=[p_1,\dots,p_n]$ (where $\sum_i p_i=1$, $p_i\ge0, i=1,\dots,n$) on the training set, which are chosen in such a way that patterns poorly classified by the current master hypothesis are more emphasized than other patterns.

Recently, several researchers [101], [102], [103], [104] have noticed that AdaBoost implements a constraint gradient descent (coordinate-descent) method on an exponential function of the margins. From this understanding, it is apparent that other algorithms can be derived [101], [102], [103], [104].[9] A slight modification of AdaBoost – called Arc-GV – has been proposed in [105].[10] For Arc-GV it can be proven that it asymptotically (with the number of iterations) finds a convex combination of all possible base hypotheses that maximizes the margin – very much in spirit to the hard margin SVM mentioned in Section IV-A. Let $H:=\{h_j\mid j=1,\dots,J\}$ be the set of hypotheses, from which the base learner can potentially select hypotheses. Then the solution of Arc-GV is the same as the one of the following linear program [105], that maximizes the smallest margin ρ:

$$\begin{aligned}&\max_{\mathbf{w}\in\mathcal{F},\rho\in\mathbb{R}_+} && \rho\\ &\text{subject to} && y_i\sum_{j=1}^{J}w_jh_j(\mathbf{x}_i)\ge\rho \quad\text{for}\quad i=1,\dots,n\\ &&& \|\mathbf{w}\|_1=1.\end{aligned}\tag{21}$$

Let us recall that SVMs and KFD implicitly compute scalar products in feature space with the help of the kernel trick. Omitting the bias ($b\equiv0$) for simplicity, the SVM minimization of (9) subject to (8) can be restated as a maximization of the margin ρ (cf. Fig. 3)

$$\begin{aligned}&\max_{\mathbf{w}\in\mathcal{F},\rho\in\mathbb{R}_+} && \rho\\ &\text{subject to} && y_i\sum_{j=1}^{N}w_j\,\mathrm{P}_j[\Phi(\mathbf{x}_i)]\ge\rho \quad\text{for}\quad i=1,\dots,n\\ &&& \|\mathbf{w}\|_2=1,\end{aligned}\tag{22}$$

where $N=\dim(\mathcal{F})$ and P_j is the operator projecting onto the j-th coordinate in feature space. The use of the ℓ_2–norm of $\mathbf{w}$ in the last constraint implies that the resulting hyperplane is chosen such that the minimum ℓ_2–distance of a training pattern to the hyperplane is maximized (cf. Section II-B). More generally, using an arbitrary ℓ_p–norm

[7]Roughly speaking, a reason for the induced sparseness is the fact that vectors far from the coordinate axes are "larger" with respect to the ℓ_1–norm than with respect to ℓ_p-norms with $p>1$. For example, consider the vectors $(1,0)$ and $(1/\sqrt{2},1/\sqrt{2})$. For the two norm, $\|(1,0)\|_2=\|(1/\sqrt{2},1/\sqrt{2})\|_2=1$, but for the ℓ_1–norm, $1=\|(1,0)\|_1<\|(1/\sqrt{2},1/\sqrt{2})\|_2=\sqrt{2}$. Note that using the ℓ_1–norm as regularizer the optimal solution is always a vertex solution (or can be expressed as such) and tends to be very sparse.

[8]A method that builds a strong PAC learning algorithm from a weak PAC learning algorithm is called a PAC boosting algorithm [96].

[9]cf. also [96] for an investigation in which potentials lead to PAC boosting algorithms.

[10]A generalization of Arc-GV using slack variables as in Eq. (12) can be found in [106], [92].

constraint on the weight vector leads to maximizing the ℓ_q–distance between hyperplane and training points [107], where $\frac{1}{q}+\frac{1}{p}=1$. Thus, in (21) one maximizes the minimum ℓ_∞–distance of the training points to the hyperplane.

On the level of the mathematical programs (22) and (21), one can clearly see the relation between Boosting and SVMs. The connection can be made even more explicit by observing that any hypothesis set H implies a mapping Φ by

$$\Phi : \mathbf{x} \mapsto [h_1(\mathbf{x}), \dots, h_N(\mathbf{x})]^\top,$$

and therefore also a kernel $k(\mathbf{x},\mathbf{y}) = (\Phi(y)\cdot\Phi(\mathbf{y})) = \sum_{j=1}^N h_j(\mathbf{x})h_j(\mathbf{y})$, which could in principle be used for SVM learning. Thus, any hypothesis set H spans a feature space $\mathcal{F}$. Furthermore, for any feature space $\mathcal{F}$, which is spanned by some mapping Φ, the corresponding hypothesis set H can be readily constructed by $h_j = \mathrm{P}_j[\Phi]$.

Boosting, in contrast to SVMs, performs the computation *explicitly* in feature space. This is well-known to be prohibitive, if the solution $\mathbf{w}$ is not sparse, as the feature space might be very high dimensional. As mentioned in Section IV-B (cf. Footnote 7), using the ℓ_1–norm instead of the ℓ_2–norm, one can expect to get sparse solutions in $\mathbf{w}$.[11] This might be seen as one important ingredient for Boosting, as it relies on the fact that there are only a few hypotheses/dimensions $h_j = \mathrm{P}_j[\Phi]$ needed to express the solution, which Boosting tries to find during each iteration. Basically, Boosting considers only the most important dimensions in feature space and can this way be very efficient.

D. Wrapping Up

SVMs, KFD and Boosting work in very high-dimensional feature spaces. They differ, however, in how they deal with the algorithmic problems that this can cause. One can think of boosting as an SV approach in a high dimensional feature space spanned by the base hypothesis of some function set H. The problem becomes tractable since Boosting uses effectively a ℓ_1–norm regularizer. This induces sparsity, hence one never really works in the full space, but always in a small subspace. Vice versa, one can think of SVMs and KFD as a "boosting approach" in a high-dimensional space. There we use the kernel trick and therefore never explicitly work in the feature space. Thus, SVMs and KFD get away without having to use ℓ_1–norm regularizers; indeed, they *could* not use them on $\mathbf{w}$, as the kernel only allows computation of the ℓ_2-norm in feature space. SVM and Boosting lead to sparse solutions (as does KFD with the appropriate regularizer [89]), although in different spaces, and both algorithms are constructed to exploit the form of sparsity they produce. Besides providing insight, this correspondence has concrete practical benefits for designing new algorithms. Almost any new development in the field of SVMs can be translated to a corresponding Boosting algorithm using the ℓ_1–norm instead of the ℓ_2–norm and vice versa (cf. [106], [109], [110]).

[11] Note that the solution of SVMs is under rather mild assumption not sparse in $\mathbf{w} = \sum_{i=1}^n \alpha_i \Phi(\mathbf{x}_i)$ [108], but in $\boldsymbol{\alpha}$.

V. Unsupervised Learning

In unsupervised learning only the data $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^N$, is given, i.e. the labels are missing. Standard questions of unsupervised learning are clustering, density estimation, and data description (see e.g. [111], [51]). As already outlined above, the kernel trick cannot only be applied in supervised learning scenarios, but also for unsupervised learning, *given that the base algorithm can be written in terms of scalar products*. In the following sections we will first review one of the most common statistical data analysis algorithm, PCA, and explain its "kernelized" variant: kernel PCA (see [11]). Subsequently, single-class classification is explained. Here the support of a given data set is being estimated (see e.g. [14], [112], [113], [110]). Recently, single-class SVMs are frequently used in outlier or novelty detection applications.

A. Kernel PCA

The basic idea of PCA is depicted in Figure 8. For N-

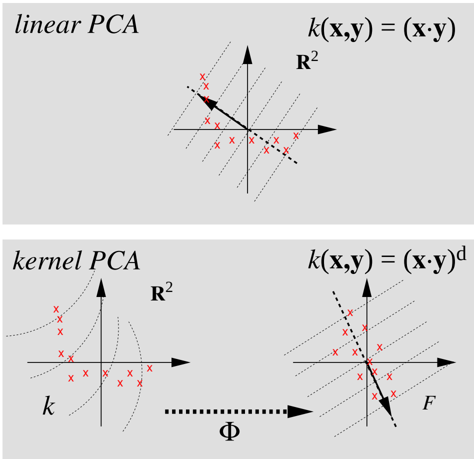


Fig. 8. By using a kernel function, Kernel-PCA is implicitly performing a linear PCA in some high dimensional feature space, that is nonlinearly related to input space. Linear PCA in the input space (top) is not sufficient to describe the most interesting direction in this toy example. Contrary, using a suitable nonlinear mapping Φ and performing linear PCA on the mapped patterns (Kernel PCA), the resulting *nonlinear* direction in the input space can find the most interesting direction (bottom) (figure from [11]).

dimensional data, a set of orthogonal directions – capturing most of the variance in the data – is computed, i.e. the first k projections ($k = 1, \dots, N$) allow to reconstruct the data with minimal quadratic error. In practice one typically wants to describe the data with reduced dimensionality by extracting a few meaningful components, while at the same time one is retaining most existing structure in the data (see e.g. [114]). Since PCA is a linear algorithm it is clearly beyond its capabilities to extract nonlinear structures in the data as, e.g., the one observed in Figure 8. It is here, where the *Kernel-PCA* algorithm sets in. To derive Kernel-PCA we first map the data $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^N$ into a feature space $\mathcal{F}$ (cf. Section III) and compute the covariance matrix

$$C = \frac{1}{n}\sum_{j=1}^{n} \Phi(\mathbf{x}_j)\Phi(\mathbf{x}_j)^\top.$$

The principal components are then computed by solving the Eigenvalue problem: find $\lambda > 0, \mathbf{V} \neq 0$ with

$$\lambda\mathbf{V} = C\mathbf{V} = \frac{1}{n}\sum_{j=1}^{n}(\Phi(\mathbf{x}_j)\cdot\mathbf{V})\Phi(\mathbf{x}_j). \qquad (23)$$

Furthermore, as can be seen from (23) all Eigenvectors with non–zero Eigenvalue must be in the span of the mapped data, i.e. $\mathbf{V} \in \mathrm{span}\{\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_n)\}$. This can be written as

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

By multiplying with $\Phi(\mathbf{x}_k)$ from the left (23) reads

$$\lambda(\Phi(\mathbf{x}_k)\cdot\mathbf{V}) = (\Phi(\mathbf{x}_k)\cdot C\mathbf{V}) \text{ for all } k = 1,\ldots,n.$$

Defining an $n \times n$-matrix

$$K_{ij} := (\Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j)) = \mathrm{k}(\mathbf{x}_i,\mathbf{x}_j) \qquad (24)$$

one computes an Eigenvalue problem for the expansion coefficients α_i, that is now solely dependent on the kernel function

$$\lambda\boldsymbol{\alpha} = K\boldsymbol{\alpha} \quad (\boldsymbol{\alpha} = (\alpha_1,\ldots,\alpha_n)^\top).$$

The solutions $(\lambda_k, \boldsymbol{\alpha}^k)$ further need to be normalized by imposing $\lambda_k(\boldsymbol{\alpha}^k\cdot\boldsymbol{\alpha}^k) = 1$ in $\mathcal{F}$. Also – as in every PCA algorithm – the data needs to be centered in $\mathcal{F}$. This can be done by simply substituting the kernel-matrix K with

$$\hat{K} = K - 1_n K - K 1_n + 1_n K 1_n,$$

where $(1_n)_{ij} = 1/n$; for details see [11].

For extracting features of a new pattern $\mathbf{x}$ with kernel PCA one simply projects the mapped pattern $\Phi(\mathbf{x})$ onto $\mathbf{V}^k$

$$\begin{aligned}(\mathbf{V}^k\cdot\Phi(\mathbf{x})) &= \sum_{i=1}^{M}\alpha_i^k(\Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x})) \\ &= \sum_{i=1}^{M}\alpha_i^k\,\mathrm{k}(\mathbf{x}_i,\mathbf{x}). \end{aligned} \qquad (25)$$

Note that in this algorithm for nonlinear PCA the nonlinearity enters the computation only at two points that do not change the nature of the algorithm: (a) in the calculation of the matrix elements of K (24), and (b) in the evaluation of the expansion (25). So, for obtaining the Kernel-PCA components one only needs to solve a similar linear Eigenvalue problem as before for linear PCA, the only difference being that one has to deal with an $n \times n$ problem instead of an $N \times N$ problem. Clearly, the size of this problem becomes problematic for large n. [115] proposes to solve this by using a sparse approximation of the matrix K which still describes the leading Eigenvectors sufficiently well. In [116] a sparse kernel PCA approach is proposed, set within a Bayesian framework. Finally, the approach given in [117] places a ℓ_1–regularizer into the (kernel) PCA problem with the effect of obtaining sparse solutions as well at a comparably low computational cost. Figures 9 – 11 show examples for feature extraction with linear PCA and Kernel-PCA for artificial data sets. Further applications of kernel PCA for real world data can be found in Section VII-A.1 for OCR or in Section VII-C.1 for denoising problems, other applications are found in e.g. [6], [12], [119].

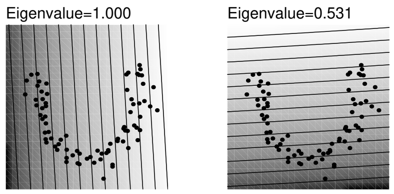


Fig. 9. Linear PCA, or, equivalently, Kernel-PCA using $\mathrm{k}(\mathbf{x},\mathbf{y}) = (\mathbf{x}\cdot\mathbf{y})$. Plotted are two linear PCA features (sorted according to the size of the Eigenvalues) on an artificial data set. Similar grey values denote areas of similar feature value (cf. (25)). The first feature (left) projects to the direction of maximal variance in the data. Clearly, one cannot identify the nonlinear structure in the underlying data using linear PCA only (figure from [118]).

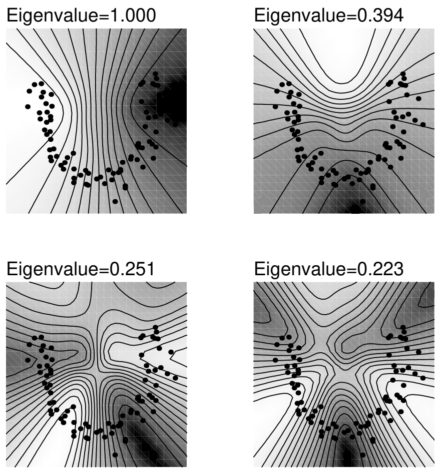


Fig. 10. The first 4 nonlinear features of Kernel-PCA using a sigmoidal Kernel on the data set from Figure 9. The Kernel-PCA components capture the nonlinear structure in the data, e.g. the first feature (upper left) is better adapted to the curvature of the data than the respective linear feature from Figure 9 (figure from [118]).

B. Single-Class Classification

A classical unsupervised learning task is density estimation. Assuming that the unlabeled observations $\mathbf{x}_1,\ldots,\mathbf{x}_n$ were generated i.i.d. according to some unknown distribution $P(\mathbf{x})$, the task is to estimate its density. However, there are several difficulties to this task. First, a density need not always exist — there are distributions that do not possess a density. Second, estimating densities exactly is

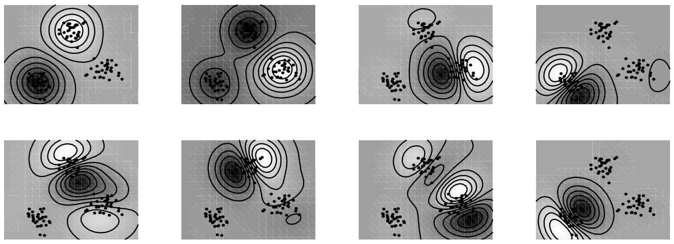


Fig. 11. The first 8 nonlinear features of Kernel-PCA using a RBF Kernel on a toy data set consisting of 3 Gaussian clusters (see [11]). Upper left: the first and second component split the data into three clusters. Note that Kernel-PCA is not primarily built to achieve such a clustering. Rather it tries to find a good description of the data in feature space and in this case the cluster structure extracted has the maximal variance in feature space. The higher components depicted split each cluster in halves (components 3 – 5), finally features 6 – 8 achieve orthogonal splits with respect to the previous splits (figure from [11]).

known to be a hard task. In many applications it is enough to estimate the support of a data distribution instead of the full density. Single-class SVMs avoid solving the harder density estimation problem and concentrate on the simpler task [3], i.e. estimating quantiles of the multivariate distribution, i.e. its support. So far there are two independent algorithms to solve the problem in a kernel feature space. They differ slightly in spirit and geometric notion [113], [14]. It is, however, not quite clear which of them is to be preferred in practice (cf. Figures 12 and 13). One solution of the single-class SVM problem by Tax and Duin [113] uses *spheres* with soft margins to describe the data in feature space, close in spirit to the algorithm of [120]. For certain classes of kernels, such as Gaussian RBF ones, this sphere single-class SVM algorithm can be shown to be equivalent to the second Ansatz which is due to Schölkopf et. al. [14]. For brevity we will focus on this second approach as it is more in the line of this review since it uses margin arguments. It computes a hyperplane in feature space such that a pre-specified fraction of the training example will lie beyond that hyperplane, while at the same time the hyperplane has maximal distance (margin) to the origin. For an illustration see Figure 12. To this end, we solve the

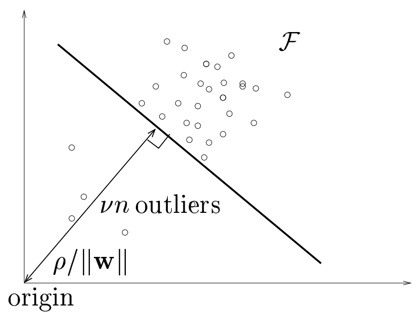


Fig. 12. Illustration of single-class idea. Solving Eq.(26), a hyperplane in $\mathcal{F}$ is constructed that maximizes the distance to the origin while allowing for ν outliers.

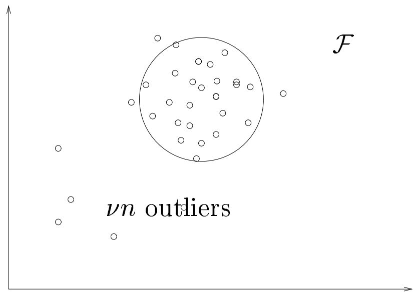


Fig. 13. Illustration of single-class idea. Construction of the smallest soft sphere in $\mathcal{F}$ that contains the data.

following quadratic program [14]:

$$\min_{\mathbf{w}\in F,\boldsymbol{\xi}\in\mathbb{R}^n,\rho\in\mathbb{R}} \quad \tfrac{1}{2}\|\mathbf{w}\|^2 + \tfrac{1}{\nu n}\textstyle\sum_i \xi_i - \rho \tag{26}$$

$$\text{subject to} \quad (\mathbf{w}\cdot\Phi(\mathbf{x}_i)) \geq \rho - \xi_i,\ \xi_i \geq 0. \tag{27}$$

Here, $\nu \in (0, 1]$ is a parameter akin to the one described above for the case of pattern recognition. Since nonzero slack variables ξ_i are penalized in the objective function, we can expect that if $\mathbf{w}$ and ρ solve this problem, then the decision function $f(\mathbf{x}) = \text{sign}((\mathbf{w}\cdot\Phi(\mathbf{x}))-\rho)$ will be positive for most examples $\mathbf{x}_i$ contained in the training set, while the SV type regularization term $\|\mathbf{w}\|$ will still be small. The actual trade-off between these two goals is controlled by ν. Deriving the dual problem, the solution can be shown to have a SV expansion (again, patterns $\mathbf{x}_i$ with nonzero α_i are called SVs)

$$f(\mathbf{x}) = \text{sign}\left(\sum_i \alpha_i\,\mathrm{k}(\mathbf{x}_i,\mathbf{x}) - \rho\right),$$

where the coefficients are found as the solution of the dual problem:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\sum_{ij}\alpha_i\alpha_j\,\mathrm{k}(\mathbf{x}_i,\mathbf{x}_j) \qquad (28)\\ \text{subject to} \quad & 0 \leq \alpha_i \leq 1/(\nu n), i = 1,\dots,n \\ & \textstyle\sum_{i=1}^n \alpha_i = 1. \end{aligned}$$

This problem can be solved with standard QP routines. It does, however, possess features that sets it apart from generic QPs, most notably the simplicity of the constraints. This can be exploited by applying a variant of SMO developed for this purpose [14].

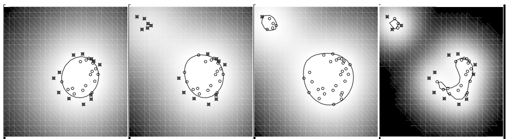
The offset ρ can be recovered by exploiting that for any α_i which is not at the upper or lower bound, the corresponding pattern $\mathbf{x}_i$ satisfies $\rho = (\mathbf{w}\cdot\Phi(\mathbf{x}_i)) = \sum_j \alpha_j\,\mathrm{k}(\mathbf{x}_j,\mathbf{x}_i)$.

Note that if ν approaches 0, the upper boundaries on the Lagrange multipliers tend to infinity, i.e. the first inequality constraint in (28) becomes void. The problem then resembles the corresponding *hard margin* algorithm, since the penalization of errors becomes infinite, as can be seen from the primal objective function (26). It can be shown that if the data set is separable from the origin, then this algorithm will find the unique supporting hyperplane with the properties that it separates all data from the origin, and its distance to the origin is maximal among all such hyperplanes. If, on the other hand, ν equals 1, then the constraints alone only allow one solution: the one where all α_i are at the upper bound $1/(\nu n)$. In this case, for kernels with integral 1, such as normalized versions of (7), the decision function corresponds to a thresholded Parzen windows estimator. For the parameter ν one can show that it controls the fraction of errors and SVs (along the lines of Section IV-A).

Theorem 2 ([14]) Assume the solution of (27) satisfies $\rho \neq 0$. The following statements hold:
(i) ν is an upper bound on the fraction of outliers.
(ii) ν is a lower bound on the fraction of SVs.
(iii) Suppose the data were generated independently from a distribution $P(\mathbf{x})$ which does not contain discrete components. Suppose, moreover, that the kernel is analytic and non-constant. When the number n of samples goes to infinity, with probability 1, ν equals both the fraction of SVs and the fraction of outliers.

We have thus described an algorithm which will compute a region that captures a certain fraction of the training examples. It is a "nice" region, as it will correspond to a small value of $\|\mathbf{w}\|^2$, thus the underlying function will be smooth [58]. How about test examples? Will they also lie inside the computed region? This question is the subject of single-class generalization error bounds [14]. Roughly, they state the following: suppose the estimated hyperplane has a small $\|\mathbf{w}\|^2$ and separates part of the training set from the origin by a certain margin $\rho/\|\mathbf{w}\|$. Then the probability that *test* examples coming from the same distribution lie outside of a slightly *larger* region will not be much larger than the fraction of training outliers.

Figure 14 displays 2-D toy examples, and shows how the parameter settings influence the solution. For further applications, including an outlier detection task in handwritten character recognition, cf. [14].



ν, width c	0.5, 0.5	0.5, 0.5	0.1, 0.5	0.5, 0.1
frac. SVs/OLs	0.54, 0.43	0.59, 0.47	0.24, 0.03	0.65, 0.38
margin $\rho/\|\mathbf{w}\|$	0.84	0.70	0.62	0.48

Fig. 14. A single-class SVM using RBF kernel (7) applied to a toy problem; domain: $[-1,1]^2$. *First two pictures:* Note how in both cases, at least a fraction of ν of all examples is in the estimated region (cf. table). The large value of ν causes the additional data points in the upper left corner to have almost no influence on the decision function. For smaller values of ν, such as 0.1 *(third picture)*, the points cannot be ignored anymore. Alternatively, one can force the algorithm to take these 'outliers' into account by changing the kernel width (7): in the *fourth picture*, using $c = 0.1, \nu = 0.5$, the data is effectively analyzed on a different length scale which leads the algorithm to consider the outliers as meaningful points. Figure taken from [14].

VI. Model Selection

In the kernel methods discussed so far, the choice of the kernel has a crucial effect on the performance, i.e. if one does not choose the kernel properly, one will not achieve the excellent performance reported in many papers. *Model selection* techniques provide principled ways to select a proper kernel. Usually, the candidates of optimal kernels are prepared using some heuristic rules, and the one which minimizes a given criterion is chosen. There are three typical ways for model selection with different criteria, each of which is a prediction of the generalization error

(i) **Bayesian evidence framework** The training of a SVM is interpreted as Bayesian inference, and the model selection is done by maximizing the marginal likelihood (i.e. evidence), e.g. [121], [91].
(ii) **PAC** The generalization error is upper bounded using a capacity measure depending both on the weights and the model, and these are optimized to minimize the bound. The kernel selection methods for SVM following this approach are reported e.g. in [122], [36], [123].
(iii) **Cross validation** Here, the training samples are divided to k subsets, each of which have the same number of samples. Then, the classifier is trained k-times: In the i-th ($i = 1, \ldots, k$) iteration, the classifier is trained on all subsets except the i-th one. Then the classification error is computed for the i-th subset. It is known that the average of these k errors is a rather good estimate of the generalization error [124]. The extreme case, where k is equal to the number of training samples, is called *leave-one-out* cross validation. Note that bootstrap [125], [126] is also a principled resampling method which is often used for model selection.

Other approaches, namely asymptotic statistical methods such as AIC [41] and NIC [43] can be used. However, since these methods need a large amount of samples by assumption, they have not been used in kernel methods so far. For (i) and (ii), the generalization error is approximated by expressions that can be computed efficiently. For small sample sizes, these values are sometimes not very accurate, but it is known that nevertheless often acceptable good models are selected. Among the three approaches, the most frequently used method is (iii) [124], but the problem is that the computational cost is the highest, because the learning problem must be solved k times. For SVM, there is an approximate way to evaluate the n-fold cross validation error (i.e. the leave-one-out classification error) called *span bound* [127]. If one assumes that the support vectors do not change even when a sample is left out, the leave-one-out classification result of this sample can be computed exactly. Under this assumption, we can obtain an estimate of the leave-one-out error – without retraining the SVM many times. Although this assumption is rather crude and not true in many cases, this approach gives a close approximation of the true leave-one-out error in experiments. For KFD there exists a similar result.

Now we would like to describe a particular efficient model selection method that has in practice often been used [128],

[102], [89], [7], [129], [130] in conjunction with the benchmark data sets described in Section VII-B.

In model selection for SVMs and KFD we have to determine the kernel parameters (one (RBF) or more (e.g. polynomial kernel)) and the regularization constant C or ν, while for Boosting one needs to choose the model-parameters of the base learner, a regularization constant and the number of boosting iterations. Given a certain benchmark data set, one usually has a number, say M (e.g. 100), realizations, i.e. splits into training and test set, available (cf. Section VII-B). The different splits are often necessary to average the results in order to get more reliable estimates of the generalization error.

One possibility to do model-selection would be to consider each realization independently from all others and to perform the cross-validation procedure M times. Then, for each realization one would end-up with different model parameters, as the model selection on each realizations will typically have various results.

It is less computationally expensive to have only one model for all realizations of one data set: To find this model, we run a 5-fold-cross validation procedure only on a few, say five, realizations of the data set. This is done in two stages: first a global search (i.e. over a wide range of the parameter space) is done to find a good guess of the parameter, which becomes more precise in the second stage. Finally, the model parameters are computed as the median of the five estimations and are used throughout the training on all M realization of the data set. This way of estimating the parameters is computationally still quite expensive, but much less expensive than the full cross validation approach mentioned above.

VII. Applications

This section describes selected[12] interesting applications of supervised and unsupervised learning with kernels. It serves to demonstrate that kernel based approaches achieve competitive results over a whole range of benchmarks with different noise levels and robustness requirements.

A. *Supervised Learning*

A.1 OCR

Historically the first real-world experiments of SVMs[13] – all done on OCR benchmarks (see Fig. 15) – exhibited quite high accuracies for SVMs [2], [120], [4], [131] comparably to state–of–the–art results achieved with convolutive multi-layer perceptrons [132], [133], [134], [135]. Table III shows the classification performance of SVMs in comparison to other state–of–the art classifiers on the US-Postal Service (USPS) benchmark. Plain SVM give a performance very similar to other state–of–the–art methods. However SVMs can be strongly improved by using prior knowledge. For instance in [4] virtual support vectors have been generated by transforming the set of support vectors with an appropriate invariance transformation and retraining the machine on these vectors. Furthermore one can structure kernels such that they induce local invariances like translations, line thickening or rotations or that e.g. products of neighboring pixels in an image [131], that are thought to contain more information, are emphasized. So, prior knowledge can be used for engineering a larger data set or problem specific kernels (see also Section VII-A.2 for an application of this idea to DNA analysis). In a two stage process we also used kernel-PCA to extract features from the USPS data in the first step. A subsequent linear classification on these nonlinear features allowed to achieve an error rate of 4%, which is better by a factor of two than operating on linear PCA features (8.7%, cf. [11]).

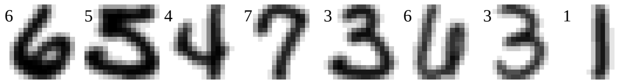


Fig. 15. Typical handwritten digits from the US-Postal Service (USPS) benchmark data set with 7291 training and 2007 test patterns (16 × 16 gray scale images).

TABLE III

Classification error in % for off-line handwritten character recognition on the (USPS) with 7291 patterns. Invariant SVMs are only slightly below the best existing results (parts of the table are from [136]). This is even more remarkable since in [137], [135], [136], a larger training set was used, containing some additional machine-printed digits which have been found to improve the accuracy.

linear PCA & linear SVM (Schölkopf et. al. [11])	8.7%
k-Nearest Neighbor	5.7%
LeNet1 (LeCun et. al. [132], [133], [134])	4.2%
Regularized RBF Networks (Rätsch [128])	4.1%
Kernel-PCA & linear SVM (Schölkopf et. al. [11])	4.0%
SVM (Schölkopf et. al. [120])	4.0%
Virtual SVM (Schölkopf [4])	3.0%
Invariant SVM (Schölkopf et. al. [131])	3.0%
Boosting (Drucker et. al. [137])	2.6%
Tangent Distance (Simard et. al. [135], [136])	2.5%
Human error rate	2.5%

A benchmark problem larger than the USPS data set (7291 patterns) was collected by NIST and contains 120000 handwritten digits. Invariant SVMs achieved the record error rate of 0.6% [18] on this challenging and more realistic data set, better than tangent distance (1.1%) and convolutional neural networks (LeNet 5: 0.9%). With an error rate of 0.7%, an ensemble of LeNet 4 networks that was trained on a vast number of artificially generated patterns (using invariance transformations) almost matches the performance of the best SVM [134].

A.2 Analyzing DNA Data

The genomic text contains untranslated regions and so called coding sequences (CDS) that encode proteins. In or-

[12] Note that for our own convenience we have biased the selection towards applications pursued by the IDA group while adding abundant references to other work.

[13] performed at AT&T Bell Labs.

der to extract protein sequences from nucleotide sequences, it is a central problem in computational biology to recognize the translation initiation sites (TIS) from which coding starts to determine which parts of a sequence will be translated and which not.

Coding sequences can in principle be characterized with alignment methods that use homologous proteins (e.g. [138]) or intrinsic properties of the nucleotide sequence that are learned for instance with Hidden Markov models (e.g. [139]). A radically different approach that has turned out to be even more successful is to model the task of finding TIS as a classification problem (see e.g. [140], [28]). A potential start codon is typically a ATG[14] triplet. The classification task is therefore to decide whether or not a binary coded (fixed length) sequence window[15] around the ATG indicates a true TIS. The machine learning algorithm, for example the neural network [140] or the SVM [28] gets a training set consisting of an input of binary coded strings in a window around the ATG together with a label indicating true/false TIS. In contrast to alignment methods, both neural networks and the SVM algorithm are finding important structure in the data by learning in the respective feature space to successfully classify from the labeled data.

As indicated in Section VII-A.1, one can incorporate prior knowledge to SVMs e.g. by using a proper feature space $\mathcal{F}$. In particular in the task of TIS recognition it turned out to be very helpful to include biological knowledge by engineering an appropriate kernel function [28]. We will give three examples for kernels that are particularly useful for start codon recognition. While certain local correlations are typical for TIS, dependencies between distant positions are of minor importance or are a priori known to not even exist. We want the feature space to reflect this. Thus, we modify the kernel utilizing a technique that was originally described for OCR in [131]: At each sequence position, we compare the two sequences locally, within a small window of length $2l+1$ around that position. We count matching nucleotides, multiplied with weights $\mathbf{p}$ increasing from the boundaries to the center of the window. The resulting weighted counts are taken to the d_1^{th} power

$$\mathrm{win}_p(\mathbf{x},\mathbf{y}) = \left(\sum_{j=-l}^{+l} p_j \ \mathrm{match}_{p+j}(\mathbf{x},\mathbf{y}) \right)^{d_1},$$

where d_1 reflects the order of local correlations (within the window) that we expect to be of importance. Here, $\mathrm{match}_{p+j}(\mathbf{x},\mathbf{y})$ is 1 for matching nucleotides at position $p+j$ and 0 otherwise. The window scores computed with win_p are summed over the whole length of the sequence. Correlations between up to d_2 windows are taken into account by applying potentiation with d_2 to the resulting sum.

$$\mathrm{k}(\mathbf{x},\mathbf{y}) = \left(\sum_{p=1}^{l} \mathrm{win}_p(\mathbf{x},\mathbf{y}) \right)^{d_2}.$$

We call this kernel locality-improved (contrary to a plain polynomial kernel), as it emphasizes *local* correlations.

In an attempt to further improve performance we aimed to incorporate another piece of biological knowledge into the kernel, this time concerning the codon-structure of the coding sequence. A codon is a triplet of adjacent nucleotides that codes for one amino acid. By definition the difference between a true TIS and a pseudo site is that downstream of a TIS there is CDS (which shows codon structure), while upstream there is not. CDS and non-coding sequences show statistically different compositions. It is likely that the SVM exploits this difference for classification. We could hope to improve the kernel by reflecting the fact that CDS shifted by three nucleotides still looks like CDS. Therefore, we further modify the locality-improved kernel function to account for this translation-invariance. In addition to counting matching nucleotides on corresponding positions, we also count matches that are shifted by three positions. We call this kernel codon-improved. Again, it can be shown to be a valid mercer kernel function by explicitly deriving the monomial features.

TABLE IV
COMPARISON OF CLASSIFICATION ERRORS (MEASURED ON THE TEST SETS) ACHIEVED WITH DIFFERENT LEARNING ALGORITHMS. FOR DETAILS SEE TEXT.

algorithm	**parameter setting**	**overall error**
neural network		15.4%
Salzberg method		13.8%
SVM, simple polynomial	d=1	13.2%
SVM, locality-improved kernel	d_1=4,l=4	11.9%
SVM, codon-improved kernel	d_1=2,l=3	12.2%
SVM, Salzberg kernel	d_1=3,l=1	11.4%

A third direction for the modification of the kernel function is obtained by the Salzberg method, where we essentially represent each data point by a sequence of log odd scores relating, individually for each position, two probabilities: first, how likely the observed nucleotide at that position derives from a true TIS and second, how likely that nucleotide occurs at the given position relative to any ATG triplet. We then proceed analogously to the locality-improved kernel, replacing the sparse bit representation by the sequence of these scores. As expected, this leads to a further increase in classification performance. In the strict sense this is not a kernel but corresponds to preprocessing.

The result of an experimental comparison of SVMs using these kernel functions with other approaches are summa-

[14] DNA has a four letter alphabet: A,C,G,T.

[15] We define the input space by the same sparse bit-encoding scheme as used by Pedersen and Nielsen (personal communication): each nucleotide is encoded by five bits, exactly one of which is set. The position of the set bit indicates whether the nucleotide is A, C, G or T, or if it is unknown. This leads to an input space of dimension $n = 1000$ for a symmetric window of size 100 to the left and right of the ATG sequence.

rized in Table IV. All results are averages over six data partitions (about 11000 patterns for training and 3000 patterns for testing). SVMs are trained on 8000 data points. An optimal set of model-parameters is selected according to the error on the remaining training data and the average errors on the remaining test set are reported in Table IV. Note that the windows consist of $2l + 1$ nucleotides. The NN results are those achieved by Pedersen and Nielsen ([140], personal communication). There, model selection seems to have involved test data, which might lead to slightly over-optimistic performance estimates. Positional conditional preference scores are calculated analogously to Salzberg [141], but extended to the same amount of input data also supplied to the other methods. Note that the performance measure shown depends on the value of the classification function threshold. For SVMs, the thresholds are by-products of the training process; for the Salzberg method, "natural" thresholds are derived from prior probabilities by Bayesian reasoning. Overall error denotes the ratio of false predictions to total predictions. The sensitivity versus specificity trade-off can be controlled by varying the threshold.

In conclusion, all three engineered kernel functions clearly outperform the NN as devised by Pedersen and Nielsen or the Salzberg method by reducing the overall number of misclassifications drastically: up to 25% compared to the neural network.

Further successful applications of SVMs have emerged in the context of gene expression profile analysis [26], [27], DNA and protein analysis [29], [30], [31].

B. Benchmarks

To evaluate a newly designed algorithm it is often desirable to have some standardized benchmark data sets. For this purpose there exists some benchmark repositories, including UCI [142], DELVE [143] and STATLOG [144]. Some of them also provide results of some standard algorithms on these data sets. The problem about these repositories and the given results is that

- it is unclear how the model selection was performed,
- it is not in all cases stated how large the training and test samples have been,
- usually there is no information how reliable these results are (error bars),
- the data sometimes needs preprocessing,
- the problems are often multi-class problems,

Some of these factors might influence the result of the learning machine at hand and makes a comparison with results e.g. in other papers difficult.

Thus, another (very clean) repository – the *IDA repository* [145] – has been created, which contains thirteen artificial and real world data sets collected from the repositories above. The IDA repository is designed to cover a variety of different data sets: from small to high expected error rates, from low to high dimensional data and from small and large sample sizes. For each of the data sets banana (toy data set introduced in [128], [102]), breast cancer[16], diabetes, german, heart, image segment, ringnorm, flare solar, splice, thyroid, titanic, twonorm, waveform), the repository includes

- a short description of the dataset,
- 100 predefined splits into training and test samples,
- the simulation results for several kernel based and Boosting methods on each split including the parameters that have been used for each method,
- a simulation summary including means and standard deviations on the 100 realizations of the data.

To build the IDA repository for problems that are originally not binary classification problems, a random partition into two classes is used[17]. Furthermore for all sets preprocessing is performed and 100 different partitions into training and test set (mostly $\approx 60\% : 40\%$) have been generated. On each partition a set of different classifiers is trained, the best model is selected by cross-validation and then its test set error is computed. Some of the results are stated in Table V. This repository has been used so far to evaluate kernel and boosting methods e.g. in [128], [102], [89], [7], [129], [130], [97].

In Table V we show experimental comparisons between SVM, RBF, KFD and AdaBoost variants [8]. Due to the careful model selection performed in the experiments, all kernel based methods exhibit a similarly good performance. Note that we can expect such a result since they use similar implicit regularization concepts by employing the same kernel [58]. The remaining differences arise from their different loss functions which induce different margin optimization strategies: KFD maximizes the average margin whereas SVM maximizes the soft margin (ultimately the minimum margin). In practice, KFD or RVM have the advantage that – if required (e.g. medical application, motion tracking) – they can also supply a confidence measures for a decision. Furthermore, the solutions for KFD with a sparsity regularization are as sparse as for RVM [91] (i.e. much higher sparsity than for SVMs can be achieved), yet using an order of magnitude less computing time than the RVM [89].

B.1 Miscellaneous Applications

The high dimensional problem of text categorization seems to be another application where SVMs have been performing particularly well. A popular benchmark is the Reuters-22173 text corpus, where Reuters collected 21450 news stories from 1997, and partitioned and indexed them into 135 different categories, to simplify the access. The feature typically used to classify Reuters documents are 10000-dimensional vectors containing word frequencies within a document. With such a coding SVMs have been achieving excellent results, see e.g. [146], [78].

[16]The breast cancer domain was obtained from the University Medical Center, Inst. of Oncology, Ljubljana, Yugoslavia. Thanks go to M. Zwitter and M. Soklic for providing the data.

[17]A random partition generates a mapping $\mathbf{m}$ of n to two classes. For this a random ± 1 vector $\mathbf{m}$ of length n is generated. The positive classes (and the negative respectively) are then concatenated.

TABLE V

COMPARISON [8] BETWEEN SUPPORT VECTOR MACHINES, THE KERNEL FISHER DISCRIMINANT (KFD), A SINGLE RADIAL BASIS FUNCTION CLASSIFIER (RBF), ADABOOST (AB), AND REGULARIZED ADABOOST (AB_R) ON 13 DIFFERENT BENCHMARK DATASETS (SEE TEXT). BEST RESULT IN BOLD FACE, SECOND BEST IN ITALICS.

	SVM	KFD	RBF	AB	AB_R
Banana	11.5±0.07	**10.8±0.05**	**10.8±0.06**	12.3±0.07	*10.9±0.04*
B.Cancer	*26.0±0.47*	**25.8±0.46**	27.6±0.47	30.4±0.47	26.5±0.45
Diabetes	*23.5±0.17*	**23.2±0.16**	24.3±0.19	26.5±0.23	23.8±0.18
German	**23.6±0.21**	*23.7±0.22*	24.7±0.24	27.5±0.25	24.3±0.21
Heart	**16.0±0.33**	*16.1±0.34*	17.6±0.33	20.3±0.34	16.5±0.35
Image	*3.0±0.06*	3.3±0.06	3.3±0.06	**2.7±0.07**	**2.7±0.06**
Ringnorm	1.7±0.01	**1.5±0.01**	1.7±0.02	1.9±0.03	*1.6±0.01*
F.Sonar	**32.4±0.18**	*33.2±0.17*	34.4±0.20	35.7±0.18	34.2±0.22
Splice	10.9±0.07	10.5±0.06	*10.0±0.10*	10.1±0.05	**9.5±0.07**
Thyroid	4.8±0.22	**4.2±0.21**	4.5±0.21	*4.4±0.22*	4.6±0.22
Titanic	**22.4±0.10**	23.2±0.20	23.3±0.13	*22.6±0.12*	*22.6±0.12*
Twonorm	3.0±0.02	**2.6±0.02**	2.9±0.03	3.0±0.03	*2.7±0.02*
Waveform	*9.9±0.04*	*9.9±0.04*	10.7±0.11	10.8±0.06	**9.8±0.08**

Further applications of SVM include object and face recognition tasks as well as image retrieval [147], [148]. SVMs have also been successfully applied to solve inverse problems [5], [149].

C. Unsupervised Learning

C.1 Denoising

Kernel PCA as a nonlinear feature extractor has proven powerful as a preprocessing step for classification algorithms. But considering it as a natural generalization of linear PCA the question arises, how to use nonlinear features for data compression, reconstruction, and de-noising, applications common in linear PCA. This is a nontrivial task, as the results provided by kernel PCA live in the high dimensional feature space and need not have an exact representation by a single vector in input space. In practice this issue has been alleviated by computing approximate pre–images [12], [13], [116].

Formally, one defines a projection operator P_k which for each test point $\mathbf{x}$ computes the projection onto the first k (nonlinear) principal components, i.e.

$$\mathrm{P}_k\,\Phi(\mathbf{x}) = \textstyle\sum_{i=1}^k \beta_i \mathbf{V}^i$$

where $\beta_i := (\mathbf{V}^i \cdot \Phi(\mathbf{x})) = \sum_{j=1}^n \alpha_j^i\, \mathrm{k}(\mathbf{x}, \mathbf{x}_j)$. Lets assume that the Eigenvectors $\mathbf{V}$ are ordered with decreasing Eigenvalue size. It can be shown that these projections have similar optimality properties as linear PCA [12] making them good candidates for the following applications:

Denoising. Given a noisy $\mathbf{x}$, map it into $\Phi(\mathbf{x})$, discard higher components to obtain $\mathrm{P}_k\,\Phi(\mathbf{x})$, 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 k 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 $\boldsymbol{\alpha}^i$ and a small number of features β_i of $\Phi(\mathbf{x})$, but not $\mathbf{x}$, compute a pre-image as an approximate reconstruction of $\mathbf{x}$. This is useful if k is smaller than the dimensionality of the input data.

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

This can be achieved by computing a vector $\mathbf{z}$ satisfying $\Phi(\mathbf{z}) = \mathrm{P}_k\,\Phi(\mathbf{x})$. The hope is that for the kernel used, such a $\mathbf{z}$ will be a good approximation of $\mathbf{x}$ in input space. However, (i) such a $\mathbf{z}$ will not always exist and (ii) if it exists, it need be not unique (cf. [12], [13]). When the vector $\mathrm{P}_k\,\Phi(\mathbf{x})$ has no pre-image $\mathbf{z}$ one can approximate it by minimizing

$$\rho(\mathbf{z}) = \|\Phi(\mathbf{z}) - \mathrm{P}_k\,\Phi(\mathbf{x})\|^2, \tag{29}$$

what can be seen as a special case of the reduced set method [150], [13]. The optimization of (29) can be formulated using kernel functions. Especially for RBF kernels (cf. (7)) there exists an efficient fixed–point iteration. For further details of how to optimize (29) and for details of the experiments reported below the reader is referred to [13].

C.1.a Example. The example shown here (taken from [12]) was carried out with Gaussian kernels, minimizing (29). Figure 16 illustrates the pre–image approach in an artificial de-noising task on the USPS database. In these experiments, linear and kernel PCA were trained with the original data. To the test set

(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 $p = 0.2$.

was added. For the noisy test sets, projections onto the first k linear and nonlinear components were computed and the reconstruction was carried out for each case. The results were compared by taking the mean squared distance of each reconstructed digit of the noisy test set to its original counterpart.

For the optimal number of components in linear and kernel PCA, the non–linear approach did better by a factor of

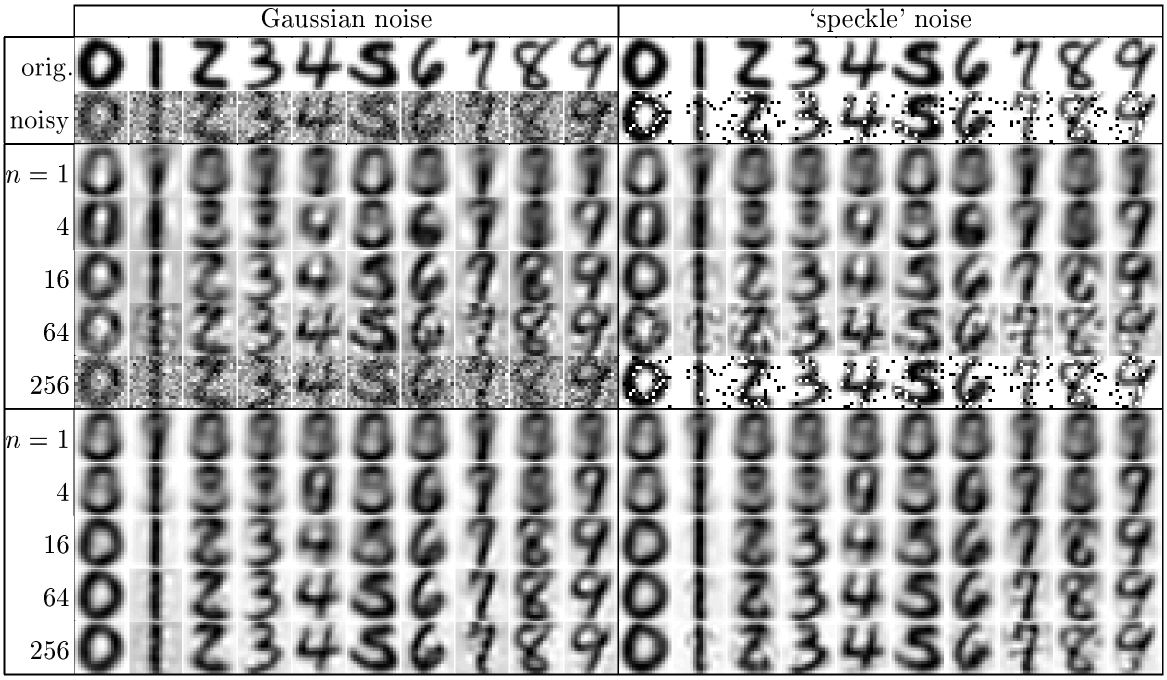


Fig. 16. De-noising of USPS data (see text). The left half shows: *top:* the first occurrence of each digit in the test set, *second row:* the upper digit with additive Gaussian noise ($\sigma = 0.5$), *following five rows:* the reconstruction for linear PCA using $k = 1, 4, 16, 64, 256$ components, and, *last five rows:* the results of the approximate pre–image approach using the same number of components. The right half shows the same but for 'speckle' noise with probability $p = 0.2$ (figure from [12]).

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. Recently, in [116] a similar approach was used together with sparse kernel PCA on real world images showing far superior performance compared to linear PCA as well.

Other applications of Kernel PCA can be found in [151] for object detection, and in [4], [119], [152] for preprocessing in regression and classification tasks.

VIII. Conclusion and Discussion

The goal of the present article was to give a simple introduction into the exciting field of kernel based learning methods. We only briefly touched learning theory and feature spaces – omitting many details of VC theory (e.g. [5]) – and instead focused on how to use and work with the algorithms. In the supervised learning part, we dealt with classification, however, a similar reasoning leads to algorithms for regression with KFD (e.g. [89]), Boosting (e.g. [108]) or SVMs (e.g. [33]).

We proposed a conceptual framework for KFD, Boosting and SVMs as algorithms that essentially differ in how they handle the high dimensionality of kernel feature spaces. One can think of boosting as a "kernel algorithm" in a space spanned by the basis hypotheses. The problem becomes only tractable since Boosting uses a ℓ_1–norm regularizer, which induces sparsity, i.e. we essentially only work in a small subspace. In SVMs and KFD, on the other hand, we use the kernel trick to *only implicitly* work in feature space. The three methods use different optimization strategies, each well suited to maximize the (average) margin in the respective feature space and to achieve sparse solutions.

The unsupervised learning part reviewed (i) kernel PCA, a nonlinear extension of PCA for finding projections that give useful nonlinear descriptors of the data and (ii) the single-class SVM algorithm that estimates the support (or, more generally, quantiles) of a data set and is an elegant approach to the outlier detection problem in high dimensions. Similar unsupervised single-class algorithms can also be constructed for Boosting [110] or KFD.

Selected real-world applications served to exemplify that kernel based learning algorithms are indeed highly competitive on a variety of problems with different characteristics.

To conclude, we would like to encourage the reader to follow the presented methodology of (re-)formulating linear, scalar product based algorithms into nonlinear algorithms to obtain further powerful kernel based learning machines.

References

[1] B.E. Boser, I.M. Guyon, and V.N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, D. Haussler, Ed., 1992, pp. 144–152.

[2] C. Cortes and V.N. Vapnik, "Support vector networks," *Machine Learning*, vol. 20, pp. 273 – 297, 1995.

[3] V.N. Vapnik, *The nature of statistical learning theory*, Springer Verlag, New York, 1995.

[4] B. Schölkopf, *Support vector learning*, Oldenbourg Verlag, Munich, 1997.

[5] V.N. Vapnik, *Statistical Learning Theory*, Wiley, New York, 1998.

[6] B. Schölkopf, C.J.C. Burges, and A.J. Smola, *Advances in*

Kernel Methods — Support Vector Learning, MIT Press, Cambridge, MA, 1999.

[7] S. Mika, G. Rätsch, J. Weston, B. Schölkopf, and K.-R. Müller, "Fisher discriminant analysis with kernels," in *Neural Networks for Signal Processing IX*, Y.-H. Hu, J. Larsen, E. Wilson, and S. Douglas, Eds. 1999, pp. 41–48, IEEE.

[8] S. Mika, G. Rätsch, J. Weston, B. Schölkopf, A.J. Smola, and K.-R. Müller, "Invariant feature extraction and classification in kernel spaces," in *Advances in Neural Information Processing Systems 12*, S.A. Solla, T.K. Leen, and K.-R. Müller, Eds. 2000, pp. 526–532, MIT Press.

[9] V. Roth and V. Steinhage, "Nonlinear discriminant analysis using kernel functions," in *Advances in Neural Information Processing Systems 12*, S.A. Solla, T.K. Leen, and K.-R. Müller, Eds. 2000, pp. 568–574, MIT Press.

[10] G. Baudat and F. Anouar, "Generalized discriminant analysis using a kernel approach," *Neural Computation*, vol. 12, no. 10, pp. 2385–2404, 2000.

[11] B. Schölkopf, A.J. Smola, and K.-R. Müller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural Computation*, vol. 10, pp. 1299–1319, 1998.

[12] S. Mika, B. Schölkopf, A.J. 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*, M.S. Kearns, S.A. Solla, and D.A. Cohn, Eds. 1999, pp. 536–542, MIT Press.

[13] B. Schölkopf, S. Mika, C.J.C. Burges, P. Knirsch, K.-R. Müller, G. Rätsch, and A.J. Smola, "Input space vs. feature space in kernel-based methods," *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 1000–1017, September 1999.

[14] B. Schölkopf, J. Platt, J. Shawe-Taylor, A.J. Smola, and R.C. Williamson, "Estimating the support of a high-dimensional distribution," TR 87, Microsoft Research, Redmond, WA, 1999, To appear in *Neural Computation.*

[15] S. Mukherjee, E. Osuna, and F. Girosi, "Nonlinear prediction of chaotic time series using a support vector machine," in *Neural Networks for Signal Processing VII — Proceedings of the 1997 IEEE Workshop*, J. Principe, L. Gile, N. Morgan, and E. Wilson, Eds., New York, 1997, pp. 511 – 520, IEEE.

[16] Y.A. LeCun, L.D. Jackel, L. Bottou, A. Brunot, C. Cortes, J.S. Denker, H. Drucker, I. Guyon, U.A. Müller, E. Säckinger, P.Y. Simard, and V.N. Vapnik, "Learning algorithms for classification: A comparism on handwritten digit recognition," *Neural Networks*, pp. 261–276, 1995.

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

[18] D. DeCoste and B. Schölkopf, "Training invariant support vector machines," *Machine Learning*, 2001, to appear. Also: Technical report JPL-MLTR-00-1, Jet Propulsion Laboratory, Pasadena.

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

[20] D. Roobaert and M.M. Van Hulle, "View-based 3d object recognition with support vector machines," in *Proc. IEEE Neural Networks for Signal Processing Workshop 1999*, 1999.

[21] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in *Proceedings of the European Conference on Machine Learning*, Berlin, 1998, pp. 137 – 142, Springer.

[22] S. Dumais, J. Platt, D. Heckerman, and M. Sahami, "Inductive learning algorithms and representations for text categorization," in *7th International Conference on Information and Knowledge Management, 1998*, 1998.

[23] H. Drucker, D. Wu, and V.N. Vapnik, "Support vector machines for span categorization," *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 1048–1054, 1999.

[24] K.-R. Müller, A.J. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V.N. Vapnik, "Predicting time series with support vector machines," in *Artificial Neural Networks — ICANN'97*, W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud, Eds., Berlin, 1997, pp. 999 – 1004, Springer Lecture Notes in Computer Science, Vol. 1327.

[25] D. Mattera and S. Haykin, "Support vector machines for dynamic reconstruction of a chaotic system," in *Advances in Kernel Methods — Support Vector Learning*, B. Schölkopf, C.J.C. Burges, and A.J. Smola, Eds., Cambridge, MA, 1999, pp. 211–242, MIT Press.

[26] M.P.S. Brown, W.N. Grundy, D. Lin, N. Cristianini, C. Sugnet, T.S. Furey, M. Ares, and D. Haussler, "Knowledge-based analysis of microarray gene expression data using support vector machines," *Proceedings of the National Academy of Sciences*, vol. 97, no. 1, pp. 262–267, 2000.

[27] T. Furey, N. Cristianini, N. Duffy, D. Bednarski, M. Schummer, and D. Haussler, "Support vector machine classification and validation of cancer tissue samples using microarray expression data," *Bioinformatics*, 2001, to appear.

[28] A. Zien, G. Rätsch, S. Mika, B. Schölkopf, T. Lengauer, and K.-R. Müller, "Engineering support vector machine kernels that recognize translation initiation sites in DNA," *BioInformatics*, 2001, to appear.

[29] T.S. Jaakkola, M. Diekhans, and D. Haussler, "A discriminative framework for detecting remote protein homologies," Unpublished, available from http://www.cse.ucsc.edu/~research/compbio/research.html, Oct. 1999.

[30] D. Haussler, "Convolution kernels on discrete structures," Tech. Rep. UCSC-CRL-99-10, UC Santa Cruz, July 1999.

[31] C. Watkins, "Dynamic alignment kernels," in *Advances in Large Margin Classifiers*, A.J. Smola, P.L. Bartlett, B. Schölkopf, and D. Schuurmans, Eds., Cambridge, MA, 2000, pp. 39–50, MIT Press.

[32] C.J.C. Burges, "A tutorial on support vector machines for pattern recognition," *Knowledge Discovery and Data Mining*, vol. 2, no. 2, pp. 121–167, 1998.

[33] A. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and Computing*, 2001, Forthcoming.

[34] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines*, Cambridge University Press, Cambridge, UK, 2000.

[35] A.J. Smola and B. Schölkopf, "On a kernel–based method for pattern recognition, regression, approximation and operator inversion," *Algorithmica*, vol. 22, pp. 211–231, 1998.

[36] A. J. Smola, *Learning with Kernels*, Ph.D. thesis, Technische Universität Berlin, 1998.

[37] G.S. Kimeldorf and G. Wahba, "Some results on Tchebycheffian spline functions," *J. Math. Anal. Applic.*, vol. 33, pp. 82–95, 1971.

[38] A.N. Tikhonov and V.Y. Arsenin, *Solutions of Ill-posed Problems*, W.H. Winston, Washington, D.C., 1977.

[39] T. Poggio and F. Girosi, "Regularization algorithms for learning that are equivalent to multilayer networks," *Science*, vol. 247, pp. 978–982, 1990.

[40] D.D. Cox and F. O'Sullivan, "Asymptotic analysis of penalized likelihood and related estimates," *The Annals of Statistics*, vol. 18, no. 4, pp. 1676–1695, 1990.

[41] H. Akaike, "A new look at the statistical model identification," *IEEE Trans. Automat. Control*, vol. 19, no. 6, pp. 716–723, 1974.

[42] J. Moody, "The *effective* number of parameters: An analysis of generalization and regularization in non-linear learning systems," in *Advances in Neural information processings systems 4*, S. J. Hanson J. Moody and R. P. Lippman, Eds., San Mateo, CA, 1992, pp. 847–854, Morgan Kaufman.

[43] N. Murata, S. Amari, and S. Yoshizawa, "Network information criterion — determining the number of hidden units for an artificial neural network model," *IEEE Transactions on Neural Networks*, vol. 5, pp. 865–872, 1994.

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

[45] R.C. Williamson, A.J. Smola, and B. Schölkopf, "Generalization performance of regularization networks and support vector machines via entropy numbers of compact operators," NeuroCOLT Technical Report NC-TR-98-019, Royal Holloway College, University of London, UK, 1998, To appear in *IEEE Transactions on Information Theory.*

[46] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks

and the bias/variance dilemma," *Neural Computation*, vol. 4, no. 1, pp. 1–58, 1992.

[47] J. Shawe-Taylor, P.L. Bartlett, R.C. Williamson, and M. Anthony, "A framework for structural risk minimization," in *Proc. COLT*. 1996, Morgan Kaufmann.

[48] B. Schölkopf and A. J. Smola, *Learning with Kernels*, MIT Press, Cambridge, MA, 2001, Forthcoming.

[49] J. Moody and C. Darken, "Fast learning in networks of locally-tuned processing units," *Neural Computation*, vol. 1, no. 2, pp. 281–294, 1989.

[50] S. Haykin, *Neural Networks : A Comprehensive Foundation*, Macmillan, New York, 1994.

[51] C.M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, 1995.

[52] G. Orr and K.-R. Müller, Eds., *Neural Networks: Tricks of the Trade*, vol. 1524, Springer LNCS, 1998.

[53] Y. Freund and R.E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, August 1997.

[54] J. Schürmann, *Pattern Classification: a unified view of statistical and neural approaches*, Wiley, New York, 1996.

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

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

[57] F. Girosi, M. Jones, and T. Poggio, "Priors, stabilizers and basis functions: From regularization to radial, tensor and additive splines," Tech. Rep. A.I. Memo No. 1430, Massachusetts Institute of Technology, June 1993.

[58] A.J. Smola, B. Schölkopf, and K.-R. Müller, "The connection between regularization operators and support vector kernels," *Neural Networks*, vol. 11, pp. 637–649, 1998.

[59] J. Mercer, "Functions of positive and negative type and their connection with the theory of integral equations," *Philos. Trans. Roy. Soc. London*, vol. A 209, pp. 415–446, 1909.

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

[61] M. Stitson, A. Gammerman, V.N. Vapnik, V. Vovk, C. Watkins, and J. Weston, "Support vector regression with ANOVA decomposition kernels," Tech. Rep. CSD-97-22, Royal Holloway, University of London, 1997.

[62] K.P. Bennett and O.L. Mangasarian, "Robust linear programming discrimination of two linearly inseparable sets," *Optimization Methods and Software*, vol. 1, pp. 23–34, 1992.

[63] O.L. Mangasarian and D.R. Musicant, "Lagrangian support vector machines," *Journal of Machine Learning Research*, 2000, in press.

[64] V.N. Vapnik, *Estimation of Dependences Based on Empirical Data*, Springer-Verlag, Berlin, 1982.

[65] B. Schölkopf, A. Smola, R.C. Williamson, and P.L. Bartlett, "New support vector algorithms," *Neural Computation*, vol. 12, pp. 1207 – 1245, 2000.

[66] M. Opper and D. Haussler, "Generalization performance of Bayes optimal classification algorithm for learning a perceptron," *Physical Review Letters*, vol. 66, pp. 2677, 1991.

[67] J. Shawe-Taylor and R.C. Williamson, "A PAC analysis of a Bayesian estimator," Tech. Rep. NC2-TR-1997-013, Royal Holloway, University of London, 1997.

[68] T. Graepel R. Herbrich and C. Campbell, "Bayes point machines: Estimating the bayes point in kernel space," in *Proceedings of IJCAI Workshop Support Vector Machines*, 1999, pp. 23–27.

[69] T. Watkin, "Optimal learning with a neural network," *Europhysics Letters*, vol. 21, pp. 871–877, 1993.

[70] P. Rul'n, "Playing billiard in version space," *Neural Computation*, vol. 9, pp. 197–238, 1996.

[71] R. Herbrich and T. Graepel, "Large scale Bayes point machines," in *Advances in Neural Information System Processing 13*, 2001, accepted for publication.

[72] Ralf Herbrich, Thore Graepel, and Colin Campbell, "Bayesian learning in reproducing kernel Hilbert spaces," Tech. Rep., Technical University of Berlin, 1999, TR 99-11.

[73] R.J. Vanderbei, "Interior-point methods: Algorithms and formulations," *ORSA Journal on Computing*, vol. 6, no. 1, pp. 32–34, 1994.

[74] D.P. Bertsekas, *Nonlinear Programming*, Athena Scientific, Belmont, MA, 1995.

[75] C. Saunders, M.O. Stitson, J. Weston, L. Bottou, B. Schölkopf, and A.J. Smola, "Support vector machine reference manual," Tech. Rep. CSD-TR-98-03, Royal Holloway University, London, 1998.

[76] E. Osuna, R. Freund, and F. Girosi, "Support vector machines: Training and applications," A.I. Memo AIM-1602, MIT A.I. Lab, 1996.

[77] E. Osuna, R. Freund, and F. Girosi, "An improved training algorithm for support vector machines," in *Neural Networks for Signal Processing VII — Proceedings of the 1997 IEEE Workshop*, J. Principe, L. Gile, N. Morgan, and E. Wilson, Eds., New York, 1997, pp. 276 – 285, IEEE.

[78] T. Joachims, "Making large–scale SVM learning practical," in *Advances in Kernel Methods — Support Vector Learning*, B. Schölkopf, C.J.C. Burges, and A.J. Smola, Eds., Cambridge, MA, 1999, pp. 169–184, MIT Press.

[79] J. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods — Support Vector Learning*, B. Schölkopf, C.J.C. Burges, and A.J. Smola, Eds., Cambridge, MA, 1999, pp. 185–208, MIT Press.

[80] S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, and K.R.K. Murthy, "Improvements to Platt's SMO algorithm for SVM classifier design," Tech. Rep. CD-99-14, National University of Singapore, 1999, `http://guppy.mpe.nus.edu.sg/~mpessk`.

[81] T.-T. Frieß, N. Cristianini, and C. Campbell., "The kernel adatron algorithm: A fast and simple learning procedure for support vector machines.," in *Proc. ICML'98*, J. Shavlik, Ed. 1998, pp. 188–196, Morgan Kaufmann Publishers.

[82] J.K. Anlauf and M. Biehl, "The adatron: an adaptive perceptron algorithm," *Europhys. Letters*, vol. 10, pp. 687 – 692, 1989.

[83] P.S. Bradley, U.M Fayyad, and O.L Mangasarian, "Mathematical programming for data mining: Formulations and challenges," *Journal of Computing*, 1998.

[84] "`http://www.kernel-machines.org`," A collection of literature, software and web pointers dealing with SVM and Gaussian processes.

[85] R.A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of Eugenics*, vol. 7, pp. 179–188, 1936.

[86] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, San Diego, 2nd edition, 1990.

[87] J.H. Friedman, "Regularized discriminant analysis," *Journal of the American Statistical Association*, vol. 84, no. 405, pp. 165–175, 1989.

[88] T.J. Hastie, A. Buja, and R.J. Tibshirani, "Penalized discriminant analysis," *Annals of Statistics*, 1995.

[89] S. Mika, G. Rätsch, and K.-R. Müller, "A mathematical programming approach to the Kernel Fisher algorithm," in *Advances in Neural Information Processing Systems 13*, 2001, to appear.

[90] S. Mika, A.J. Smola, and B. Schölkopf, "An improved training algorithm for kernel fisher discriminants," in *Proceedings AISTATS 2001*. 2001, Morgan Kaufmann, to appear.

[91] M.E. Tipping, "The relevance vector machine," in *Advances in Neural Information Processing Systems 12*, S.A. Solla, T.K. Leen, and K.-R. Müller, Eds. 2000, pp. 652–658, MIT Press.

[92] K.P. Bennett, A. Demiriz, and J. Shawe-Taylor, "A column generation algorithm for boosting," in *Prooceedings, 17th ICML*, P. Langley, Ed., San Francisco, 2000, pp. 65–72, Morgan Kaufmann.

[93] R.E. Schapire, "The strength of weak learnability," *Machine Learning*, vol. 5, no. 2, pp. 197–227, 1990.

[94] R.E. Schapire, Y. Freund, P.L. Bartlett, and W.S. Lee, "Boosting the margin: a new explanation for the effectiveness of voting methods," in *Proc. 14th International Conference on Machine Learning*. 1997, pp. 322–330, Morgan Kaufmann.

[95] R.E. Schapire, "A brief introduction to boosting," in *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1999.

[96] N. Duffy and D.P. Helmbold, "Potential boosters?," in *Advances in Neural Information Processing Systems 12*, S.A. Solla, T.K. Leen, and K.-R. Müller, Eds. 2000, pp. 258–264, MIT Press.

[97] "`http://www.boosting.org`," A collection of references, soft-

ware and web pointers concerned with Boosting and ensemble learning methods.

[98] R.E. Schapire, *The Desig and Analysis of Efficient Learning Algorithms*, Ph.D. thesis, MIT Press, 1992.

[99] M. Kearns and L. Valiant, "Cryptographic limitations on learning Boolean formulae and finite automata," *Journal of the ACM*, vol. 41, no. 1, pp. 67–95, Jan. 1994.

[100] L.G. Valiant, "A theory of the learnable," *Communications of the ACM*, vol. 27, no. 11, pp. 1134–1142, Nov. 1984.

[101] L. Breiman, "Arcing the edge," Technical Report 486, Statistics Department, University of California, June 1997.

[102] G. Rätsch, T. Onoda, and K.-R. Müller, "Soft margins for AdaBoost," *Machine Learning*, vol. 42, no. 3, pp. 287–320, Mar. 2001, also NeuroCOLT Technical Report NC-TR-1998-021.

[103] N. Duffy and D.P. Helmbold, "A geometric approach to leveraging weak learners," in *Computational Learning Theory: 4th European Conference (EuroCOLT '99)*, P. Fischer and H. U. Simon, Eds., Mar. 1999, pp. 18–33, Long version to appear in TCS.

[104] L. Mason, J. Baxter, P.L. Bartlett, and M. Frean, "Functional gradient techniques for combining hypotheses," in *Advances in Large Margin Classifiers*, A.J. Smola, P.L. Bartlett, B. Schölkopf, and D. Schuurmans, Eds., pp. 221–247. MIT Press, Cambridge, MA, 2000.

[105] L. Breiman, "Prediction games and arcing algorithms," Technical Report 504, Statistics Department, University of California, December 1997.

[106] G. Rätsch, B. Schölkopf, A.J. Smola, S. Mika, T. Onoda, and K.-R. Müller, "Robust ensemble learning," in *Advances in Large Margin Classifiers*, A.J. Smola, P.L. Bartlett, B. Schölkopf, and D. Schuurmans, Eds., pp. 207–219. MIT Press, Cambridge, MA, 2000.

[107] O.L. Mangasarian, "Arbitrary-norm separating plane," *Operation Research Letters*, vol. 24, no. 1, pp. 15–23, 1999.

[108] G. Rätsch, M. Warmuth, S. Mika, T. Onoda, S. Lemm, and K.-R. Müller, "Barrier boosting," in *Proc. COLT*, Stanford, Feb. 2000, pp. 170–179, Morgan Kaufmann.

[109] G. Rätsch, A. Demiriz, and K. Bennett, "Sparse regression ensembles in infinite and finite hypothesis spaces," NeuroCOLT2 Technical Report 85, Royal Holloway College, London, September 2000, Machine Learning, to appear.

[110] G. Rätsch, B. Schölkopf, S. Mika, and K.-R. Müller, "SVM and Boosting: One class," Tech. Rep. 119, GMD FIRST, Berlin, November 2000.

[111] R.O. Duda and P.E. Hart, *Pattern classification and scene analysis*, John Wiley & Sons, 1973.

[112] B. Schölkopf, R.C. Williamson, A.J. Smola, J. Shawe-Taylor, and J.C. Platt, "Support vector method for novelty detection," in *Advances in Neural Information Processing Systems 12*, S.A. Solla, T.K. Leen, and K.-R. Müller, Eds. 2000, pp. 582–588, MIT Press.

[113] D. Tax and R. Duin, "Data domain description by support vectors," in *Proc. ESANN*, M. Verleysen, Ed., Brussels, 1999, pp. 251–256, D. Facto Press.

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

[115] A.J. Smola and B. Schölkopf, "Sparse greedy matrix approximation for machine learning," in *Proc. ICML'00*, P. Langley, Ed., San Francisco, 2000, pp. 911–918, Morgan Kaufmann.

[116] M. Tipping, "Sparse kernel principal component analysis," in *Advances in Neural Information Processing Systems 13*. 2001, MIT-Press, to appear.

[117] A.J. Smola, O.L. Mangasarian, and B. Schölkopf, "Sparse kernel feature analysis," Tech. Rep. 99-04, University of Wisconsin, Data Mining Institute, Madison, 1999.

[118] B. Schölkopf, K.-R. Müller, and A.J. Smola, "Lernen mit Kernen," *Informatik Forschung und Entwicklung*, vol. 14, pp. 154 – 163, 1999.

[119] R. Rosipal, M. Girolami, and L. Trejo, "Kernel PCA feature extraction of event–related potentials for human signal detection performance," in *Proceedings of Intl. Conf. on Artificial Neural Networks in Medicine and Biology*, Malmgren, Borga, and Niklasson, Eds., 2000, pp. 321–326.

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

[121] J. Kwok, "Integrating the evidence framework and the support vector machine," in *Proc. ESANN'99*, M. Verleysen, Ed., Brussels, 1999, pp. 177–182.

[122] B. Schölkopf, J. Shawe-Taylor, and A.J. Smola R.C. Williamson, "Kernel dependent support vector error bounds," in *Proceedings of ICANN'99*, D. Willshaw and A. Murray, Eds. 1999, vol. 1, pp. 103–108, IEE Press.

[123] K. Tsuda, "Optimal hyperplane classifier based on entropy number bound," in *Proceedings of ICANN'99*, D. Willshaw and A. Murray, Eds. 1999, vol. 1, pp. 419–424, IEE Press.

[124] J.K. Martin and D.S. Hirschberg, "Small sample statistics for classification error rates I: Error rate measurements," Tech. Rep. 96-21, Department of Information and Computer Science, UC Irvine, 1996.

[125] B. Efron and R.J. Tibshirani, *An Introduction to the Bootstrap*, Chapman and Hall, New York, 1994.

[126] B. Efron and R.J. Tibshirani, "Improvements on cross-validation: the .632+ bootstrap method," *J. Amer. Statist. Assoc*, vol. 92, pp. 548–560, 1997.

[127] V.N. Vapnik and O. Chapelle, "Bounds on error expectation for support vector machines," *Neural Computation*, vol. 12, no. 9, Sept. 2000.

[128] G. Rätsch, "Ensemble learning methods for classification," M.S. thesis, Dep. of Computer Science, University of Potsdam, Apr. 1998, In German.

[129] J. Weston, "LOO-Support Vector Machines," in *Proceedings of IJCNN'99*, A.J. Smola, P.L. Bartlett, B. Schölkopf, and D. Schuurmans, Eds., 1999.

[130] J. Weston and R. Herbrich, "Adaptive margin support vector machines," in *Advances in Large Margin Classifiers*, A.J. Smola, P.L. Bartlett, B. Schölkopf, and D. Schuurmans, Eds., Cambridge, MA, 2000, pp. 281–296, MIT Press.

[131] B. Schölkopf, P.Y. Simard, A.J. Smola, and V.N. Vapnik, "Prior knowledge in support vector kernels," in *Advances in Neural Information Processing Systems 10*, M. Jordan, M. Kearns, and S. Solla, Eds., Cambridge, MA, 1998, pp. 640–646, MIT Press.

[132] Y.A. 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*, vol. 1, pp. 541 – 551, 1989.

[133] L. Bottou, C. Cortes, J.S. Denker, H. Drucker, I. Guyon, L.D. Jackel, Y.A. LeCun, U.A. Müller, E. Säckinger, P.Y. Simard, and V.N. Vapnik, "Comparison of classifier methods: a case study in handwritten digit recognition," in *Proceedings of the 12th International Conference on Pattern Recognition and Neural Networks, Jerusalem*. 1994, pp. 77 – 87, IEEE Computer Society Press.

[134] Y.A. LeCun, L.D. Jackel, L. Bottou, A. Brunot, C. Cortes, J.S. Denker, H. Drucker, I. Guyon, U.A. Müller, E. Säckinger, P.Y. Simard, and V.N. Vapnik, "Comparison of learning algorithms for handwritten digit recognition," in *Proceedings ICANN'95 — International Conference on Artificial Neural Networks*, F. Fogelman-Soulié and P. Gallinari, Eds., Nanterre, France, 1995, vol. II, pp. 53 – 60, EC2.

[135] P.Y. Simard, Y.A. LeCun, and J.S. Denker, "Efficient pattern recognition using a new transformation distance," in *Advances in Neural Information Processing Systems 5*, S. J. Hanson, J. D. Cowan, and C. L. Giles, Eds., San Mateo, CA, 1993, pp. 50–58, Morgan Kaufmann.

[136] P.Y. Simard, Y.A. LeCun, J.S. Denker, and B. Victorri, "Transformation invariance in pattern recognition – tangent distance and tangent propagation," in *Neural Networks: Tricks of the Trade*, G. Orr and K.-R. Müller, Eds. 1998, vol. 1524, pp. 239–274, Springer LNCS.

[137] H. Drucker, R. Schapire, and P.Y. Simard, "Boosting performance in neural networks," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 7, pp. 705 – 719, 1993.

[138] W.R. Pearson, T. Wood, Z. Zhang, and W. Miller, "Comparison of DNA Sequences with Protein Sequences," *Genomics*, vol. 46, no. 1, pp. 24–36, Nov. 1997.

[139] C. Iseli, C.V. Jongeneel, and P. Bucher, "ESTScan: a program for detecting, evaluating, and reconstructing potential coding regions in EST sequences," in *ISMB'99*, T. Lengauer, R. Schneider, P. Bork, D. Brutlag, J. Glasgow, H.-W. Mewes,

and R. Zimmer, Eds., Menlo Park, California 94025, Aug. 1999, pp. 138–148, AAAI Press.

[140] A.G. Pedersen and H. Nielsen, "Neural Network Prediction of Translation Initiation Sites in Eukaryotes: Perspectives for EST and Genome analysis," in *ISMB'97*, 1997, vol. 5, pp. 226–233.

[141] S.L. Salzberg, "A method for identifying splice sites and translational start sites in eukaryotic mRNA," *Computational Applied Bioscience*, vol. 13, no. 4, pp. 365–376, 1997.

[142] University of California Irvine, "`http://www.ics.uci.edu/~mlearn`," UCI-Benchmark repository – a huge collection of artificial and real-world data sets.

[143] University of Toronto, "`http://www.cs.utoronto.ca/~delve/data/datasets.html`," DELVE-Benchmark repository – a collection of artificial and real-world data sets.

[144] "`ftp://ftp.ncc.up.pt/pub/statlog`," Benchmark repository used for the STATLOG competition.

[145] "`http://ida.first.gmd.de/~raetsch/data/benchmarks.htm`," IDA Benchmark repository used in several Boosting, KFD and SVM papers.

[146] S. Dumais, "Using SVMs for text categorization," *IEEE Intelligent Systems*, vol. 13(4), 1998, In: M.A. Hearst, B. Schölkopf, S. Dumais, E. Osuna, and J. Platt: Trends and Controversies — Support Vector Machines.

[147] E. Osuna, R. Freund, and F. Girosi, "Training support vector machines: An application to face detection," in *Proceedings CVPR'97*, 1997.

[148] B. Bradshaw, B. Schölkopf, and J. Platt, "Kernel methods for extracting local image semantics," unpublished manuscript, private communication, 2000.

[149] J. Weston, A. Gammerman, M. Stitson, V.N. Vapnik, V. Vovk, and C. Watkins, "Support vector density estimation," in *Advances in Kernel Methods — Support Vector Learning*, B. Schölkopf, C.J.C. Burges, and A.J. Smola, Eds., pp. 293 – 305. MIT Press, Cambridge, MA, 1999.

[150] C.J.C. Burges, "Simplified support vector decision rules," in *Proc. ICML'96*, L. Saitta, Ed., San Mateo, CA, 1996, pp. 71–77, Morgan Kaufmann.

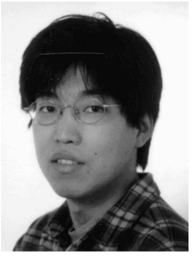
[151] S. Romdhani, S. Gong, and A. Psarrou, "A multiview nonlinear active shape model using kernel PCA," in *Proceedings of BMVC*, Nottingham, UK, 1999, pp. 483–492.

[152] R. Rosipal, M. Girolami, and L. Trejo, "Kernel PCA for feature extraction and de–noising in non–linear regression," submitted, see `http://www.researchindex.com`, Jan. 2000.

Klaus-Robert Müller received the Diplom degree in mathematical physics 1989 and the Ph.D. in theoretical computer science in 1992, both from University of Karlsruhe, Germany. From 1992 to 1994 he worked as a Postdoctoral fellow at GMD FIRST, in Berlin where he started to built up the intelligent data analysis (IDA) group. From 1994 to 1995 he was a European Community STP Research Fellow at University of Tokyo in Prof. Amari's Lab. From 1995 on he is department head of the IDA group at GMD FIRST in Berlin and since 1999 he holds a joint associate Professor position of GMD and University of Potsdam. He has been lecturing at Humboldt University, Technical University Berlin and University of Potsdam. In 1999 he received the annual national prize for pattern recognition (Olympus Prize) awarded by the German pattern recognition society DAGM. He serves in the editorial board of Computational Statistics, IEEE Transactions on Biomedical Engineering and in program and organization committees of various international conferences. His research areas include statistical physics and statistical learning theory for neural networks, support vector machines and ensemble learning techniques. His present interests are expanded to time-series analysis, blind source separation techniques and to statistical denoising methods for the analysis of biomedical data.

Sebastian Mika is a doctoral student at GMD FIRST (IDA), Berlin. He received the Diplom in computer science from the Technical University of Berlin in 1998. His scientific interests are in the fields of Machine Learning and Kernel methods.

Gunnar Rätsch is a doctoral student at GMD FIRST (IDA), Berlin. He received the Diplom in computer science from the University of Potsdam (Germany) in 1998, along with the prize for the best student of the faculty of Natural Sciences. His scientific interests are in the fields of Boosting and Kernel methods.



Koji Tsuda is a researcher at Electrotechnical Laboratory, Tsukuba, Japan. From 2000 to 2001 he was a visiting researcher at GMD FIRST (IDA), Berlin. He received a Doctor of Engineering in information science from Kyoto University in 1998. His scientific interests are in the fields of Machine learning and Kernel methods.

Bernhard Schölkopf received an M.Sc. in mathematics and the Lionel Cooper Memorial Prize from the University of London in 1992, followed in 1994 by the Diplom in physics from the Eberhard-Karls-Universität, Tübingen, with a thesis written at the Max-Planck-Institute for biological cybernetics. Three years later, he obtained a Ph.D. in computer science from the Technical University Berlin. His thesis on Support Vector Learning won the annual dissertation prize of the German Association for Computer Science (GI) 1997. He has worked at AT&T Bell Labs (with V. Vapnik), at GMD FIRST (IDA), Berlin, at the Australian National University and Microsoft Research Cambridge (UK); currently he is with Barnhill Technologies. He has taught at the Humboldt University and the Technical University Berlin. His scientific interests include machine learning and perception.