Support Vector Machines and Kernel Algorithms

Bernhard Schölkopf
Max-Planck-Institut für biologische Kybernetik
72076 Tübingen, Germany
`Bernhard.Schoelkopf@tuebingen.mpg.de`

Alex Smola
RSISE, Australian National University
Canberra 0200 ACT, Australia
`Alex.Smola@anu.edu.au`

March 20, 2002

Contact Author:

Alexander J. Smola
Tel: (+61) 2 6125-8652
Fax: (+61) 2 6125-8651
Cel: (+61) 410 457 686
Alex.Smola@anu.edu.au

Articles authored or co-authored in 2e:

Support Vector Machines and Kernel Algorithms
Bernhard Schölkopf and Alexander J. Smola

INTRODUCTION

One of the fundamental problems of learning theory is the following: suppose we are given two classes of objects. We are then faced with a new object, and we have to assign it to one of the two classes. This problem, referred to as *(binary) pattern recognition*, can be formalized as follows: we are given empirical data

$$(x_1, y_1), \ldots, (x_m, y_m) \in \mathcal{X} \times \{\pm 1\}, \tag{1}$$

and we want to estimate a *decision function* $f : \mathcal{X} \to \{\pm 1\}$. Here, $\mathcal{X}$ is some nonempty set from which the *patterns* x_i are taken, usually referred to as the *domain*; the y_i are called *labels* or *targets*.

A good decision function will have the property that it *generalizes* to unseen data points, achieving a small value of the *risk*

$$R[f] = \int \frac{1}{2}|f(x) - y| \; d\mathrm{P}(x, y). \tag{2}$$

In other words, on average over an unknown distribution P which is assumed to generate both training and test data, we would like to have a small error. Here, the error is measured by means of the *zero-one loss function* $c(x, y, f(x)) := \frac{1}{2}|f(x) - y|$. The loss is 0 if (x, y) is classified correctly, and 1 otherwise.

It should be emphasized that so far, the patterns could be just about anything, and we have made no assumptions on $\mathcal{X}$ other than it being a set endowed with a probability measure P (note that the labels y may, but need not depend on x in a deterministic fashion). Moreover, (2) does not tell us how to *find* a function with a small risk. In fact, it does not even tell us how to *evaluate* the risk of a given function, since the probability measure P is assumed to be unknown.

We therefore introduce an additional type of structure, pertaining to what we are actually given — the training data. Loosely speaking, to generalize, we want to choose y such that (x, y) is in some sense similar to the training examples (1). To this end, we need notions of *similarity* in $\mathcal{X}$ and in $\{\pm 1\}$.

Characterizing the similarity of the outputs $\{\pm 1\}$ is easy: in binary classification, only two situations can occur: two labels can either be identical or different. The choice of the similarity measure for the inputs, on the other hand, is a deep question that lies at the core of the problem of machine learning.

One of the advantages of kernel methods is that the learning algorithms developed are quite independent of the choice of the similarity measure. This allows us to adapt the latter to the specific problems at hand without the need to reformulate the learning algorithm itself.

KERNELS AS SIMILARITY MEASURES

Let us consider a symmetric similarity measure of the form

$$\begin{aligned} k : \mathcal{X} \times \mathcal{X} &\rightarrow \mathbb{R} \\ (x, x') &\mapsto k(x, x'), \end{aligned} \tag{3}$$

that is, a function that, given two patterns x and x', returns a real number characterizing their similarity. The function k is often called a *kernel*.

General similarity measures of this form are rather difficult to study. Let us therefore start from a particularly simple case, and generalize it subsequently. A simple type of similarity measure that is of particular mathematical appeal is a *dot product*. For instance, given two vectors $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^N$, the *canonical dot product* is defined as

$$\langle \mathbf{x}, \mathbf{x}' \rangle := \sum_{i=1}^{N} [\mathbf{x}]_i [\mathbf{x}']_i. \tag{4}$$

Here, $[\mathbf{x}]_i$ denotes the ith entry of $\mathbf{x}$.

The geometric interpretation of the canonical dot product is that it computes the cosine of the angle between the vectors $\mathbf{x}$ and $\mathbf{x}'$, provided they are normalized to length 1. Moreover, it allows computation of the *length* (or *norm*) of a vector $\mathbf{x}$ as

$$\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}. \tag{5}$$

Being able to compute dot products amounts to being able to carry out all geometric constructions that can be formulated in terms of angles, lengths and distances.

Note, however, that the dot product approach is not really sufficiently general to deal with many interesting problems.

- First, we have deliberately not made the assumption that the patterns actually exist in a dot product space. So far, they could be any kind of object. In order to be able to use a dot product as a similarity measure, we therefore first need to represent the patterns as vectors in some dot product space $\mathcal{H}$, called the *feature space*. To this end, we use a map

$$\begin{aligned} \Phi : \mathcal{X} &\rightarrow \mathcal{H} \\ x &\mapsto \mathbf{x} := \Phi(x). \end{aligned} \tag{6}$$

  Note that we use a boldface $\mathbf{x}$ to denote the vectorial representation of x in the feature space.

- Second, even if the original patterns lie in a dot product space, we may still want to consider more general similarity measures obtained by applying a nonlinear map (6). An example that we will consider below is a map which computes products of entries of the input patterns.

Embedding the data into $\mathcal{H}$ via Φ has two main benefits. First, it allows us to deal with the patterns geometrically, and thus lets us study learning algorithms using linear algebra and analytic geometry. Second, it lets us define a similarity measure from the dot product in $\mathcal{H}$,

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

The freedom to choose the mapping Φ enables us to design a large variety of similarity measures and learning algorithms.

A SIMPLE PATTERN RECOGNITION ALGORITHM

We now design a simple pattern recognizer. We make use of the structure introduced in the previous section; that is, we assume that our data are embedded into a dot product space $\mathcal{H}$. The basic idea of the algorithm is to assign a previously unseen pattern to the class with closer mean. We thus begin by computing the means of the two classes in feature space;

$$\mathbf{c}_+ = \frac{1}{m_+} \sum_{\{i|y_i=+1\}} \mathbf{x}_i, \tag{8}$$

$$\mathbf{c}_- = \frac{1}{m_-} \sum_{\{i|y_i=-1\}} \mathbf{x}_i, \tag{9}$$

where m_+ and m_- are the number of examples with positive and negative labels, respectively, with $m_+, m_- > 0$. We assign a new point $\mathbf{x}$ to the class whose mean is closest (Figure 1). This geometric construction can be formulated in terms of the dot product $\langle \cdot, \cdot \rangle$, leading to

$$\begin{aligned} y &= \operatorname{sgn} \langle (\mathbf{x} - \mathbf{c}), \mathbf{w} \rangle \\ &= \operatorname{sgn} \langle (\mathbf{x} - (\mathbf{c}_+ + \mathbf{c}_-)/2), (\mathbf{c}_+ - \mathbf{c}_-) \rangle \\ &= \operatorname{sgn} (\langle \mathbf{x}, \mathbf{c}_+ \rangle - \langle \mathbf{x}, \mathbf{c}_- \rangle + b). \end{aligned} \tag{10}$$

Here, we have defined the offset

$$b := \frac{1}{2}(\|\mathbf{c}_-\|^2 - \|\mathbf{c}_+\|^2), \tag{11}$$

which vanishes if the class means have the same distance to the origin.

It is instructive to rewrite (10) in terms of the input patterns x_i, using the kernel k to compute the dot products. To this end, substitute (8) and (9) into (10) to get the *decision function*

$$\begin{aligned} y &= \operatorname{sgn}\left(\frac{1}{m_+}\sum_{\{i|y_i=+1\}}\langle\mathbf{x},\mathbf{x}_i\rangle - \frac{1}{m_-}\sum_{\{i|y_i=-1\}}\langle\mathbf{x},\mathbf{x}_i\rangle + b\right) \\ &= \operatorname{sgn}\left(\frac{1}{m_+}\sum_{\{i|y_i=+1\}}k(x,x_i) - \frac{1}{m_-}\sum_{\{i|y_i=-1\}}k(x,x_i) + b\right). \end{aligned} \tag{12}$$

Similarly, the offset can be expressed in terms of the kernel. Surprisingly, it turns out that this rather simple-minded approach contains a well-known statistical classification method as a special case. Assume that the class means have the same distance to the origin (hence $b = 0$, cf. (11)), and that k can be viewed as a probability density when one of its arguments is fixed.

In this case, (12) takes the form of the so-called Bayes classifier separating the two classes, subject to the assumption that the two classes of patterns were generated by sampling from two probability distributions that are correctly estimated by the *Parzen windows* estimators of the two class densities,

$$p_+(x) := \frac{1}{m_+}\sum_{\{i|y_i=+1\}}k(x,x_i) \text{ and } p_-(x) := \frac{1}{m_-}\sum_{\{i|y_i=-1\}}k(x,x_i), \tag{13}$$

where $x \in \mathcal{X}$.

The classifier (12) is quite close to Support Vector Machines (SVMs). In both cases, the decision function is a kernel expansion corresponding to a separating hyperplane in a feature space.

SVMs deviate from (12) in the selection of the patterns on which the kernels are centered and in the choice of weights that are placed on the individual kernels in the decision function. It will no longer be the case that *all* training patterns appear in the kernel expansion, and the weights of the kernels in the expansion will no longer be uniform within the classes.

EXAMPLES OF KERNELS

So far, we have used the kernel notation as an abstract similarity measure. We now give some concrete examples of kernels, mainly for the case where the inputs x_i are already taken from a dot product space. The role of the kernel then is to implicitly change the representation of the data into another (usually higher dimensional) feature space. One of the most common kernels used is the polynomial one,

$$k(x,x') = \langle x,x'\rangle^d, \text{ where } d \in \mathbb{N}. \tag{14}$$

It can be shown to correspond to a feature space spanned by *all* products of order d of input variables, i.e., all products of the form $[x]_{i_1} \cdot \ldots \cdot [x]_{i_d}$. The dimension of this space is of the order N^d, but using the kernel to evaluate dot products, this does not affect us. Another popular choice is the Gaussian

$$k(x,x') = \exp\left(-\frac{\|x-x'\|^2}{2\,\sigma^2}\right), \tag{15}$$

with a suitable width $\sigma > 0$. Likewise the function

$$k(x,x') = \mathbf{E}[f(x)f(x')] = \int f(x)f(x')d\mathrm{P}(x,x') \tag{16}$$

is a kernel. Such functions are typically used in Gaussian Process estimation. See Williams (1998) for further details. Examples of more sophisticated kernels, defined not on dot product spaces but on discrete objects such as strings, are the string matching kernels proposed by Watkins (2000) and Haussler (1999).

In general, there are several ways of deciding whether a given function k qualifies as a valid kernel. One way is to appeal to *Mercer's theorem*. This classical result of functional analysis states that the kernel of a positive definite integral operator can be diagonalized in terms of an eigenvector expansion with nonnegative eigenvalues. From the expansion, the feature map Φ can explicitly be constructed. Another approach exploits the fact that the class of admissible kernels coincides with the class of *positive definite* kernels, i.e., functions k such that $\sum_{ij} a_i a_j k(x_i, x_j) \geq 0$ for all a_i, x_i $(i = 1, \ldots, m)$. For positive definite kernels, the feature space can be constructed as the associated *reproducing kernel Hilbert space*.

HYPERPLANE CLASSIFIERS

STATISTICAL LEARNING THEORY, or VC (Vapnik-Chervonenkis) theory, shows that it is imperative to restrict the set of functions from which f is chosen to one that has a *capacity* suitable for the amount of available training data. VC theory provides *bounds* on the test error, depending on both the empirical risk and the capacity of the function class. The minimization of these bounds leads to the principle of *structural risk minimization* (Vapnik, 1995).

SVMs can be considered an approximate implementation of this principle, by trying to minimize a combination of the *training error* (or *empirical risk*),

$$R_{\mathrm{emp}}[f] = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}|f(x_i) - y_i|, \tag{17}$$

and a capacity term derived for the class of hyperplanes in a dot product space $\mathcal{H}$ (Vapnik, 1995),

$$\langle \mathbf{w}, \mathbf{x} \rangle + b = 0 \text{ where } \mathbf{w} \in \mathcal{H}, b \in \mathbb{R}, \tag{18}$$

corresponding to decision functions

$$f(\mathbf{x}) = \mathrm{sgn}\,(\langle \mathbf{w}, \mathbf{x} \rangle + b). \tag{19}$$

Consider first problems which are separable by hyperplanes. Among all hyperplanes separating the data, there exists a unique *optimal hyperplane* (Vapnik, 1995), distinguished by the maximum margin of separation between any training point and the hyperplane. It is the solution of

$$\underset{\mathbf{w}\in\mathcal{H},b\in\mathbb{R}}{\text{maximize}} \ \min\{\|\mathbf{x}-\mathbf{x}_i\| \,|\, \mathbf{x}\in\mathcal{H}, \langle\mathbf{w},\mathbf{x}\rangle+b=0, i=1,\ldots,m\}. \tag{20}$$

Moreover, the capacity of the class of separating hyperplanes can be shown to decrease with increasing margin. The latter is the basis of the *statistical* justification of the approach; in addition, it is *computationally* attractive, since we will show below that it can be constructed by solving a quadratic programming problem for which efficient algorithms exist.

Note that although the form of the decision function (19) is similar to our earlier example (10), the training is different. In the earlier example, the normal vector of the hyperplane was trivially computed from the class means as $\mathbf{w}=\mathbf{c}_+-\mathbf{c}_-$. In the present case, we need to do some additional work. To construct the optimal hyperplane, we have to solve

$$\underset{\mathbf{w}\in\mathcal{H},b\in\mathbb{R}}{\text{minimize}} \qquad \tau(\mathbf{w})=\frac{1}{2}\|\mathbf{w}\|^2 \tag{21}$$

$$\text{subject to} \qquad y_i(\langle\mathbf{w},\mathbf{x}_i\rangle+b)\geq 1 \text{ for all } i=1,\ldots,m. \tag{22}$$

Note that the constraints (22) ensure that $f(\mathbf{x}_i)$ will be $+1$ for $y_i=+1$, and -1 for $y_i=-1$. Now one might argue that for this to be the case, we don't actually need the "≥ 1" on the right hand side of (22). However, without it, it would not be meaningful to minimize the length of $\mathbf{w}$: to see this, imagine we wrote "> 0" instead of "≥ 1." Now assume that the solution is $(\mathbf{w},b)$. Let us rescale this solution by multiplication with some $0<\lambda<1$. Since $\lambda>0$, the constraints are still satisfied. Since $\lambda<1$, however, the length of $\mathbf{w}$ has decreased. Hence $(\mathbf{w},b)$ cannot be the minimizer of $\tau(\mathbf{w})$.

Let us now try to get an intuition for why we should be minimizing the length of $\mathbf{w}$, as in (21). If $\|\mathbf{w}\|$ were 1, then the left hand side of (22) would equal the distance from $\mathbf{x}_i$ to the hyperplane (cf. (20)). In general, we have to divide $y_i(\langle\mathbf{w},\mathbf{x}_i\rangle+b)$ by $\|\mathbf{w}\|$ to transform it into this distance. Hence, if we can satisfy (22) for all $i=1,\ldots,m$ with an $\mathbf{w}$ of minimal length, then the overall margin will be maximized (see Figure 2).

The constrained optimization problem (21) is dealt with by introducing *Lagrange multipliers* $\alpha_i\geq 0$ and a *Lagrangian*

$$L(\mathbf{w},b,\boldsymbol{\alpha})=\frac{1}{2}\|\mathbf{w}\|^2-\sum_{i=1}^m\alpha_i\left(y_i(\langle\mathbf{x}_i,\mathbf{w}\rangle+b)-1\right). \tag{23}$$

We use boldface Greek letters as a shorthand for corresponding vectors $\boldsymbol{\alpha}=(\alpha_1,\ldots,\alpha_m)$. The Lagrangian L has a *saddle point* in $\mathbf{w},b$ and $\boldsymbol{\alpha}$ at the optimal solution of the primal optimization problem. This means that it should be minimized with respect to the *primal variables* $\mathbf{w}$ and b and maximized with respect to the *dual variables* α_i. Note that the constraints appear in the second term of the Lagrangian.

Let us try to get some intuition for this way of dealing with constrained optimization problems. If a constraint (22) is violated, then $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - 1 < 0$, in which case L can be increased by increasing the corresponding α_i. At the same time, $\mathbf{w}$ and b will have to change such that L decreases. To prevent $\alpha_i \left(y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - 1 \right)$ from becoming an arbitrarily large negative number, the change in $\mathbf{w}$ and b will ensure that, provided the problem is separable, the constraint will eventually be satisfied. Similarly, one can understand that for all constraints which are not precisely met as equalities (that is, for which $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - 1 > 0$), the corresponding α_i must be 0: this is the value of α_i that maximizes L. The latter is the statement of the Karush-Kuhn-Tucker (KKT) complementarity conditions of optimization theory.

To minimize w.r.t. the primal variables, we require

$$\frac{\partial}{\partial b} L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0 \text{ and } \frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0, \tag{24}$$

leading to

$$\sum_{i=1}^{m} \alpha_i y_i = 0 \tag{25}$$

and

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i. \tag{26}$$

The solution thus has an expansion (26) in terms of a subset of the training patterns, namely those patterns with non-zero α_i, called *Support Vectors (SVs)*. Often, only few of the training examples actually end up being SVs. By the KKT conditions,

$$\alpha_i \left[y_i \left(\langle \mathbf{x}_i, \mathbf{w} \rangle + b \right) - 1 \right] = 0 \text{ for all } i = 1, \ldots, m, \tag{27}$$

the SVs lie on the margin (cf. Figure 2) — this can be exploited to compute b once the α_i have been found. All remaining training examples $(\mathbf{x}_j, y_j)$ are irrelevant: their constraint $y_j(\langle \mathbf{w}, \mathbf{x}_j \rangle + b) \geq 1$ (cf. (22)) could just as well be left out, and they do not appear in the expansion (26). This nicely captures our intuition of the problem: as the hyperplane (cf. Figure 2) is completely determined by the patterns closest to it, the solution should not depend on the other examples.

By substituting (25) and (26) into the Lagrangian (23), one eliminates the primal variables $\mathbf{w}$ and b, arriving at the so-called *dual optimization problem*, which is the problem that one usually solves in practice:

$$\underset{\boldsymbol{\alpha} \in \mathbb{R}^m}{\text{maximize}} \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \tag{28}$$

$$\text{subject to} \quad \alpha_i \geq 0 \text{ for all } i = 1, \ldots, m \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0. \tag{29}$$

Using (26), the hyperplane decision function (19) can thus be written as

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{m} y_i \alpha_i \langle \mathbf{x}, \mathbf{x}_i \rangle + b\right), \tag{30}$$

where b is computed by exploiting (27). For details, see Vapnik (1995); Burges (1998); Cristianini and Shawe-Taylor (2000); Schölkopf and Smola (2002); Herbrich (2002).

SUPPORT VECTOR CLASSIFICATION

We now have all the tools to describe SVMs (Figure 3). Everything in the last section was formulated in a dot product space, which we think of as the feature space $\mathcal{H}$ (see (6)). To express the formulas in terms of the input patterns in $\mathcal{X}$, we employ (7). This substitution, which is sometimes referred to as the *kernel trick*, was used by Boser, Guyon, and Vapnik (1992) to develop nonlinear Support Vector Machines. It can be applied since all feature vectors only occurred in dot products (see (28) and (30)). We obtain decision functions of the form (cf. (30))

$$f(x) = \operatorname{sgn}\left(\sum_{i=1}^{m} y_i \alpha_i \langle \Phi(x), \Phi(x_i) \rangle + b\right) = \operatorname{sgn}\left(\sum_{i=1}^{m} y_i \alpha_i k(x, x_i) + b\right), \tag{31}$$

and the following quadratic program (cf. (28)):

$$\underset{\boldsymbol{\alpha} \in \mathbb{R}^m}{\text{maximize}} \qquad W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \tag{32}$$

$$\text{subject to} \qquad \alpha_i \geq 0 \text{ for all } i = 1, \dots, m, \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0. \tag{33}$$

Figure 4 shows a toy example.

In practice, a separating hyperplane may not exist, e.g., if a high noise level causes a large overlap of the classes. To accommodate this case, one introduces slack variables

$$\xi_i \geq 0 \text{ for all } i = 1, \dots, m, \tag{34}$$

in order to relax the constraints (22) to

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \xi_i \text{ for all } i = 1, \dots, m. \tag{35}$$

A classifier that generalizes well is then found by controlling both the classifier capacity (via $\|\mathbf{w}\|$) and the sum of the slacks $\sum_i \xi_i$. The latter can be shown to provide an upper bound on the number of training errors.

One possible realization of such a *soft margin* classifier is obtained by minimizing the objective function

$$\tau(\mathbf{w}, \boldsymbol{\xi}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m}\xi_i \tag{36}$$

subject to the constraints (34) and (35), where the constant $C > 0$ determines the trade-off between margin maximization and training error minimization. This again leads to the problem of maximizing (32), subject to modified constraint where the only difference from the separable case is an upper bound C on the Lagrange multipliers α_i.

Another realization uses the more natural ν-parametrization. In it, the parameter C is replaced by a parameter $\nu \in (0, 1]$ which can be shown to provide lower and upper bounds for the fraction of examples that will be SVs and those that will have non-zero slack variables, respectively. Its dual can be shown to consist in maximizing the quadratic part of (32), subject to $0 \leq \alpha_i \leq 1/(\nu m)$, $\sum_i \alpha_i y_i = 0$ and the additional constraint $\sum_i \alpha_i = 1$.

SUPPORT VECTOR REGRESSION

Rather than dealing with outputs $y \in \{\pm 1\}$, *regression estimation* is concerned with estimating real-valued functions using $y \in \mathbb{R}$.

To generalize the SV algorithm to the regression case, Vapnik (1995) proposed the *ε-insensitive loss function* (Figure 5),

$$c(x, y, f(x)) := |y - f(\mathbf{x})|_\varepsilon := \max\{0, |y - f(\mathbf{x})| - \varepsilon\}. \tag{37}$$

To estimate a linear regression

$$f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b, \tag{38}$$

one minimizes

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m}|y_i - f(\mathbf{x}_i)|_\varepsilon. \tag{39}$$

Here, the term $\|\mathbf{w}\|^2$ is the same as in pattern recognition (cf. (36)). We can transform this into a constrained optimization problem by introducing slack variables, akin to the soft margin case.

An analysis similar to the one above leads to the dual (for $C, \varepsilon \geq 0$ chosen a priori):

$$\begin{aligned} \underset{\boldsymbol{\alpha}, \boldsymbol{\alpha}^* \in \mathbb{R}^m}{\text{maximize}} \quad W(\boldsymbol{\alpha}, \boldsymbol{\alpha}^*) \quad = \quad & -\varepsilon\sum_{i=1}^{m}(\alpha_i^* + \alpha_i) + \sum_{i=1}^{m}(\alpha_i^* - \alpha_i)y_i \\ & -\frac{1}{2}\sum_{i,j=1}^{m}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)k(x_i, x_j) \end{aligned} \tag{40}$$

$$\text{subject to} \qquad 0 \leq \alpha_i, \alpha_i^* \leq C \text{ for all } i = 1, \ldots, m, \text{ and } \sum_{i=1}^{m}(\alpha_i - \alpha_i^*) = 0. \tag{41}$$

The regression estimate takes the form

$$f(x) = \sum_{i=1}^{m} (\alpha_i^* - \alpha_i) k(x_i, x) + b, \tag{42}$$

where $\alpha_i^* - \alpha_i$ is zero for all points that lie inside the ε-tube (Figure 5). Note the similarity to the pattern recognition case (cf. (31) and Figure 6).

KERNEL PRINCIPAL COMPONENT ANALYSIS

The kernel trick can be used to develop nonlinear generalizations of any algorithm that can be cast in terms of dot products, such as PRINCIPAL COMPONENT ANALYSIS (PCA).

PCA in feature space leads to an algorithm called *kernel PCA*. It is derived as follows. We wish to find eigenvectors $\mathbf{v}$ and eigenvalues λ of the so-called *covariance matrix* $\mathbf{C}$ in the feature space, where

$$\mathbf{C} := \frac{1}{m} \sum_{i=1}^{m} \Phi(x_i)\Phi(x_i)^\top. \tag{43}$$

In the case when $\mathcal{H}$ is very high dimensional, the computational costs of doing this directly are prohibitive. Fortunately, one can show that all solutions to

$$\mathbf{C}\mathbf{v} = \lambda \mathbf{v} \tag{44}$$

with $\lambda \neq 0$ must lie in the span of Φ-images of the training data. Thus, we may expand the solution $\mathbf{v}$ as

$$\mathbf{v} = \sum_{i=1}^{m} \alpha_i \Phi(x_i), \tag{45}$$

thereby reducing the problem to that of finding the α_i. It turns out that this leads to a dual eigenvalue problem for the expansion coefficients,

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

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)^\top$ is normalized to satisfy $\|\boldsymbol{\alpha}\|^2 = 1/\lambda$.

To extract nonlinear features from a test point x, we compute the dot product between $\Phi(x)$ and the nth normalized eigenvector in feature space,

$$\langle \mathbf{v}^n, \Phi(x) \rangle = \sum_{i=1}^{m} \alpha_i^n k(x_i, x). \tag{47}$$

A toy example is given in Figure 7. As in the case of SVMs, the architecture can be visualized by Figure 6.

IMPLEMENTATION AND EMPIRICAL RESULTS

An initial weakness of SVMs was that the size of the quadratic programming problem scaled with the number of SVs. This was due to the fact that in (32), the quadratic part contained at least all SVs — the common practice was to extract the SVs by going through the training data in chunks while regularly testing for the possibility that patterns initially not identified as SVs become SVs at a later stage. This procedure is referred to as *chunking*; note that without chunking, the size of the matrix in the quadratic part of the objective function would be $m \times m$, where m is the number of all training examples.

What happens if we have a high-noise problem? In this case, many of the slack variables ξ_i become nonzero, and all the corresponding examples become SVs. For this case, decomposition algorithms were proposed, based on the observation that not only can we leave out the non-SV examples (the x_i with $\alpha_i = 0$) from the current chunk, but also some of the SVs, especially those that hit the upper boundary ($\alpha_i = C$). The chunks are usually dealt with using quadratic optimizers. Several public domain SV packages and optimizers are listed on the web page http://www.kernel-machines.org.

Modern SVM implementations made it possible to train on some rather large problems. Success stories include the 60,000 example MNIST digit recognition benchmark (with record results, see DeCoste and Schölkopf (2002)), as well as problems in text categorization (e.g., Joachims (1999)) and bioinformatics. For overviews of further applications as well as extensions of the approach to problems such as multi-class classification, density estimation, and novelty detection, the interested reader is referred to Burges (1998); Vapnik (1998); Schölkopf et al. (1999); Smola et al. (2000); Cristianini and Shawe-Taylor (2000); Schölkopf and Smola (2002).

DISCUSSION

During the last few years, SVMs and other kernel methods have rapidly advanced into the standard toolkit of techniques for machine learning and high-dimensional data analysis. This was probably due to a number of advantages compared to neural networks, such as the absence of spurious local minima in the optimization procedure, the fact that there are only few parameters to tune, enabling fast deployment in applications, the modularity in the design, where various kernels can be combined with a number of different learning algorithms, and the excellent performance on high-dimensional data.

Of particular interest for the cognitive science community are the way in which kernel methods connect similarity measures, nonlinearities, and data representations in linear spaces where simple geometric algorithms are performed. Moreover, it is worthwhile to note that as approaches based on a mathematical analysis of the problem of learning (in the formulation given by Vapnik and Chervonenkis), they are in principle not bound to any particular implementation.

Along with the observation that SV sets contain all the information required to solve a given classification task, this has led to speculations whether they could be related to prototypes believed to be used by the brain for categorization (cf. Poggio (1990)). This question, along with the one of whether SVMs might help explain some aspects of how the brain works, is as yet unanswered.

REFERENCES

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

(*) Burges, C. J. C., A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

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

DeCoste, D, and Schölkopf, B., Training invariant support vector machines. *Machine Learning*, 46:161–190, 2002.

Haussler, D., Convolutional kernels on discrete structures. Technical Report UCSC-CRL-99-10, Computer Science Department, University of California at Santa Cruz, 1999.

(*) Herbrich, R., *Learning Kernel Classifiers: Theory and Algorithms*. MIT Press, 2002.

Joachims, T., Making large–scale SVM learning practical. In Schölkopf, B., Burges, C. J. C., and Smola, A.J., editors, *Advances in Kernel Methods — Support Vector Learning*, pages 169–184, Cambridge, MA, 1999. MIT Press.

Poggio, T., A theory of how the brain might work. A.I. Memo No. 1253, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1990.

Schölkopf, B., Burges, C. J. C., and Smola, A. J., *Advances in Kernel Methods — Support Vector Learning*. MIT Press, Cambridge, MA, 1999.

(*) Schölkopf, B., and Smola, A. J., *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

Smola, A. J., Bartlett, P. L., Schölkopf, B., and Schuurmans, D., *Advances in Large Margin Classifiers*. MIT Press, Cambridge, MA, 2000.

(*) Vapnik, V., *The Nature of Statistical Learning Theory*. Springer, NY, 1995.

Vapnik, V., *Statistical Learning Theory*. Wiley, NY, 1998.

C. Watkins. Dynamic alignment kernels. In Schölkopf, B., Burges, C. J. C., and Smola, A.J., editors, *Advances in Large Margin Classifiers*, pages 39–50, Cambridge, MA, 2000. MIT Press.

(*) Williams, C. K. I., Prediction with Gaussian processes: From linear regression to linear prediction and beyond. In Jordan, M. I., editor, *Learning and Inference in Graphical Models*. Kluwer, 1998.

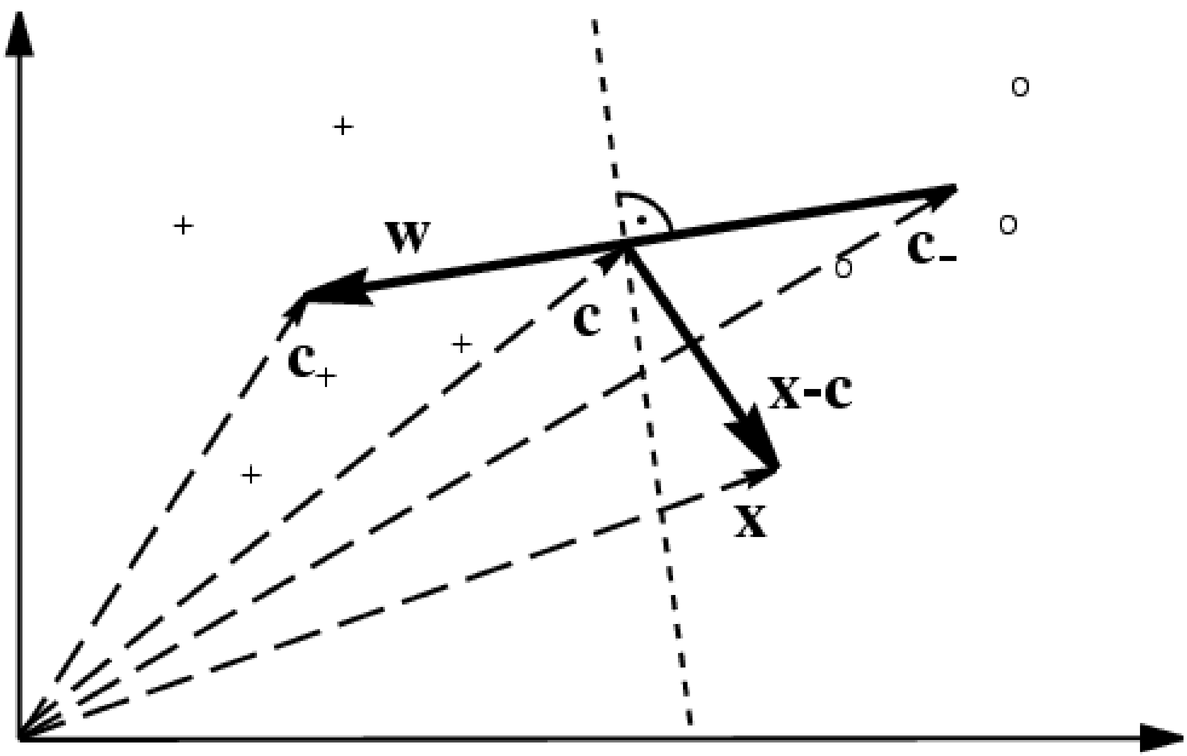

Figure 1: A simple geometric classification algorithm: given two classes of points (depicted by 'o' and '+'), compute their means $\mathbf{c}_+, \mathbf{c}_-$ and assign a test pattern $\mathbf{x}$ to the one whose mean is closer. This can be done by looking at the dot product between $\mathbf{x} - \mathbf{c}$ (where $\mathbf{c} = (\mathbf{c}_+ + \mathbf{c}_-)/2$) and $\mathbf{w} := \mathbf{c}_+ - \mathbf{c}_-$, which changes sign as the enclosed angle passes through $\pi/2$ (from Schölkopf and Smola (2002)).

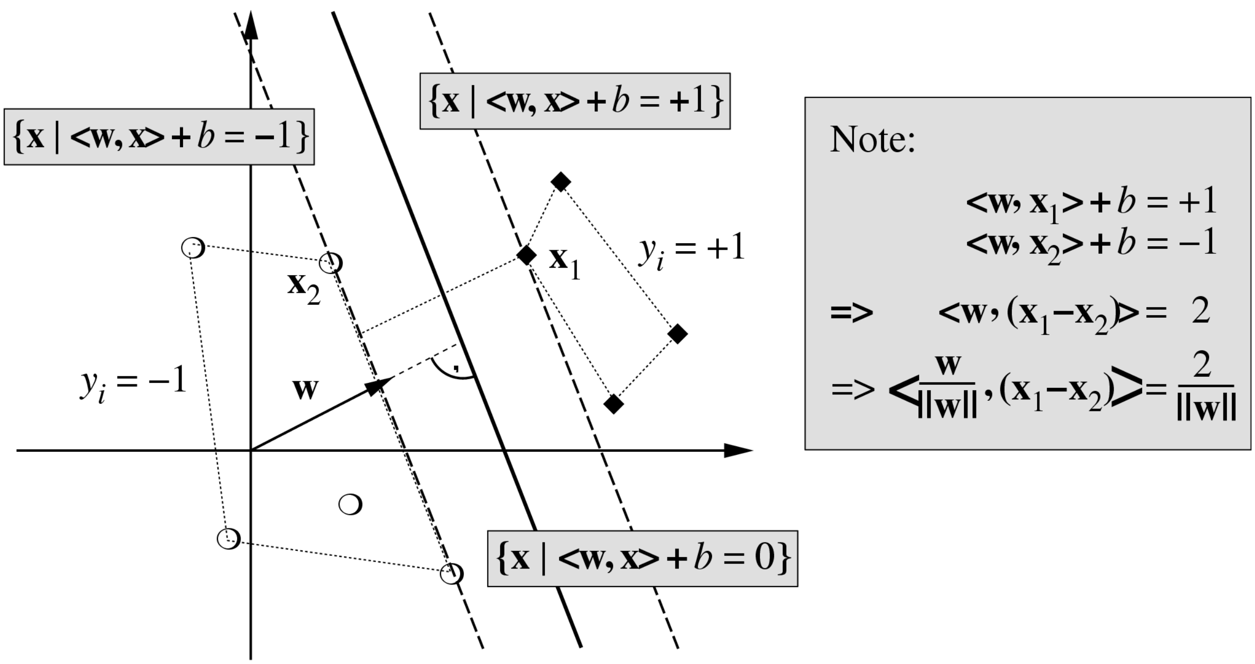

Figure 2: A binary classification toy problem: separate balls from diamonds. The *optimal hyperplane* (20) is shown as a solid line. The problem being separable, there exists a weight vector $\mathbf{w}$ and a threshold b such that $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) > 0$ $(i = 1, \ldots, m)$. Rescaling $\mathbf{w}$ and b such that the point(s) closest to the hyperplane satisfy $|\langle \mathbf{w}, \mathbf{x}_i \rangle + b| = 1$, we obtain a *canonical* form $(\mathbf{w}, b)$ of the hyperplane, satisfying $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1$. Note that in this case, the *margin* (the distance of the closest point to the hyperplane) equals $1/\|\mathbf{w}\|$. This can be seen by considering two points $\mathbf{x}_1, \mathbf{x}_2$ on opposite sides of the margin, that is, $\langle \mathbf{w}, \mathbf{x}_1 \rangle + b = 1, \langle \mathbf{w}, \mathbf{x}_2 \rangle + b = -1$, and projecting them onto the hyperplane normal vector $\mathbf{w}/\|\mathbf{w}\|$ (from Schölkopf and Smola (2002)).

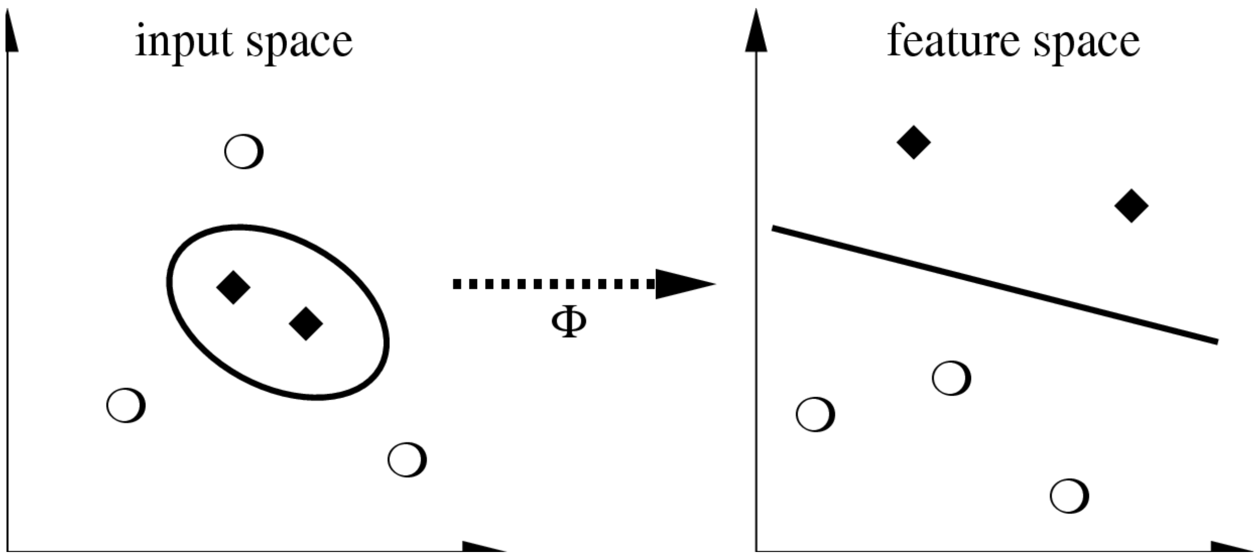

Figure 3: The idea of SVMs: map the training data into a higher-dimensional feature space via Φ, and construct a separating hyperplane with maximum margin there. This yields a nonlinear decision boundary in input space. By the use of a kernel (3), it is possible to compute the separating hyperplane without explicitly carrying out the map into the feature space (from Schölkopf and Smola (2002)).

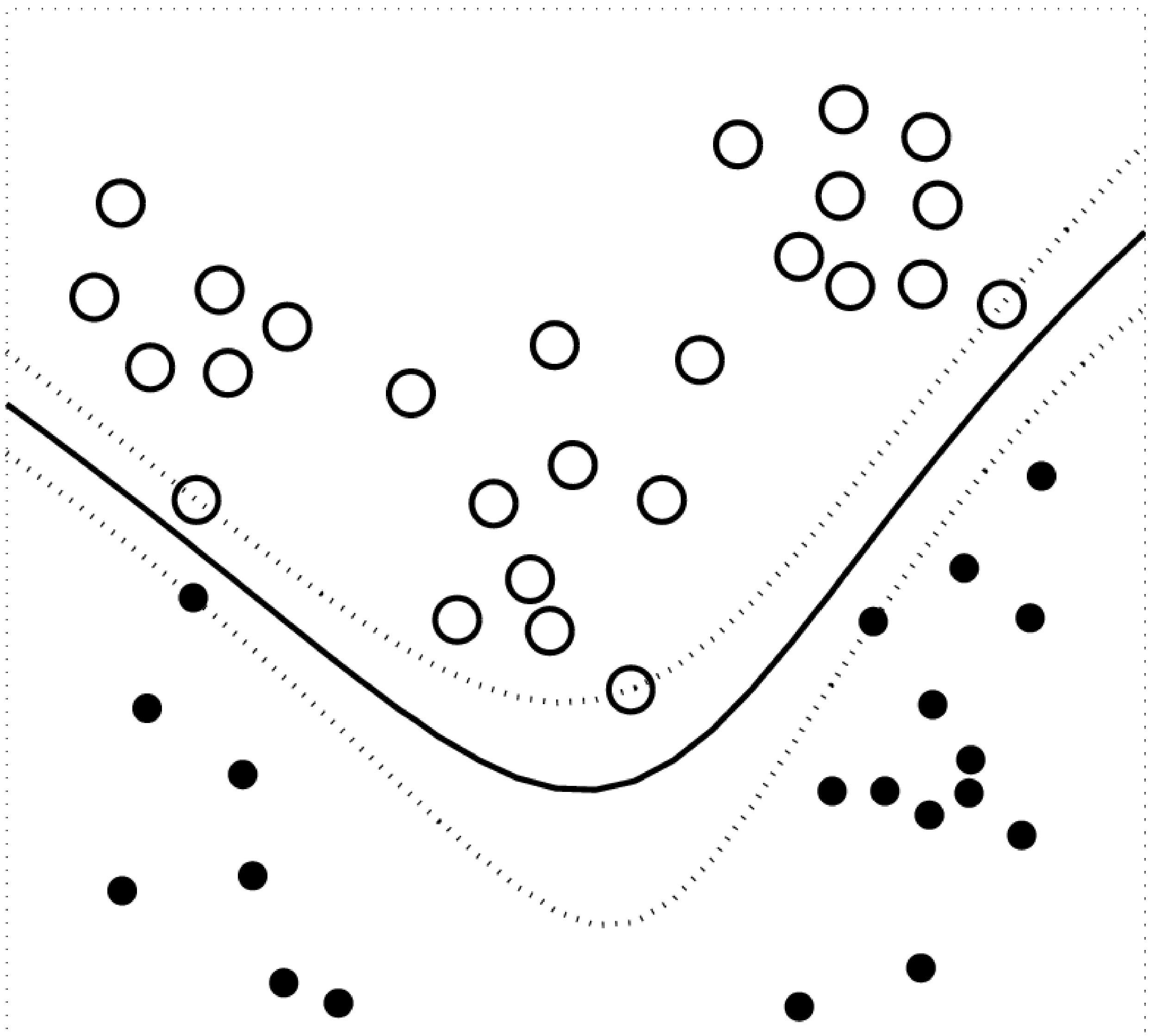

Figure 4: Example of an SV classifier found using a radial basis function kernel $k(x, x') = \exp(-\|x - x'\|^2)$. Circles and points are two classes of training examples; the middle line is the decision surface; the outer lines precisely meet the constraint (22). Note that the SVs found by the algorithm (sitting on the constraint lines) are not centers of clusters, but examples which are critical for the given classification task.

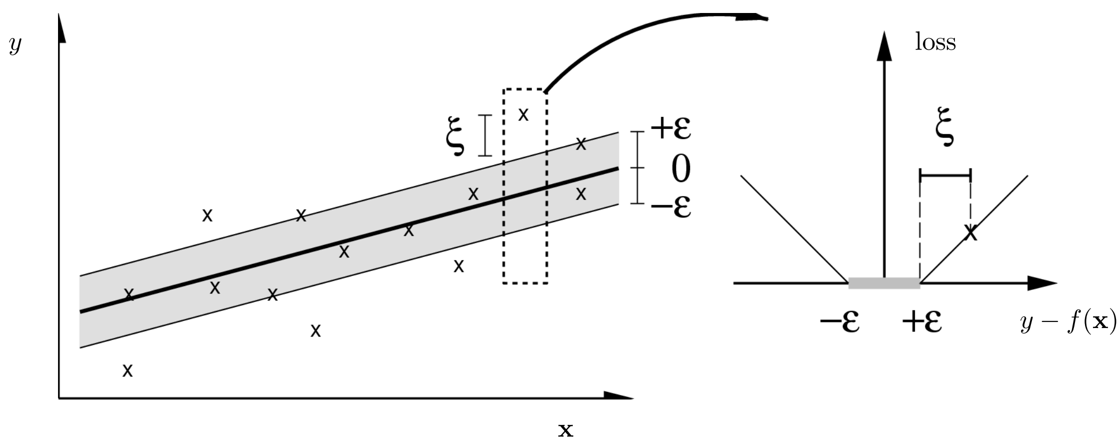

Figure 5: In SV regression, a tube with radius ε is fitted to the data. The trade-off between model complexity and points lying outside of the tube (with positive slack variables ξ) is determined by minimizing (39) (from Schölkopf and Smola (2002)).

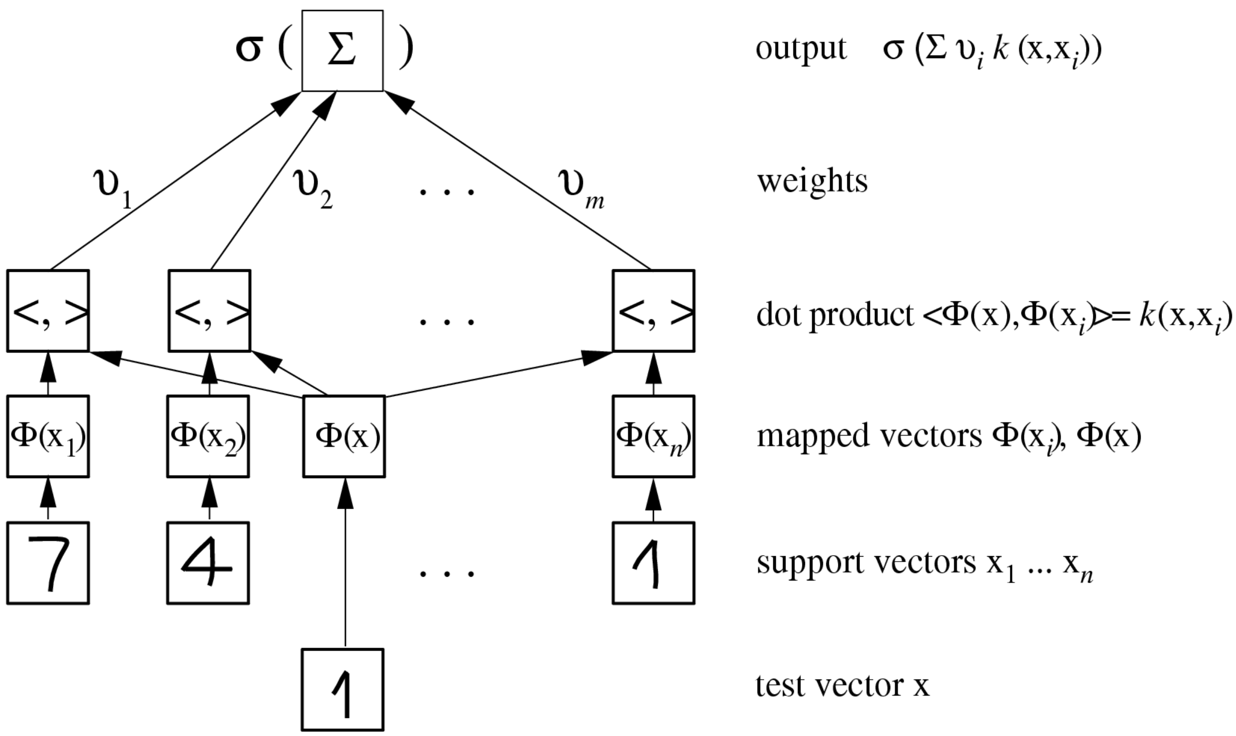

Figure 6: Architecture of SVMs and related kernel methods. The input x and the expansion patterns (SVs) x_i (we assume that we are dealing with handwritten digits) are nonlinearly mapped (by Φ) into a feature space $\mathcal{H}$ where dot products are computed. Through the use of the kernel k, these two layers are in practice computed in one step. The results are linearly combined using weights υ_i, found by solving a quadratic program (in pattern recognition, $\upsilon_i = y_i\alpha_i$; in regression estimation, $\upsilon_i = \alpha_i^* - \alpha_i$) or an eigenvalue problem (Kernel PCA). The linear combination is fed into the function σ (in pattern recognition, $\sigma(x) = \operatorname{sgn}(x + b)$; in regression estimation, $\sigma(x) = x + b$; in Kernel PCA, $\sigma(x) = x$) (from Schölkopf and Smola (2002)).

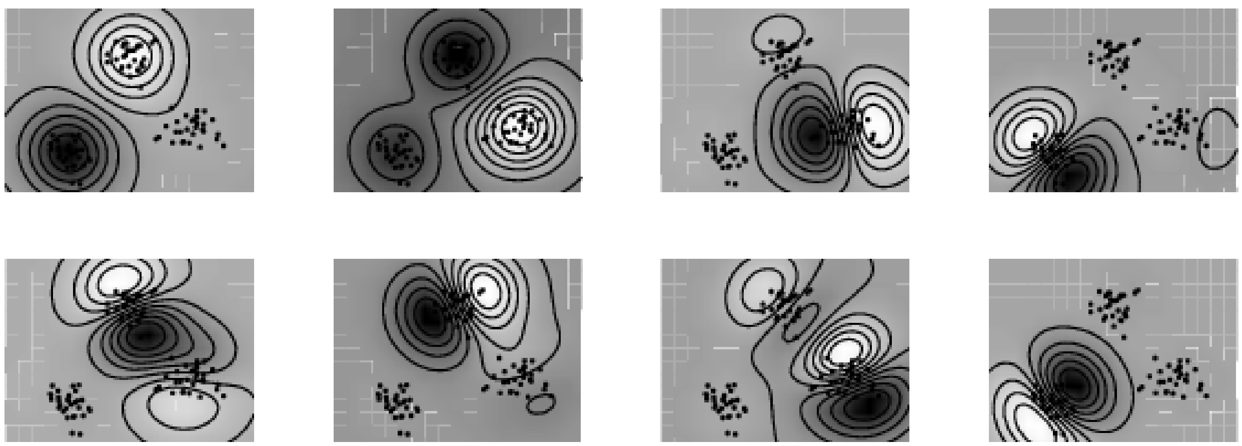

Figure 7: Contour plots of the first 8 nonlinear features of Kernel PCA using a RBF Kernel on a toy data set consisting of 3 Gaussian clusters. Upper left panel: the first and second component split the data into three clusters. Note that Kernel PCA is not deliberately designed to perform clustering — it tries to find a PCA description of the data in feature space by looking for directions of large variance. In input space, this reveals the cluster structure. The following three components depicted split each cluster in halves (components 3–5), while the last three achieve splits to the earlier ones.