Statistical Learning and Kernel Methods

Bernhard Schölkopf

Microsoft Research Limited,
1 Guildhall Street, Cambridge CB2 3NH, UK
bsc@microsoft.com
http://research.microsoft.com/~bsc

February 29, 2000

Technical Report
MSR-TR-2000-23

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

Lecture notes for a course to be taught at the Interdisciplinary College 2000, Günne, Germany, March 2000.

Abstract

We briefly describe the main ideas of statistical learning theory, support vector machines, and kernel feature spaces.

Contents

1 An Introductory Example

Suppose we are given empirical data

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

Here, the *domain* $\mathcal{X}$ is some nonempty set that the *patterns* x_i are taken from; the y_i are called *labels* or *targets*.

Unless stated otherwise, indices i and j will always be understood to run over the training set, i.e. $i, j = 1, \ldots, m$.

Note that we have not made any assumptions on the domain $\mathcal{X}$ other than it being a set. In order to study the problem of learning, we need additional structure. In learning, we want to be able to *generalize* to unseen data points. In the case of pattern recognition, this means that given some new pattern $x \in \mathcal{X}$, we want to predict the corresponding $y \in \{\pm 1\}$. By this we mean, loosely speaking, that we choose y such that (x, y) is in some sense similar to the training examples. To this end, we need similarity measures in $\mathcal{X}$ and in $\{\pm 1\}$. The latter is easy, as two target values can only be identical or different. For the former, we require a similarity measure

$$\begin{aligned} k : \mathcal{X} \times \mathcal{X} &\to \mathbb{R}, \\ (x, x') &\mapsto k(x, x'), \end{aligned} \tag{2}$$

i.e., a function that, given two examples x and x', returns a real number characterizing their similarity. For reasons that will become clear later, the function k is called a *kernel* [13, 1, 8].

A type of similarity measure that is of particular mathematical appeal are dot products. For instance, given two vectors $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^N$, the canonical dot product is defined as

$$(\mathbf{x} \cdot \mathbf{x}') := \sum_{i=1}^{N} (\mathbf{x})_i (\mathbf{x}')_i. \tag{3}$$

Here, $(\mathbf{x})_i$ denotes the i-th entry of $\mathbf{x}$.

The geometrical interpretation of this 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 of a vector $\mathbf{x}$ as $\sqrt{(\mathbf{x} \cdot \mathbf{x})}$, and of the distance between two vectors as the length of the difference vector. Therefore, being able to compute dot products amounts to being able to carry out all geometrical constructions that can be formulated in terms of angles, lenghts and distances.

Note, however, that we have not made the assumption that the patterns live in a dot product space. In order to be able to use a dot product as a similarity measure, we therefore first need to embed them into some dot product space F, which need not be identical to $\mathbb{R}^N$. To this end, we use a map

$$\begin{aligned} \Phi : \mathcal{X} &\to F \\ x &\mapsto \mathbf{x}. \end{aligned} \tag{4}$$

The space F is called a *feature space*. To summarize, embedding the data into F has three benefits.

1. It lets us define a similarity measure from the dot product in F,

$$k(x, x') := (\mathbf{x} \cdot \mathbf{x}') = (\Phi(x) \cdot \Phi(x')). \tag{5}$$

2. It allows us to deal with the patterns geometrically, and thus lets us study learning algorithm using linear algebra and analytic geometry.

3. The freedom to choose the mapping Φ will enable us to design a large variety of learning algorithms. For instance, consider a situation where the inputs already live in a dot product space. In that case, we could directly define a similarity measure as the dot product. However, we might still choose to first apply a nonlinear map Φ to change the representation into one that is more suitable for a given problem and learning algorithm.

We are now in the position to describe a pattern recognition learning algorithm that is arguably one of the simplest possible. The basic idea is to compute the means of the two classes in feature space,

$$\mathbf{c}_1 = \frac{1}{m_1} \sum_{\{i: y_i = +1\}} \mathbf{x}_i, \tag{6}$$

$$\mathbf{c}_2 = \frac{1}{m_2} \sum_{\{i: y_i = -1\}} \mathbf{x}_i, \tag{7}$$

where m_1 and m_2 are the number of examples with positive and negative labels, respectively. We then assign a new point $\mathbf{x}$ to the class whose mean is closer to it. This geometrical construction can be formulated in terms of dot products. Half-way in between $\mathbf{c}_1$ and $\mathbf{c}_2$ lies the point $\mathbf{c} := (\mathbf{c}_1 + \mathbf{c}_2)/2$. We compute the class of $\mathbf{x}$ by checking whether the vector connecting $\mathbf{c}$ and $\mathbf{x}$ encloses an angle smaller than $\pi/2$ with the vector $\mathbf{w} := \mathbf{c}_1 - \mathbf{c}_2$ connecting the class means, in other words

$$\begin{aligned} y &= \operatorname{sgn}\left((\mathbf{x} - \mathbf{c}) \cdot \mathbf{w}\right) \\ y &= \operatorname{sgn}\left((\mathbf{x} - (\mathbf{c}_1 + \mathbf{c}_2)/2) \cdot (\mathbf{c}_1 - \mathbf{c}_2)\right) \\ &= \operatorname{sgn}\left((\mathbf{x} \cdot \mathbf{c}_1) - (\mathbf{x} \cdot \mathbf{c}_2) + b\right). \end{aligned} \tag{8}$$

Here, we have defined the offset

$$b := \frac{1}{2}\left(\|\mathbf{c}_2\|^2 - \|\mathbf{c}_1\|^2\right). \tag{9}$$

It will prove instructive to rewrite this expression in terms of the patterns x_i in the input domain $\mathcal{X}$. To this end, note that we do not have a dot product in $\mathcal{X}$, all we have is the similarity measure k (cf. (5)). Therefore, we need to rewrite everything in terms of the kernel k evaluated on input patterns. To this end, substitute (6) and (7) into (8) to get the *decision function*

$$\begin{aligned} y &= \operatorname{sgn}\left(\frac{1}{m_1}\sum_{\{i:y_i=+1\}}(\mathbf{x}\cdot\mathbf{x}_i) - \frac{1}{m_2}\sum_{\{i:y_i=-1\}}(\mathbf{x}\cdot\mathbf{x}_i) + b\right) \\ &= \operatorname{sgn}\left(\frac{1}{m_1}\sum_{\{i:y_i=+1\}}k(x,x_i) - \frac{1}{m_2}\sum_{\{i:y_i=-1\}}k(x,x_i) + b\right). \end{aligned} \tag{10}$$

Similarly, the offset becomes

$$b := \frac{1}{2}\left(\frac{1}{m_2^2}\sum_{\{(i,j):y_i=y_j=-1\}}k(x_i,x_j) - \frac{1}{m_1^2}\sum_{\{(i,j):y_i=y_j=+1\}}k(x_i,x_j)\right). \tag{11}$$

Let us consider one well-known special case of this type of classifier. Assume that the class means have the same distance to the origin (hence $b = 0$), and that k can be viewed as a density, i.e. it is positive and has integral 1,

$$\int_{\mathcal{X}} k(x,x')dx = 1 \quad \text{for all } x' \in \mathcal{X}. \tag{12}$$

In order to state this assumption, we have to require that we can define an integral on $\mathcal{X}$.

If the above holds true, then (10) corresponds to the so-called Bayes decision boundary separating the two classes, subject to the assumption that the two classes were generated from two probability distributions that are correctly estimated by the *Parzen windows* estimators of the two classes,

$$p_1(x) := \frac{1}{m_1}\sum_{\{i:y_i=+1\}}k(x,x_i) \tag{13}$$

$$p_2(x) := \frac{1}{m_2}\sum_{\{i:y_i=-1\}}k(x,x_i). \tag{14}$$

Given some point x, the label is then simply computed by checking which of the two, $p_1(x)$ or $p_2(x)$, is larger, which directly leads to (10). Note that this decision is the best we can do if we have no prior information about the probabilities of the two classes.

The classifier (10) is quite close to the types of learning machines that we will be interested in. It is linear in the feature space, while in the input domain, it is represented by a kernel expansion. It is example-based in the sense that the kernels are centered on the training examples, i.e. one of the two arguments of the kernels is always a training example. The main point where the more sophisticated techniques to be discussed later will deviate from (10) is in the selection of the examples that the kernels are centered on, and in the weight that is put on the individual kernels in the decision function. Namely, it will no longer be the case that *all* training examples appear in the kernel expansion, and the weights of the kernels in the expansion will no longer be uniform. In the feature space representation, this statement corresponds to saying that we will study all normal vectors $\mathbf{w}$ of decision hyperplanes that can be represented as linear combinations of the training examples. For instance, we might want to remove the influence of patterns that are very far away from the decision boundary, either since we expect that they will not improve the generalization error of the decision function, or since we would like to reduce the computational cost of evaluating the decision function (cf. (10)). The hyperplane will then only depend on a subset of training examples, called *support vectors*.

2 Learning Pattern Recognition from Examples

With the above example in mind, let us now consider the problem of pattern recognition in a more formal setting [27, 28], following the introduction of [19]. In two-class pattern recognition, we seek to estimate a function

$$f : \mathcal{X} \to \{\pm 1\} \tag{15}$$

based on input-output training data (1). We assume that the data were generated independently from some unknown (but fixed) probability distribution $P(x, y)$. Our goal is to learn a function that will correctly classify unseen examples (x, y), i.e. we want $f(x) = y$ for examples (x, y) that were also generated from $P(x, y)$.

If we put no restriction on the class of functions that we choose our estimate f from, however, even a function which does well on the training data, e.g. by satisfying $f(x_i) = y_i$ for all $i = 1, \ldots, m$, need not generalize well to unseen examples. To see this, note that for each function f and any test set $(\bar{x}_1, \bar{y}_1), \ldots, (\bar{x}_{\bar{m}}, \bar{y}_{\bar{m}}) \in \mathbb{R}^N \times \{\pm 1\}$, satisfying $\{\bar{x}_1, \ldots, \bar{x}_{\bar{m}}\} \cap \{x_1, \ldots, x_m\} = \{\}$, there exists another function f^* such that $f^*(x_i) = f(x_i)$ for all $i = 1, \ldots, m$, yet $f^*(\bar{x}_i) \neq f(\bar{x}_i)$ for all $i = 1, \ldots, \bar{m}$. As we are only given the training data, we have no means of selecting which of the two functions (and hence which of the completely different sets of test label predictions) is preferable. Hence, only minimizing the training error (or *empirical risk*),

$$R_{emp}[f] = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} |f(x_i) - y_i|, \tag{16}$$

does not imply a small test error (called *risk*), averaged over test examples drawn from the underlying distribution $P(x, y)$,

$$R[f] = \int \frac{1}{2} |f(x) - y| \, dP(x, y). \tag{17}$$

Statistical learning theory [31, 27, 28, 29], or VC (Vapnik-Chervonenkis) theory, shows that it is imperative to restrict the class of functions that f is chosen from to one which has a *capacity* that is suitable for the amount of available training data. VC theory provides *bounds* on the test error. The minimization of these bounds, which depend on both the empirical risk and the capacity of the function class, leads to the principle of *structural risk minimization* [27]. The best-known capacity concept of VC theory is the *VC dimension*, defined as the largest number h of points that can be separated in all possible ways using functions of the given class. An example of a VC bound is the following: if $h < m$ is the VC dimension of the class of functions that the learning machine can implement, then for all functions of that class, with a probability of at least $1 - \eta$, the bound

$$R(\alpha) \leq R_{emp}(\alpha) + \phi\left(\frac{h}{m}, \frac{\log(\eta)}{m}\right) \tag{18}$$

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

$$\phi\left(\frac{h}{m}, \frac{\log(\eta)}{m}\right) = \sqrt{\frac{h\left(\log\frac{2m}{h} + 1\right) - \log(\eta/4)}{m}}. \tag{19}$$

Tighter bounds can be formulated in terms of other concepts, such as the *annealed VC entropy* or the *Growth function*. These are usually considered to be harder to evaluate, but they play a fundamental role in the conceptual part of VC theory [28]. Alternative capacity concepts that can be used to formulate bounds include the *fat shattering dimension* [2].

The bound (18) deserves some further explanatory remarks. Suppose we wanted to learn a "dependency" where $P(x, y) = P(x) \cdot P(y)$, i.e. where the pattern x contains no information about the label y, with uniform $P(y)$. Given a training sample of fixed size, we can then surely come up with a learning machine which achieves zero training error (provided we have no examples contradicting each other). However, in order to reproduce the random labellings, this machine will necessarily require a large VC dimension h. Thus, the confidence term (19), increasing monotonically with h, will be large, and the bound (18) will *not* support possible hopes that due to the small training error, we should expect a small test error. This makes it understandable how (18) can hold independent of assumptions about the underlying distribution $P(x, y)$: it always holds (provided that $h < m$), but it does not always make a nontrivial prediction — a bound on an error rate becomes void if it is larger than the maximum error rate. In order to get nontrivial predictions from (18), the function space must be restricted such that the capacity (e.g. VC dimension) is small enough (in relation to the available amount of data).

3 Hyperplane Classifiers

In the present section, we shall describe a hyperplane learning algorithm that can be performed in a dot product space (such as the feature space that we introduced previously). As described in the previous section, to design learning algorithms, one needs to come up with a class of functions whose capacity can be computed.

[32] and [30] considered the class of hyperplanes

$$(\mathbf{w} \cdot \mathbf{x}) + b = 0 \quad \mathbf{w} \in \mathbb{R}^N, b \in R, \tag{20}$$

corresponding to decision functions

$$f(\mathbf{x}) = \operatorname{sgn}((\mathbf{w} \cdot \mathbf{x}) + b), \tag{21}$$

and proposed a learning algorithm for separable problems, termed the *Generalized Portrait*, for constructing f from empirical data. It is based on two facts. First, among all hyperplanes separating the data, there exists a unique one yielding the maximum margin of separation between the classes,

$$\max_{\mathbf{w}, b} \min\{\|\mathbf{x} - \mathbf{x}_i\| : \mathbf{x} \in \mathbb{R}^N, (\mathbf{w} \cdot \mathbf{x}) + b = 0, i = 1, \ldots, m\}. \tag{22}$$

Second, the capacity decreases with increasing margin.

To construct this *Optimal Hyperplane* (cf. Figure 1), one solves the following optimization problem:

$$\text{minimize} \qquad \tau(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 \tag{23}$$

$$\text{subject to} \qquad y_i \cdot ((\mathbf{w} \cdot \mathbf{x}_i) + b) \geq 1, \quad i = 1, \ldots, m. \tag{24}$$

This constrained optimization problem is dealt with by introducing Lagrange multipliers $\alpha_i \geq 0$ and a Lagrangian

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{m} \alpha_i \left(y_i \cdot ((\mathbf{x}_i \cdot \mathbf{w}) + b) - 1\right). \tag{25}$$

The Lagrangian L has to be minimized with respect to the *primal variables* $\mathbf{w}$ and b and maximized with respect to the *dual variables* α_i (i.e. a saddle point has to be found). Let us try to get some intuition for this. If a constraint (24) is violated, then $y_i \cdot ((\mathbf{w} \cdot \mathbf{x}_i) + b) - 1 < 0$, 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 $-\alpha_i \left(y_i \cdot ((\mathbf{w} \cdot \mathbf{x}_i) + b) - 1\right)$ from becoming arbitrarily large, 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, i.e. for which $y_i \cdot ((\mathbf{w} \cdot \mathbf{x}_i) + b) - 1 > 0$, 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 complementarity conditions of optimization theory [6].

The condition that at the saddle point, the derivatives of L with respect to the primal variables must vanish,

$$\frac{\partial}{\partial b} L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0, \quad \frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0, \tag{26}$$

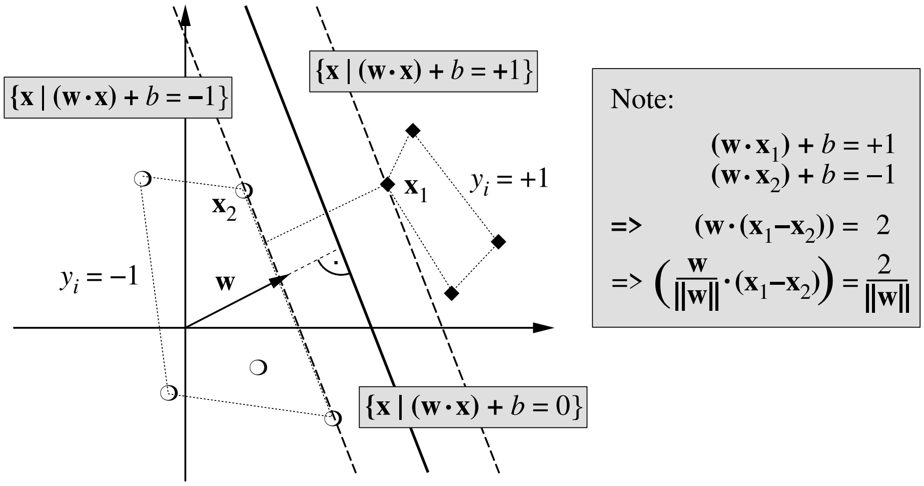

Figure 1: A binary classification toy problem: separate balls from diamonds. The *optimal hyperplane* is orthogonal to the shortest line connecting the convex hulls of the two classes (dotted), and intersects it half-way between the two classes. The problem being separable, there exists a weight vector $\mathbf{w}$ and a threshold b such that $y_i \cdot ((\mathbf{w} \cdot \mathbf{x}_i) + b) > 0$ $(i = 1, \ldots, m)$. Rescaling $\mathbf{w}$ and b such that the point(s) closest to the hyperplane satisfy $|(\mathbf{w} \cdot \mathbf{x}_i) + b| = 1$, we obtain a *canonical* form $(\mathbf{w}, b)$ of the hyperplane, satisfying $y_i \cdot ((\mathbf{w} \cdot \mathbf{x}_i) + b) \geq 1$. Note that in this case, the *margin*, measured perpendicularly to the hyperplane, equals $2/\|\mathbf{w}\|$. This can be seen by considering two points $\mathbf{x}_1, \mathbf{x}_2$ on opposite sides of the margin, i.e. $(\mathbf{w} \cdot \mathbf{x}_1) + b = 1, (\mathbf{w} \cdot \mathbf{x}_2) + b = -1$, and projecting them onto the hyperplane normal vector $\mathbf{w}/\|\mathbf{w}\|$.

leads to

$$\sum_{i=1}^{m} \alpha_i y_i = 0 \tag{27}$$

and

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i. \tag{28}$$

The solution vector thus has an expansion in terms of a subset of the training patterns, namely those patterns whose α_i is non-zero, called *Support Vectors*. By the Karush-Kuhn-Tucker complementarity conditions

$$\alpha_i \cdot [y_i((\mathbf{x}_i \cdot \mathbf{w}) + b) - 1] = 0, \qquad i = 1, \ldots, m, \tag{29}$$

the Support Vectors lie on the margin (cf. Figure 1). All remaining examples of the training set are irrelevant: their constraint (24) does not play a role in the optimization, and they do not appear in the expansion (28). This nicely captures our intuition of the problem: as the hyperplane (cf. Figure 1) is completely determined by the patterns closest to it, the solution should not depend on the other examples.

By substituting (27) and (28) into L, one eliminates the primal variables and arrives at the Wolfe dual of the optimization problem [e.g. 6]: find multipliers α_i which

$$\text{maximize} \qquad W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \tag{30}$$

$$\text{subject to} \qquad \alpha_i \geq 0, \quad i = 1, \ldots, m, \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0. \tag{31}$$

The hyperplane decision function can thus be written as

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{m} y_i \alpha_i \cdot (\mathbf{x} \cdot \mathbf{x}_i) + b \right) \tag{32}$$

where b is computed using (29).

The structure of the optimization problem closely resembles those that typically arise in Lagrange's formulation of mechanics. Also there, often only a subset of the constraints become active. For instance, if we keep a ball in a box, then it will typically roll into one of the corners. The constraints corresponding to the walls which are not touched by the ball are irrelevant, the walls could just as well be removed.

Seen in this light, it is not too surprising that it is possible to give a mechanical interpretation of optimal margin hyperplanes [9]: If we assume that each support vector $\mathbf{x}_i$ exerts a perpendicular force of size α_i and sign y_i on a solid plane sheet lying along the hyperplane, then the solution satisfies the requirements of mechanical stability. The constraint (27) states that the forces on the sheet sum to zero; and (28) implies that the torques also sum to zero, via $\sum_i \mathbf{x}_i \times y_i \alpha_i \cdot \mathbf{w}/\|\mathbf{w}\| = \mathbf{w} \times \mathbf{w}/\|\mathbf{w}\| = 0$.

There are theoretical arguments supporting the good generalization performance of the optimal hyperplane ([31, 27, 35, 4]). In addition, it is computationally attractive, since it can be constructed by solving a quadratic programming problem.

4 Support Vector Classifiers

We now have all the tools to describe support vector machines [28, 19, 26]. Everything in the last section was formulated in a dot product space. We think of this space as the feature space F described in Section 1. To express the formulas in terms of the input patterns living in $\mathcal{X}$, we thus need to employ (5), which expresses the dot product of bold face feature vectors $\mathbf{x}, \mathbf{x}'$ in terms of the kernel k evaluated on input patterns x, x',

$$k(x, x') = (\mathbf{x} \cdot \mathbf{x}'). \tag{33}$$

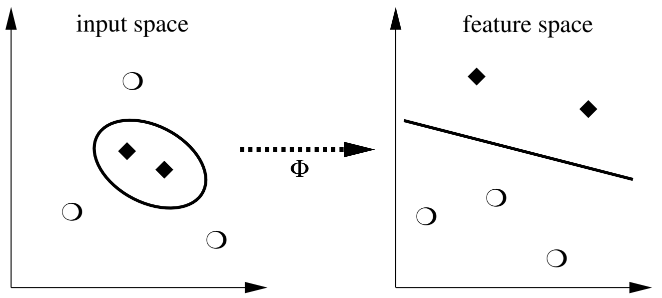

Figure 2: The idea of SV machines: map the training data into a higher-dimensional feature space via Φ, and construct a separating hyperplane with maximum margin there. This yields a nonlinear decision boundary in input space. By the use of a kernel function (2), it is possible to compute the separating hyperplane without explicitly carrying out the map into the feature space.

This can be done since all feature vectors only occured in dot products. The weight vector (cf. (28)) then becomes an expansion in feature space, and will thus typically no more correspond to the image of a single vector from input space. We thus obtain decision functions of the more general form (cf. (32))

$$\begin{aligned} f(x) &= \operatorname{sgn}\left(\sum_{i=1}^{m} y_i\alpha_i \cdot (\Phi(x)\cdot\Phi(x_i)) + b\right) \\ &= \operatorname{sgn}\left(\sum_{i=1}^{m} y_i\alpha_i \cdot k(x, x_i) + b\right), \end{aligned} \tag{34}$$

and the following quadratic program (cf. (30)):

$$\text{maximize} \qquad W(\alpha) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y_i y_j k(x_i, x_j) \tag{35}$$

$$\text{subject to} \qquad \alpha_i \geq 0, \quad i = 1, \dots, m, \text{ and } \sum_{i=1}^{m}\alpha_i y_i = 0. \tag{36}$$

In practice, a separating hyperplane may not exist, e.g. if a high noise level causes a large overlap of the classes. To allow for the possibility of examples violating (24), one introduces slack variables [10, 28, 22]

$$\xi_i \geq 0, \qquad i = 1, \dots, m \tag{37}$$

in order to relax the constraints to

$$y_i \cdot ((\mathbf{w}\cdot\mathbf{x}_i) + b) \geq 1 - \xi_i, \qquad i = 1, \dots, m. \tag{38}$$

A classifier which 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 is done as it can

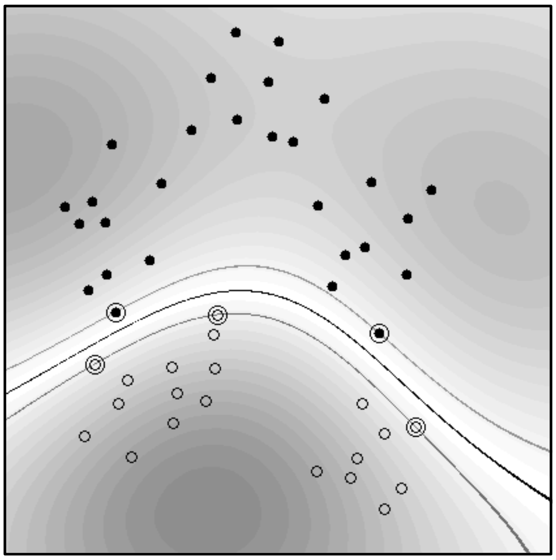

Figure 3: Example of a Support Vector classifier found by using a radial basis function kernel $k(x, x') = \exp(-\|x - x'\|^2)$. Both coordinate axes range from -1 to +1. Circles and disks are two classes of training examples; the middle line is the decision surface; the outer lines precisely meet the constraint (24). Note that the Support Vectors found by the algorithm (marked by extra circles) are not centers of clusters, but examples which are critical for the given classification task. Grey values code the modulus of the argument $\sum_{i=1}^{m} y_i \alpha_i \cdot k(x, x_i) + b$ of the decision function (34).)

be shown to provide an upper bound on the number of training errors which leads to a convex optimization problem.

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

$$\tau(\mathbf{w}, \boldsymbol{\xi}) = \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{m} \xi_i \tag{39}$$

subject to the constraints (37) and (38), for some value of the constant $C > 0$ determining the trade-off. Here and below, we use boldface greek letters as a shorthand for corresponding vectors $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_m)$. Incorporating kernels, and rewriting it in terms of Lagrange multipliers, this again leads to the problem of maximizing (35), subject to the constraints

$$0 \leq \alpha_i \leq C, \quad i = 1, \ldots, m, \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0. \tag{40}$$

The only difference from the separable case is the upper bound C on the Lagrange multipliers α_i. This way, the influence of the individual patterns (which

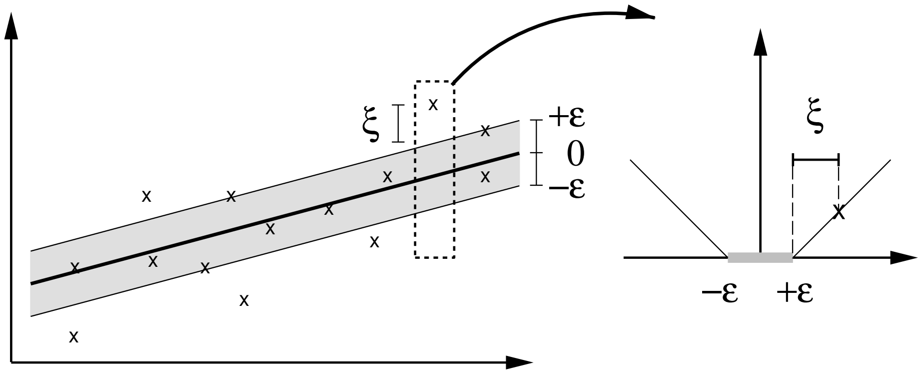

Figure 4: In SV regression, a tube with radius ε is fitted to the data. The trade-off between model complexity and points lying outside of the tube (with positive slack variables ξ) is determined by minimizing (46).

could be outliers) gets limited. As above, the solution takes the form (34). The threshold b can be computed by exploiting the fact that for all SVs x_i with $\alpha_i < C$, the slack variable ξ_i is zero (this again follows from the Karush-Kuhn-Tucker complementarity conditions), and hence

$$\sum_{j=1}^{m} y_j \alpha_j \cdot k(x_i, x_j) + b = y_i. \tag{41}$$

Another possible realization of a soft margin variant of the optimal hyperplane uses the ν-parametrization [22]. In it, the paramter C is replaced by a parameter $\nu \in [0, 1]$ which can be shown to lower and upper bound the number of examples that will be SVs and that will come to lie on the wrong side of the hyperplane, respectively. It uses a primal objective function with the error term $\frac{1}{\nu m} \sum_i \xi_i - \rho$, and separation constraints

$$y_i \cdot ((\mathbf{w} \cdot \mathbf{x}_i) + b) \geq \rho - \xi_i, \quad i = 1, \ldots, m. \tag{42}$$

The margin parameter ρ is a variable of the optimization problem. The dual can be shown to consist of maximizing the quadratic part of (35), subject to $0 \leq \alpha_i \leq 1/(\nu m)$, $\sum_i \alpha_i y_i = 0$ and the additional constraint $\sum_i \alpha_i = 1$.

5 Support Vector Regression

The concept of the margin is specific to pattern recognition. To generalize the SV algorithm to regression estimation [28], an analogue of the margin is constructed in the space of the target values y (note that in regression, we have $y \in \mathbb{R}$) by using Vapnik's ε-insensitive loss function (Figure 4)

$$|y - f(\mathbf{x})|_\varepsilon := \max\{0, |y - f(\mathbf{x})| - \varepsilon\}. \tag{43}$$

To estimate a linear regression

$$f(\mathbf{x}) = (\mathbf{w} \cdot \mathbf{x}) + b \tag{44}$$

with precision ε, one minimizes

$$\frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{m} |y_i - f(\mathbf{x}_i)|_\varepsilon. \tag{45}$$

Written as a constrained optimization problem, this reads:

$$\begin{aligned}
&\text{minimize} && \tau(\mathbf{w}, \boldsymbol{\xi}, \boldsymbol{\xi}^*) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m}(\xi_i + \xi_i^*) && (46)\\
&\text{subject to} && ((\mathbf{w} \cdot \mathbf{x}_i) + b) - y_i \leq \varepsilon + \xi_i && (47)\\
& && y_i - ((\mathbf{w} \cdot \mathbf{x}_i) + b) \leq \varepsilon + \xi_i^* && (48)\\
& && \xi_i, \xi_i^* \geq 0 && (49)
\end{aligned}$$

for all $i = 1, \ldots, m$. Note that according to (47) and (48), any error smaller than ε does not require a nonzero ξ_i or ξ_i^*, and hence does not enter the objective function (46).

Generalization to kernel-based regression estimation is carried out in complete analogy to the case of pattern recognition. Introducing Lagrange multipliers, one thus arrives at the following optimization problem: for $C > 0, \varepsilon \geq 0$ chosen a priori,

$$\begin{aligned}
&\text{maximize} && W(\boldsymbol{\alpha}, \boldsymbol{\alpha}^*) = -\varepsilon\sum_{i=1}^{m}(\alpha_i^* + \alpha_i) + \sum_{i=1}^{m}(\alpha_i^* - \alpha_i)y_i \\
& && -\frac{1}{2}\sum_{i,j=1}^{m}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)k(x_i, x_j) && (50)\\
&\text{subject to} && 0 \leq \alpha_i, \alpha_i^* \leq C, \quad i = 1, \ldots, m, \text{ and } \sum_{i=1}^{m}(\alpha_i - \alpha_i^*) = 0 && (51)
\end{aligned}$$

The regression estimate takes the form

$$f(x) = \sum_{i=1}^{m}(\alpha_i^* - \alpha_i)k(x_i, x) + b, \tag{52}$$

where b is computed using the fact that (47) becomes an equality with $\xi_i = 0$ if $0 < \alpha_i < C$, and (48) becomes an equality with $\xi_i^* = 0$ if $0 < \alpha_i^* < C$.

Several extensions of this algorithm are possible. From an abstract point of view, we just need some target function which depends on the vector $(\mathbf{w}, \boldsymbol{\xi})$ (cf. (46)). There are multiple degrees of freedom for constructing it, including some freedom how to penalize, or regularize, different parts of the vector, and some freedom how to use the kernel trick. For instance, more general loss

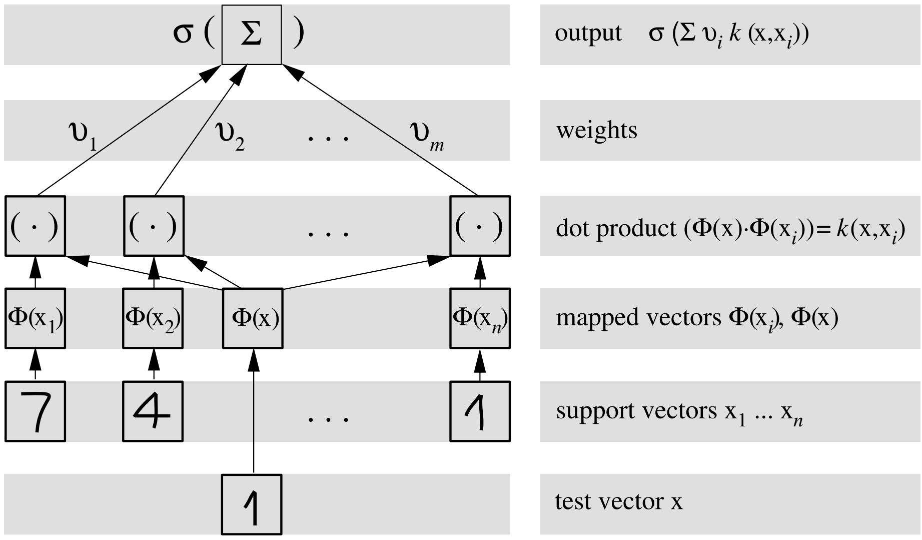

Figure 5: Architecture of SV machines. The input x and the Support Vectors x_i are nonlinearly mapped (by Φ) into a feature space F, where dot products are computed. By the use of the kernel k, these two layers are in practice computed in one single step. The results are linearly combined by weights υ_i, found by solving a quadratic program (in pattern recognition, $\upsilon_i = y_i\alpha_i$; in regression estimation, $\upsilon_i = \alpha_i^* - \alpha_i$). The linear combination is fed into the function σ (in pattern recognition, $\sigma(x) = \operatorname{sgn}(x+b)$; in regression estimation, $\sigma(x) = x + b$).

functions can be used for $\boldsymbol{\xi}$, leading to problems that can still be solved efficiently [24]. Moreover, norms other than the 2-norm $\|.\|$ can be used to regularize the solution. Yet another example is that polynomial kernels can be incorporated which consist of multiple layers, such that the first layer only computes products within certain specified subsets of the entries of $\mathbf{w}$ [17].

Finally, the algorithm can be modified such that ε need not be specified a priori. Instead, one specifies an upper bound $0 \leq \nu \leq 1$ on the fraction of points allowed to lie outside the tube (asymptotically, the number of SVs) and the corresponding ε is computed automatically. This is achieved by using as primal objective function

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\left(\nu m\varepsilon + \sum_{i=1}^{m} |y_i - f(\mathbf{x}_i)|_\varepsilon\right) \tag{53}$$

instead of (45), and treating $\varepsilon \geq 0$ as a parameter that we minimize over [22].

6 Further Developments

Having described the basics of SV machines, we now summarize some empirical findings and theoretical developments which were to follow.

By the use of kernels, the optimal margin classifier was turned into a classifier which became a serious competitor of high-performance classifiers. Surprisingly, it was noticed that when different kernel functions are used in SV machines, they empirically lead to very similar classification accuracies and SV sets [18]. In this sense, the SV set seems to characterize (or *compress*) the given task in a manner which up to a certain degree is independent of the type of kernel (i.e. the type of classifier) used.

Initial work at AT&T Bell Labs focused on OCR (optical character recognition), a problem where the two main issues are classification accuracy and classification speed. Consequently, some effort went into the improvement of SV machines on these issues, leading to the *Virtual SV* method for incorporating prior knowledge about transformation invariances by transforming SVs, and the *Reduced Set* method for speeding up classification. This way, SV machines became competitive with the best available classifiers on both OCR and object recognition tasks [7, 9, 17].

Another initial weakness of SV machines, less apparent in OCR applications which are characterized by low noise levels, was that the size of the quadratic programming problem scaled with the number of Support Vectors. This was due to the fact that in (35), 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 some of the patterns that were initially not identified as SVs turn out to become SVs at a later stage (note that without chunking, the size of the matrix 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 will become nonzero, and all the corresponding examples will become SVs. For this case, a decomposition algorithm was proposed [14], which is based on the observation that not only can we leave out the non-SV examples (i.e. the x_i with $\alpha_i = 0$) from the current chunk, but also some of the SVs, especially those that hit the upper boundary (i.e. $\alpha_i = C$). In fact, one can use chunks which do not even contain all SVs, and maximize over the corresponding sub-problems. SMO [15, 25, 20] explores an extreme case, where the sub-problems are chosen so small that one can solve them analytically. Several public domain SV packages and optimizers are listed on the web page http://www.kernel-machines.org. For more details on the optimization problem, see [19].

On the theoretical side, the least understood part of the SV algorithm initially was the precise role of the kernel, and how a certain kernel choice would influence the generalization ability. In that respect, the connection to regularization theory provided some insight. For kernel-based function expansions, one can show that given a regularization operator P mapping the functions of the learning machine into some dot product space, the problem of minimizing the regularized risk

$$R_{reg}[f] = R_{emp}[f] + \frac{\lambda}{2}\|Pf\|^2 \tag{54}$$

(with a regularization parameter $\lambda \geq 0$) can be written as a constrained optimization problem. For particular choices of the loss function, it further reduces to a SV type quadratic programming problem. The latter thus is not specific to SV machines, but is common to a much wider class of approaches. What gets lost in the general case, however, is the fact that the solution can usually be expressed in terms of a small number of SVs. This specific feature of SV machines is due to the fact that the type of regularization and the class of functions that the estimate is chosen from are intimately related [11, 23]: the SV algorithm is equivalent to minimizing the regularized risk on the set of functions

$$f(x) = \sum_i \alpha_i k(x_i, x) + b, \tag{55}$$

provided that k and P are interrelated by

$$k(x_i, x_j) = ((Pk)(x_i, .) \cdot (Pk)(x_j, .)) \,. \tag{56}$$

To this end, k is chosen as a Green's function of P^*P, for in that case, the right hand side of (56) equals

$$(k(x_i, .) \cdot (P^*Pk)(x_j, .)) = (k(x_i, .) \cdot \delta_{x_j}(.)) = k(x_i, x_j). \tag{57}$$

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

In SV machines, the kernel thus plays a dual role: firstly, it determines the class of functions (55) that the solution is taken from; secondly, via (56), the kernel determines the type of regularization that is used.

We conclude this section by noticing that the kernel method for computing dot products in feature spaces is not restricted to SV machines. Indeed, it has been pointed out that it can be used to develop nonlinear generalizations of any algorithm that can be cast in terms of dot products, such as principal component analysis [21], and a number of developments have followed this example.

7 Kernels

We now take a closer look at the issue of the similarity measure, or kernel, k.

In this section, we think of $\mathcal{X}$ as a subset of the vector space $\mathbb{R}^N$, $(N \in \mathbb{N})$, endowed with the canonical dot product (3).

7.1 Product Features

Suppose we are given patterns $x \in \mathbb{R}^N$ where most information is contained in the d-th order products (monomials) of entries $[x]_j$ of x,

$$[x]_{j_1} \cdot \ldots \cdot [x]_{j_d}, \tag{58}$$

where $j_1, \ldots, j_d \in \{1, \ldots, N\}$. In that case, we might prefer to *extract* these product features, and work in the feature space F of all products of d entries. In visual recognition problems, where images are often represented as vectors, this would amount to extracting features which are products of individual pixels.

For instance, in $\mathbb{R}^2$, we can collect all monomial feature extractors of degree 2 in the nonlinear map

$$\Phi : \mathbb{R}^2 \rightarrow F = \mathbb{R}^3 \tag{59}$$

$$([x]_1, [x]_2) \mapsto ([x]_1^2, [x]_2^2, [x]_1[x]_2). \tag{60}$$

This approach works fine for small toy examples, but it fails for realistically sized problems: for N-dimensional input patterns, there exist

$$N_F = \frac{(N+d-1)!}{d!(N-1)!} \tag{61}$$

different monomials (58), comprising a feature space F of dimensionality N_F. For instance, already 16×16 pixel input images and a monomial degree $d = 5$ yield a dimensionality of 10^{10}.

In certain cases described below, there exists, however, a way of *computing dot products* in these high-dimensional feature spaces without explicitely mapping into them: by means of kernels nonlinear in the input space $\mathbb{R}^N$. Thus, if the subsequent processing can be carried out using dot products exclusively, we are able to deal with the high dimensionality.

The following section describes how dot products in polynomial feature spaces can be computed efficiently.

7.2 Polynomial Feature Spaces Induced by Kernels

In order to compute dot products of the form $(\Phi(x) \cdot \Phi(x'))$, we employ kernel representations of the form

$$k(x, x') = (\Phi(x) \cdot \Phi(x')), \tag{62}$$

which allow us to compute the value of the dot product in F without having to carry out the map Φ. This method was used by [8] to extend the *Generalized Portrait* hyperplane classifier of [31] to nonlinear Support Vector machines. In [1], F is termed the *linearization space*, and used in the context of the potential function classification method to express the dot product between elements of F in terms of elements of the input space.

What does k look like for the case of polynomial features? We start by giving an example [28] for $N = d = 2$. For the map

$$C_2 : ([x]_1, [x]_2) \mapsto ([x]_1^2, [x]_2^2, [x]_1[x]_2, [x]_2[x]_1), \tag{63}$$

dot products in F take the form

$$(C_2(x) \cdot C_2(x')) = [x]_1^2[x']_1^2 + [x]_2^2[x']_2^2 + 2[x]_1[x]_2[x']_1[x']_2 = (x \cdot x')^2, \tag{64}$$

i.e. the desired kernel k is simply the square of the dot product in input space. The same works for arbitrary $N, d \in \mathbb{N}$ [8]: as a straightforward generalization of a result proved in the context of polynomial approximation [16, Lemma 2.1], we have:

Proposition 1 *Define C_d to map $x \in \mathbb{R}^N$ to the vector $C_d(x)$ whose entries are all possible d-th degree ordered products of the entries of x. Then the corresponding kernel computing the dot product of vectors mapped by C_d is*

$$k(x, x') = (C_d(x) \cdot C_d(x')) = (x \cdot x')^d. \tag{65}$$

Proof. We directly compute

$$\begin{aligned}
(C_d(x) \cdot C_d(x')) &= \sum_{j_1, \ldots, j_d = 1}^{N} [x]_{j_1} \cdot \ldots \cdot [x]_{j_d} \cdot [x']_{j_1} \cdot \ldots \cdot [x']_{j_d} && (66) \\
&= \left(\sum_{j=1}^{N} [x]_j \cdot [x']_j \right)^d = (x \cdot x')^d. && (67)
\end{aligned}$$

∎

Instead of ordered products, we can use unordered ones to obtain a map Φ_d which yields the same value of the dot product. To this end, we have to compensate for the multiple occurence of certain monomials in C_d by scaling the respective entries of Φ_d with the square roots of their numbers of occurence. Then, by this definition of Φ_d, and (65),

$$(\Phi_d(x) \cdot \Phi_d(x')) = (C_d(x) \cdot C_d(x')) = (x \cdot x')^d. \tag{68}$$

For instance, if n of the j_i in (58) are equal, and the remaining ones are different, then the coefficient in the corresponding component of Φ_d is $\sqrt{(d-n+1)!}$ [for the general case, cf. 23]. For Φ_2, this simply means that [28]

$$\Phi_2(x) = ([x]_1^2, [x]_2^2, \sqrt{2}\,[x]_1[x]_2). \tag{69}$$

If x represents an image with the entries being pixel values, we can use the kernel $(x \cdot x')^d$ to work in the space spanned by products of any d pixels — provided that we are able to do our work solely in terms of dot products, without any explicit usage of a mapped pattern $\Phi_d(x)$. Using kernels of the form (65), we take into account higher-order statistics without the combinatorial explosion (cf. (61)) of time and memory complexity which goes along already with moderately high N and d.

To conclude this section, note that it is possible to modify (65) such that it maps into the space of all monomials *up to* degree d, defining [28]

$$k(x, x') = ((x \cdot x') + 1)^d. \tag{70}$$

8 Representing Similarities in Linear Spaces

In what follows, we will look at things the other way round, and start with the kernel. Given some kernel function, can we construct a feature space such that the kernel computes the dot product in that feature space? This question has been brought to the attention of the machine learning community by [1, 8, 28]. In functional analysis, the same problem has been studied under the heading of *Hilbert space representations* of kernels. A good monograph on the functional analytic theory of kernels is [5]; indeed, a large part of the material in the present section is based on that work.

There is one more aspect in which this section differs from the previous one: the latter dealt with vectorial data. The results in the current section, in contrast, hold for data drawn from domains which need no additional structure other than them being nonempty sets $\mathcal{X}$. This generalizes kernel learning algorithms to a large number of situations where a vectorial representation is not readily available [17, 12, 34].

We start with some basic definitions and results.

Definition 2 (Gram matrix) *Given a kernel k and patterns $x_1, \ldots, x_m \in \mathcal{X}$, the $m \times m$ matrix*

$$K := (k(x_i, x_j))_{ij} \tag{71}$$

is called the Gram matrix *(or* kernel matrix*) of k with respect to $x_1, \ldots, x_m$.*

Definition 3 (Positive matrix) *An $m \times m$ matrix K_{ij} satisfying*

$$\sum_{i,j} c_i \bar{c}_j K_{ij} \geq 0 \tag{72}$$

for all $c_i \in \mathbb{C}$ is called positive.[1]

Definition 4 ((Positive definite) kernel) *Let $\mathcal{X}$ be a nonempty set. A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{C}$ which for all $m \in \mathbb{N}, x_i \in \mathcal{X}$ gives rise to a positive Gram matrix is called a* positive definite kernel*. Often, we shall refer to it simply as a* kernel.

The term *kernel* stems from the first use of this type of function in the study of integral operators. A function k which gives rise to an operator T via

$$(Tf)(x) = \int_{\mathcal{X}} k(x, x') f(x') \, dx' \tag{73}$$

is called the kernel of T. One might argue that the term *positive definite kernel* is slightly misleading. In matrix theory, the term *definite* is usually used to denote the case where equality in (72) only occurs if $c_1 = \ldots = c_m = 0$. Simply using the term *positive kernel*, on the other hand, could be confused with a kernel whose *values* are positive. In the literature, a number of different terms

[1] The bar in $\bar{c_j}$ denotes complex conjugation.

are used for positive definite kernels, such as *reproducing kernel*, *Mercer kernel*, or *support vector kernel*.

The definitions for (positive definite) kernels and positive matrices differ only in the fact that in the former case, we are free to choose the points on which the kernel is evaluated.

Positive definitness implies *positivity on the diagonal*,

$$k(x_1, x_1) \geq 0 \text{ for all } x_1 \in \mathcal{X}, \tag{74}$$

(use $m = 1$ in (72)), and *symmetry*, i.e.

$$k(x_i, x_j) = \overline{k(x_j, x_i)}. \tag{75}$$

Note that in the complex-valued case, our definition of symmetry includes complex conjugation, depicted by the bar. The definition of symmetry of matrices is analogous, i.e. $K_{ij} = \bar{K}_{ji}$.

Obviously, real-valued kernels, which are what we will mainly be concerned with, are contained in the above definition as a special case, since we did not require that the kernel take values in $\mathbb{C} \setminus \mathbb{R}$. However, it is not sufficient to require that (72) hold for real coefficients c_i. If we want to get away with real coefficients only, we additionally have to require that the kernel be symmetric,

$$k(x_i, x_j) = k(x_j, x_i). \tag{76}$$

It can be shown that whenever k is a (complex-valued) positive definite kernel, its real part is a (real-valued) positive definite kernel.

Kernels can be regarded as generalized dot products. Indeed, any dot product can be shown to be a kernel; however, linearity does not carry over from dot products to general kernels. Another property of dot products, the Cauchy-Schwarz inequality, does have a natural generalization to kernels:

Proposition 5 *If k is a positive definite kernel, and $x_1, x_2 \in \mathcal{X}$, then*

$$|k(x_1, x_2)|^2 \leq k(x_1, x_1) \cdot k(x_2, x_2). \tag{77}$$

Proof. For sake of brevity, we give a non-elementary proof using some basic facts of linear algebra. The 2×2 Gram matrix with entries $K_{ij} = k(x_i, x_j)$ is positive. Hence both its eigenvalues are nonnegative, and so is their product, K's determinant, i.e.

$$0 \leq K_{11}K_{22} - K_{12}K_{21} = K_{11}K_{22} - K_{12}\bar{K}_{12} = K_{11}K_{22} - |K_{12}|^2. \tag{78}$$

Substituting $k(x_i, x_j)$ for K_{ij}, we get the desired inequality. ∎

We are now in a position to construct the feature space associated with a kernel k.

We define a map from $\mathcal{X}$ into the space of functions mapping $\mathcal{X}$ into $\mathbb{C}$, denoted as $\mathbb{C}^{\mathcal{X}}$, via

$$\begin{aligned} \Phi : \mathcal{X} &\rightarrow \mathbb{C}^{\mathcal{X}} \\ x &\mapsto k(.,x). \end{aligned} \tag{79}$$

Here, $\Phi(x) = k(.,x)$ denotes the function that assigns the value $k(x',x)$ to $x' \in \mathcal{X}$.

We have thus turned each pattern into a function on the domain $\mathcal{X}$. In a sense, a pattern is now represented by its similarity to *all* other points in the input domain $\mathcal{X}$. This seems a very rich representation, but it will turn out that the kernel allows the computation of the dot product in that representation.

We shall now construct a dot product space containing the images of the input patterns under Φ. To this end, we first need to endow it with the linear structure of a vector space. This is done by forming linear combinations of the form

$$f(.) = \sum_{i=1}^{m} \alpha_i k(.,x_i). \tag{80}$$

Here, $m \in \mathbb{N}$, $\alpha_i \in \mathbb{C}$ and $x_i \in \mathcal{X}$ are arbitrary.

Next, we define a dot product between f and another function

$$g(.) = \sum_{j=1}^{m'} \beta_j k(.,x'_j) \tag{81}$$

($m' \in \mathbb{N}$, $\beta_j \in \mathbb{C}$ and $x'_j \in \mathcal{X}$) as

$$\langle f, g \rangle := \sum_{i=1}^{m} \sum_{j=1}^{m'} \bar{\alpha}_i \beta_j k(x_i, x'_j). \tag{82}$$

To see that this is well-defined, although it explicitly contains the expansion coefficients (which need not be unique), note that

$$\langle f, g \rangle = \sum_{j=1}^{m'} \beta_j \overline{f(x'_j)}, \tag{83}$$

using $k(x'_j, x_i) = \overline{k(x_i, x'_j)}$. The latter, however, does not depend on the particular expansion of f. Similarly, for g, note that

$$\langle f, g \rangle = \sum_{i=1}^{m} \bar{\alpha}_i g(x_i). \tag{84}$$

The last two equations also show that $\langle .,. \rangle$ is antilinear in the first argument and linear in the second one. It is symmetric, as $\langle f, g \rangle = \overline{\langle g, f \rangle}$. Moreover, given functions $f_1, \ldots, f_n$, and coefficients $\gamma_1, \ldots, \gamma_n \in \mathbb{C}$, we have

$$\sum_{i,j=1}^{n} \bar{\gamma}_i \gamma_j \langle f_i, f_j \rangle = \left\langle \sum_i \gamma_i f_i, \sum_i \gamma_i f_i \right\rangle \geq 0, \tag{85}$$

hence $\langle .,. \rangle$ is actually a positive definite kernel on our function space.

For the last step in proving that it even is a dot product, we will use the following interesting property of Φ, which follows directly from the definition: for all functions (80), we have

$$\langle k(.,x), f \rangle = f(x) \tag{86}$$

— k is the *representer of evaluation.* In particular,

$$\langle k(.,x), k(.,x') \rangle = k(x,x'). \tag{87}$$

By virtue of these properties, positive kernels k are also called *reproducing kernels* [3, 5, 33, 17].

By (86) and Proposition 5, we have

$$|f(x)|^2 = |\langle k(.,x), f \rangle|^2 \leq k(x,x) \cdot \langle f, f \rangle. \tag{88}$$

Therefore, $\langle f, f \rangle = 0$ directly implies $f = 0$, which is the last property that was left to prove in order to establish that $\langle .,. \rangle$ is a dot product.

One can complete the space of functions (80) in the norm corresponding to the dot product, i.e. add the limit points of sequences that are convergent in that norm, and thus gets a Hilbert space H, usually called a *reproducing kernel Hilbert space.*[2]

The case of real-valued kernels is included in the above; in that case, H can be chosen as a real Hilbert space.

9 Examples of Kernels

Besides (65), [8] and [28] suggest the usage of Gaussian radial basis function kernels [1]

$$k(x,x') = \exp\left(-\frac{\|x-x'\|^2}{2\,\sigma^2}\right) \tag{89}$$

and sigmoid kernels

$$k(x,x') = \tanh(\kappa(x \cdot x') + \Theta). \tag{90}$$

Note that all these kernels have the convenient property of unitary invariance, i.e. $k(x,x') = k(Ux, Ux')$ if $U^\top = U^{-1}$ (if we consider complex numbers, then U^* instead of $U^\top$ has to be used).

The radial basis function kernel additionally is translation invariant. Moreover, as it satisfies $k(x,x) = 1$ for all $x \in \mathcal{X}$, each mapped example has unit length, $\|\Phi(x)\| = 1$. In addition, as $k(x,x') > 0$ for all $x, x' \in \mathcal{X}$, all points lie inside the same orthant in feature space. To see this, recall that for unit lenght vectors, the dot product (3) equals the cosine of the enclosed angle. Hence

$$\cos(\angle(\Phi(x), \Phi(x'))) = (\Phi(x) \cdot \Phi(x')) = k(x,x') > 0, \tag{91}$$

[2]A Hilbert space H is defined as a complete dot product space. Completeness means that all sequences in H which are convergent in the norm corresponding to the dot product will actually have their limits in H, too.

which amounts to saying that the enclosed angle between any two mapped examples is smaller than $\pi/2$.

The examples given so far apply to the case of vectorial data. Let us at least give one example where $\mathcal{X}$ is not a vector space.

Example 6 (Similarity of probabilistic events) *If $\mathcal{A}$ is a σ-algebra, and P a probability measure on $\mathcal{A}$, then*

$$k(A, B) = P(A \cap B) - P(A)P(B) \tag{92}$$

is a positive definite kernel.

Further examples include kernels for string matching, as proposed by [34, 12].

10 Representating Dissimilarities in Linear Spaces

We now move on to a larger class of kernels. It is interesting in several regards. First, it will turn out that some kernel algorithms work with this larger class of kernels, rather than only with positive definite kernels. Second, their relationship to positive definite kernels is a rather interesting one, and a number of connections between the two classes provide understanding of kernels in general. Third, they are intimately related to a question which is a variation on the central aspect of positive definite kernels: the latter can be thought of as dot products in feature spaces; the former, on the other hand, can be embedded as *distance measures* arising from norms in feature spaces.

The following definition differs only in the additional constraint on the sum of the c_i from Definition 3.

Definition 7 (Conditionally positive matrix) *A symmetric $m \times m$ matrix K_{ij} $(m \geq 2)$ satisfying*

$$\sum_{i,j=1}^{m} c_i \bar{c}_j K_{ij} \geq 0 \tag{93}$$

for all $c_i \in \mathbb{C}$ with

$$\sum_{i=1}^{m} c_i = 0 \tag{94}$$

is called conditionally positive.

Definition 8 (Conditionally positive definite kernel) *Let $\mathcal{X}$ be a nonempty set. A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{C}$ which for all $m \geq 2, x_i \in \mathcal{X}$ gives rise to a conditionally positive Gram matrix is called a* conditionally positive definite kernel.

The definitions for the real-valued case look exactly the same. Note that symmetry is required, also in the complex case. Due to the additional constraint on the coefficients c_i, it does not follow automatically anymore.

It is trivially true that whenever k is positive definite, it is also conditionally positive definite. However, the latter is strictly weaker: if k is conditionally positive definite, and $b \in \mathbb{C}$, then $k+b$ is also conditionally positive definite. To see this, simply apply the definition to get, for $\sum_i c_i = 0$,

$$\sum_{i,j} c_i \bar{c}_j (k(x_i, x_j) + b) = \sum_{i,j} c_i \bar{c}_j k(x_i, x_j) + b \left| \sum_i c_i \right|^2 = \sum_{i,j} c_i \bar{c}_j k(x_i, x_j) \geq 0. \tag{95}$$

A standard example of a conditionally positive definite kernel which is not positive definite is

$$k(x, x') = -\|x - x'\|^2, \tag{96}$$

where $x, x' \in \mathcal{X}$, and $\mathcal{X}$ is a dot product space.

To see this, simply compute, for some pattern set $x_1, \ldots, x_m$,

$$\begin{aligned}
\sum_{i,j} c_i c_j k(x_i, x_j) &= -\sum_{i,j} c_i c_j \|x_i - x_j\|^2 && (97) \\
&= -\sum_{i,j} c_i c_j \left(\|x_i\|^2 + \|x_j\|^2 - 2(x_i \cdot x_j) \right) \\
&= -\sum_i c_i \sum_j c_j \|x_j\|^2 - \sum_j c_j \sum_i c_i \|x_i\|^2 + 2 \sum_{i,j} c_i c_j (x_i \cdot x_j) \\
&= 2 \sum_{i,j} c_i c_j (x_i \cdot x_j) \geq 0, && (98)
\end{aligned}$$

where the last line follows from (94) and the fact that $k(x, x') = (x \cdot x')$ is a positive definite kernel. Note that without (94), (97) can also be negative (e.g., put $c_1 = \ldots = c_m = 1$), hence the kernel is not a positive definite one.

Without proof, we add that in fact,

$$k(x, x') = -\|x - x'\|^\beta \tag{99}$$

is conditionally positive definite for $0 < \beta \leq 2$.

Let us consider the kernel (96), which can be considered the canonical conditionally positive kernel on a dot product space, and see how it is related to the dot product. Clearly, the distance induced by the norm is invariant under translations, i.e.

$$\|x - x'\| = \|(x - x_0) - (x' - x_0)\| \tag{100}$$

for all $x, x', x_0 \in \mathcal{X}$. In other words, even complete knowledge of $\|x - x'\|$ for all $x, x' \in \mathcal{X}$ would not completely determine the underlying dot product, the reason being that the dot product is *not* invariant under translations. Therefore, one needs to *fix* an origin x_0 when going from the distance measure to the dot product. To this end, we need to write the dot product of $x - x_0$ and $x' - x_0$ in terms of distances:

$$\begin{aligned}
((x - x_0) \cdot (x' - x_0)) &= (x \cdot x') + \|x_0\|^2 + (x \cdot x_0) + (x_0 \cdot x') \\
&= \frac{1}{2} \left(-\|x - x'\|^2 + \|x - x_0\|^2 + \|x_0 - x'\|^2 \right) && (101)
\end{aligned}$$

By construction, this will always result in a positive definite kernel: the dot product is a positive definite kernel, and we have only translated the inputs. We have thus established the connection between (96) and a class of positive definite kernels corresponding to the dot product in different coordinate systems, related to each other by translations. In fact, a similar connection holds for a wide class of kernels:

Proposition 9 *Let $x_0 \in \mathcal{X}$, and let k be a symmetric kernel on $\mathcal{X} \times \mathcal{X}$, satisfying $k(x_0, x_0) \geq 0$. Then*

$$\tilde{k}(x, x') := k(x, x') - k(x, x_0) - k(x_0, x') \tag{102}$$

is positive definite if and only if k is conditionally positive definite.

This result can be generalized to $k(x_0, x_0) < 0$. In this case, we simply need to add $k(x_0, x_0)$ on the right hand side of (102). This is necessary, for otherwise, we would have $\tilde{k}(x_0, x_0) < 0$, contradicting (74). Without proof, we state that it is also sufficient.

Using this result, one can prove another interesting connection between the two classes of kernels:

Proposition 10 *A kernel k is conditionally positive definite if and only if $\exp(tk)$ is positive definite for all $t > 0$.*

Positive definite kernels of the form $\exp(tk)$ $(t > 0)$ have the interesting property that their n-th root $(n \in \mathbb{N})$ is again a positive definite kernel. Such kernels are called *infinitely divisible.* One can show that, disregarding some technicalities, the logarithm of an infinitely divisible positive definite kernel mapping into $\mathbb{R}_0^+$ is a conditionally positive definite kernel.

Conditionally positive definite kernels are a natural choice whenever we are dealing with a translation invariant problem, such as the support vector machine: maximization of the margin of separarion between two classes of data is independent of the origin's position. This can be seen from the dual optimization problem (36): the constraint $\sum_{i=1}^{m} \alpha_i y_i = 0$ projects out the same subspace as (94) in the definition of conditionally positive matrices [17, 23].

We have seen that positive definite kernels and conditionally positive definite kernels are closely related to each other. The former can be represented as dot products in Hilbert spaces. The latter, it turns out, essentially correspond to distance measures associated with norms in Hilbert spaces:

Proposition 11 *Let k be a real-valued conditionally positive definite kernel on $\mathcal{X}$, satisfying $k(x, x) = 0$ for all $x \in \mathcal{X}$. Then there exists a Hilbert space H of real-valued functions on $\mathcal{X}$, and a mapping $\Phi : \mathcal{X} \to H$, such that*

$$k(x, x') = -\|\Phi(x) - \Phi(x')\|^2. \tag{103}$$

There exist generalizations to the case where $k(x, x) \neq 0$ and where k maps into $\mathbb{C}$. In these cases, the representation looks slightly more complicated.

The significance of this proposition is that using conditionally positive definite kernels, we can thus generalize all algorithms based on distances to corresponding algorithms operating in feature spaces. This is an analogue of the kernel trick for distances rather than dot products, i.e. dissimilarities rather than similarities.

Acknowledgements. Thanks to A. Smola and R. Williamson for discussions, and to C. Watkins for pointing out, in his NIPS'99 SVM workshop talk, that distances and dot products differ in the way they deal with the origin.

References

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

[2] N. Alon, S. Ben-David, N. Cesa-Bianchi, and D. Haussler. Scale–sensitive Dimensions, Uniform Convergence, and Learnability. *Journal of the ACM*, 44(4):615–631, 1997.

[3] N. Aronszajn. Theory of reproducing kernels. *Trans. Amer. Math. Soc.*, 68:337–404, 1950.

[4] P. L. Bartlett and J. Shawe-Taylor. Generalization performance of support vector machines and other pattern classifiers. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 43–54, Cambridge, MA, 1999. MIT Press.

[5] C. Berg, J.P.R. Christensen, and P. Ressel. *Harmonic Analysis on Semigroups.* Springer-Verlag, New York, 1984.

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

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

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

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

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

[11] F. Girosi, M. Jones, and T. Poggio. Regularization theory and neural networks architectures. *Neural Computation*, 7(2):219–269, 1995.

[12] D. Haussler. Convolutional kernels on discrete structures. Technical Report

UCSC-CRL-99-10, Computer Science Department, University of California at Santa Cruz, 1999.

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

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

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

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

[17] B. Schölkopf. *Support Vector Learning.* R. Oldenbourg Verlag, München, 1997. Doktorarbeit, TU Berlin.

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

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

[20] B. Schölkopf, J. Platt, J. Shawe-Taylor, A.J. Smola, and R.C. Williamson. Estimating the support of a high-dimensional distribution. TR MSR 99 - 87, Microsoft Research, Redmond, WA, 1999.

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

[22] B. Schölkopf, A. Smola, R. C. Williamson, and P. L. Bartlett. New support vector algorithms. *Neural Computation*, 12:1083 – 1121, 2000.

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

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

[25] A. J. Smola and B. Schölkopf. A tutorial on support vector regression. NeuroCOLT Technical Report NC-TR-98-030, Royal Holloway College, University of London, UK, 1998.

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

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

[28] V. Vapnik. *The Nature of Statistical Learning Theory.* Springer, N.Y., 1995.

[29] V. Vapnik. *Statistical Learning Theory.* Wiley, N.Y., 1998.

[30] V. Vapnik and A. Chervonenkis. A note on one class of perceptrons. *Automation and Remote Control*, 25, 1964.

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

[32] V. Vapnik and A. Lerner. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24, 1963.

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

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

[35] R. C. Williamson, A. J. Smola, and B. Schölkopf. Generalization performance of regularization networks and support vector machines via entropy numbers of compact operators. Technical Report 19, NeuroCOLT, http://www.neurocolt.com, 1998. Accepted for publication in IEEE Transactions on Information Theory.