# Statistical Learning and Kernel Methods in Bioinformatics

Bernhard Schölkopf,[*†] Isabelle Guyon,[‡] and Jason Weston[†]
[*] *Biowulf Technologies, New York*
[†] *Max-Planck-Institut für biologische Kybernetik, Tübingen,*
[‡] *Biowulf Technologies, Berkeley*
bernhard.schoelkopf@tuebingen.mpg.de,
isabelle@clopinet.com,
jason.weston@biowulf.com.

**Abstract.** We briefly describe the main ideas of statistical learning theory, support vector machines, and kernel feature spaces. In addition, we present an overview of applications of kernel methods in bioinformatics.[1]

## 1 An Introductory Example

In this Section, we formalize the problem of pattern recognition as that of classifying objects called "pattern" into one of two classes. We introduce a simple pattern recognition algorithm that illustrates the mechanism of kernel methods.

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, 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 easier, 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* [28, 1, 9].

[1] The present article is partly based on Microsoft TR-2000-23, Redmond, WA.

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

$$(\mathbf{x} \cdot \mathbf{x}') := \sum_{n=1}^{N} [\mathbf{x}]_n [\mathbf{x}']_n. \tag{3}$$

Here, $[\mathbf{x}]_n$ denotes the $n$-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, lengths 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 $\mathcal{H}$, which need not be identical to $\mathbb{R}^N$. To this end, we use a map

$$\begin{aligned} \Phi : \mathcal{X} &\to \mathcal{H} \\ x &\mapsto \mathbf{x} := \Phi(x). \end{aligned} \tag{4}$$

The space $\mathcal{H}$ is called a *feature space*. To summarize, embedding the data into $\mathcal{H}$ has three benefits.

1. It lets us define a similarity measure from the dot product in $\mathcal{H}$,

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

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

3. The freedom to choose the mapping $\Phi$ 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 use the dot product as a similarity measure. However, we might still choose to first apply another nonlinear map 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 simple pattern recognition algorithm. The 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

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

So, our simple pattern recognition algorithm is of the general form of a linear discriminant function:

$$y = \operatorname{sgn}\left((\mathbf{w} \cdot \mathbf{x}) + b\right) \tag{10}$$

It will prove instructive to rewrite this expression in terms of the patterns $x_i$ in the input domain $\mathcal{X}$. 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*

$$\begin{aligned} y &= \operatorname{sgn}\left(\frac{1}{m_1}\sum_{\{i:y_i=+1\}}(\mathbf{x} \cdot \mathbf{x}_i) - \frac{1}{m_2}\sum_{\{i:y_i=-1\}}(\mathbf{x} \cdot \mathbf{x}_i) + b\right) \\ &= \operatorname{sgn}\left(\frac{1}{m_1}\sum_{\{i:y_i=+1\}}k(x, x_i) - \frac{1}{m_2}\sum_{\{i:y_i=-1\}}k(x, x_i) + b\right). \end{aligned} \tag{11}$$

Similarly, the offset becomes

$$b := \frac{1}{2}\left(\frac{1}{m_2^2}\sum_{\{(i,j):y_i=y_j=-1\}}k(x_i, x_j) - \frac{1}{m_1^2}\sum_{\{(i,j):y_i=y_j=+1\}}k(x_i, x_j)\right). \tag{12}$$

So, our simple pattern recognition algorithm is also of the general form of a kernel classifier:

$$y = \operatorname{sgn}\left(\sum_i \alpha_i k(x, x_i) + b\right) \tag{13}$$

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,

$$\int_{\mathcal{X}} k(x, x')dx = 1 \quad \text{for all } x' \in \mathcal{X}. \tag{14}$$

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 (11) 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,

$$p_1(x) := \frac{1}{m_1} \sum_{\{i:y_i=+1\}} k(x, x_i) \tag{15}$$

$$p_2(x) := \frac{1}{m_2} \sum_{\{i:y_i=-1\}} k(x, x_i). \tag{16}$$

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, leading to (11). Note that this decision is the best we can do if we have no prior information about the probabilities of the two classes, or a uniform prior distribution. For further details, see [38].

The classifier (11) is quite close to the types of learning machines that we will be interested in. It is linear in the feature space (Equation (10)), while in the input domain, it is represented by a kernel expansion (Equation (13)). 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 (11) 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. (11)). 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, highlighting some ideas developed in statistical learning theory [39]. In two-class pattern recognition, we seek to estimate a function

$$f : \mathcal{X} \to \{\pm 1\} \tag{17}$$

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*),

$$R_{emp}[f] = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}|f(x_i) - y_i|, \tag{18}$$

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

$$R[f] = \int \frac{1}{2}|f(x) - y|\, dP(x, y). \tag{19}$$

Here, we denote by $|.|$ the absolute value.

Statistical learning theory ([41], [39], [40]), 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*. 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

$$R(f) \leq R_{emp}(f) + \psi\left(\frac{h}{m}, \frac{\log(\eta)}{m}\right) \tag{20}$$

holds, where $m$ is the number of training examples and the *confidence term* $\psi$ is defined as

$$\psi\left(\frac{h}{m}, \frac{\log(\eta)}{m}\right) = \sqrt{\frac{h\left(\log\frac{2m}{h} + 1\right) - \log(\eta/4)}{m}}. \tag{21}$$

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 [39]. Alternative capacity concepts that can be used to formulate bounds include the *fat shattering dimension* [3].

The bound (20) 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 (21), increasing monotonically with $h$, will be large, and the bound (20) will *not* support possible hopes that due to the small training error, we should expect a small test error. This makes it understandable how (20) can hold independently 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 (20), 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).

The principles of statistical learning theory that we just sketched provide a prescription to bias the choice of function space towards small capacity ones. The rationale behind that prescription is to try to achieve better bounds on the test error $R[f]$. This is related to model selection prescriptions that bias towards choosing simple models (e.g., Occam's razor, minimum description length, small number of free parameters). Yet, the prescription of statistical learning theory sometimes differs markedly from the others. A family of functions with only one free parameter may have infinite VC dimension. Also, statistical learning theory predicts that the kernel classifiers operating in spaces of infinite dimension that we shall introduce can have a large probability of a low test error.

## 3 Optimal Margin 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.

Vapnik and Lerner [42] considered the class of hyperplanes

$$(\mathbf{w} \cdot \mathbf{x}) + b = 0 \quad \mathbf{w} \in \mathbb{R}^N, b \in \mathbb{R}, \tag{22}$$

corresponding to decision functions

$$f(\mathbf{x}) = \operatorname{sgn}\left((\mathbf{w} \cdot \mathbf{x}) + b\right), \tag{23}$$

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 (assuming that the data is separable), there exists a unique one yielding the maximum margin of separation between the classes,

$$\max_{\mathbf{w},b} \ \min\{\|\mathbf{x} - \mathbf{x}_i\| : \mathbf{x} \in \mathbb{R}^N, (\mathbf{w} \cdot \mathbf{x}) + b = 0, i = 1, \ldots, m\}. \tag{24}$$

Second, the capacity can be shown to decrease with increasing margin.

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

$$\text{minimize} \qquad \tau(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 \tag{25}$$

$$\text{subject to} \qquad y_i \cdot ((\mathbf{w} \cdot \mathbf{x}_i) + b) \geq 1, \quad i = 1, \ldots, m. \tag{26}$$

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

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{m} \alpha_i \left(y_i \cdot ((\mathbf{x}_i \cdot \mathbf{w}) + b) - 1\right). \tag{27}$$

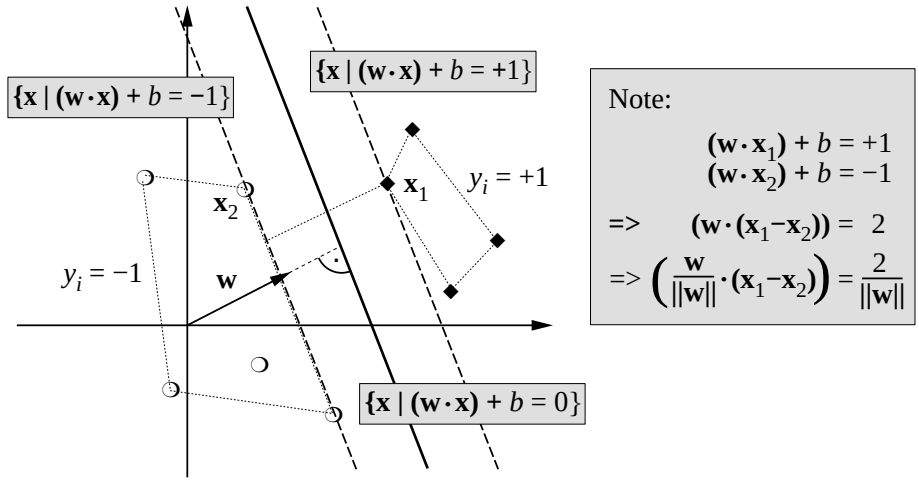


Figure 1: A binary classification toy problem: separate balls from diamonds. The *Optimum Margin Hyperplane* is orthogonal to the shortest line connecting the convex hulls of the two classes (dotted), and intersects it halfway between the two classes. The problem being separable, there exists a weight 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 $\mathbf{w}/\|\mathbf{w}\|$ (from [35]).

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* $\alpha_i$ (i.e., a saddle point has to be found). Let us try to get some intuition for this. If a constraint (26) 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 $\alpha_i$. 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 $\alpha_i$ must be 0, for this is the value of $\alpha_i$ that maximizes $L$. This is the statement of the Karush-Kuhn-Tucker conditions of optimization theory [6].

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

$$\frac{\partial}{\partial b} L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0, \quad \frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0, \tag{28}$$

leads to

$$\sum_{i=1}^{m} \alpha_i y_i = 0 \tag{29}$$

and

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i. \tag{30}$$

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

$$\alpha_i \cdot [y_i((\mathbf{x}_i \cdot \mathbf{w}) + b) - 1] = 0, \quad i = 1, \ldots, m, \tag{31}$$

the Support Vectors satisfy $y_i((\mathbf{x}_i \cdot \mathbf{w}) + b) - 1 = 0$, i.e., they lie on the margin (cf. Figure 1). All remaining examples of the training set are irrelevant: their constraint (26) does not play a role in the optimization, and they do not appear in the expansion (30). This nicely captures our intuition of the problem: as the hyperplane (cf. Figure 1) is geometrically completely determined by the patterns closest to it, the solution should not depend on the other examples.

By substituting (29) and (30) into $L$, one eliminates the primal variables and arrives at the Wolfe dual of the optimization problem (e.g., [6]): find multipliers $\alpha_i$ which

$$\text{maximize} \qquad W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \tag{32}$$

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

By substituting (30) into (23), the hyperplane decision function can thus be written as

$$f(\mathbf{x}) = \text{sgn} \left( \sum_{i=1}^{m} y_i \alpha_i \cdot (\mathbf{x} \cdot \mathbf{x}_i) + b \right), \tag{34}$$

where $b$ is computed using (31).

The structure of the optimization problem closely resembles those that typically arise in Lagrange's formulation of mechanics. 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 [11]: If we assume that each support vector $\mathbf{x}_i$ exerts a perpendicular force of size $\alpha_i$ and sign $y_i$ on a solid plane sheet lying along the hyperplane, then the solution satisfies the requirements of mechanical stability. The constraint (29) states that the forces on the sheet sum to zero; and (30) 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 ([41], [47], [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 [9, 39, 37, 15, 38]. Everything in the last section was formulated in a dot product space. We think of this space as the feature space $\mathcal{H}$ 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 $x, x'$,

$$k(x, x') = (\mathbf{x} \cdot \mathbf{x}'). \tag{35}$$

This can be done since all feature vectors only occured in dot products. The weight vector (cf. (30)) then becomes an expansion in feature space, and will thus typically no longer correspond to the image of a single vector from input space. We thus obtain decision functions

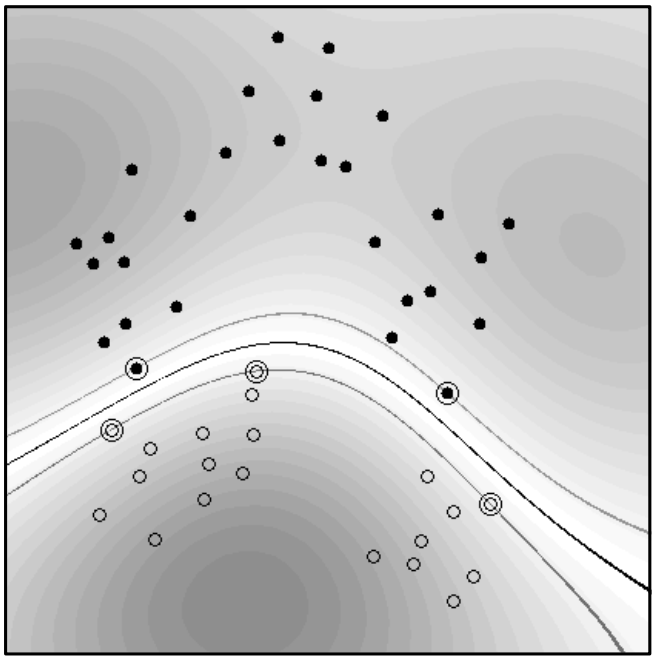

Figure 2: Example of a Support Vector classifier found by using a radial basis function kernel $k(x, x') = \exp(-\|x - x'\|^2)$. Both coordinate axes range from -1 to +1. Circles and disks are two classes of training examples; the middle line is the decision surface; the outer lines precisely meet the constraint (26). 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 (36) (from [35]).)

of the more general form (cf. (34))

$$\begin{aligned} f(x) &= \text{sgn}\left(\sum_{i=1}^{m} y_i \alpha_i \cdot (\Phi(x) \cdot \Phi(x_i)) + b\right) \\ &= \text{sgn}\left(\sum_{i=1}^{m} y_i \alpha_i \cdot k(x, x_i) + b\right), \end{aligned} \tag{36}$$

and the following quadratic program (cf. (32)):

$$\text{maximize} \qquad W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \tag{37}$$

$$\text{subject to} \qquad \alpha_i \geq 0, \;\; i = 1, \ldots, m, \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0. \tag{38}$$

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 (26), one introduces slack variables [14]

$$\xi_i \geq 0, \quad i = 1, \ldots, m \tag{39}$$

in order to relax the constraints to

$$y_i \cdot ((\mathbf{w} \cdot \mathbf{x}_i) + b) \geq 1 - \xi_i, \quad i = 1, \ldots, m. \tag{40}$$

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 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

$$\tau(\mathbf{w}, \boldsymbol{\xi}) = \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{m} \xi_i \tag{41}$$

subject to the constraints (39) and (40), for some value of the constant $C > 0$ determining the trade-off. Here, we use the shorthand $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_m)$. Incorporating kernels, and rewriting it in terms of Lagrange multipliers, this again leads to the problem of maximizing (37), subject to the constraints

$$0 \leq \alpha_i \leq C, \;\; i = 1, \ldots, m, \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0. \tag{42}$$

The only difference from the separable case is the upper bound $C$ on the Lagrange multipliers $\alpha_i$. This way, the influence of the individual patterns (which could be outliers) gets limited. As above, the solution takes the form (36). The threshold $b$ can be computed by exploiting the fact that for all SVs $x_i$ with $\alpha_i < C$, the slack variable $\xi_i$ is zero (this again follows from the Karush-Kuhn-Tucker complementarity conditions), and hence

$$\sum_{j=1}^{m} y_j \alpha_j \cdot k(x_i, x_j) + b = y_i. \tag{43}$$

Another possible realization of a soft margin variant of the optimal hyperplane uses the $\nu$-parametrization [38]. In it, the parameter $C$ 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

$$y_i \cdot ((\mathbf{w} \cdot \mathbf{x}_i) + b) \geq \rho - \xi_i, \quad i = 1, \ldots, m. \tag{44}$$

The margin parameter $\rho$ is a variable of the optimization problem. The dual can be shown to consist of maximizing the quadratic part of (37), subject to $0 \leq \alpha_i \leq 1/(\nu m)$, $\sum_i \alpha_i y_i = 0$ and the additional constraint $\sum_i \alpha_i = 1$. The advantage of the $\nu$-SVM is its more intuitive parametrization.

We conclude this section by noting that the SV algorithm has been generalized to problems such as regression estimation [39] as well as one-class problems and novelty detection [38]. The algorithms and architectures involved are similar to the case of pattern recognition described above (see Figure 3). Moreover, 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 [38], and a number of developments have followed this example.

## 5 Polynomial 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). Unlike in cases where $\mathcal{X}$ does not have a dot product, we thus *could* use the canonical dot product as a similarity measure $k$. However, in many cases, it is advantageous to use a different $k$, corresponding to a better data representation.

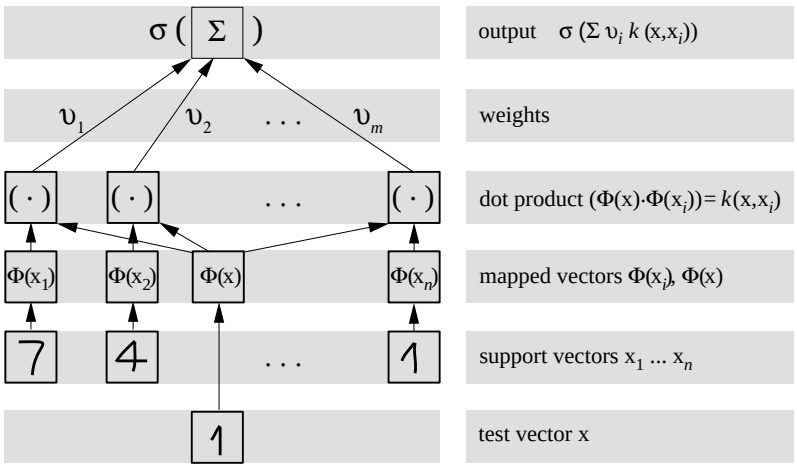


Figure 3: Architecture of SV machines. The input $x$ and the Support Vectors $x_i$ are nonlinearly mapped (by $\Phi$) into a feature space $\mathcal{H}$, where dot products are computed. By the use of the kernel $k$, these two layers are in practice computed in one single step. The results are linearly combined by weights $\upsilon_i$, found by solving a quadratic program (in pattern recognition, $\upsilon_i = y_i\alpha_i$). The linear combination is fed into the function $\sigma$ (in pattern recognition, $\sigma(x) = \operatorname{sgn}(x + b)$) (from [35]).

### *5.1 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 $x$,

$$[x]_{j_1}[x]_{j_2}\ldots[x]_{j_d}, \tag{45}$$

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 $\mathcal{H}$ 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

$$\Phi : \mathbb{R}^2 \rightarrow \mathcal{H} = \mathbb{R}^3 \tag{46}$$

$$([x]_1, [x]_2) \mapsto ([x]_1^2, [x]_2^2, [x]_1[x]_2). \tag{47}$$

Here the dimension of input space is $N = 2$ and that of feature space is $N_{\mathcal{H}} = 3$. This approach works fine for small toy examples, but it fails for realistically sized problems: for general $N$-dimensional input patterns, there exist

$$N_{\mathcal{H}} = \frac{(N + d - 1)!}{d!(N - 1)!} \tag{48}$$

different monomials (45), comprising a feature space $\mathcal{H}$ of dimensionality $N_{\mathcal{H}}$. For instance, already $16 \times 16$ 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.

### 5.2 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

$$k(x,x') = (\Phi(x)\cdot\Phi(x')), \tag{49}$$

which allow us to compute the value of the dot product in $\mathcal{H}$ without having to carry out the map $\Phi$. This method was used by Boser, Guyon and Vapnik [9] to extend the *Generalized Portrait* hyperplane classifier of Vapnik and Chervonenkis [41] to nonlinear Support Vector machines. Aizerman et al. [1] call $\mathcal{H}$ the *linearization space*, and used it in the context of the potential function classification method to express the dot product between elements of $\mathcal{H}$ 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 [39] for $N = d = 2$. For the map

$$C_2 : ([x]_1,[x]_2) \mapsto ([x]_1^2,[x]_2^2,[x]_1[x]_2,[x]_2[x]_1), \tag{50}$$

dot products in $\mathcal{H}$ take the form

$$(C_2(x)\cdot C_2(x')) = [x]_1^2[x']_1^2 + [x]_2^2[x']_2^2 + 2[x]_1[x]_2[x']_1[x']_2 = (x\cdot x')^2, \tag{51}$$

i.e., the desired kernel $k$ is simply the square of the dot product in input space. The same works for arbitrary $N, d \in \mathbb{N}$ [9]:

**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*

$$k(x,x') = (C_d(x)\cdot C_d(x')) = (x\cdot x')^d. \tag{52}$$

*Proof.* We directly compute

$$(C_d(x)\cdot C_d(x')) = \sum_{j_1,\ldots,j_d=1}^{N} [x]_{j_1}\cdot\ldots\cdot[x]_{j_d}\cdot[x']_{j_1}\cdot\ldots\cdot[x']_{j_d} \tag{53}$$

$$= \left(\sum_{j=1}^{N}[x]_j\cdot[x']_j\right)^d = (x\cdot x')^d. \tag{54}$$

□

Instead of ordered products, we can use unordered ones to obtain a map $\Phi_d$ 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 $\Phi_d$ with the square roots of their numbers of occurence. Then, by this definition of $\Phi_d$, and (52),

$$(\Phi_d(x)\cdot\Phi_d(x')) = (C_d(x)\cdot C_d(x')) = (x\cdot x')^d. \tag{55}$$

For instance, if $p$ of the $j_i$ in (45) are equal, and the remaining ones are different, then the coefficient in the corresponding component of $\Phi_d$ is $\sqrt{(d-p+1)!}$ (for the general case, cf. [38]). For $\Phi_2$, this simply means that [39]

$$\Phi_2(x) = ([x]_1^2,[x]_2^2,\sqrt{2}\,[x]_1[x]_2). \tag{56}$$

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 (52), we take into account higher-order statistics without the combinatorial explosion (cf. (48)) of time and memory complexity which goes along already with moderately high $N$ and $d$.

Finally, note that it is possible to modify (52) such that it maps into the space of all monomials *up to* degree $d$, defining

$$k(x, x') = ((x \cdot x') + 1)^d. \tag{57}$$

## 6 Examples of Kernels

When considering feature maps, it is also possible to look at things the other way around, and start with the kernel. Given a kernel function satisfying a mathematical condition termed *positive definiteness*, it is possible to construct a feature space such that the kernel computes the dot product in that feature space. This has been brought to the attention of the machine learning community by [1], [9], and [39]. In functional analysis, the issue has been studied under the heading of *Hilbert space representations* of kernels. A good monograph on the theory of kernels is [5].

Besides (52), [9] and [39] suggest the usage of Gaussian radial basis function kernels [1]

$$k(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\,\sigma^2}\right) \tag{58}$$

and sigmoid kernels

$$k(x, x') = \tanh(\kappa(x \cdot x') + \Theta). \tag{59}$$

where $\sigma, \kappa$, and $\Theta$ are real parameters.

The examples given so far apply to the case of vectorial data. In fact it is possible to construct kernels that are used to compute similarity scores for data drawn from rather different domains. This generalizes kernel learning algorithms to a large number of situations where a vectorial representation is not readily available ([35], [20], [43]). Let us next give an example where $\mathcal{X}$ is not a vector space.

**Example 1 (Similarity of probabilistic events).** *If $\mathcal{A}$ is a $\sigma$-algebra, and $P$ a probability measure on $\mathcal{A}$, and $A$ and $B$ two events in $\mathcal{A}$, then*[2]

$$k(A, B) = P(A \cap B) - P(A)P(B) \tag{60}$$

*is a positive definite kernel.*

Further examples include kernels for string matching, as proposed by [43] and [20].

There is an analogue of the kernel trick for distances rather than dot products, i.e., dissimilarities rather than similarities. This leads to the class of *conditionally positive definite kernels*, which contain the standard SV kernels as special cases. Interestingly, it turns out that SVMs and kernel PCA can be applied also with this larger class of kernels, due to their being translation invariant in feature space [38].

[2] A $\sigma$-algebra is a type of a collection of sets which represent probabilistic events, and $P$ assigns probabilities to the events.

## 7 Applications

Having described the basics of SV machines, we now summarize some empirical findings.

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 [36]. 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 (or, in some cases, superior to) the best available classifiers on both OCR and object recognition tasks ([8], [11], [16]).

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 (37), 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 this "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 $\xi_i$ will become nonzero, and all the corresponding examples will become SVs. For this case, a decomposition algorithm was proposed [30], 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 ([33]) explores an extreme case, where the subproblems 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 [38].

Let us now discuss some SVM applications in bioinformatics. Many problems in bioinformatics involve variable selection as a subtask. Variable selection refers to the problem of selecting input variables that are most predictive of a given outcome.[3] Examples are found in diagnosis applications where the outcome may be the prediction of disease *vs.* normal [17, 29, 45, 19, 12] or in prognosis applications where the outcome may be the time of recurrence of a disease after treatment [27, 23]. The input variables of such problems may include clinical variables from medical examinations, laboratory test results, or the measurements of high throughput assays like DNA microarrays. Other examples are found in the prediction of biochemical properties such as the binding of a molecule to a drug target ([46, 7], see below). The input variables of such problems may include physico-chemical descriptors of the drug candidate molecule such as the presence or absence of chemical groups and their

[3] We make a distinction between variable and features to avoid the confusion between the input space and the feature space in which kernel machines operate.

relative position. The objectives of variable selection may be multiple: reducing the cost of production of the predictor, increasing its speed, improving its prediction performance and/or providing an interpretable model.

Algorithmically, SVMs can be combined with any variable selection method used as a filter (preprocessing) that pre-selects a variable subset [17]. However, directly optimizing an objective function that combines the original training objective function and a penalty for large numbers of variables often yields better performance. Because the number of variables itself is a discrete quantity that does not lend itself to the use of simple optimization techniques, various substitute approaches have been proposed, including training kernel parameters that act as variable scaling coefficients [45, 13]. Another approach is to minimize the $\ell_1$ norm (the sum of the absolute values of the $\mathbf{w}$ weights) instead of the $\ell_2$ norm commonly used for SVMs [27, 23, 24, 7]. The use of the $\ell_1$ norm tends to drive to zero a number of weights automatically. Similar approaches are used in statistics [38]. The authors of [44] proposed to reformulate the SVM problem as a constrained minimization of the $\ell_0$ "norm" of the weight vector $\mathbf{w}$ (i.e., the number of nonzero components). Their algorithm amounts to performing multiplicative updates leading to the rapid decay of useless weights. Additionally, classical wrapper methods used in machine learning [22] can be applied. These include greedy search techniques such as backward elimination that was introduced under the name SVM RFE [19, 34].

SVM applications in bioinformatics are not limited to ones involving variable selection. One of the earliest applications was actually in sequence analysis, looking at the task of translation initiation site (TIS) recognition. It is commonly believed that only parts of the genomic text code for proteins. Given a piece of DNA or mRNA sequence, it is a central problem in computational biology to determine whether it contains coding sequence. The beginning of coding sequence is referred to as a TIS. In [48], an SVM is trained on neighbourhoods of ATG triplets, which are potential start codons. The authors use a polynomial kernel which takes into account nonlinear relationships between nucleotides that are spatially close. The approach significantly improves upon competing neural network based methods.

Another important task is the prediction of gene function. The authors of [10] argue that SVMs have many mathematical features that make them attractive for such an analysis, including their flexibility in choosing a similarity measure, sparseness of solution when dealing with large datasets, the ability to handle large feature spaces, and the possibility to identify outliers. Experimental results show that SVMs outperform other classification techniques (C4.5, MOC1, Parzen windows and Fisher's linear discriminant) in the task of identifying sets of genes with a common function using expression data. In [32] this work is extended to allow SVMs to learn from heterogeneous data: the microarray data is supplemented by phylogenetic profiles. Phylogenetic profiles measure whether a gene of interest has a close homolog in a corresponding genome, and hence such a measure can capture whether two genes are similar on the sequence level, and whether they have a similar pattern of occurence of their homologs across species, both factors indicating a functional link. The authors show how a type of kernel combination and a type of feature scaling can help improve performance in using these data types together, resulting in improved performance over using a more naive combination method, or only a single type of data.

Another core problem in statistical bio-sequence analysis is the annotation of new protein sequences with structural and functional features. To a degree, this can be achieved by relating the new sequences to proteins for which such structural properties are already known. Although numerous techniques have been applied to this problem with some success, the

detection of remote protein homologies has remained a challenge. The challenge for SVM researchers in applying kernel techniques to this problem is that standard kernel functions work for fixed length vectors and not variable length sequences like protein sequences. In [21] an SVM method for detecting remote protein homologies was introduced and shown to outperform the previous best method, a Hidden Markov Model (HMM) in classifying protein domains into super-families. The method is a variant of SVMs using a new kernel function. The kernel function (the so-called Fisher kernel) is derived from a generative statistical model for a protein family; in this case, the best performing HMM. This general approach of combining generative models like HMMs with discriminative methods such as SVMs has applications in other areas of bioinformatics as well, such as in promoter region-based classification of genes [31]. Since the work of Jaakkola *et al.* [21], other researchers have investigated using SVMs in various other ways for the problem of protein homology detection. In [26] the Smith-Waterman algorithm, a method for generating pairwise sequence comparison scores, is employed to encode proteins as fixed length vectors which can then be fed into the SVM as training data. The method was shown to outperform the Fisher kernel method on the SCOP 1.53 database. Finally, another interesting direction of SVM research is given in [25] where the authors employ string matching kernels first pioneered by [43] and [20] which induce feature spaces directly from the (variable length) protein sequences.

There are many other important application areas in bioinformatics, only some of which have been tackled by researchers using SVMs and other kernel methods. Some of these problems are waiting for practitioners to apply these methods. Other problems remain difficult because of the scale of the data or because they do not yet fit into the learning framework of kernel methods. It is the task of researchers in the coming years to develop the algorithms to make these tasks solvable. We conclude this survey with three case studies.

**Lymphoma Feature Selection** As an example of variable selection with SVMs, we show results on DNA microarray measurements performed on lymphoma tumors and normal tissues [2, 12]. The dataset includes 96 tissue samples (72 cancer and 24 non-cancer) for which 4026 gene expression coefficients were recorded (the input variables). A simple preprocessing (standardization) was performed and missing values were replaced by zeros. The dataset was split into training and test set in various proportions and each experiment was repeated on 96 different splits. Variable selection was performed with the RFE algorithm [19] by removing genes with smallest weights and retraining repeatedly. The gene set size was decreased logarithmically, apart from the last 64 genes,which were removed one at a time. In Figure 4, we show the learning curves when the number of genes varies in the gene elimination process.

For comparison, we show in Figure 5 the results obtained by a competing technique [18] that uses a correlation coefficient to rank order genes. Classification is performed using the top ranking genes each contributing to the final decision by voting according to the magnitude of their correlation coefficient. Other comparisons with a number of other methods including Fisher's discriminant, decision trees, and nearest neighbors have confirmed the superiority of SVMs [19, 12, 34].

**KDD Cup: Thrombin Binding** The *Knowledge Discovery and Data Mining (KDD)* is the premier international meeting of the data mining community. It holds an annual competition, called the *KDD Cup* (http://www.cs.wisc.edu/∼dpage/kddcup2001/), consisting of several

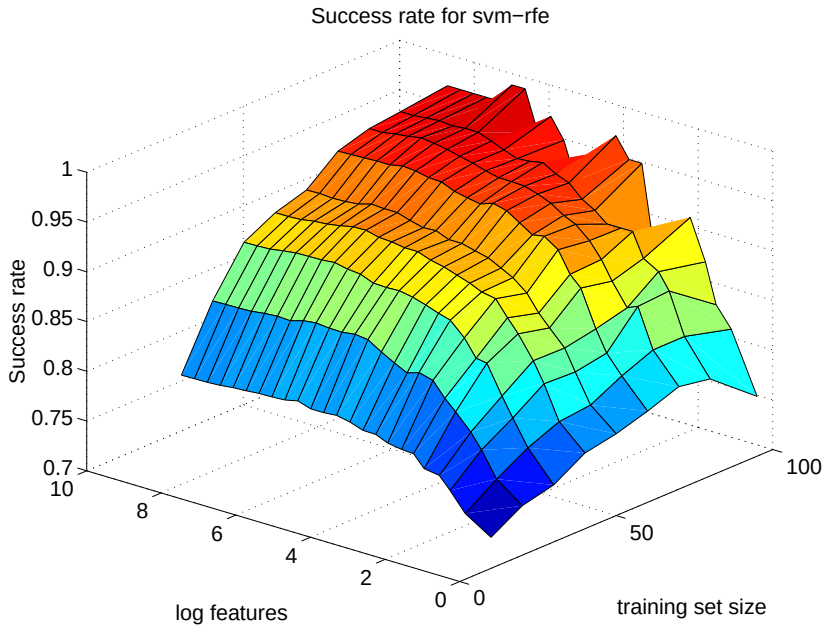


Figure 4: Variable selection performed by the SVM RFE method. The success rate is represented as a function of the training set size and the number of genes retained in the gene elimination process.

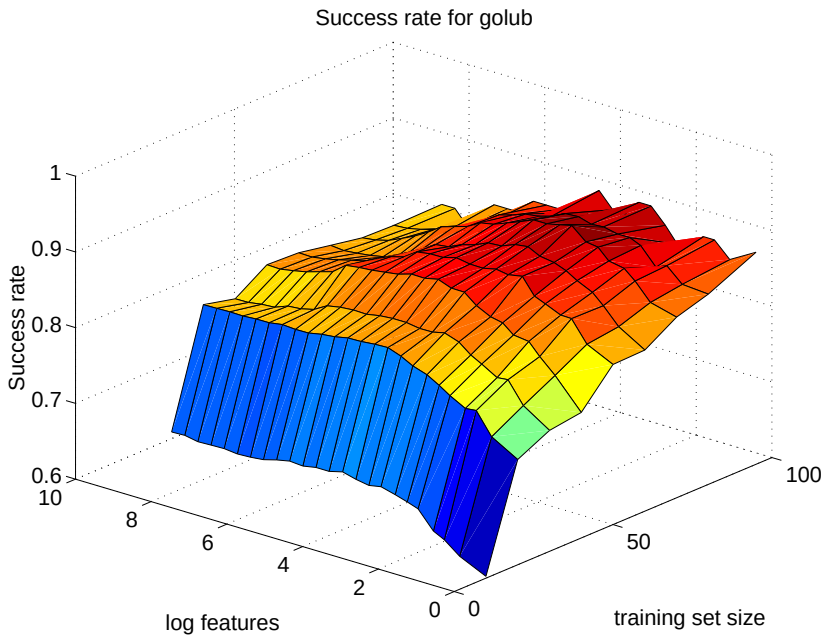


Figure 5: Variable selection performed by the S2N method. The success rate is represented as a function of the training set size and the number of top ranked genes used for classification.

datasets to be analyzed. One of the tasks in the 2001 competition was to predict binding of compounds to a target site on Thrombin (a key receptor in blood clotting). Such a predictor can be used to speed up the drug design process. The input data, which was provided by DuPont, consists of 1909 binary feature vectors of dimension 139351, which describe three-dimensional properties of the respective molecule. For each of these feature vectors, one is additionally given the information whether it binds or not. As a test set, there are 636 additional compounds, represented by the same type of feature vectors. Several characteristics of the dataset render the problem hard: there are very few positive training examples, but a very large number of input features, and rather different distributions between training and test data. The latter is due to test molecules being compounds engineered based on previous (training set) results.

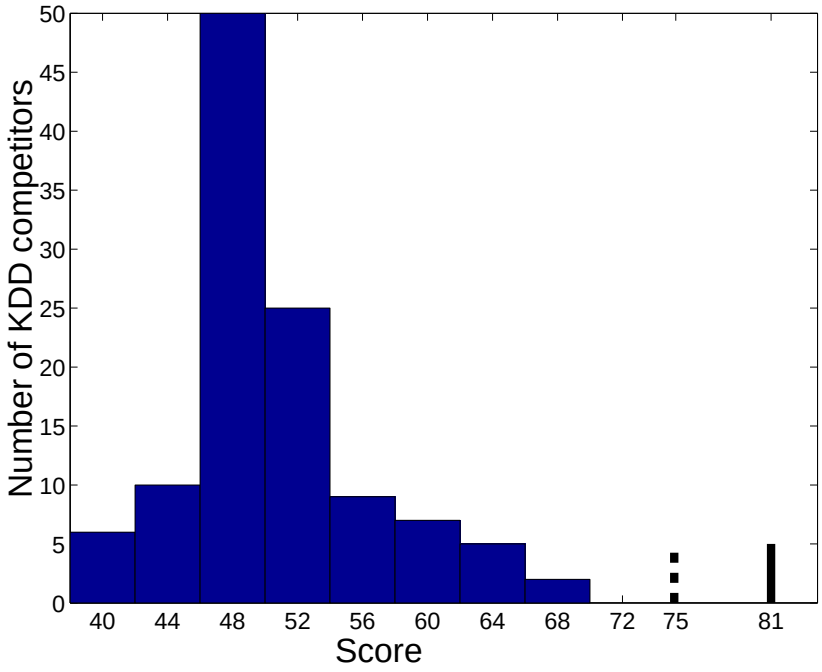


Figure 6: Results on the KDD cup Thrombin binding problem. Bar plot histogram of all entries in the competition (e.g., the bin labelled '68' gives the number of competition entries with performance in the range from 64 to 68), as well as results from [46] using inductive (dashed line) and transductive (solid line) feature selection systems.

There were more than 100 entries in the KDD cup for the Thrombin dataset alone, with the winner achieving a performance score of 68%. After the competition took place, using a type of correlation score designed to cope with the small number of positive examples to perform feature selection combined with an SVM, a 75% success rate was obtained [46]. See Figure 6 for an overview of the results of all entries to the competition, as well as the results of [46]. This result was improved further by modifying the SVM classifier to adapt to the distribution of the unlabeled test data. To do this, the so-called transductive setting was employed, where (unlabeled) test feature vectors are used in the training stage (this is possible if during training it is already known for which compounds we want to predict whether they bind or not.) This method achieved a 81% success rate. It is noteworthy that these results were obtained selecting a subset of only 10 of the 139351 features, thus the solutions can provide not only prediction accuracy but also a determination of the crucial properties of a compound with respect to its binding activity.

## 8 Conclusion

One of the most appealing features of kernel algorithms is the solid foundation provided by both statistical learning theory and functional analysis. Kernel methods let us interpret (and design) learning algorithms geometrically in feature spaces nonlinearly related to the input space, and combine statistics and geometry in a promising way. This theoretical elegance is also matched by their practical performance. SVMs and other kernel methods have yielded promising results in the field of bioinformatics, and we anticipate that the popularity of machine learning techniques in bioinformatics is still increasing. It is our hope that this combination of theory and practice will lead to further progress in the future for both fields.

## References


[1] M. A. Aizerman, É. 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] A. A. Alizadeh et al. Distinct types of diffuse large b-cell lymphoma identified by gene expression profiling. *Nature*, 403:503–511, 2000. Data available from http://llmpp.nih.gov/lymphoma.

[3] 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.

[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] J. Bi, K. P. Bennett, M. Embrechts, and C. Breneman. Dimensionality reduction via sparse support vector machine. In *NIPS'2001 Workshop on Variable and Feature Selection*, 2001. Slides available at http://www.clopinet.com/isabelle/Projects/NIPS2001/bennett-nips01.ppt.gz.

[8] 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.

[9] B. E. Boser, I. M. Guyon, and V. 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.

[10] M. P. S. Brown, W. N. Grundy, D. Lin, N. Cristianini, C. Sugnet, T. S. Furey, M. Ares, and D. Haussler. Knowledge-based analysis of microarray gene expression data using support vector machines. *Proceedings of the National Academy of Sciences*, 97(1):262–267, 2000.

[11] 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.

[12] J. Cai, A. Dayanik, N. Hasan, T. Terauchi, and H. Yu. Supervised machine learning algorithms for classification of cancer tissue types using microarray gene expression data. Technical report, Columbia University, 2001. http://www.cpmc.columbia.edu/homepages/jic7001/cs4995/project1.htm.

[13] O. Chapelle and J. Weston. Feature selection for non-linear SVMs using a gradient descent algorithm. In *NIPS'2001 workshop on Variable and Feature Selection*, 2001.

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

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

[16] D. DeCoste and B. Schölkopf. Training invariant support vector machines. *Machine Learning*, 46:161–190, 2002. Also: Technical Report JPL-MLTR-00-1, Jet Propulsion Laboratory, Pasadena, CA, 2000.

[17] T. S. Furey, N. Duffy, N. Cristianini, D. Bednarski, M. Schummer, and D. Haussler. Support vector machine classification and validation of cancer tissue samples using microarray expression data. *Bioinformatics*, 16(10):906–914, 2000.

[18] T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, and E. S. Lander. Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science*, 286:531–537, 1999.

[19] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46:389–422, 2002.

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

[21] T. S. Jaakkola, M. Diekhans, and D. Haussler. A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology*, 7:95–114, 2000.

[22] R. Kohavi and G. John. Wrappers for feature selection. *Artificial Intelligence*, 97:12:273–324, 1997.

[23] Yuh-Jye Lee, O. L. Mangasarian, and W. H. Wolberg. Breast cancer survival and chemotherapy: A support vector machine analysis. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 55:1–20, 2000.

[24] Yuh-Jye Lee, O. L. Mangasarian, and W. H. Wolberg. Survival-time classification of breast cancer patients. Technical Report 01-03, Data Mining Institute, March 2001. Data link ftp://ftp.cs.wisc.edu/math-prog/cpo-dataset/machine-learn/WPBCC/.

[25] C. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: A string kernel for SVM protein classification. *Proceedings of the Pacific Symposium on Biocomputing*, 2002.

[26] Liao, Li, and W. S. Noble. Combining pairwise sequence similarity and support vector machines for remote protein homology detection. In *Proceedings of the Sixth Annual International Conference on Research in Computational Molecular Biology*, 2002.

[27] O. L. Mangasarian, W. Nick Street, and W. H. Wolberg. Breast cancer diagnosis and prognosis via linear programming. *Operations Research*, 43:570–577, 1995.

[28] J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society, London*, A 209:415–446, 1909.

[29] S. Mukherjee, P. Tamayo, D. Slonim, A. Verri, T. Golub, J. P. Mesirov, and T. Poggio. Support vector machine classification of microarray data. Technical report, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 2000.

[30] 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.

[31] P. Pavlidis, T. S. Furey, M. Liberto, and W. N. Grundy. Promoter region-based classification of genes. *Proceedings of the Pacific Symposium on Biocomputing*, pages 151–163, 2001.

[32] P. Pavlidis, J. Weston, J. Cai, and W. N. Grundy. Learning gene functional classifications from multiple data types. *Journal of Computational Biology*, 2002.

[33] 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.

[34] S. Ramaswamy et al. Multiclass cancer diagnosis using tumor gene expression signatures. *Proceedings of the National Academy of Science*, 98:15149–15154, 2001.

[35] B. Schölkopf. *Support Vector Learning*. R. Oldenbourg Verlag, München, 1997. Doktorarbeit, Technische Universität Berlin. Available from http://www.kyb.tuebingen.mpg.de/∼bs.

[36] 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.

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

[38] B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

[39] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, NY, 1995.

[40] V. Vapnik. *Statistical Learning Theory*. Wiley, NY, 1998.

[41] 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).

[42] V. Vapnik and A. Lerner. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24:774–780, 1963.

[43] 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.

[44] J. Weston, A. Elisseeff, and B. Schölkopf. Use of the $\ell_0$-norm with linear models and kernel methods. Technical report, Biowulf Technologies, New York, 2001.

[45] J. Weston, S. Mukherjee, O. Chapelle, M. Pontil, T. Poggio, and V. Vapnik. Feature selection for SVMs. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press, Cambridge, MA, 2000.

[46] J. Weston, F. Pérez-Cruz, O. Bousquet, O. Chapelle, A. Elisseeff, and B. Schölkopf. KDD cup 2001 data analysis: prediction of molecular bioactivity for drug design – binding to thrombin. Technical report, BIOwulf, 2001. http://www.biowulftech.com/people/jweston/kdd/kdd.html.

[47] R. C. Williamson, A. J. Smola, and B. Schölkopf. Generalization performance of regularization networks and support vector machines via entropy numbers of compact operators. *IEEE Transactions on Information Theory*, 47(6):2516–2532, 2001.

[48] A. Zien, G. Rätsch, S. Mika, B. Schölkopf, T. Lengauer, and K.-R. Müller. Engineering support vector machine kernels that recognize translation initiation sites. *Bioinformatics*, 16(9):799–807, 2000.