Kernel Methods and Support Vector Machines

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

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

June 23, 2003

1 Introduction

Over the past ten years kernel methods such as Support Vector Machines and Gaussian Processes have become a staple for modern statistical estimation and machine learning. The groundwork for this field was laid in the second half of the 20th century by Vapnik and Chervonenkis (geometrical formulation of an optimal separating hyperplane, capacity measures for margin classifiers), Mangasarian (linear separation by a convex function class), Aronszajn (Reproducing Kernel Hilbert Spaces), Aizerman, Braverman, and Rozonoér (nonlinearity via kernel feature spaces), Arsenin and Tikhonov (regularization and ill-posed problems), and Wahba (regularization in Reproducing Kernel Hilbert Spaces).

However, it took until the early 90s until positive definite kernels became a popular and viable means of estimation. Firstly this was due to the lack of sufficiently powerful hardware, since kernel methods require the computation of the so-called kernel matrix, which requires quadratic storage in the number of data points (a computer of at least a few megabytes of memory is required to deal with 1000+ points). Secondly, many of the previously mentioned techniques lay dormant or existed independently and only recently the (in hindsight obvious) connections were made to turn this into a practical estimation tool. Nowadays, a variety of good reference books exist and anyone serious about dealing with kernel methods is recommended to consult one of the following works for further information [15, 5, 8, 12]. Below, we will summarize the main ideas of kernel method and support vector machines, building on the summary given in [13].

2 Learning from Data

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

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

and we want to estimate a *decision function* $f : \mathcal{X} \to \{\pm 1\}$. Here, $\mathcal{X}$ is some nonempty set from which the *patterns* x_i are taken, usually referred to as the *domain*; the y_i are called *labels* or *targets*. A good decision function will have the property that it *generalizes* to unseen data points, achieving a small value of the *risk*

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

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

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

We therefore introduce an additional type of structure, pertaining to what we are actually given — the training data. Loosely speaking, to generalize, we want to choose y such that (x, y) is in some sense similar to the training examples (1). To this end, we need notions of *similarity* in $\mathcal{X}$ and in $\{\pm 1\}$. Characterizing the similarity of the outputs $\{\pm 1\}$ is easy: in binary classification, only two situations can occur: two labels can either be identical or different. The choice of the similarity measure for the inputs, on the other hand, is a deep question that lies at the core of the problem of machine learning.

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

3 Kernels

Let us consider a symmetric similarity measure of the form

$$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}, \text{ where } (x, x') \mapsto k(x, x'),$$

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

*Corresponding Author

3.1 Kernels as Similarity Measures

General similarity measures of this form are rather difficult to study. Let us therefore start from a particularly simple case, the *dot product* $\langle \mathbf{x}, \mathbf{x}' \rangle$, and generalize it subsequently.

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

$$\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}. \tag{3}$$

Being able to compute dot products amounts to being able to carry out all geometric constructions that can be formulated in terms of angles, lengths and distances. However, this is not really sufficiently general to deal with many interesting problems.

- First, we have deliberately not made the assumption that the patterns actually exist in a dot product space (they could be any kind of object). We therefore first need to represent the patterns as vectors in some dot product space $\mathcal{H}$, called the *feature space* using a map

$$\Phi : \mathcal{X} \to \mathcal{H} \text{ where } x \mapsto \mathbf{x} := \Phi(x). \tag{4}$$

  Note that we use a boldface $\mathbf{x}$ to denote the vectorial representation of x in the feature space.
- Second, even if the original patterns lie in a dot product space, we may still want to consider more general similarity measures obtained by applying the map (4).

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

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

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

3.2 Examples of Kernels

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

$$k(x, x') = \langle x, x' \rangle^d , \text{ where } d \in \mathbb{N}. \tag{6}$$

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

$$k(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\,\sigma^2} \right), \tag{7}$$

with a suitable width $\sigma > 0$. Examples of more sophisticated kernels, defined not on dot product spaces but on discrete objects such as strings, are the string matching kernels proposed by [16] and [7].

In general, there are several ways of deciding whether a given function k qualifies as a valid kernel. One way is to appeal to *Mercer's theorem*. This classical result of functional analysis states that the kernel of a positive definite integral operator can be diagonalized in terms of an eigenvector expansion with nonnegative eigenvalues. From the expansion, the feature map Φ can explicitly be constructed. Another approach exploits the fact that k is the kernel of a *Reproducing Kernel Hilbert Space*. See [12] for references and details.

4 Support Vector Classifiers

Statistical Learning Theory shows that it is imperative to restrict the set of functions from which f is chosen to one that has a *capacity* suitable for the amount of available training data. It provides *bounds* on the test error, depending on both the empirical risk and the capacity of the function class. The minimization of these bounds leads to the principle of *structural risk minimization* [15].

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

$$R_{\text{emp}}[f] = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} |f(x_i) - y_i|, \tag{8}$$

and a capacity term derived for the class of hyperplanes in a dot product space $\mathcal{H}$ [15],

$$\langle \mathbf{w}, \mathbf{x} \rangle + b = 0 \text{ where } \mathbf{w} \in \mathcal{H}, b \in \mathbb{R}, \tag{9}$$

corresponding to decision functions

$$f(\mathbf{x}) = \operatorname{sgn}\left(\langle \mathbf{w}, \mathbf{x} \rangle + b \right). \tag{10}$$

4.1 Hard Margin Solution

Consider first problems which are linearly separable. There exists a unique *optimal hyperplane* [15], distinguished by the maximum margin of separation between any training point and the hyperplane. It is the solution of

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

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

One can see from Figure 1 that in order to construct the optimal hyperplane, we need to solve

$$\begin{array}{ll} \underset{\mathbf{w} \in \mathcal{H}, b \in \mathbb{R}}{\text{minimize}} & \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{subject to} & y_i (\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 \text{ for all } i = 1, \ldots, m. \end{array} \tag{11}$$

Note that the constraints ensure that $f(\mathbf{x}_i)$ will be $+1$ for $y_i = +1$, and -1 for $y_i = -1$.[1]

[1] One might argue that for this to be the case, we don't actually need the constraint "≥ 1". However, without it, it would not be meaningful to minimize the length of $\mathbf{w}$: to see this, imagine we wrote "> 0" instead of "≥ 1." Now assume that the solution is $(\mathbf{w}, b)$. Let us rescale this solution by multiplication with some $0 < \lambda < 1$. Since $\lambda > 0$, the constraints are still satisfied. Since $\lambda < 1$, however, the length of $\mathbf{w}$ has decreased. Hence $(\mathbf{w}, b)$ cannot be the minimizer of (11).

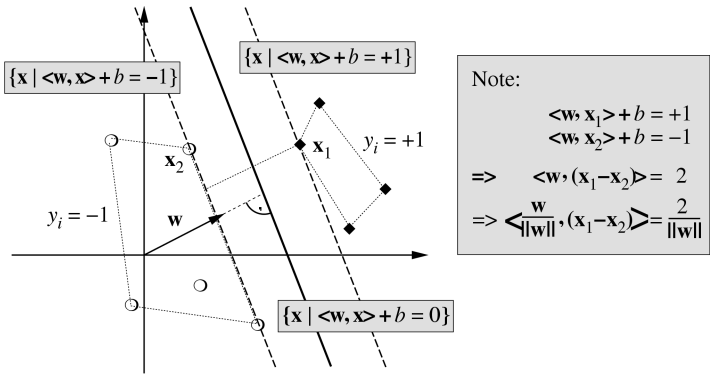

Figure 1: A binary classification toy problem: separate balls from diamonds. The *optimal hyperplane* is shown as a solid line. The problem being separable, there exists a weight vector $\mathbf{w}$ and a threshold b such that $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) > 0$ ($i = 1, \ldots, m$). Rescaling $\mathbf{w}$ and b such that the point(s) closest to the hyperplane satisfy $|\langle \mathbf{w}, \mathbf{x}_i \rangle + b| = 1$, we obtain a *canonical* form $(\mathbf{w}, b)$ of the hyperplane, satisfying $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1$. Note that in this case, the *margin* (the distance of the closest point to the hyperplane) equals $1/\|\mathbf{w}\|$. This can be seen by considering two points $\mathbf{x}_1, \mathbf{x}_2$ on opposite sides of the margin, that is, $\langle \mathbf{w}, \mathbf{x}_1 \rangle + b = 1$, $\langle \mathbf{w}, \mathbf{x}_2 \rangle + b = -1$, and projecting them onto the hyperplane normal vector $\mathbf{w}/\|\mathbf{w}\|$ (from [12]).

The constrained optimization problem (11) is dealt with by introducing *Lagrange multipliers* $\alpha_i \geq 0$ ($\boldsymbol{\alpha} := (\alpha_1, \ldots, \alpha_m)$) and a *Lagrangian*

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{m} \alpha_i \left(y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) - 1 \right). \quad (12)$$

L has a *saddle point* in $\mathbf{w}, b$ and $\boldsymbol{\alpha}$ at the optimal solution of the primal optimization problem. This means that it should be minimized with respect to the *primal variables* $\mathbf{w}$ and b and maximized with respect to the *dual variables* α_i. Furthermore, the product between constraints and Lagrange multipliers in L vanish at optimality, that is

$$\alpha_i(y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) - 1) = 0 \text{ for all } i = 1, \ldots, m. \quad (13)$$

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

$$\frac{\partial}{\partial b} L(\mathbf{w}, b, \boldsymbol{\alpha}) = -\sum_{i=1}^{m} \alpha_i y_i = 0 \quad (14)$$

$$\frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}, b, \boldsymbol{\alpha}) = \mathbf{w} - \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i = 0 \quad (15)$$

The solution thus has an expansion (15) in terms of a subset of the training patterns, namely those patterns with non-zero α_i, called *Support Vectors (SVs)*. Often, only few of the training examples actually end up being SVs.

By the *Karush-Kuhn-Tucker conditions* (13) known from optimization theory, the SVs lie on the margin (cf. Figure 1) — this can be exploited to compute b once the α_i have been found. All remaining training examples $(\mathbf{x}_j, y_j)$ are irrelevant: their constraint $y_j(\langle \mathbf{w}, \mathbf{x}_j \rangle + b) \geq 1$ could just as well be left out. In other words, the hyperplane is completely determined by the patterns closest to it.

By substituting (14) and (15) into the Lagrangian (12), one eliminates the primal variables $\mathbf{w}$ and b, arriving at the so-called *dual optimization problem*, which is the problem usually solved in practice:

$$\begin{aligned} \underset{\boldsymbol{\alpha} \in \mathbb{R}^m}{\text{maximize}} \quad & \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j K_{ij} \\ \text{subject to} \quad & \alpha_i \geq 0 \text{ for all } i = 1, \ldots, m \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0. \end{aligned} \quad (16)$$

where $K_{ij} := \langle \mathbf{x}_i, \mathbf{x}_j \rangle$. Using (15), the decision function (10) can thus be written as

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{m} y_i \alpha_i \langle \mathbf{x}, \mathbf{x}_i \rangle + b \right), \quad (17)$$

where b is computed via (13). For details, see [15, 5, 12, 8].

4.2 The Kernel Trick

We now have all the tools to describe SVMs. Everything above was formulated in a dot product space, which we think of as the feature space $\mathcal{H}$ (see (4)). To express the formulae in terms of the input patterns in $\mathcal{X}$, we employ (5) and replace $\langle \mathbf{x}, \mathbf{x}' \rangle$ by $k(\mathbf{x}, \mathbf{x}')$ wherever it occurs. This substitution, which is sometimes referred to as the *kernel trick*, was used by Boser et al. [3] to develop nonlinear SVMs. Now f can be rewritten as

$$f(x) = \operatorname{sgn}\left(\sum_{i=1}^{m} y_i \alpha_i k(x, x_i) + b \right). \quad (18)$$

Furthermore, in the quadratic program (16) the definition of K_{ij} becomes $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. Figure 2 shows a toy example.

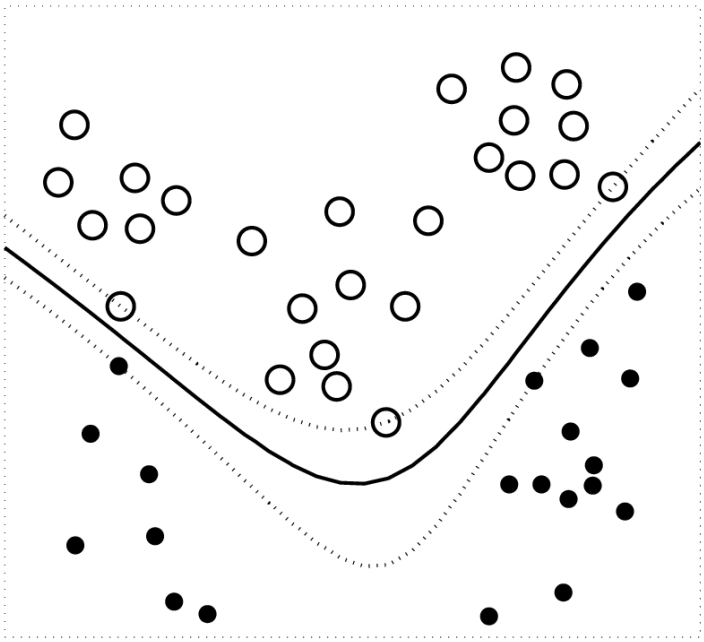

Figure 2: Example of an SV classifier found using a radial basis function kernel $k(x, x') = \exp(-\|x - x'\|^2)$. Circles and points are two classes of training examples; the middle line is the decision surface; the outer lines precisely meet the constraint of (11). Note that the SVs found by the algorithm (sitting on the dotted constraint lines) are not centers of clusters, but examples which are critical for the given classification task (from [13]).

4.3 Soft Margin Solution

In practice, a separating hyperplane may not exist, e.g., if a high noise level causes a large overlap of the classes. To accommodate this case, one introduces slack variables $\xi_i \geq 0$ for all $i = 1, \ldots, m$ in order to relax the constraints of (11) to

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \xi_i \text{ for all } i = 1, \ldots, m. \tag{19}$$

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

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

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m} \xi_i \tag{20}$$

subject to the constraints on ξ_i and (19), where the constant $C > 0$ determines the trade-off between margin maximization and training error minimization. This again leads to the problem of maximizing (16), subject to modified constraint where the only difference from the separable case is an upper bound C on the Lagrange multipliers α_i.

Another realization uses the more natural ν-parametrization. In it, the parameter C is replaced by a parameter $\nu \in (0, 1]$ which can be shown to provide lower and upper bounds for the fraction of examples that will be SVs and those that will have non-zero slack variables, respectively. Its dual can be shown to consist in maximizing the quadratic part of (16), subject to

$$0 \leq \alpha_i \leq 1/(\nu m), \ \sum_i \alpha_i y_i = 0, \text{ and } \sum_i \alpha_i = 1.$$

5 Discussion

5.1 Extensions

The applicability of the "kernel trick" extends significantly beyond the classification setting and in recent years a large number of kernel algorithms have been proposed to solve as diverse tasks as the estimation of the support (or, more generally, quantiles) of a distribution, of a regression function, or of a nonlinear manifold. Below we give a brief overview of the most popular methods:

Regression: Just as classification can be formulated as a quadratic optimization problem, so can regression. Here, the maximum margin condition is replaced by the requirement of finding the *flattest* function which performs a regression within ϵ deviation from the observations.

Principal Component Analysis: It can be extended to nonlinear settings by replacing PCA in input space by a feature space representation. The final algorithm consists of solving an eigenvector problem for the kernel matrix.

Similar modifications can be carried out to obtain nonlinear versions of projection pursuit, e.g., via *sparse kernel feature analysis*.

Independent Component Analysis: Recently, an algorithm was suggested in [2] to find independent components via a modification of canonical correlation analysis. This is currently an active topic of research and it is likely to lead to novel criteria for factorizing distributions.

Quantiles of a Distribution: In this problem one attempts to find sets such that the probability of data occurring outside this set is controlled. This is done by ensuring that the set contains a certain fraction of the training data while at the same time keeping the set "simple" (where simplicity is determined by an SVM-style regularization term). This can be done also for high-dimensional problems, and one can show that it can be cast as a classification problem with only one class. Kernel extensions exist.

Estimation of Manifolds: Here one aims at finding smooth manifolds which approximate a dataset. Again, one can find an optimization problems similar to the SV optimization problem (i.e., a regularization term plus a misprediction cost) and generate a kernel expansion.

These and many more kernel methods plus the corresponding references can be found in [12].

5.2 Implementations

An initial weakness of SVMs was that the size of the quadratic programming problem scaled with the number of SVs. This was due to the fact that in (16), the quadratic part contained at least all SVs — the common practice was to extract the SVs by going through the training data in chunks while regularly testing for the possibility that patterns initially not identified as SVs become SVs at a later stage. This procedure is referred to as *chunking*; note that without chunking, the size of the matrix in the quadratic part of the objective function would be $m \times m$, where m is the number of all training examples.

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

5.3 Empirical Results and Applications

Modern SVM implementations made it possible to train on some rather large problems. Success stories include the 60,000 example MNIST digit recognition benchmark (with record results), as well as problems in text categorization and bioinformatics, where two main areas of application are worth mentioning:

Firstly there are classification and gene selection problems in DNA microarray analysis. Given the high dimensionality of the data to begin with, the use of kernels is not advisable in this case. Instead, a linear classifier with a suitable penalty on the expansion coefficients favoring sparse expansions is found. See [4, 6] for further details and references. Finding suitable variable selection criteria is an active area of research.

Secondly, sequence analysis can often be cast into the form of a classification problem, requiring the design of custom tailored kernels for this purpose. Such research has led to excellent results (see [9, 7**?** , 16, 14, 10] and the references therein for further details).

5.4 Conclusion

During the last few years, SVMs and other kernel methods have rapidly advanced into the standard toolkit of tech-

niques for machine learning and high-dimensional data analysis. This was probably due to a number of advantages compared to neural networks, such as the absence of spurious local minima in the optimization procedure, the fact that there are only few parameters to tune, enabling fast deployment in applications, the modularity in the design, where various kernels can be combined with a number of different learning algorithms, and the excellent performance on high-dimensional data.

References

[1] C. Ambroise and G.J. McLachlan. Selection bias in gene extraction on the basis of microarray gene-expression data. *Proc. Natl. Acad. Sci. USA*, 99(10):6562–6566, 2002.

[2] F. R. Bach and M. I. Jordan. Kernel independent component analysis. *Journal of Machine Learning Research*, 3:1–48, 2002.

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

[4] 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.

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

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

[7] D. Haussler. Convolutional kernels on discrete structures. Technical Report UCSC-CRL-99-10, Computer Science Department, UC Santa Cruz, 1999.

[8] R. Herbrich. *Learning Kernel Classifiers: Theory and Algorithms*. MIT Press, 2002.

[9] T. S. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems 11*, pages 487–493, Cambridge, MA, 1999. MIT Press.

[10] C. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: A string kernel for SVM protein classification. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 564–575, 2002.

[11] D. Meyer, F. Leisch, and K. Hornik. The support vector machine under test. *Neurocomputing*, 2003. Forthcoming.

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

[13] B. Schölkopf and A. J. Smola. Support vector machines. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 1119–1125. MIT Press, 2nd edition, 2003.

[14] K. Tsuda, T. Kin, and K. Asai. Marginalized kernels for biological sequences. *Bioinformatics*, 18 (Suppl. 2):S268–S275, 2002.

[15] V. Vapnik. *Statistical Learning Theory*. John Wiley and Sons, New York, 1998.

[16] 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.

[17] 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, September 2000.