Support Vector Machines

Konrad Rieck[1], Sören Sonnenburg[1], Sebastian Mika[2], Christin Schäfer[3], Pavel Laskov[4], David Tax[5], and Klaus-Robert Müller[1]

[1] Berlin Institute of Technology, Franklinstraße 28/29, 10587 Berlin, Germany
[2] idalab GmbH, Mohrenstraße 63, 10117 Berlin, Germany
[3] Fraunhofer Institute FIRST, Kekuléstraße 7, 12489 Berlin, Germany
[4] University of Tübingen, Sand 1, 72076 Tübingen, Germany
[5] Delft University of Technology, Mekelweg 4, 2628 CD Delf, Netherlands

1 Introduction

In this chapter we introduce basic concepts and ideas of the Support Vector Machines (SVM). In the first section we formulate the learning problem in a statistical framework. A special focus is put on the concept of consistency, which leads to the principle of structural risk minimization (SRM). Application of these ideas to classification problems brings us to the basic, linear formulation of the SVM, described in Section 3. We then introduce the so called 'kernel trick' as a tool for building a non-linear SVM as well as applying an SVM to non-vectorial data (Section 4). The practical issues of implementation of the SVM training algorithms and the related optimization problems are the topic of Section 5. Extensions of the SVM algorithms for the problems of non-linear regression and novelty detection are presented in Section 6. A brief description of the most successful applications of the SVM is given in Section 7. Finally, in the last Section 8 we summarize the main ideas of the chapter.

2 Learning from Examples

2.1 General Setting of Statistical Learning

The main objective of statistical learning is to find a description of an unknown dependency between measurements of objects and certain properties of these objects. The measurements, to be also called "input variables", are assumed to be observable in all objects of interest. On the contrary, the objects' properties, or "output variables", are in general available only for a small subset of objects known as *examples*. The purpose of estimating the dependency between the input and output variables is to be able to determine the values of output variables for any object of interest.

The problem of estimating an unknown dependency occurs in various practical applications. For example, the input variables can be the prices for a set of stocks and the output variable the direction of change in a certain stock price. As another example, the input can be some medical parameters and the output the probability of a patient having a certain disease. An essential feature of statistical learning is that the information is assumed to be contained in a limited set of examples (the sample), and the estimated dependency should be as accurate as possible for *all* objects of interest.

To proceed with a formal description of main properties of statistical learning, let us fix some notation. Let $\mathcal{X}$ denote the space of input variables representing the objects, and let $\mathcal{Y}$ be the space of output variables. The structure of $\mathcal{Y}$ defines the learning task. For example, if $\mathcal{Y} = \mathbb{R}$, the learning amounts to a regression problem, for $\mathcal{Y} = \{1, 2, 3\}$, the task is a classification problem with three classes, etc.

Let $\mathcal{Z} = \{(\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y} \mid i = 1, \ldots, M\}$ be a given sample. We assume that there exists some unknown but fixed probability distribution $P(X, Y)$ over the space $\mathcal{X} \times \mathcal{Y}$ generating our data; that is, $(\mathbf{x}_i, y_i) \in \mathcal{Z}$ are drawn identically and independently from $P(X, Y)$.

The dependency to be estimated takes the form of a function $f : \mathcal{X} \to \mathcal{Y}$. To decide which of many possible functions best describes the dependency observed in the training sample, we introduce the concept of a *loss function*:

$$\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}. \tag{1}$$

Such a loss function should be bounded from below and should measure the cost $\ell(f(\mathbf{x}), y)$ of discrepancy between the predicted value $f(\mathbf{x}) \in \mathcal{Y}$ and the true value $y \in \mathcal{Y}$. Then the *risk*, i.e. the expected loss incurred from using a particular prediction function f, can be defined as:

$$\mathrm{R}(f) = \mathbb{E}_P[\ell(f(\mathbf{x}), y)], \tag{2}$$

where $\mathbb{E}_P$ denotes the expectation with respect to the joint distribution $P(X, Y)$ of input and output variables.

Notice that, if we would know the joint distribution $P(X, Y)$, the learning problem can be easily solved. For example, in the classification case one could calculate the conditional probability $P(Y|X)$ and compute the so called "Bayes-optimal solution":

$$f^*(\mathbf{x}) = \operatorname*{argmax}_{y_1 \in \mathcal{Y}} \int_{y_2 \in \mathcal{Y}} \ell(y_1, y_2)\ P(Y = y_2 | X = \mathbf{x}). \tag{3}$$

However, in our setup $P(X, Y)$ is unknown, and only a sample $\mathcal{Z}$ is available. One possible solution would be to estimate $P(X, Y)$ or $P(Y|X)$ from the sample $\mathcal{Z}$. In many theoretical and practical approaches the inference is carried out exactly in this way (Duda et al., 2001; Bishop, 1995; Devroye et al., 1996). But it is also well known that estimating a density from empirical

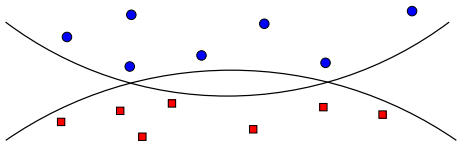

Fig. 1. Two functions that separate two classes of data points with zero empirical risk. Without further information it is impossible to decide for one of them.

data is a hard problem, especially in the multi-dimensional case. The number of examples one needs in order to get a reliable estimate of a density in N dimensions grows exponentially with N—a well-known difficulty denoted as *curse of dimensionality*.

In the approach to be followed in this chapter we shall attempt to estimate the function f directly from $\mathcal{Z}$ without using $P(X,Y)$ or $P(Y|X)$. For this, the following three steps are necessary. First, a class of functions $\mathcal{F}$ needs to be defined. Second, a suitable loss ℓ is to be fixed. Finally, a method has to be provided to find the function f which minimizes the risk $\mathrm{R}(f)$ among all $f \in \mathcal{F}$. Such method is called an "induction principle". Desirable properties of such an induction principle are discussed in the next section.

2.2 Desirable properties for Induction Principles

The most commonly used induction principle is the one of minimizing the empirical risk

$$\mathrm{R}_{\mathrm{emp}}(f) = \frac{1}{M}\sum_{i=1}^{M} \ell(f(\mathbf{x}_i), y_i), \tag{4}$$

which is the empirical counterpart of the expected risk (2). The goal of learning in our setup is to find an algorithm that, given a training sample $\mathcal{Z}$, finds a function $f \in \mathcal{F}$ that minimizes (4). Notice that this will not necessarily result in a unique solution. As one can see in Figure 1 more than one function can have the same (e.g. zero) empirical risk on the same data sample. However, these functions can take arbitrary values at other points in $\mathcal{X}$; hence the solution that minimizes the empirical risk is not guaranteed to minimize the true risk (2).

The other two phenomena arising in relation with the minimization of the empirical risk (4) are *overfitting* and *underfitting*. An overly complex function f might describe the training data well but does not generalize to unseen examples. The converse could also happen. Assume the function class $\mathcal{F}$ we can choose from is very small, e.g. it contains only a single, fixed function. Then our learning machine would trivially be consistent, since $\mathrm{R}(f) = const$ for all $f \in \mathcal{F}$. But if this single $f \in \mathcal{F}$ is not by accident the rule that generates our data, the decisions are unrelated to the concept generating our data. This phenomenon is called underfitting (cf. Figure 2). Apparently we

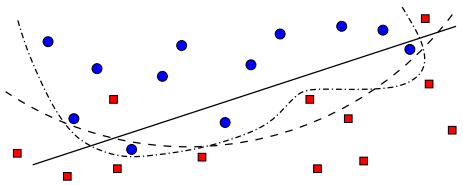

Fig. 2. An illustration of underfitting and overfitting on a small sample. The simple linear function (solid line) underfits the data and already makes training errors. The complex one (dash-dotted line) has no training error but may not generalize well on unseen data. The function with intermediate complexity (dashed line) seems to capture the decision boundary best.

need some way of controlling how large the class of functions $\mathcal{F}$ is, such that we avoid overfitting and underfitting and obtain solutions that generalize well (i.e. with reasonable complexity). The questions of consistency, overfitting and underfitting are closely related and will lead us to a concept known as regularization (e.g. Tikhonov and Arsenin, 1977; Morozov, 1984) and to the principle of Structural Risk Minimization (Vapnik, 1998).

Regularization

In the previous paragraphs we have shown that for successful learning it is not enough to find a function with minimal empirical risk. If we are interested in a good estimation of the true risk on all possible data points, we need to introduce a complexity control and choose our solution by minimizing the following objective function:

$$\mathrm{R}_{\mathrm{emp}}(f, \mathcal{Z}) + \lambda\, \Omega(f). \tag{5}$$

This equation shows a regularization approach. We add a penalty term to make the trade-off between the complexity of the function class and the empirical error. Using such a regularization, a bound for the true risk can be derived.

There are several possibilities to choose λ and Ω in order to derive a consistent inductive principle. In the following sections we will describe the choice inspired by the work of Vapnik. Other possible choices are for example Akaike information criterion (Akaike, 1974) or Mallows Cp (Mallows, 1973), used in classical statistics, as well as spline-regularization (Wahba, 1980), wavelet regularization (Donoho et al., 1996), CART (Breiman et al., 1984) and many other modern approaches. A general foundation for regularization in model selection is given in (Barron et al., 1999). Bartlett and Mendelson (2002) investigate regularization in the context of SVM.

Consistency

Let us define more closely what consistency means and how it can be characterized. Let us denote by f^M the function $f \in \mathcal{F}$ that minimizes (4) for a

given training sample $\mathcal{Z}$ of size M. The notion of *consistency* implies that, as $M \to \infty$, $|\,\mathrm{R}(f^M) - \mathrm{R}_{\mathrm{emp}}(f^M)| \to 0$ in probability. We have already seen in a previous example that such convergence may not be the case in general, the reason being that f^M now depends on the sample $\mathcal{Z}$. One can show that a necessary and sufficient condition for consistency is *uniform* convergence, over all functions in $\mathcal{F}$, of the difference between the expected and the empirical risk to zero. This insight is summarized in the following theorem:

Theorem 1 (Vapnik and Chervonenkis, 1991). *One-sided uniform convergence in probability, i.e.*

$$\lim_{M\to\infty} \mathbb{P}\left[\sup_{f\in\mathcal{F}} (\mathrm{R}(f) - \mathrm{R}_{\mathrm{emp}}(f)) > \epsilon\right] = 0, \tag{6}$$

for all $\epsilon > 0$, is a necessary and sufficient condition for (nontrivial) consistency of empirical risk minimization.

Since the condition in the theorem is not only sufficient but also necessary it seems reasonable that any "good" learning machine implementing a specific function class should satisfy condition (6).

2.3 Structural Risk Minimization

Consequently, the question arises how one can choose function classes that satisfy Theorem 1 in practice? It will turn out that this is possible and it crucially depends on the question how complex the functions in the class $\mathcal{F}$ are, a question we have already seen to be equally important when talking about overfitting and underfitting. But what does complexity mean and how can one *control* the size of a function class?

The complexity of a function class can be measured by the number of different possible combinations of outcome assignments when choosing functions from this class. This quantity is usually difficult to obtain theoretically for useful classes of functions. Popular approximations of this measure are covering numbers (Shawe-Taylor et al., 1998), annealed entropy and fat-shattering dimension (Bartlett et al., 1996), VC-entropy and VC-dimension (Vapnik, 1998), or Rademacher and Gaussian complexity (Bartlett and Mendelson, 2002). We will not go into detail about these quantities here.

A specific way of controlling the complexity of a function class is given by VC-theory and the principle of *Structural Risk Minimization* (Vapnik, 1998). Here the concept of complexity is captured by the VC-dimension h of the function class $\mathcal{F}$. Roughly speaking, the VC-dimension measures how many (training) points can be shattered (i.e. separated for all possible labellings) using functions of the class. This quantity can be used to bound the probability that the expected error deviates much from the empirical error for any function from the class, that is VC-style bounds usually take the form

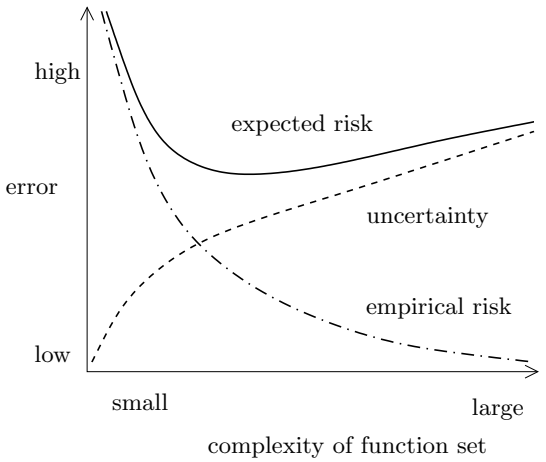

Fig. 3. Schematic illustration of (8). The dash-dotted line represents the training error (empirical risk), the dashed line the upper bound on the complexity term. With higher complexity the empirical error decreases but the upper bound on the risk uncertainty becomes worse. For a certain complexity of the function class the best expected risk (solid line) is obtained. Thus, in practice the goal is to find the best trade-off between empirical error and complexity.

$$\left[\sup_{f\in\mathcal{F}} \left(\mathrm{R}(f) - \mathrm{R}_{\mathrm{emp}}(f,\mathcal{Z})\right) > \epsilon\right] \leq H(\mathcal{F}, M, \epsilon), \tag{7}$$

where H is some function that depends on properties of the function class $\mathcal{F}$, e.g. the VC-dimension, the size M of the training set and the desired closeness ϵ. By equating the right-hand side of (7) to $\delta > 0$ and solving $H = \delta$ for ϵ one can turn these bounds into expressions of the following form: with probability at least $1 - \delta$ over the random draw of the training sample $\mathcal{Z}$,

$$\mathrm{R}(f) \leq \mathrm{R}_{\mathrm{emp}}(f,\mathcal{Z}) + \widetilde{H}(\mathcal{F}, M, \delta), \tag{8}$$

where $\widetilde{H}$ is the penalty term that measures our degree of uncertainty. If the function class is simple then $\widetilde{H}$ is small. This penalty term usually increases if we require a higher precision (e.g. with $\log(\frac{1}{\delta})$) and decreases if we observe more examples (e.g. with $\frac{1}{M}$ or $\frac{1}{\sqrt{M}}$). Note that this prototypical bound is structurally identical to the regularized risk functional in equation (5). The practical implication of bounds like (8) is that our learning machine should be constructed such that

1. it finds a function with a small empirical error, and
2. at the same time keeps the penalty term $\widetilde{H}$ small.

Only if our learning principle can control both quantities we have a guarantee that the expected error of our estimate will be small (cf. Figure 3).

One of the most famous of these VC-style bounds is due to Vapnik and Chervonenkis:

Theorem 2 (Vapnik and Chervonenkis, 1991). *Let h denote the VC-dimension of the function class $\mathcal{F}$ and let $\mathrm{R}_{\mathrm{emp}}$ be defined by (4) using the 0/1-loss. For all $\delta > 0$ and $f \in \mathcal{F}$ the inequality bounding the risk*

$$\mathrm{R}(f) \leq \mathrm{R}_{\mathrm{emp}}(f, \mathcal{Z}) + \sqrt{\frac{h\left(\ln \frac{2M}{h} + 1\right) - \ln(\delta/4)}{M}} \tag{9}$$

holds with probability of at least $1 - \delta$ for $M > h$ over the random draw of the training sample $\mathcal{Z}$.

This theorem lays the ground for the SVM algorithm that we will consider in more detail in Section 3.

Based on Theorem 2 the principle of Structural Risk Minimization (SRM) has been derived (e.g. Cortes and Vapnik, 1995; Vapnik, 1998). According to this principle a nested family of function classes $\mathcal{F}_1 \subseteq \cdots \subseteq \mathcal{F}_k$ with non-decreasing VC-dimension $h_1 \leq \cdots \leq h_k$ is constructed. After the solutions $f_1, \ldots, f_k$ of the empirical risk minimization (4) in the function classes $\mathcal{F}_1, \ldots, \mathcal{F}_k$ have been found, the principle chooses the function class $\mathcal{F}_i$ (and the function f_i) such that an upper bound on the generalization error like (9) is minimized.

3 Linear SVM: Learning Theory in Practice

Having summarized the prerequisites from statistical learning theory, we now give an example of a particular learning machine that builds upon these insights. The Support Vector Machine algorithm (SVM) developed by Vapnik and others (e.g. Boser et al., 1992; Cortes and Vapnik, 1995; Vapnik, 1998; Cristianini and Shawe-Taylor, 2000; Müller et al., 2001; Schölkopf and Smola, 2002, and numerous others) is one of the most successful classification techniques over the last decade, especially after being combined with the kernel idea which we shall discuss in Section 4.

3.1 Linear Separation Planes

We are now going to discuss how one could possibly control the size of a function class and how to select the empirical risk minimizer in this class.

In the following, let us assume that we are dealing with a two class classification problem (i.e. $\mathcal{Y} = \{-1, +1\}$) in a real-valued vector space, e.g. $\mathcal{X} = \mathbb{R}^N$. Further, we assume that the distribution of these two classes is such that they are linearly separable, i.e. one can find a linear function of the inputs $\mathbf{x} \in \mathcal{X}$ such that $f(\mathbf{x}) < 0$ whenever the label $y = -1$ and $f(\mathbf{x}) \geq 0$ otherwise. This

can be conveniently expressed by a hyperplane in the space $\mathcal{X}$, i.e. we are looking for a function f of the form

$$f(\mathbf{x}) = (\mathbf{w}^\top \mathbf{x}) + b. \tag{10}$$

Assume that the function class $\mathcal{F}$ we choose our solution from is the one containing all possible hyperplanes, i.e. $\mathcal{F} = \{f : \mathcal{X} \to \mathbb{R} \mid f(\mathbf{x}) = (\mathbf{w}^\top \mathbf{x}) + b\}$. For $\mathcal{X} = \mathbb{R}^N$ it is rather straightforward to show that the VC-dimension of this class of functions will be $h = N + 1$, that is, in an N-dimensional space the maximal number of points that can be separated for an arbitrary labelling using a hyperplane is $N + 1$.

3.2 Canonical Hyperplanes and Margins

To apply the SRM principle in practice, not only must the VC-dimension of the class of hyperplanes be finite, rather a nested structure of function classes must be defined. To this end we define the function classes

$$\mathcal{F}_\Lambda = \{f : \mathbb{R}^N \to \mathbb{R} \mid f(x) = (\mathbf{w}^\top \mathbf{x}) + b, \|\mathbf{w}\| \leq \Lambda\}. \tag{11}$$

Clearly $\mathcal{F}_{\Lambda_1} \subseteq \mathcal{F}_{\Lambda_2}$ whenever $\Lambda_1 \leq \Lambda_2$. But what effect does constraining the norm of the weight vector have on the corresponding VC-dimensions of $\mathcal{F}_\Lambda$? It turns out that we also get $h(\mathcal{F}_{\Lambda_1}) \leq h(\mathcal{F}_{\Lambda_2})$ for $\Lambda_1 \leq \Lambda_2$, as we will see shortly in (12).

The crucial ingredient in making the function classes $\mathcal{F}_\Lambda$ nested is to define a unique representation for each hyperplane. We introduce the concept of canonical hyperplanes and the notion of margins. If the data are separable by $(\mathbf{w}, b)$ then they are also separable by any (positive) multiple of $(\mathbf{w}, b)$ and hence there exist an infinite number of representations for the same separating hyperplane. In particular, all function classes $\mathcal{F}_\Lambda$ would have the same VC-dimension as they would contain the same functions in different representations.

A canonical hyperplane with respect to a sample $\mathcal{Z}$ of M points is defined as a function

$$f(\mathbf{x}) = (\mathbf{w}^\top \mathbf{x}) + b,$$

where $\mathbf{w}$ is normalized such that

$$\min_{i=1,\ldots,M} |f(\mathbf{x}_i)| = 1.$$

The notion of a canonical hyperplane is illustrated in Figure 4. Notice that none of the training examples produces an absolute output that is smaller than one and the examples closest the hyperplane have exactly an output of one, i.e. $(\mathbf{w}^\top \mathbf{x}) + b = \pm 1$. In section 5, we will see that the latter objects will be used in the description of the hyperplane, and they are therefore called the *support vectors*. In Figure 4 these are the objects which are connected to the

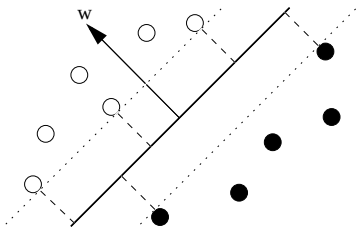

Fig. 4. Linear SVM and margins. A linear SVM classifier is defined by the normal vector $\mathbf{w}$ of a hyperplane and an offset b. The decision boundary is $\{\mathbf{x}|(\mathbf{w}^\top\mathbf{x})+b=0\}$ (solid line). Each of the two half spaces induced by this hyperplane corresponds to one class, i.e. $f(\mathbf{x}) = \mathrm{sgn}((\mathbf{w}^\top\mathbf{x})+b)$. The margin of a linear classifier is the minimal distance of any training point to the hyperplane. For the case shown in the picture it is the distance between the dotted lines and the solid line.

decision boundary by the dashed lines. Since we assumed the sample $\mathcal{Z}$ to be linearly separable, we can turn any f that separates the data into a canonical hyperplane by suitably normalizing the weight vector $\mathbf{w}$ and adjusting the threshold b correspondingly.

The *margin* is defined to be the minimal Euclidean distance between any training example $\mathbf{x}_i$ and the separating hyperplane. Intuitively, the margin measures how good the separation between the two classes by a hyperplane is. If this hyperplane is in the canonical form, the margin can be measured by the length of the weight vector $\mathbf{w}$. Consider two support vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ from different classes. The margin is given by the projection of the distance between these two points on the direction perpendicular to the hyperplane. This distance can be computed as (e.g. Vapnik, 1998)

$$\left(\frac{\mathbf{w}^\top}{\|\mathbf{w}\|}(\mathbf{x}_1 - \mathbf{x}_2)\right) = \frac{2}{\|\mathbf{w}\|}.$$

The smaller the norm of the weight vector $\mathbf{w}$ in the canonical representation, the larger the margin.

More generally, it was shown (e.g. Vapnik, 1998) that if the hyperplane is constructed under the constraint $\|\mathbf{w}\|_2 \leq \Lambda$ then the VC-dimension of the class $\mathcal{F}_\Lambda$ is bounded by

$$h \leq \min(\Lambda^2 R^2 + 1, N + 1), \tag{12}$$

where R is the radius of the smallest sphere around the data. Thus, if we bound the margin of a function class from below, say by $\frac{2}{\Lambda}$, we can control its VC-dimension and hence apply the SRM principle as shown in Figure 5.

A particularly important insight is that the complexity only indirectly depends on the dimensionality of the data. This is very much in contrast to density estimation, where the problems become more difficult as the dimensionality of the data increases. For SVM classifier, if we can achieve a large margin the problem remains simple.

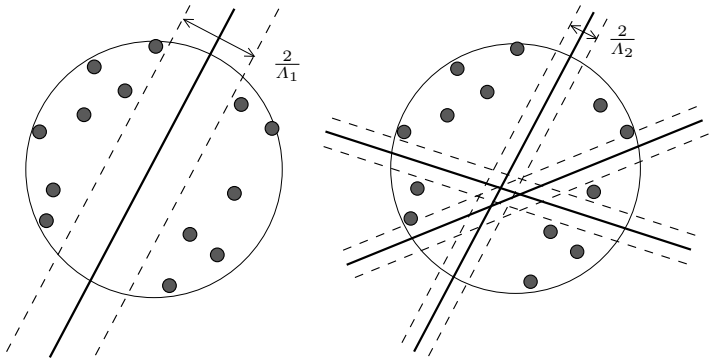

Fig. 5. Illustration of why a large margin reduces the complexity of a linear hyperplane classifier. If we choose hyperplanes with a large margin, there is only a small number of possibilities to separate the data, i.e. the VC-dimension of $\mathcal{F}_{\Lambda_1}$ is small (left panel). On the contrary, if we allow smaller margins there are more separating hyperplanes, i.e. the VC-dimension of $\mathcal{F}_{\Lambda_2}$ is large (right panel).

4 Kernel Functions

In the previous section we have seen that by restricting ourselves to linear functions one can control the complexity of a learning machine. We have thus avoided the problem of dealing with too complex functions at the price of being able to solve only linearly separable problems. In the following we show how to extend the linear SVM for constructing a rich set of non-linear decision functions by abstracting the task of learning from the actual data representation. Based on this abstraction we then introduce techniques for learning with structured data, such as strings and trees.

Central to the success of non-linear SVM was the re-discovery of the so called *Reproducing Kernel Hilbert Spaces* (RKHS) and *Mercer's Theorem* (Boser et al., 1992). There is a large body of literature dealing with kernel functions, their theory and applicability, see e.g. Kolmogorov (1941); Aronszajn (1950); Aizerman et al. (1964); Boser et al. (1992) or Schölkopf and Smola (2002); Shawe-Taylor and Cristianini (2004) for an overview. We only recall the basic definitions and properties necessary for turning our linear, hyperplane based learning technique into a very powerful algorithm capable of finding non-linear decision functions with controllable complexity.

4.1 The Kernel Trick

The basic idea of the so called *kernel methods* is to first preprocess the data by some non-linear mapping Φ and then to apply the same linear algorithm as before but in the image space of Φ (cf. Figure 6 for an illustration).

More formally we apply the mapping

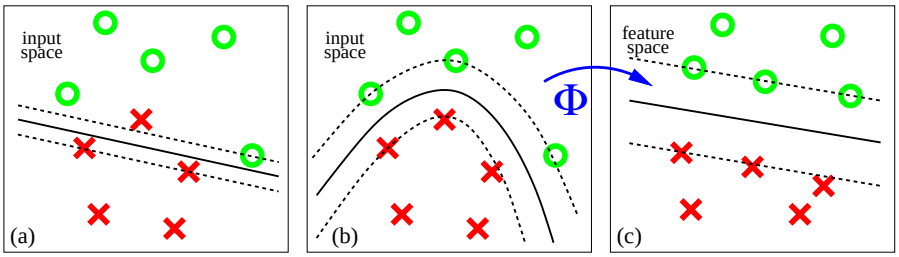

Fig. 6. Three views on the same two class separation problem (Zien et al., 2000). (a) A linear separation of the input points is not possible without errors. Even allowing misclassification of one data point results in a small margin. (b) A better separation is provided by a non-linear surface in the input space. (c) This non-linear surface corresponds to a linear surface in a feature space. Data points are mapped from input space to feature space by the function Φ induced by the kernel function k.

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

to the data $\mathbf{x}_1, \ldots, \mathbf{x}_M \in \mathcal{X}$ and consider our algorithm in $\mathcal{E}$ instead of $\mathcal{X}$, i.e. the sample is preprocessed as

$$\{(\Phi(\mathbf{x}_1), y_1), \ldots, (\Phi(\mathbf{x}_M), y_M)\} \subseteq (\mathcal{E} \times \mathcal{Y})^M.$$

In certain applications we might have sufficient knowledge about our problem such that we can design an appropriate Φ by hand (e.g. Zien et al., 2000; Blankertz et al., 2002). An alternative strategy is to consider a class of mappings and choose the Φ providing the best representation for a particular learning task (Braun et al., 2008). If this mapping is not too complex to compute and the space $\mathcal{E}$ is not too high-dimensional, we might just explicitly apply this mapping to our data. Similar transformations are applied in neural networks (Bishop, 1995), radial basis networks (e.g. Moody and Darken, 1989) or Boosting algorithms (Freund and Schapire, 1997), where the input data is mapped to some representation given by the hidden layer, the RBF bumps or the hypotheses space, respectively (Rätsch et al., 2002). The difference with kernel methods, however, is that for a suitably chosen Φ we get an algorithm that has powerful non-linearities but is still very intuitive and retains most of the favorable properties of its linear input space version.

The problem with explicitly using the mapping Φ to construct a feature space is that the resulting space can be extremely high-dimensional. As an example consider the case when the input space $\mathcal{X}$ consists of images of 16×16 pixels, i.e. 256 dimensional vectors, and we choose 5^{th} order monomials as non-linear features. The dimensionality of such space would be

$$\binom{5 + 256 - 1}{5} \approx 10^{10}.$$

Such a mapping would clearly be intractable to carry out explicitly. We are not only facing the technical problem of storing the data and doing the computations, but we are also introducing problems due to the fact that we are now working in an extremely sparsely sampled space.

The problems concerning the storage and the manipulation of the high dimensional data, however, can be alleviated. It turns out that for a certain class of mappings we are well able to compute scalar products in this new space even if it is extremely high dimensional. Simplifying the above example of computing all 5^{th} order products of 256 pixels to that of computing all 2^{nd} order products of two "pixels", i.e.

$$\mathbf{x} = (x_1, x_2) \text{ and } \Phi(\mathbf{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2),$$

the computation of a scalar product between two such feature space vectors can be readily reformulated in terms of a so-called kernel function k:

$$\begin{aligned}(\Phi(\mathbf{x})^\top \Phi(\mathbf{z})) &= (x_1^2, \sqrt{2}\, x_1x_2, x_2^2)(z_1^2, \sqrt{2}\, z_1z_2, z_2^2)^\top \\ &= ((x_1, x_2)(z_1, z_2)^\top)^2 \\ &= (\mathbf{x}^\top \mathbf{z})^2 \\ &=: \mathrm{k}(\mathbf{x}, \mathbf{z}).\end{aligned}$$

This finding generalizes: For $\mathbf{x}, \mathbf{z} \in \mathbb{R}^N$, and $d \in \mathbb{N}$ the kernel function

$$\mathrm{k}(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\top \mathbf{z})^d$$

computes a scalar product in the space of all products of d vector entries (monomials) of $\mathbf{x}$ and $\mathbf{z}$ (Vapnik, 1998; Schölkopf et al., 1998b).

The *kernel trick* (Aizerman et al., 1964; Boser et al., 1992; Vapnik, 1998) is to take the original algorithm and formulate it such, that we only use $\Phi(\mathbf{x})$ in scalar products. Then, if we can efficiently evaluate these scalar products, we do not need to carry out the mapping Φ explicitly and can still solve the problem in the huge feature space $\mathcal{E}$. Furthermore, we do not need to know the mapping Φ but only the kernel function.

Now we can ask two questions:

1. For which mappings Φ does there exist a simple way to evaluate the scalar product?
2. Under which conditions does a function $\mathrm{k} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ correspond to a scalar product?

The first question is difficult to answer in general. But for the second question there exists an answer which we present in the following.

4.2 Feature Spaces

To address the question whether a kernel function $\mathrm{k} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ equals a scalar product in some feature space, let us first introduce some more notation

and definitions. Given a training sample $\{\mathbf{x}_1, \ldots, \mathbf{x}_M\} \subseteq \mathcal{X}$, the $M \times M$ matrix K with elements $K_{ij} = \mathrm{k}(\mathbf{x}_i, \mathbf{x}_j)$ is called the kernel matrix or the Gram matrix. An $M \times M$ matrix K (and any other symmetric matrix) is said to be positive semi-definite if any quadratic form over K is positive or zero, i.e. for all $r_i \in \mathbb{R}$, $i = 1, \ldots, M$, we have

$$\sum_{i,j=1}^{M} r_i r_j K_{ij} \geq 0. \tag{13}$$

Positive semi-definite kernels are exactly those giving rise to a positive semi-definite kernel matrix K for all M and all sets $\{\mathbf{x}_1, \ldots, \mathbf{x}_M\} \subseteq \mathcal{X}$. Note that for a kernel (and a matrix) to be positive semi-definite, it is necessary to be symmetric and non-negative on the diagonal.

For any positive semi-definite kernel k we can construct a mapping Φ into a feature space $\mathcal{E}$, such that k acts as a scalar product over Φ. As a matter of fact, it is possible to construct more than one of these spaces. We will omit many crucial details and only present the central results. For more details see Schölkopf and Smola (2002).

The Feature Map

Given a real-valued, positive semi-definite kernel function k, defined over a non-empty set $\mathcal{X}$, we define the feature space $\mathcal{E}$ as the space of all functions mapping from $\mathcal{X}$ to $\mathbb{R}$, i.e. as $\mathcal{E} = \mathbb{R}^{\mathcal{X}} = \{f \mid f : \mathcal{X} \to \mathbb{R}\}$. Notice that, unlike the example in Figure 6, this feature space is not a usual Euclidean space but rather a vector space of functions. The mapping Φ is now defined as

$$\Phi : \mathcal{X} \to \mathbb{R}^{\mathcal{X}}, \Phi(\mathbf{x}) = \mathrm{k}(\cdot, \mathbf{x}), \tag{14}$$

i.e. Φ maps each $\mathbf{x}$ to the function $\mathrm{k}(\cdot, \mathbf{x})$, i.e. the kernel k where the first argument is free and the second is fixed to $\mathbf{x}$ (e.g., Schölkopf et al., 1999). One can show that the set of all linear combinations of the form

$$f(\cdot) = \sum_{i=1}^{M} \alpha_i \, \mathrm{k}(\cdot, \mathbf{x}_i), \tag{15}$$

for arbitrary M, $\alpha_i \in \mathbb{R}$, and $\mathbf{x}_1, \ldots, \mathbf{x}_M$ forms a vector space. Especially, for all functions of the form (15) one gets

$$\langle \mathrm{k}(\cdot, \mathbf{x}), f \rangle_{\mathcal{H}} = f(\mathbf{x}),$$

where $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ denotes the scalar product in some Hilbert space that will become clearer below. In particular we have

$$\begin{aligned} \langle \mathrm{k}(\cdot, \mathbf{x}), \mathrm{k}(\cdot, \mathbf{z}) \rangle_{\mathcal{H}} &= \langle \Phi(\mathbf{x}), \Phi(\mathbf{z}) \rangle_{\mathcal{E}} \\ &= \mathrm{k}(\mathbf{x}, \mathbf{z}). \end{aligned}$$

The last property is the reason why positive semi-definite kernels are also called reproducing kernels: they reproduce the evaluation of f on $\mathbf{x}$. It also shows that k indeed computes, as desired, the scalar product in $\mathcal{E}$ for $\Phi(\mathbf{x})$ defined as in (14). Hence (14) is one possible realization of the mapping associated with a kernel and is called the feature map (for its empirical counterpart see e.g. Mika (2002)). The following is a formal definition of a *Reproducing Kernel Hilbert Space* (cf. Schölkopf and Smola, 2002).

Definition 1 (Reproducing Kernel Hilbert Space (RKHS)). *Let $\mathcal{X}$ be a nonempty set and $\mathcal{H}$ a Hilbert space of functions $f : \mathcal{X} \to \mathbb{R}$. Then $\mathcal{H}$ is called a* reproducing kernel Hilbert space *endowed with the dot product $\langle \cdot, \cdot \rangle$ if there exists a function $\mathrm{k} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ with the properties that*

1. k *has the reproducing property $\langle f, \mathrm{k}(\cdot, \mathbf{x}) \rangle = f(\mathbf{x})$ for all $f \in \mathcal{H}$, in particular $\langle \mathrm{k}(\cdot, \mathbf{x}), \mathrm{k}(\cdot, \mathbf{z}) \rangle = \mathrm{k}(\mathbf{x}, \mathbf{z})$, and*
2. k *spans $\mathcal{H}$, i.e. $\mathcal{H} = \overline{\mathrm{span}\{\mathrm{k}(\cdot, \mathbf{x}) | \mathbf{x} \in \mathcal{X}\}}$, where $\overline{A}$ denotes the completion of the set A.*

One can show, that the kernel k for such a RKHS is uniquely determined.

Mercer Kernels

As a second way to identify a feature space associated with a kernel k one can use a technique derived from Mercer's Theorem.

The Mercer's Theorem, which we will reproduce in the following, states that if a function k gives rise to a positive integral operator, the evaluation of $\mathrm{k}(\mathbf{x}, \mathbf{z})$ can be expressed as a finite or infinite, absolute and uniformly convergent series, almost everywhere. This series defines a feature space and an associated mapping connected to the function k.

Let $\mathcal{X}$ be a finite measure space, i.e. a space with a σ-algebra and a measure μ satisfying $\mu(\mathcal{X}) \leq \infty$.

Theorem 3 (Mercer, 1909). *Suppose $\mathrm{k} \in L_\infty(\mathcal{X}^2, \mu)$ is a symmetric real-valued function such that the integral operator*

$$T_\mathrm{k} : L_2(\mathcal{X}, \mu) \to L_2(\mathcal{X}, \mu), \ (T_\mathrm{k} f)(\mathbf{x}) := \int_{\mathcal{X}} \mathrm{k}(\mathbf{x}, \mathbf{z}) f(\mathbf{z}) d\mu(\mathbf{z})$$

is positive semi-definite, i.e. for all $f \in L_2(\mathcal{X}, \mu)$

$$\int_{\mathcal{X}^2} \mathrm{k}(\mathbf{x}, \mathbf{z}) f(\mathbf{x}) f(\mathbf{z}) d\mu(\mathbf{x}) d\mu(\mathbf{z}) \geq 0.$$

Let $\varphi_j \in L_2(\mathcal{X}, \mu)$ be the normalized orthogonal eigenfunctions of T_k associated with the eigenvalues $\lambda_j \geq 0$, sorted in non-increasing order. Then

1. $(\lambda_j)_j \in l_1$

2. $\mathrm{k}(\mathbf{x},\mathbf{z}) = \sum_{j=1}^{N_\mathcal{E}} \lambda_j \varphi_j(\mathbf{x})\varphi_j(\mathbf{z})$ *holds for almost all* $\mathbf{x},\mathbf{z}$. *Either* $N_\mathcal{E} \in \mathbb{N}$ *or* $N_\mathcal{E} = \infty$*; in the latter case, the series converges absolutely and uniformly for almost all* $\mathbf{x},\mathbf{z}$.

If we choose as feature space $\mathcal{E} = l_2^{N_\mathcal{E}}$ and the mapping Φ as

$$\Phi : \mathcal{X} \to l_2^{N_\mathcal{E}},\ \Phi(\mathbf{x}) = (\sqrt{\lambda_j}\varphi_j(\mathbf{x}))_{j=1,\ldots,N_\mathcal{E}},$$

we see from the second statement in Theorem 3 that the kernel k corresponds to the dot product in $l_2^{N_\mathcal{E}}$, i.e. $\mathrm{k}(\mathbf{x},\mathbf{z}) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{z}) \rangle$.

The kernels satisfying the Mercer's Theorem are called *Mercer kernels*. It can be shown that, if the set $\mathcal{X}$ on which the kernel is defined, is compact, a kernel is a Mercer kernel if and only if it is a positive semi-definite kernel (cf. Smola et al., 1998). Table 1 lists some of the most widely used kernel functions in machine learning applications.

Table 1. Common kernel functions:

Gaussian RBF:	$\mathrm{k}(\mathbf{x},\mathbf{z}) = \exp\left(\frac{-\|\mathbf{x}-\mathbf{z}\|^2}{c}\right)$	(16)
Polynomial:	$\mathrm{k}(\mathbf{x},\mathbf{z}) = ((\mathbf{x}^\top \mathbf{z}) + \theta)^d$	(17)
Inverse multi-quadric:	$\mathrm{k}(\mathbf{x},\mathbf{z}) = \frac{1}{\sqrt{\|\mathbf{x}-\mathbf{z}\|^2 + c^2}}$	

Note that recently Braun et al. (2008) have observed that the excellent generalization that is typically observed when using SVMs in high-dimensional applications with few samples is due to its very economic representation in the feature space $\mathcal{E}$. Given the appropriate kernel, only a very low dimensional subspace is task relevant (see Figure 7).

4.3 Kernels for Structured Data

Another important feature of kernel functions is that they are not restricted to operate on vectorial data. Kernels can be defined over any type of data including discrete and structured representations. Consequently, a large body of research has studied kernel functions for structured data, such as for analysis of strings and sequences (e.g. Watkins, 2000; Lodhi et al., 2002; Sonnenburg et al., 2007a), hierarchical representations and trees (e.g. Collins and Duffy, 2002; Kashima and Koyanagi, 2002; Rieck et al., 2010) as well as network and graph structures (e.g. Gärtner et al., 2004; Kashima et al., 2004; Vishwanathan et al., 2010). We herein provide a brief introduction to kernel functions defined over strings and trees. An extensive discussion of kernels for structured data is provided by Shawe-Taylor and Cristianini (2004).

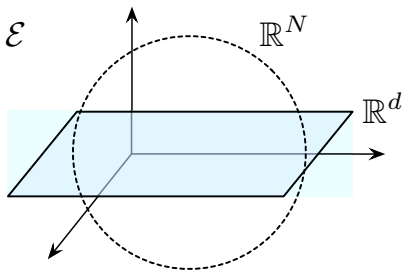

Fig. 7. Illustration of the feature space representation. The data is embedded in a high-dimensional feature space $\mathcal{E}$. Since only, say, N data points exist, the data given through a normalized kernel is situated in a N-dimensional ball in $\mathcal{E}$. However, only a small d-dimensional subspace of the N-dimensional ball in $\mathcal{E}$ is task relevant, e.g. for the classification or regression at hand. Thus, if the kernel is well chosen, then kernel methods make very economical use of the data as they map the data into a effectively very low dimensional task relevant subspace of $\mathcal{E}$ (see also Braun et al. (2008) for further discussion and proofs).

String Kernels

Strings and sequences are a natural representation of data in many areas of computer science. For example, several applications in bioinformatics are concerned with studying strings of DNA and many tasks of information retrieval center around analysis of text documents. Before introducing kernels for strings, let us introduce some notation. A *string* or *sequence* $\mathbf{x}$ is a concatenation of symbols from an *alphabet* $\mathcal{A}$, such as the characters in text or the bases of DNA. The set of all possible concatenations of symbols from $\mathcal{A}$ is denoted by $\mathcal{A}^*$ and the set of all concatenations of length n by $\mathcal{A}^n$

For characterizing the content of sequential data, most string kernels make use of a predefined set $L \subseteq \mathcal{A}^*$ of relevant strings. This set L can be interpreted as a language that is used to capture structure contained in strings and may range from a simple selection of interesting terms to complex constructs involving gaps and wildcards. We focus on two basic definitions that are widely used for learning with string kernels: *words* and *n-grams* (Figure 8).

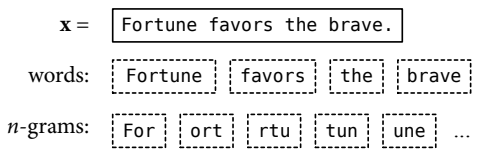

Fig. 8. Representations of a string $\mathbf{x}$ using words and n-grams.

In the domain of information retrieval and natural language processing, the set L is often defined as *words* of a natural language, such as English or German. In this setting, L is either given explicitly by providing a dictionary of terms or implicitly by partitioning strings according to a set of delimiter symbols $\mathcal{D} \subset \mathcal{A}$, such that

$$L = (\mathcal{A} \setminus \mathcal{D})^*$$

where L corresponds to all concatenations of non-delimiter symbols. Based on this definition, the content of a string can be described in terms of contained words from the set L—a representation often denoted as a *"bag of words"* (Joachims, 1999).

In several applications, however, the structure underlying sequential data is unknown and no set of appropriate words can be defined a priori. An alternative technique for defining the set L is to move a sliding window of length n over a string and extract *n-grams* (substrings of length n, cf. Damashek, 1995). Formally, this set can be defined as

$$L = A^n.$$

Using this definition of L, the content of a string can be described in terms of contained n-grams, even if no prior knowledge about its structure is available, for example as in many applications of bioinformatics involving DNA and protein sequences.

Based on the set L, a feature map Φ can be defined which embeds strings in an $|L|$-dimensional vector space spanned by the strings of L, that is,

$$\Phi : \mathcal{A}^* \to \mathbb{R}^{|L|}, \quad \Phi(\mathbf{x}) = \big(\#_w(\mathbf{x})\big)_{w \in L}, \tag{18}$$

where $\#_w(\mathbf{x})$ returns the number of occurrences of the string w in the string $\mathbf{x}$. Alternatively, $\#_w(\mathbf{x})$ may be defined as frequency, probability or binary flag for the occurrences of w in $\mathbf{x}$.

This feature map Φ provides the basis for constructing a *string kernel* which takes the form

$$\mathrm{k} : \mathcal{A}^* \times \mathcal{A}^* \to \mathbb{R}, \quad \mathrm{k}(\mathbf{x}, \mathbf{z}) = \sum_{w \in L} \#_w(\mathbf{x}) \cdot \#_w(\mathbf{z}) \tag{19}$$

and correspond to an inner product in the feature space spanned by the strings of L. Depending on the complexity of the set L, however, the dimension of this features space may grow almost arbitrarily. Thus in practice, computation of string kernels is rarely conducted using explicit vectors, but carried out by means of advanced data structures, such as hash maps, Tries and suffix trees (cf. Sonnenburg et al., 2007a). The corresponding realizations of (19) for words and n-grams are denoted as *Bag-of-Words Kernel* (Joachims, 1999) and *Spectrum Kernel* (Leslie et al., 2002), respectively.

Due to the ease of incorporation with kernel-based learning methods, string kernels have gained considerable attention in research, starting from first realizations of Haussler (1999) and Watkins (2000), and extending to domain-specific kernels for natural language processing (Lodhi et al., 2002; Cancedda

et al., 2003) and bioinformatics (Zien et al., 2000). In particular, the challenge of uncovering structure in DNA has influenced several extensions of the feature map in (18), for example by incorporating generative models (Jaakkola et al., 2000; Tsuda et al., 2002), inexact and position-dependent matching (Leslie et al., 2003; Leslie and Kuang, 2004; Rätsch et al., 2005; Sonnenburg et al., 2006) as well as sequence alignments (Vert et al., 2004; Cuturi et al., 2007). A discussion of several string kernels and their implementations is provided by Sonnenburg et al. (2007a).

Tree Kernels

Besides sequences and strings, several applications of statistics and machine learning involve tree-structured data, for example in form of parse trees in natural language processing or molecule structures in chemistry. Formally, a *tree* $\mathbf{x}$ is an acyclic graph with a dedicated root, where we additionally require each *node* x to be labeled with a symbol. To navigate in a tree, we address the i-th child of a node x by x_i and denote the number of children by $|x|$. Moreover, we denote the set of all possible trees by T. Similar to strings, we construct a feature map Φ which embeds a tree $\mathbf{x}$ in a $|T|$-dimensional vector space spanned by all possible trees, that is

$$\Phi : T \to \mathbb{R}^{|T|}, \quad \Phi(\mathbf{x}) = \big(\#_t(\mathbf{x})\big)_{t \in T}, \tag{20}$$

where $\#_t(\mathbf{x})$ counts the occurrences of the (sub)tree t in the tree $\mathbf{x}$. In contrast to (18), the set T is more involved than a list of strings and thus requires special techniques for constructing a feasible kernel function.

A generic technique for defining kernels over structured data is the convolution of local kernels defined over sub-structures (Haussler, 1999). This concept has been applied to tree data by Collins and Duffy (2002) for constructing a *tree kernel* which implicitly computes an inner product by counting shared subtrees. Given two trees $\mathbf{x}$ and $\mathbf{y}$, this kernel is defined as

$$\mathrm{k}(\mathbf{x}, \mathbf{z}) = \langle \Phi(\mathbf{x}) \Phi(\mathbf{z}) \rangle = \sum_{x \in \mathbf{x}} \sum_{z \in \mathbf{z}} c(x, z) \tag{21}$$

where the counting function c recursively determines the number of shared subtrees rooted in the tree nodes x and z.

The function c is defined as $c(x, z) = 0$ if x and z have different labels and $c(x, z) = 1$ if x and z are leave nodes of the same label. In all other cases, the definition of c follows a recursive rule given by

$$c(x, z) = \prod_{i=1}^{|x|} (1 + c(x_i, z_i)). \tag{22}$$

To understand how the counting of subtrees relates to an inner product, let us consider two trees $\mathbf{x}, \mathbf{z}$ and a subtree t, which occurs m times in $\mathbf{x}$ and n

times in $\mathbf{z}$. Clearly, both trees share the subtree t and we can count mn distinct pairs of t common to $\mathbf{x}$ and $\mathbf{z}$. If we consider the feature map Φ given in (20), we have $\Phi_t(\mathbf{x}) = m$ and $\Phi_t(\mathbf{z}) = n$ and also obtain $\Phi_t(\mathbf{x})\Phi_t(\mathbf{z}) = mn$. Hence, by counting all shared subtrees of $\mathbf{x}$ and $\mathbf{z}$, we arrive at an inner product over the vectors $\Phi(\mathbf{x})$ and $\Phi(\mathbf{z})$. As an example, Figure 9 illustrates two simple trees and their shared subtrees.

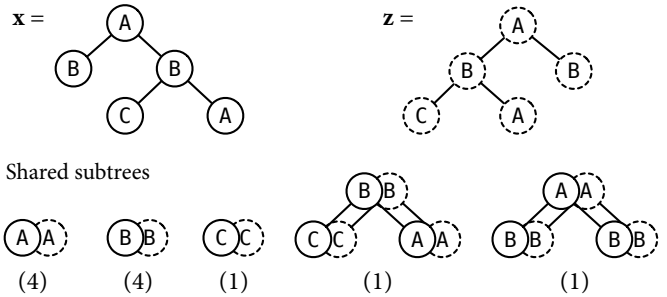

Fig. 9. Shared subtrees of two trees. The numbers in brackets indicate the number of occurrences for each shared subtree pair.

Several extensions and refinements of the kernel in (21) have been proposed to increase its expressiveness for specific learning tasks. For example, there exists several variations of (22) which account for different types of subtrees, such as complete trees (Vishwanathan and Smola, 2004) and ordered trees (Kashima and Koyanagi, 2002). A generic extension by Moschitti (2006) allows for controlling the vertical as well as the horizontal contribution of subtree counts. Furthermore, different techniques have been studied for alleviating the quadratic run-time of counting shared subtrees, most notably the approximation framework of Rieck et al. (2010) which enables computing tree kernels in almost linear time.

4.4 Properties of Kernels

Besides being useful tools for constructing non-linear classifiers or learning with structured data, kernels possess some additional properties that make them an interesting choice in algorithms. It was shown (Girosi et al., 1993) that using a particular positive semi-definite kernel corresponds to an *implicit* choice of a regularization operator. For translation-invariant kernels, the regularization properties can be expressed conveniently in Fourier space in terms of the frequencies (Smola et al., 1998; Girosi, 1998). For example, Gaussian kernels (cf. (16)) correspond to a general smoothness assumption in all k-th order derivatives (Smola et al., 1998). Vice versa, using this correspondence kernels matching a certain prior about the frequency content of the data can be constructed so as to reflect our prior problem knowledge.

Furthermore, many algorithms can be formulated using so called *conditionally positive definite* kernels (cf. Smola et al., 1998) which are a superclass of positive semi-definite kernels considered so far. They can be interpreted as generalized non-linear dissimilarity measures (opposed to just the scalar product) and are applicable e.g. in SVM and kernel PCA.

5 Implementation of SVM

5.1 Basic formulations

We are now at the point to merge the ideas of statistical learning, structural risk minimization and reproducing kernels into a single algorithm, Support Vector Machines, suitable for a wide range of practical application. The main goal of this algorithm is to find a weight vector $\mathbf{w}$ separating the data $\mathcal{Z}$ with the largest possible margin.

Separable Data

Assume that the data are separable. Our goal is to find the smallest possible $\mathbf{w}$ without committing any error. This can be expressed by the following quadratic optimization problem:

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|^2, \\ \text{subject to} \quad & y_i \left((\mathbf{w}^\top \mathbf{x}_i) + b \right) \geq 1, \quad \forall i = 1, \ldots, M. \end{aligned} \tag{23}$$

The constraints in (23) assure that $\mathbf{w}$ and b are chosen such that no example has a distance to the hyperplane smaller than one. The problem can be solved directly by using a quadratic optimizer. Notice that the optimal solution renders a canonical hyperplane. In contrast to many neural networks (e.g. Bishop, 1995) one can always find the *global* minimum. In fact, all minima of (23) are global minima, although they might not be unique as e.g. in the case when $M < N$, where N is the dimensionality of the data.

In the formulation (23), referred to as the primal formulation, we are bound to use the original data $\mathbf{x}_i$. In order to apply the kernel trick (cf. section 4.1) we need to transform the problem such that the only operation involving the original data $\mathbf{x}$ is an inner product between certain data points. This can be achieved by transforming the problem to the dual formulation. The notion of duality is an essential part of non-linear optimization theory, for details one can refer to any standard textbook on mathematical programming (e.g. Luenberger, 1973; Bertsekas, 1995). For our purposes it is important that for every quadratic optimization problem there exists a dual problem which is also a quadratic problem. If both the primal and the dual problems have an optimal solution then the values of the objective function at the optimal

solutions coincide. This implies that by solving the dual problem—which uses the original data $\mathbf{x}$ only through inner products—the solution to the primal problem can be reconstructed.

To derive the dual of (23), we introduce Lagrange multipliers $\alpha_i \geq 0$, $i = 1, \ldots, M$, one for each of the constraints in (23). We obtain the following Lagrangian:

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{M} \alpha_i (y_i((\mathbf{w}^\top \mathbf{x}_i) + b) - 1). \tag{24}$$

The task is to minimize (24) with respect to $\mathbf{w}, b$ and to maximize it with respect to α_i. At the optimal point, we have the following saddle point equations:

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

which translate into

$$\sum_{i=1}^{M} \alpha_i y_i = 0 \quad \text{and} \quad \mathbf{w} = \sum_{i=1}^{M} \alpha_i y_i \mathbf{x}_i. \tag{25}$$

From the right equation of (25), we find that $\mathbf{w}$ is contained in the subspace spanned by the $\mathbf{x}_i$ in the training set. By substituting (25) into (24), we get the dual quadratic optimization problem:

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{M} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{M} \alpha_i \alpha_j y_i y_j \left(\mathbf{x}_i^\top \mathbf{x}_j\right), \tag{26}$$

$$\text{subject to} \quad \alpha_i \geq 0, \ i = 1, \ldots, M, \tag{27}$$

$$\sum_{i=1}^{M} \alpha_i y_i = 0. \tag{28}$$

Thus, by solving the dual optimization problem, one obtains the coefficients α_i, $i = 1, \ldots, M$, which one needs to express the solution $\mathbf{w}$. This leads to the decision function:

$$\begin{aligned} f(\mathbf{x}) &= \operatorname{sgn}\left((\mathbf{w}^\top \mathbf{x}_i) + b\right) \\ &= \operatorname{sgn}\left(\sum_{i=1}^{M} y_i \alpha_i \left(\mathbf{x}_i^\top \mathbf{x}\right) + b\right). \end{aligned} \tag{29}$$

Note that the scalar product in this dual formulation can be directly replaced by the kernel mapping $\mathrm{k}(\mathbf{x}_i, \mathbf{x})$, opening the possibility for the non-linear classifiers. This expression does not directly depend on the dimensionality N of the data but on the number of training examples M. As long as we are able to evaluate the scalar product $(\mathbf{x}_i^\top \mathbf{x})$ the dimensionality could be arbitrary, even infinite.

Non-separable Data

So far we have only considered the separable case which corresponds to an empirical error of zero (cf. Theorem 2). However for many practical applications this assumption is violated. If the data is not linearly separable then problem (23) has no feasible solution. By allowing for some errors we might get better results and avoid overfitting effects (cf. Figure 2).

Therefore a "good" trade-off between the empirical risk and the complexity term in (9) needs to be found. Using a technique which was first proposed in (Bennett and Mangasarian, 1992) and later used for SVMs in (Cortes and Vapnik, 1995), one introduces slack-variables to relax the hard-margin constraints:

$$y_i((\mathbf{w}^\top \mathbf{x}_i) + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad i = 1, \ldots, M, \tag{30}$$

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

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

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

$$\max_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{M} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{M} \alpha_i \alpha_j y_i y_j \left(\mathbf{x}_i^\top \mathbf{x}_j\right), \tag{31}$$

$$\text{subject to} \quad 0 \leq \alpha_i \leq C, \; i = 1, \ldots, M, \tag{32}$$

$$\sum_{i=1}^{M} \alpha_i y_i = 0. \tag{33}$$

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

The threshold b can be computed by exploiting the fact that for all support vectors $\mathbf{x}_i$ with $0 < \alpha_i < C$, the slack variable ξ_i is zero. This follows from the Karush-Kuhn-Tucker (KKT) conditions (cf. (34) below). Thus, for any support vector $\mathbf{x}_i$ with $\alpha_i < C$ holds:

$$y_i \left(\sum_{j=1}^{M} y_j \alpha_j \left(\mathbf{x}_i^\top \mathbf{x}_j\right) + b \right) = 1.$$

Averaging over these patterns I yields a numerically stable solution:

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

Sparsity

The Karush-Kuhn-Tucker (KKT) conditions are the necessary conditions for an optimal solution of a non-linear programming problem (e.g. Bertsekas, 1995; Luenberger, 1973). The conditions are particularly simple for the dual SVM problem (31), (32) and (33) (Vapnik, 1982):

$$\begin{aligned} \alpha_i = 0 &\Rightarrow \ y_i f(\mathbf{x}_i) \geq 1 \ \text{ and } \ \xi_i = 0, \\ 0 < \alpha_i < C &\Rightarrow \ y_i f(\mathbf{x}_i) = 1 \ \text{ and } \ \xi_i = 0, \\ \alpha_i = C &\Rightarrow \ y_i f(\mathbf{x}_i) \leq 1 \ \text{ and } \ \xi_i \geq 0. \end{aligned} \tag{34}$$

They reveal one of the most important properties of SVMs: the solution is sparse in $\boldsymbol{\alpha}$. For all examples outside the margin area the optimal α_i's are zero. Specifically, the KKT conditions show that only such α_i connected to a training pattern $\mathbf{x}_i$, which is either on the edge of (i.e. $0 < \alpha_i < C$ and $y_i f(\mathbf{x}_i) = 1$) or inside the margin area (i.e. $\alpha_i = C$ and $y_i f(\mathbf{x}_i) < 1$) are non-zero. These are exactly the *support vectors* as mentioned in section 3.2.

5.2 Decomposition

The practical usefulness of SVM stems from their ability to provide arbitrary non-linear separation boundaries and at the same time to control generalization ability through the parameter C and the kernel parameters. In order to utilize these features it is necessary to work with the dual formulation (31)–(33) of the SVM training problem. This can be difficult from the computational point of view, for two reasons:

1. One needs to solve the quadratic programming problem with as many variables as the number M of available data points (this can be quite large, up to 10^5–10^6).
2. Merely to define the quadratic problem formally, one needs to store the $M \times M$ kernel matrix, which poses an insurmountable storage problem for large datasets.

Because of these implications, it is usually impossible to use the standard optimization tools (e.g. MINOS, CPLEX, LOQO) for solving the SVM training problems on datasets containing larger than 10,000 examples. In the following sections the decomposition techniques are presented, which use the special structure of the SVM problem to provide efficient training algorithms.

Basic principles

The key idea of decomposition is to freeze all but a small number of optimization variables and to solve a sequence of constant-size problems. The set of variables optimized at a current iteration is referred to as the *working set*. Because the working set is re-optimized, the value of the objective function

is improved at each iteration provided the working set is not optimal before re-optimization.

The mathematics of the decomposition technique can be best seen in the matrix notation. Let $\boldsymbol{\alpha} = (\alpha_1, \dots \alpha_M)^\top$, let $\mathbf{y} = (y_1, \dots, y_M)^\top$, let H be the matrix with the entries $H_{ij} = y_i y_j \, \mathrm{k}(\mathbf{x}_i, \mathbf{x}_j)$, and let $\mathbf{1}$ denote the vector of length M consisting of all 1s. Then the dual SVM problem (31)–(33) can be written in the matrix form as:

$$\max_{\boldsymbol{\alpha}} \quad \mathbf{1}^\top \boldsymbol{\alpha} - \frac{1}{2} \boldsymbol{\alpha}^\top H \boldsymbol{\alpha}, \tag{35}$$

$$\text{subject to} \quad \mathbf{y}^\top \boldsymbol{\alpha} = 0, \tag{36}$$

$$\boldsymbol{\alpha} - C\mathbf{1} \leq 0, \tag{37}$$

$$\boldsymbol{\alpha} \geq 0. \tag{38}$$

Let us partition the vector $\boldsymbol{\alpha}$ into $\boldsymbol{\alpha}_B$ of the variables to be included in the working set at a current iteration and the vector $\boldsymbol{\alpha}_N$ of the remaining variables. The matrix H is partitioned as

$$H = \begin{bmatrix} H_{BB} & H_{BN} \\ H_{NB} & H_{NN} \end{bmatrix},$$

with the corresponding parts determined by the index sets B and N. By re-writing the problem (35)–(38) using the partitioned matrix notation, and observing that only the free variables $\boldsymbol{\alpha}_B$ are to be considered as variables, the following sub-problem, to be solved at each iteration, is obtained:

$$\max_{\boldsymbol{\alpha}} \quad (\mathbf{1}_B^\top - \boldsymbol{\alpha}_N^\top H_{NB}) \boldsymbol{\alpha}_B - \frac{1}{2} \boldsymbol{\alpha}_B^\top H_{BB} \boldsymbol{\alpha}_B, \tag{39}$$

$$\text{subject to} \quad \mathbf{y}_B^\top \boldsymbol{\alpha}_B = -\mathbf{y}_N \boldsymbol{\alpha}_N, \tag{40}$$

$$\boldsymbol{\alpha}_B - C\mathbf{1}_B \leq 0, \tag{41}$$

$$\boldsymbol{\alpha}_B \geq 0. \tag{42}$$

Choosing the size q of the working set B relatively small (usually $q \leq 100$) one can ensure that the sub-problem (39)–(42) is easily solved by any optimization tool.

Iteration of this procedure is carried out until the following termination criteria, derived from Karush-Kuhn-Tucker conditions (34), are satisfied to the required precision ϵ:

$$b - \epsilon \leq y_i - \textstyle\sum_{j=1}^{M} y_j \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) \leq b + \epsilon, \qquad \forall i : \; 0 < \alpha_i < C, \tag{43}$$

$$y_i \left(\textstyle\sum_{j=1}^{M} y_j \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) + b \right) \geq 1 - \epsilon, \qquad \forall i : \; \alpha_i = 0, \tag{44}$$

$$y_i \left(\textstyle\sum_{j=1}^{M} y_j \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) + b \right) \leq 1 + \epsilon, \qquad \forall i : \; \alpha_i = C. \tag{45}$$

Working set selection: feasible direction algorithms

The crucial issue in the decomposition technique presented above is the selection of working sets. First, the provision that a working set must be sub-optimal before re-optimization is essential to prevent the algorithm from cycling. Second, the working set selection affects the speed of the algorithm: if sub-optimal working sets are selected more or less at random, the algorithm converges very slowly. Finally, the working set selection is important in theoretical analysis of decomposition algorithms; in particular in the convergence proofs and in the analysis of the convergence rate.

Two main approaches exist to the working set selection in the SVM decomposition algorithms: the heuristic approach and the feasible direction approach. The former has been used in the original paper of Osuna et al. (1997a) on SVM decomposition and has been mainly used in the specific flavor of decomposition algorithms called Sequential Minimal Optimization (SMO), presented in the next section. The feasible direction decomposition algorithms root in the SVM$^{\text{light}}$ algorithm of Joachims (1999) for pattern recognition, with the formal connection to the feasible direction methods of non-linear programming established by Laskov (2002).

The notion of a "feasible direction" stems from the classical techniques of non-linear programming subject to linear constraints (Zoutendijk, 1960; Bertsekas, 1995). It refers to the direction along which any step of the magnitude δ satisfying $0 < \delta \leq \delta^0$, for some fixed δ^0, results in a feasible solution to the non-linear program. For any non-linear program, finding the feasible direction amounts to a solution of a linear programming problem. In particular, for the dual SVM training problem (31)–(33) the following problem must be solved:

$$\max_{\mathbf{d}} \quad \mathbf{g}^\top \mathbf{d}, \tag{46}$$

$$\text{subject to} \quad \mathbf{y}^\top \mathbf{d} = 0, \tag{47}$$

$$d_i \geq 0, \quad \text{for all } \alpha_i = 0, \tag{48}$$

$$d_i \leq 0, \quad \text{for all } \alpha_i = C, \tag{49}$$

$$||\mathbf{d}||_2 \leq 1, \tag{50}$$

where $\mathbf{g}$ is the gradient of the objective function (31). Solving the feasible direction problem exactly at each iteration is inefficient because the linear program (46)–(50) has all M variables. However, an approximate solution to the feasible direction problem can be efficiently found by using the normalization $d_i \in \{-1, 0, 1\}$ instead of (50) and requiring that the number of positive direction components is equal to the number of the negative components. In this case, the solution is obtained by sorting all examples by the quantity $g_i y_i$, and selecting $q/2$ examples with the largest and $q/2$ examples with the smallest values. In fact, by using a Heap data structure, sorting can be avoided, and the entire selection can be executed in $O(q \log M)$ time. The motivation behind the quantity $g_i y_i$ can be traced back to the first-order Karush-Kuhn-

Tucker conditions (Laskov, 2002), which provides the solid formal background for the feasible direction SVM decomposition.

Convergence of the feasible direction SVM decomposition has been proved by Lin (2001), and the linear convergence rate has been observed experimentally (Laskov, 2002).

Sequential Minimal Optimization

The Sequential Minimal Optimization (SMO) algorithm proposed by Platt (1999) is another popular and efficient algorithm for the SVM training. In this algorithm the idea of decomposition is taken to its extreme by reducing the working sets to two points—the smallest size for which it is possible to maintain the equality constraint (33). For two points the optimal solution can be computed analytically without calling an optimization tool.

The analytical computation of the optimal solution is based on the following idea: given the solution $(\alpha_1^{\text{old}}, \alpha_2^{\text{old}})$, the optimal update is computed to obtain the solution $(\alpha_1^{\text{new}}, \alpha_2^{\text{new}})$. To carry out the update, first the constraints (32)–(33) have to be obeyed. The geometry of these constraints depends on whether the labels y_1 and y_2 are equal or not. The two possible configurations are shown in Figure 10. If $y_1 \neq y_2$ (left picture) the solution should be sought along the line $\alpha_1 - \alpha_2 = \gamma$, where $\gamma = \alpha_1^{\text{old}} + y_1 y_2 \alpha_2^{\text{old}}$. If $y_1 = y_2$ (right picture) the solution should be sought along the line $\alpha_1 + \alpha_2 = \gamma$. If the solution transgresses the bound of the box, it should be clipped at the bound.

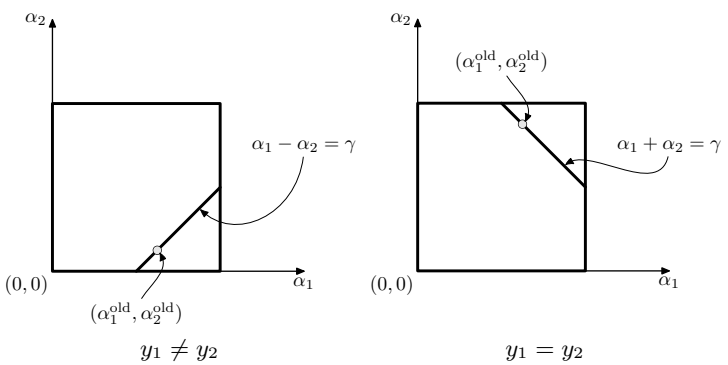

Fig. 10. Constraints of the SVM training problem with two examples.

The optimal values of the variables along the line are computed by eliminating one of the variables through the equality constraint and finding the unconstrained minimum of the objective function, for example, eliminating α_1 one obtains the following update rule for α_2:

$$\alpha_2^{\text{new}} = \alpha_2^{\text{old}} - \frac{y_2(E_1 - E_2)}{\eta}, \tag{51}$$

where

$$E_1 = \sum_{j=1}^{M} y_j \alpha_j \, \mathrm{k}(\mathbf{x}_1, \mathbf{x}_j) + b - y_1, \tag{52}$$

$$E_2 = \sum_{j=1}^{M} y_j \alpha_j \, \mathrm{k}(\mathbf{x}_2, \mathbf{x}_j) + b - y_2, \tag{53}$$

$$\eta = 2\,\mathrm{k}(\mathbf{x}_1, \mathbf{x}_2) - \mathrm{k}(\mathbf{x}_1, \mathbf{x}_1) - \mathrm{k}(\mathbf{x}_2, \mathbf{x}_2). \tag{54}$$

Next, the bound constraints should be taken care of. Depending on the geometry, one computes the following lower and upper bounds on the value of the variable α_2:

$$L = \begin{cases} \max\,(0, \alpha_2^{\mathrm{old}} - \alpha_1^{\mathrm{old}}), & \text{if } y_1 \neq y_2, \\ \max\,(0, \alpha_2^{\mathrm{old}} + \alpha_1^{\mathrm{old}} - C), & \text{if } y_1 = y_2, \end{cases}$$

$$H = \begin{cases} \min\,(C, C + \alpha_2^{\mathrm{old}} - \alpha_1^{\mathrm{old}}), & \text{if } y_1 \neq y_2, \\ \min\,(C, \alpha_2^{\mathrm{old}} + \alpha_1^{\mathrm{old}}), & \text{if } y_1 = y_2. \end{cases}$$

The constrained optimum is then found by clipping the unconstrained optimum to the ends of the line segment:

$$\alpha_2^{\mathrm{new}} := \begin{cases} H, & \text{if } \alpha_2^{\mathrm{new}} \geq H, \\ L, & \text{if } \alpha_2^{\mathrm{new}} \leq L, \\ \alpha_2^{\mathrm{new}}, & \text{otherwise.} \end{cases}$$

Finally, the value of α_1^{new} is computed:

$$\alpha_1^{\mathrm{new}} = \alpha_1^{\mathrm{old}} + y_1 y_2 (\alpha_2^{\mathrm{old}} - \alpha_2^{\mathrm{new}}). \tag{55}$$

The working set selection in the SMO algorithm is carried out by means of two heuristics. The "first choice" heuristic is responsible for the choice of the first example in each pair. Following this heuristic, all examples that violate the KKT condition (34) are used in turns as the first example in the pair. More precisely, the algorithm makes repeated passes only through the examples whose α_i is strictly between then bounds, and only when all such examples satisfy the KKT conditions the sweep over the entire data is done to bring in new examples. The "second choice" heuristic is responsible for the selection of the second example in the pair. It is intended to bring in such an example that results in the largest step taken during the joint optimization (51). As a cheap approximation of this step the numerator $|E_1 - E_2|$ is taken (the denominator, which involves evaluation of kernels, can be expensive to compute). Following the strategy to maximize $|E_1 - E_2|$, the SMO algorithm chooses the example with the largest E_2, if E_1 is negative, and the example with the smallest E_2, if E_1 is positive. Under unusual circumstances, when no progress can be made using the second heuristic above, a hierarchy of weaker heuristics is applied, the details of which are provided by Platt (1999).

5.3 Incremental Support Vector Optimization

Many real-life machine learning problems can be more naturally viewed as online rather than batch learning problems. Indeed, the data is often collected continuously in time, and, more importantly, the concepts to be learned may also evolve in time. Significant effort has been spent in the recent years on the development of online SVM learning algorithms (e.g. Rüping, 2002; Kivinen et al., 2001; Ralaivola and d'Alché Buc, 2001). An elegant solution to online SVM learning is the incremental SVM proposed by Cauwenberghs and Poggio (2001), which provides a framework for exact online learning.

The incremental SVM learning algorithm can be best presented using the following abstract form of the SVM optimization problem:

$$\max_{\mu} \min_{\substack{0 \leq \boldsymbol{\alpha} \leq C \\ \mathbf{a}^\top \boldsymbol{\alpha} + b = 0}} : W = -\mathbf{c}^\top \boldsymbol{\alpha} + \frac{1}{2} \boldsymbol{\alpha}^\top K \boldsymbol{\alpha} + \mu(\mathbf{a}^\top \boldsymbol{\alpha} + b), \tag{56}$$

where $\mathbf{c}$ and $\mathbf{a}$ are $M \times 1$ vectors, K is the $M \times M$ kernel matrix and b is a scalar. By defining the meaning of the abstract parameters $\mathbf{a}$, b and $\mathbf{c}$ for the particular SVM problem at hand, one can use the same algorithmic structure for different SVM algorithms. In particular, for the standard support vector classifiers (Vapnik, 1998), take $\mathbf{c} = \mathbf{1}, \mathbf{a} = \mathbf{y}, b = 0$ and the given regularization constant C; the same definition applies to the ν-SVM (Schölkopf et al., 2000) except that $C = \frac{1}{N\nu}$.

The incremental SVM provides a procedure for adding one example to an existing optimal solution. When a new point k is added, its weight α_k is initially set to 0. Then the weights of other points and μ should be updated, in order to obtain the optimal solution for the enlarged dataset. Likewise, when a point k is to be removed from the dataset, its weight is forced to 0, while updating the weights of the remaining points and μ so that the solution obtained with $\alpha_k = 0$ is optimal for the reduced dataset. The online learning follows naturally from the incremental learning: the new example is added while some old example is removed from the working set.

The basic principle of the incremental SVM (Cauwenberghs and Poggio, 2001) is that *updates to the state of the example k should keep the remaining examples in their optimal state*. In other words, the KKT conditions (34) expressed for the gradients g_i:

$$g_i = -c_i + K_{i,:}\boldsymbol{\alpha} + \mu a_i \begin{cases} \geq 0, & \text{if } \alpha_i = 0, \\ = 0, & \text{if } 0 < \alpha_i < C, \\ \leq 0, & \text{if } \alpha_i = C, \end{cases} \tag{57}$$

$$\frac{\partial W}{\partial \mu} = \mathbf{a}^\top \boldsymbol{\alpha} + b = 0, \tag{58}$$

must be maintained for all the examples, except possibly for the current example k. Let S denote the set of unbounded support vectors and R the set

of other examples. In order to maintain the KKT conditions (57), one can express increments to the weights of the unbounded support vectors and to the gradients of other examples as a linear function of the increment of the current example's weight $\Delta\alpha_k$:

$$\Delta\alpha_S = \beta\Delta\alpha_k, \quad \Delta g_R = \gamma\Delta\alpha_k. \tag{59}$$

The sensitivity relations (59) only make sense for the fixed composition of sets S and R. To proceed with learning, we need to detect the largest increment $\Delta\alpha_k^{\max}$ such that the sets S and R remain intact. After changing the SVM state according to the relations (59) evaluated for the increment $\Delta\alpha_k^{\max}$, the sensitivity vectors β and γ must be recomputed accordingly (depending of whether an example enters or leaves the set S). For the details of accounting the reader should refer to (Cauwenberghs and Poggio, 2001; Tax and Laskov, 2003). The iteration terminates when the current element satisfies the KKT conditions (57) as well.

5.4 Large-scale Learning with SVM

The kernel trick has enabled SVMs to be applied to several domains and it has been one of the reasons SVMs are used with great success, often delivering state-of-the-art results. Unfortunately, it is also the kernel trick that limits the speed of applying SVMs and renders them unsuitable in several large-scale applications. The reason is that for predicting the class label of a single example, we need to compare it with all support vectors, that is, compute

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{M} \alpha_i y_i \,\mathrm{k}(\mathbf{x}_i, \mathbf{x}) + b\right).$$

Consequently, the speed for application of an SVM decays linearly with the number of support vectors. It is this same operation that slows down training in most SVM implementations and potentially the cause for a shift in interest back to linear SVMs for large-scale applications. In the linear case, the decision function of an SVM (without bias term b) takes the form

$$f(\mathbf{x}) = \operatorname{sgn}\left(\mathbf{w}^T\mathbf{x}\right)$$

and can be computed in linear time in the number of dimensions of the feature space, i.e. $\mathcal{O}(\dim(\mathcal{X}))$. Furthermore, if either the vectors $\mathbf{x}$ or $\mathbf{w}$ are sparse, this run-time can be further reduced to processing non-zero entries only.

The first linear SVM for large-scale application has been proposed by Joachims (2006) (SVM^{perf}) and builds on solving the underlying optimization problem using the concept of *cutting planes* (Kelly, 1960). The resulting linear SVM achieves an ε-precise solution in time linear in the number of samples and dimensions, that is, $\mathcal{O}(M\dim(\mathcal{X}))$.

Theorem 4 (Joachims, 2006). *For any $\varepsilon > 0$, $C > 0$, and any training sample $\mathcal{Z} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_M, y_M)\}$, SVM^{perf} terminates after at most*

$$\max\left\{\frac{2}{\varepsilon}, \frac{8CR^2}{\varepsilon^2}\right\}$$

iterations, where $R = \max_{i=1}^{M} ||\mathbf{x}_i||$.

This method has been further refined by Teo et al. (2007, 2010) and later on by Franc and Sonnenburg (2008, 2009) who improve convergence rates to $\mathcal{O}(\frac{1}{\varepsilon})$. Following a different line of research, Fan et al. (2008) have developed a dual coordinate descent approach that performs similar, but drastically reduces memory requirements, as no set of cutting planes needs to be stored. For that reason we will explain this algorithm in more detail.

The idea of the dual coordinate descent method by Fan et al. (2008) is to perform coordinate descent on a single dual variable α_i while maintaining the SVM normal vector $\mathbf{w}$. Note that SMO (cf. Section 5.2) updates two variables at the same time, while coordinate descent updates a single variable only. More formally, considering the dual objective function $D(\boldsymbol{\alpha}) = \frac{1}{2}\sum_{i,j=1}^{M} \alpha_i \alpha_j y_i y_j \,\mathrm{k}(\mathbf{x}_i, \mathbf{x}_j) - \sum_{i=1}^{M} \alpha_i$, one solves the dual optimization problem for a single variable α_i via

$$\begin{aligned} \min_{d} \quad & D(\boldsymbol{\alpha} + d\,\mathbf{1_i}) \\ \text{subject to} \quad & 0 \leq \alpha_i + d \leq C \end{aligned}$$

where $\mathbf{1_i}$ is the vector whose $i-$th component is 1 while its other components are zero, i.e. $\mathbf{1_i} = (0, \ldots, 0, 1, 0, \ldots, 0)^T$. By inserting $(\boldsymbol{\alpha} + d\,\mathbf{1_i})$ into D and removing terms independent of d, we arrive at

$$\begin{aligned} \min_{d} \quad & \frac{d^2}{2}\,\mathrm{k}(\mathbf{x}_i, \mathbf{x}_i) + d\left(\sum_{j=1}^{M} y_i y_j \,\mathrm{k}(\mathbf{x}_i, \mathbf{x}_j)\alpha_j - 1\right) \\ \text{subject to} \quad & 0 \leq \alpha_i + d \leq C. \end{aligned} \tag{60}$$

If α_i is optimal, that is $d = 0$, the projected gradient is 0 (cf. PG in Algorithm 1) and we leave this coordinate unchanged. Otherwise, asserting $\mathrm{k}(\mathbf{x}_i, \mathbf{x}_i) > 0$ one computes the new value of α_i by determining the derivative of Equation (60) with respect to d. Setting the derivative to zero while ensuring $0 \leq \alpha_i \leq C$ finally leads to the following update rule

$$\alpha_i \leftarrow \min\left(\max\left(\alpha_i - \frac{\sum_{j=1}^{M} y_i y_j \,\mathrm{k}(\mathbf{x}_i, \mathbf{x}_j)\alpha_j - 1}{\mathrm{k}(\mathbf{x}_i, \mathbf{x}_i)}, 0\right), C\right).$$

It should be noted that for efficiency the $\mathrm{k}(\mathbf{x}_i, \mathbf{x}_i)$ in the denominator can be precomputed. In addition, we know from Equation (25) that $\mathbf{w} = \sum_{i=1}^{M} \alpha_i y_i \mathbf{x}_i$ and subsequently the numerator can be written as $y_i \mathbf{w}^T \mathbf{x}_i - 1$.

In addition, we avoid to recompute $\mathbf{w}$ in each iteration by starting with $\boldsymbol{\alpha} = \mathbf{0}$ and $\mathbf{w} = \mathbf{0}$, and only compute updates on $\mathbf{w}$ via $\mathbf{w} \leftarrow \mathbf{w} + (\alpha_i - \bar{\alpha}_i) y_i \mathbf{x}_i$. As a result, dual coordinate descent closely resembles a perceptron-style algorithm with a step size obtained via the SVM dual.

Algorithm 1 Dual Coordinate Descent (Fan et al., 2008)

$\boldsymbol{\alpha} = \mathbf{0}$ and $\mathbf{w} = \mathbf{0}$
repeat
  for $i = 1, \ldots, M$ **do**
    $G = y_i \mathbf{w}^T \mathbf{x}_i - 1$

$$PG = \begin{cases} \min(G, 0) & \text{if } \alpha_i = 0 \\ \max(G, 0) & \text{if } \alpha_i = C \\ G & \text{if } 0 < \alpha_i < C \end{cases}$$

    if $|PG| \neq 0$ **then**
      $\bar{\alpha}_i \leftarrow \alpha_i$
      $\alpha_i \leftarrow \min(\max(\alpha_i - G/\bar{Q}_{ii}, 0), C)$
      $\mathbf{w} \leftarrow \mathbf{w} + (\alpha_i - \bar{\alpha}_i) y_i \mathbf{x}_i$
    end if
  end for
until Optimality

The full algorithm is outlined in Algorithm 1. It can be shown that dual coordinate descent reaches a ε-precise solution $D(\alpha) \leq D(\alpha^*) + \varepsilon$ in $\mathcal{O}(\log(\frac{1}{\varepsilon}))$ iterations with an iteration cost of $\mathcal{O}(M \dim(\mathcal{X}))$ (Fan et al., 2008).

Besides dual coordinate descent, several other approaches for efficiently training linear SVMs have been proposed, for example, based on stochastic learning concepts (e.g. Shwartz et al., 2007; Bottou and Bousquet, 2008; Bordes et al., 2009; Shwartz et al., 2007; Yu et al., 2010). Finally, there have been a number of attempts to combine the advantages of fast linear SVM solvers with the non-linearity of kernels (Joachims and Yu, 2009; Chang et al., 2010; Sonnenburg and Franc, 2010)

6 Extensions of SVM

6.1 Regression

In this subsection we will give a short overview of the idea of Support Vector Regression (SVR). A regression problem is given whenever $\mathcal{Y} = \mathbb{R}$ for the training dataset $\mathcal{Z} = \{(\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y} | i = 1, \ldots, M\}$ (cf. Section 2.1) and our interest is to find a function of the form $f : \mathcal{X} \to \mathbb{R}$.

In our discussion of statistical learning in section 2.1 we have not talked about loss functions except for saying that they should be non-negative functions of the form (1). The particular form of the loss function depends on

the learning task. For the pattern recognition problem the 0/1-loss function was used (cf. Theorem 2). For the regression problem the following two loss functions are common: the simple squared loss

$$\ell(f(\mathbf{x}), y) = (f(\mathbf{x}) - y)^2, \tag{61}$$

and the ϵ-insensitive loss

$$\ell(f(\mathbf{x}), y) = \begin{cases} |f(\mathbf{x}) - y| - \epsilon, & \text{if } |f(\mathbf{x}) - y| > \epsilon, \\ 0, & \text{otherwise.} \end{cases} \tag{62}$$

For $\epsilon = 0$ the ϵ-insensitive loss equals the ℓ_1-norm, otherwise it linearly penalizes deviations from the correct predictions by more than ϵ.

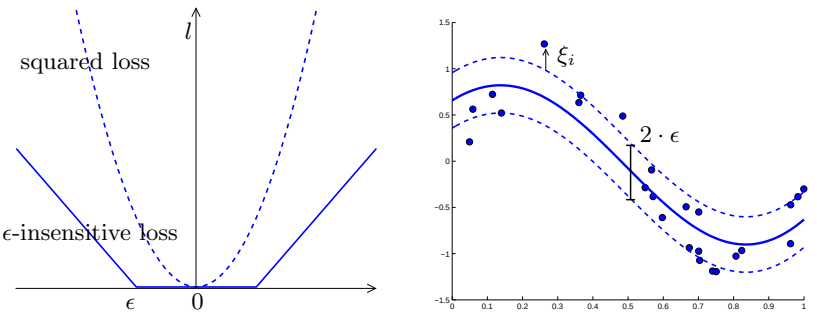

Fig. 11. The left subplot shows the two different loss functions for the regression problem. The right subplot gives a regression function derived with an ϵ-insensitive loss function. The solid line indicates the function, the dashed lines indicate the ϵ-tube around this function.

In the left subplot of Figure 11 the two error functions are shown. In the right subplot a regression function using the ϵ-insensitive loss function is shown for some artificial data. The dashed lines indicate the boundaries of the area where the loss is zero (the "tube"). Clearly most of the data is within the tube.

Similarly to the classification task, one is looking for the function that best describes the values y_i. In classification one is interested in the function that separates two classes; in contrast, in regression one looks for the function that contains the given dataset in its ϵ-tube. Some data points can be allowed to lie outside the ϵ-tube by introducing the slack-variables.

The primal formulation for the SVR is then given by:

$$\begin{aligned} \min_{\mathbf{w},b,\xi^{(*)}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{M}(\xi_i + \xi_i^*), \\ \text{subject to} \quad & ((\mathbf{w}^\top \mathbf{x}_i) + b) - y_i \leq \epsilon + \xi_i, \\ & y_i - ((\mathbf{w}^\top \mathbf{x}_i) + b) \leq \epsilon + \xi_i^*, \\ & \xi_i^{(*)} \geq 0, \quad i = 1, \ldots, M. \end{aligned}$$

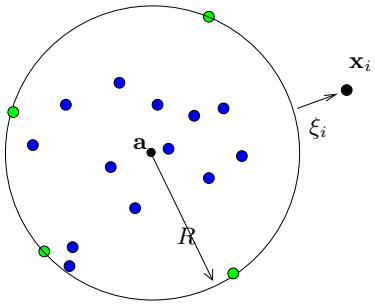

Fig. 12. Schematic illustration of the hypersphere model for describing a target class of data. The center $\mathbf{a}$ and the radius R should be optimized to minimize the volume of the captured space.

In contrast to the primal formulation for the classification task, we have to introduce two types of slack-variables ξ_i and ξ_i^*, one to control the error induced by observations that are larger than the upper bound of the ϵ-tube, and the other for the observations smaller than the lower bound. To enable the construction of a non-linear regression function, a dual formulation is obtained in the similar way to the classification SVM and the kernel trick is applied.

As a first application, SVR has been studied for analysis of time series by Müller et al. (1997). Further applications and an extensive description of SVR are provided by Vapnik (1998); Cristianini and Shawe-Taylor (2000); Schölkopf and Smola (2002).

6.2 One-class Classification

Another common problem of statistical learning is one-class classification (novelty detection). Its fundamental difference from the standard classification problem is that the training data is not identically distributed to the test data. The dataset contains two classes: one of them, the target class, is well sampled, while the other class it absent or sampled very sparsely. Schölkopf et al. (2001) have proposed an approach in which the target class is separated from the origin by a hyperplane. Alternatively (Tax and Duin, 2001), the target class can be modeled by fitting a hypersphere with minimal radius around it. We present this approach, illustrated in Figure 12, in more detail below.

Mathematically the problem of fitting a hypersphere around the data is stated as:

$$\begin{aligned} \min_{R,\mathbf{a}} \quad & R^2 + C\sum_{i=0}^{M}\xi_i, \\ \text{subject to} \quad & ||\mathbf{x}_i - \mathbf{a}||^2 \leq R^2 + \xi_i,\ i = 1,\ldots,M, \\ & \xi_i \geq 0, \end{aligned} \tag{63}$$

where $\mathbf{a}$ is the center of the sphere, and R is the radius. Similarly to the SVM we make a "soft" fit by allowing non-negative slacks ξ_i. One can likewise apply the kernel trick by deriving the dual formulation of (63):

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & \sum_{i=1}^{M} \alpha_i \, \mathrm{k}(\mathbf{x}_i, \mathbf{x}_i) - \frac{1}{2} \sum_{i,j=1}^{M} \alpha_i \alpha_j \, \mathrm{k}(\mathbf{x}_i, \mathbf{x}_j), \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C, \; i = 1, \ldots, M, \\ & \sum_{i=1}^{M} \alpha_i = 1. \end{aligned} \tag{64}$$

The parameter C can be interpreted (Schölkopf et al., 2001) as the reciprocal of the quantity $\frac{1}{M\nu}$, where ν is an upper bound for the fraction of objects outside the boundary.

To decide whether a new object belongs to the target class one should determine its position with respect to the sphere using the formula

$$f(\mathbf{x}) = \operatorname{sign}(R^2 - \mathrm{k}(\mathbf{x}, \mathbf{x}) + 2 \sum_i \alpha_i \, \mathrm{k}(\mathbf{x}, \mathbf{x}_i) - \sum_{i,j} \alpha_i \alpha_j \, \mathrm{k}(\mathbf{x}_i, \mathbf{x}_j)). \tag{65}$$

An object with positive sign belongs to the target class and vice versa. An incremental version of the one-class classification SVM can be obtained using the approach of section 5.3, with the parameters of the abstract formulation (56) defined as: $\mathbf{c} = \operatorname{diag}(K)$, $\mathbf{a} = \mathbf{y}$ and $b = -1$ (Laskov et al., 2006).

7 Applications

7.1 Optical character recognition (OCR)

One of the first real-world experiments carried out with SVM was done at AT&T Bell Labs using optical character recognition (OCR) data (Cortes and Vapnik, 1995; Schölkopf et al., 1995). These early experiments already showed the astonishingly high accuracies for SVMs which was on a par with convolutive multi-layer perceptrons. Below we list the classification performance of SVM, some variants not discussed in this chapter, and other classifiers on the USPS (US-Postal Service) benchmark (parts from (Simard et al., 1998)). The task is to classify handwritten digits into one of ten classes. Standard SVMs as presented here already exhibit a performance similar to other methods.

Linear PCA & Linear SVM (Schölkopf et al., 1998b) 8.7%
k-Nearest Neighbor 5.7%
LeNet1 (Bottou et al., 1994) 4.2%
Regularised RBF Networks (Rätsch, 1998) 4.1%
Kernel-PCA & linear SVM (Schölkopf et al., 1998b) 4.0%
SVM (Schölkopf et al., 1995) 4.0%

Invariant SVM (Schölkopf et al., 1998a) 3.0%
Boosting (Drucker et al., 1993) 2.6%
Tangent Distance (Simard et al., 1998) 2.5%
Human error rate 2.5%

A benchmark problem larger than the USPS dataset (7291 patterns) was collected by NIST and contains 120000 handwritten digits. Invariant SVMs achieve an error rate of 0.7% (DeCoste and Schölkopf, 2002) on this challenging and more realistic dataset, slightly better than the tangent distance method (1.1%) or single neural networks (LeNet 5: 0.9%). An ensemble of LeNet 4 networks that was trained on a vast number of artificially generated patterns (using invariance transformations) is on a par with the best SVM, and also achieved the accuracy of 0.7% (LeCun et al., 1995).

7.2 Text categorization and text mining

The high dimensional problem of text categorization is an application for which SVMs have performed particularly well. A popular benchmark is the Reuters-22173 text corpus containing 21450 news stories, collected by Reuters in 1997, that are partitioned and indexed into 135 different categories. The features typically used to classify Reuters documents are 10000-dimensional vectors containing word frequencies within a document. SVMs have achieved excellent results using such a coding (see e.g. Joachims, 1997).

7.3 Network Intrusion Detection

The one-class formulation of the SVM presented in Section 6.2 provides another example for a successful application in computer security. In the last years, the amount and diversity of computer attacks has drastically increased, rendering regular defenses based on manually crafted detection rules almost ineffective. By contrast, the one-class SVM provides means for identifying unknown attacks automatically as novelties and thus has gained a strong focus in security research (e.g. Eskin et al., 2002; Nassar et al., 2008; Wahl et al., 2009; Perdisci et al., 2009).

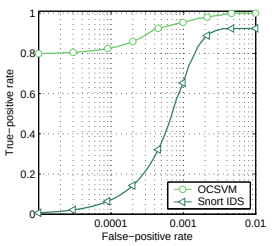

Fig. 13. One-class SVM vs. regular intrusion detection system (Rieck, 2009).

As an example of this research, we present results from an application of the one-class SVM for detection of unknown attacks in network traffic of the FTP protocol (see Rieck, 2009). Figure 13 shows the detection performance of the one-class SVM and the rule-based detection system "Snort". For analysing network data the one-class SVM is applied using a string kernel defined over n-grams as presented in Section 4.3. The one-class SVM significantly outperforms the regular detection system by identifying 80% of the unknown attacks with almost

no false alarms, demonstrating the ability of SVMs to effectively generalize from data and surpass manually crafted rules and heuristics.

7.4 Bioinformatics

SVMs are widely used in bioinformatics, defining the state-of-the-art in several applications, like splice site detection (Sonnenburg et al., 2002; Degroeve et al., 2005; Sonnenburg et al., 2007b; Rätsch et al., 2007; Sonnenburg and Franc, 2010), detection of transcription starts (Sonnenburg et al., 2006), translation initiation site detection (Zien et al., 2000) and detection of several other genomic signals (Sonnenburg et al., 2008). Various further applications have been considered, like gene array expression monitoring (Brown et al., 2000), remote protein homology detection (Jaakkola et al., 2000; Leslie et al., 2002), protein sub-cellular localization Ong and Zien (2008), to detect protein protein interactions (Ben-Hur and Noble, 2005) to analyze phylogenetic trees Vert (2002). For further information, the interested reader is referred to the tutorial on support vector machines and kernels (Ben-Hur et al., 2008).

7.5 Other Applications

There are numerous other applications to which SVM were successfully applied. Examples are object and face recognition tasks (Osuna et al., 1997b), inverse problems (Vapnik, 1998; Weston et al., 1999), drug design in computational chemistry (Warmuth et al., 2003; Müller et al., 2005) and brain computer interfacing (Blankertz et al., 2007; Müller et al., 2008). A large collection of links to SVM applications is available at `www.clopinet.com/isabelle/Projects/SVM/applist.html`.

8 Summary and Outlook

We have shown in this chapter how the problem of learning from data can be cast formally into the problem of estimating functions from given observations. We have reviewed some basic notation and concepts from statistics and especially from statistical learning theory. The latter provides us with two extremely important insights: (i) what matters the most is not the dimensionality of the data but the complexity of the function class we choose our estimate from, (ii) consistency plays an important role in successful learning. Closely related to these two questions is the issue of regularization. Regularization allows us to *control* the complexity of our learning machine and usually suffices to achieve consistency.

As an application of statistical learning theory we have presented the technique for constructing a maximum margin hyperplane. Whilst it is satisfactory to have a technique at hand that implements (at least partially) what the

theory justifies, the algorithm is only capable of finding (linear) hyperplanes. To circumvent this restriction we introduced kernel functions yielding SVMs. Kernel functions allow us to reformulate many algorithms in some kernel feature space that is non-linearly related to the input space and yield powerful, non-linear techniques. Moreover, kernel functions enable us to apply learning techniques in structured domains, such as on string, trees and graphs. This abstraction using the kernel trick is possible whenever we are able to express an algorithm such that it only uses the data in the form of scalar products. However, since the algorithms are still linear in the feature space we can use the same theory and optimization strategies as before.

Kernel algorithms have seen an extensive development over the past years. Among many theoretical (Williamson et al., 1998; Graepel et al., 2000; Bartlett et al., 2002) and algorithmic (Platt, 1999; Joachims, 1999) results on SVM itself, new algorithms using the kernel trick have been proposed (e.g. Kernel PCA (Schölkopf et al., 1998b), Kernel ICA (Harmeling et al., 2003), temporal Kernel CCA (Bießmann et al., 2009) or Bayes–Point machines (Herbrich et al., 2001)). This development is still an ongoing and exciting field of study. A large collection of SVM implementations is contained in the shogun machine learning toolbox (Sonnenburg et al., 2010). Furthermore, various kernel-based learning methods are available from `http://mloss.org/`

References

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

H. Akaike. A new look at the statistical model identification. *IEEE Trans. Automat. Control*, 19(6):716–723, 1974.

N. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68:337–404, 1950.

A. Barron, L. Birgé, and P. Massart. Risk bounds for model selection via penalization. *Probability Theory and Related Fields*, 113:301–415, 1999.

P. Bartlett and S. Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3: 463–482, Nov. 2002.

P. Bartlett, P. Long, and R. Williamson. Fat-shattering and the learnability of real-valued functions. *Journal of Computer and System Sciences*, 52(3): 434–452, June 1996.

P. Bartlett, O. Bousquet, and S. Mendelson. Localized rademacher complexities. In J. Kivinen and R. Sloan, editors, *Proceedings COLT*, volume 2375 of *Lecture Notes in Computer Science*, pages 44–58. Springer, 2002.

A. Ben-Hur and W. S. Noble. Kernel methods for predicting protein-protein interactions. *Bioinformatics*, 21 suppl 1:i38–i46, 2005.

A. Ben-Hur, C. Ong, S. Sonnenburg, B. Schölkopf, and G. Rätsch. Support vector machines and kernels for computational biology. *PLoS Computational Biology*, 4(10):e1000173, 2008.

K. Bennett and O. Mangasarian. Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software*, 1: 23–34, 1992.

D. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, MA, 1995.

F. Bießmann, F. C. Meinecke, A. Gretton, A. Rauch, G. Rainer, N. Logothetis, and K.-R. Müller. Temporal kernel canonical correlation analysis and its application in multimodal neuronal data analysis. *Machine Learning*, 79(1-2):5—27, 2009. doi: 10.1007/s10994-009-5153-3. URL `http://www.springerlink.com/content/e1425487365v2227`.

C. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

B. Blankertz, G. Curio, and K.-R. Müller. Classifying single trial EEG: Towards brain computer interfacing. In T. G. Diettrich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Inf. Proc. Systems (NIPS 01)*, volume 14, pages 157–164, 2002.

B. Blankertz, G. Dornhege, M. Krauledat, K.-R. Müller, and G. Curio. The non-invasive Berlin Brain-Computer Interface: Fast acquisition of effective performance in untrained subjects. *NeuroImage*, 37(2):539–550, 2007. URL `http://dx.doi.org/10.1016/j.neuroimage.2007.01.051`.

A. Bordes, L. Bottou, and P. Gallinari. Sgd-qn: Careful quasi-newton stochastic gradient descent. *JMLR*, 10:1737–1754, Jul 2009.

B. Boser, I. 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, 1992.

L. Bottou and O. Bousquet. The tradeoffs of large scale learning. In *NIPS 20*. MIT Press, 2008.

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

M. L. Braun, J. Buhmann, and K.-R. Müller. On relevant dimensions in kernel feature spaces. *Journal of Machine Learning Research*, 9:1875–1908, Aug 2008.

L. Breiman, J. Friedman, J. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth, 1984.

M. Brown, W. Grundy, D. Lin, N. Cristianini, C. Sugnet, T. 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.

N. Cancedda, E. Gaussier, C. Goutte, and J.-M. Renders. Word-sequence kernels. *Journal of Machine Learning Research*, 3(Feb):1059–1082, 2003.

G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. In T. Leen, T. Diettrich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 409–415, 2001.

Y.-W. Chang, C.-J. Hsieh, K.-W. Chang, M. Ringgaard, and C.-J. Lin. Training and testing low-degree polynomial data mappings via linear svm. *JMLR*, 11:1471–1490, 2010.

M. Collins and N. Duffy. Convolution kernel for natural language. In *Advances in Neural Information Proccessing Systems (NIPS)*, volume 16, pages 625–632, 2002.

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

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

M. Cuturi, J.-P. Vert, and T. Matsui. A kernel for time series based on global alignments. In *Proc. of the International Conferenc on Acoustics, Speech and Signal Processing (ICASSP)*, 2007.

M. Damashek. Gauging similarity with n-grams: Language-independent categorization of text. *Science*, 267(5199):843–848, 1995.

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

S. Degroeve, Y. Saeys, B. D. Baets, P. Rouzé, and Y. V. de Peer. Splicemachine: predicting splice sites from high-dimensional local context representations. *Bioinformatics*, 21(8):1332–8, 2005.

L. Devroye, L. Györfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Number 31 in Applications of Mathematics. Springer, New York, 1996.

D. Donoho, I. Johnstone, G. Kerkyacharian, and D. Picard. Density estimation by wavelet thresholding. *Annals of Statistics*, 24:508–539, 1996.

H. Drucker, R. Schapire, and P. Simard. Boosting performance in neural networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 7:705–719, 1993.

R. Duda, P.E.Hart, and D.G.Stork. *Pattern classification*. John Wiley & Sons, second edition, 2001.

E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo. *Applications of Data Mining in Computer Security*, chapter A geometric framework for unsupervised anomaly detection: detecting intrusions in unlabeled data. Kluwer, 2002.

R. Fan, K. Chang, C. Hsieh, X. Wang, and C. Lin. LIBLINEAR: A library for large linear classification. *JMLR*, 9:1871–1874, 2008.

V. Franc and S. Sonnenburg. OCAS optimized cutting plane algorithm for support vector machines. In *Proceedings of the 25nd International Machine Learning Conference*. ACM Press, June 2008. URL `http://cmp.felk.cvut.cz/~xfrancv/ocas/html/index.html`.

V. Franc and S. Sonnenburg. Optimized cutting plane algorithm for large-scale risk minimization. *Journal of Machine Learning Research*, 10(Oct): 2157–2192, 2009.

Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

T. Gärtner, J. Lloyd, and P. Flach. Kernels and distances for structured data. *Machine Learning*, 57(3):205–232, 2004.

F. Girosi. An equivalence between sparse approximation and support vector machines. *Neural Computation*, 10:1455–1480, 1998.

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

T. Graepel, R. Herbrich, and J. Shawe-Taylor. Generalization error bounds for sparse linear classifiers. In *Proc. COLT*, pages 298–303, San Francisco, 2000. Morgan Kaufmann.

S. Harmeling, A. Ziehe, M. Kawanabe, and K.-R. Müller. Kernel-based non-linear blind source separation. *Neural Computation*, 15:1089–1124, 2003.

D. Haussler. Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, UC Santa Cruz, July 1999.

R. Herbrich, T. Graepel, and C. Campbell. Bayes point machines. *Journal of Machine Learning Research*, 1:245–279, Aug. 2001.

T. Jaakkola, M. Diekhans, and D. Haussler. A discriminative framework for detecting remote protein homologies. *J. Comp. Biol.*, 7:95–114, 2000.

T. Joachims. Training linear SVMs in linear time. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 217–226, 2006.

T. Joachims. Text categorization with support vector machines: Learning with many relevant features. Technical Report 23, LS VIII, University of Dortmund, 1997.

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

T. Joachims and C.-N. J. Yu. Sparse kernel svms via cutting-plane training. *Mach. Learn.*, 76(2-3):179–193, 2009.

H. Kashima and T. Koyanagi. Kernels for semi-structured data. In *International Conference on Machine Learning (ICML)*, pages 291–298, 2002.

H. Kashima, K. Tsuda, and A. Inokuchi. Kernels for graphs. In *Kernels and Bioinformatics*, pages 155–170. MIT press, 2004.

J. Kelly. The cutting-plane method for solving convex programs. *Journal of the Society of Industrial Applied Mathematics*, 8:703–712, 1960.

J. Kivinen, A. Smola, and R. Williamson. Online learning with kernels. In T. G. Diettrich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Inf. Proc. Systems (NIPS 01)*, pages 785–792, 2001.

A. Kolmogorov. Stationary sequences in hilbert spaces. *Moscow University Mathematics*, 2:1–40, 1941.

P. Laskov. Feasible direction decomposition algorithms for training support vector machines. *Machine Learning*, 46:315–349, 2002.

P. Laskov, C. Gehl, S. Krüger, and K. R. Müller. Incremental support vector learning: Analysis, implementation and applications. *Journal of Machine Learning Research*, 7:1909–1936, Sept. 2006.

Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Müller, E. Säckinger, P. Simard, and V. Vapnik. Comparison of learning algorithms for handwritten digit recognition. In F. Fogelman-Soulié and P. Gallinari, editors, *Proceedings ICANN'95 — International Conference on Artificial Neural Networks*, volume II, pages 53–60, Nanterre, France, 1995. EC2.

C. Leslie and R. Kuang. Fast string kernels using inexact matching for protein sequences. *Journal of Machine Learning Research*, 5:1435–1455, 2004.

C. Leslie, E. Eskin, and W. Noble. The spectrum kernel: A string kernel for SVM protein classification. In *Proc. Pacific Symp. Biocomputing*, pages 564–575, 2002.

C. Leslie, E. Eskin, A. Cohen, J. Weston, and W. Noble. Mismatch string kernel for discriminative protein classification. *Bioinformatics*, 1(1):1–10, 2003.

C.-J. Lin. On the convergence of the decomposition method for support vector machines. *IEEE Trans. on Neural Networks*, 12(6):1288–1298, 2001.

H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444, 2002.

D. Luenberger. *Introduction to Linear and Nonlinear Programming*. Addison-Wesley, Reading, MA, 1973.

C. Mallows. Some comments on Cp. *Technometrics*, 15:661–675, 1973.

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.

S. Mika. *Kernel Fisher Discriminants*. PhD thesis, Berlin Institute of Technology, Dec. 2002.

J. Moody and C. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2):281–294, 1989.

V. Morozov. *Methods for Solving Incorrectly Posed Problems*. Springer Verlag, 1984.

A. Moschitti. Efficient convolution kernels for dependency and constituent syntactic trees. In *European Conference on Machine Learning (ECML)*, 2006.

K.-R. Müller, A. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Predicting time series with support vector machines. In W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud, editors, *Artificial Neural Networks – ICANN '97*, volume 1327 of *LNCS*, pages 999–1004, Berlin, 1997. Springer.

K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Neural Networks*, 12(2): 181–201, May 2001.

K.-R. Müller, G. Rätsch, S. Sonnenburg, S. Mika, M. Grimm, and N. Heinrich. Classifying 'drug-likeness' with kernel-based learning methods. *J. Chem. Inf. Model*, 45:249–253, 2005.

K.-R. Müller, M. Tangermann, G. Dornhege, M. Krauledat, G. Curio, and B. Blankertz. Machine learning for real-time single-trial EEG-analysis: From brain-computer interfacing to mental state monitoring. *Journal of neuroscience methods*, 167(1):82–90, 2008. URL `http://dx.doi.org/10.1016/j.jneumeth.2007.09.022`.

M. Nassar, R. State, and O. Festor. Monitoring SIP traffic using support vector machines. In *Proc. of Symposium on Recent Advances in Intrusion Detection*, pages 311–330, 2008.

C. S. Ong and A. Zien. An automated combination of kernels for predicting protein subcellular localization. In *In: Proceedings of the 8th Workshop on Algorithms in Bioinformatics (WABI)*, Lecture Notes in Bioinformatics, pages 186–179. Springer, 2008.

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

E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. In *Proceedings CVPR'97*, 1997b.

R. Perdisci, D. Ariu, P. Fogla, G. Giacinto, and W. Lee. McPAD: A multiple classifier system for accurate payload-based anomaly detection. *Computer Networks*, 2009. in press.

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

L. Ralaivola and F. d'Alché Buc. Incremental support vector machine learning: A local approach. *Lecture Notes in Computer Science*, 2130:322–329, 2001.

G. Rätsch. Ensemble learning methods for classification. Master's thesis, Dep. of Computer Science, University of Potsdam, Apr. 1998. In German.

G. Rätsch, S. Mika, B. Schölkopf, and K.-R. Müller. Constructing boosting algorithms from SVMs: an application to one-class classification. *IEEE PAMI*, 24(9):1184–1199, Sept. 2002.

G. Rätsch, S. Sonnenburg, and B. Schölkopf. RASE: recognition of alternatively spliced exons in c. elegans. *Bioinformatics*, 21:i369–i377, June 2005.

G. Rätsch, S. Sonnenburg, J. Srinivasan, H. Witte, R. Sommer, K.-R. Müller, and B. Schölkopf. Improving the c. elegans genome annotation using machine learning. *PLoS Computational Biology*, 3(2):e20, 2007.

K. Rieck. *Machine Learning for Application-Layer Intrusion Detection*. PhD thesis, Berlin Institute of Technology, July 2009.

K. Rieck, T. Krueger, U. Brefeld, and K.-R. Müller. Approximate tree kernels. *Journal of Machine Learning Research*, 11(Feb):555–580, 2010.

S. Rüping. Incremental learning with support vector machines. Technical Report TR-18, Universität Dortmund, SFB475, 2002.

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

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

B. Schölkopf, P. Simard, A. Smola, and V. Vapnik. Prior knowledge in support vector kernels. In M. Jordan, M. Kearns, and S. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10, pages 640–646, Cambridge, MA, 1998a. MIT Press.

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

B. Schölkopf, S. Mika, C. Burges, P. Knirsch, K.-R. Müller, G. Rätsch, and A. Smola. Input space vs. feature space in kernel-based methods. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 10(5):1000–1017, September 1999.

B. Schölkopf, A. Smola, R. Williamson, and P. Bartlett. New support vector algorithms. *Neural Computation*, 12:1207 – 1245, 2000.

B. Schölkopf, J. Platt, J. Shawe-Taylor, A. Smola, and R. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7):1443–1471, 2001.

J. Shawe-Taylor and N. Cristianini. *Kernel methods for pattern analysis*. Cambridge University Press, 2004.

J. Shawe-Taylor, P. Bartlett, and R. Williamson. Structural risk minimization over data-dependent hierachies. *IEEE Transactions on Information Theory*, 44(5):1926–1940, 1998.

S.-S. Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated subgradient solver for svm. In *ICML*, pages 807–814. ACM Press, 2007.

P. Simard, Y. LeCun, J. Denker, and B. Victorri. Transformation invariance in pattern recognition – tangent distance and tangent propagation. In G. Orr and K.-R. Müller, editors, *Neural Networks: Tricks of the Trade*, volume 1524, pages 239–274. Springer LNCS, 1998.

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

S. Sonnenburg and V. Franc. COFFIN: a computational framework for linear SVMs. In *Proceedings of the 27nd International Machine Learning Conference*, 2010. (accepted).

S. Sonnenburg, G. Rätsch, A. Jagota, and K.-R. Müller. New methods for splice-site recognition. In J. Dorronsoro, editor, *Proc. International conference on artificial Neural Networks – ICANN'02*, pages 329–336. LNCS 2415, Springer Berlin, 2002.

S. Sonnenburg, A. Zien, and G. Rätsch. ARTS: Accurate Recognition of Transcription Starts in Human. *Bioinformatics*, 22(14):e472–480, 2006.

S. Sonnenburg, G. Rätsch, and K. Rieck. Large scale learning with string kernels. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large Scale Kernel Machines*, pages 73–103. MIT Press, Cambridge, MA., 2007a.

S. Sonnenburg, G. Schweikert, P. Philips, J. Behr, and G. Rätsch. Accurate Splice Site Prediction. *BMC Bioinformatics, Special Issue from NIPS workshop on New Problems and Methods in Computational Biology Whistler, Canada, 18 December 2006*, 8:(Suppl. 10):S7, December 2007b.

S. Sonnenburg, A. Zien, P. Philips, and G. Rätsch. POIMs: positional oligomer importance matrices — understanding support vector machine based signal detectors. *Bioinformatics*, 24(13):i6–i14, 2008.

S. Sonnenburg, G. Rätsch, S. Henschel, C. Widmer, J. Behr, A. Zien, F. de Bona, A. Binder, C. Gehl, and V. Franc. The SHOGUN machine learning toolbox. *Journal of Machine Learning Research*, 11:1799–1802, June 2010. URL `http://www.shogun-toolbox.org`.

D. Tax and R. Duin. Uniform object generation for optimizing one-class classifiers. *Journal for Machine Learning Research*, pages 155–173, 2001.

D. Tax and P. Laskov. Online SVM learning: from classification to data description and back. In C. Molina, editor, *Proc. NNSP*, pages 499–508, 2003.

C. H. Teo, Q. Le, A. Smola, and S. Vishwanathan. A scalable modular convex solver for regularized risk minimization. In *KDD'07*, August 2007.

C. H. Teo, S. Vishwanthan, A. J. Smola, and Q. V. Le. Bundle methods for regularized risk minimization. *Journal of Machine Learning Research*, 11 (Jan):311–365, 2010.

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

K. Tsuda, M. Kawanabe, G. Rätsch, S. Sonnenburg, and K.-R. Müller. A new discriminative kernel from probabilistic models. *Neural Computation*, 14: 2397–2414, 2002.

V. Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer, Berlin, 1982.

V. Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998.

V. Vapnik and A. Chervonenkis. The necessary and sufficient conditions for consistency in the empirical risk minimization method. *Pattern Recognition and Image Analysis*, 1(3):283–305, 1991.

J.-P. Vert. A tree kernel to analyze phylogenetic profiles. *Bioinformatics*, 18: S276–S284, 2002.

J.-P. Vert, H. Saigo, and T. Akutsu. *Kernel methods in Computational Biology*, chapter Local alignment kernels for biological sequences, pages 131–154. MIT Press, 2004.

S. Vishwanathan and A. Smola. Fast kernels for string and tree matching. In K. Tsuda, B. Schölkopf, and J. Vert, editors, *Kernels and Bioinformatics*, pages 113–130. MIT Press, 2004.

S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11(Apr), 2010.

G. Wahba. Spline bases, regularization, and generalized cross-validation for solving approximation problems with large quantities of noisy data. In *Proceedings of the International Conference on Approximation theory*. Academic Press, Austin, Texas, 1980.

S. Wahl, K. Rieck, P. Laskov, P. Domschitz, and K.-R. Müller. Securing IMS against novel threats. *Bell Labs Technical Journal*, 14(1):243–257, 2009.

M. K. Warmuth, J. Liao, G. Rätsch, M. M., S. Putta, and C. Lemmem. Support Vector Machines for active learning in the drug discovery process. *Journal of Chemical Information Sciences*, 43(2):667–673, 2003.

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

J. Weston, A. Gammerman, M. Stitson, V. Vapnik, V. Vovk, and C. Watkins. Support vector density estimation. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 293–305. MIT Press, Cambridge, MA, 1999.

R. Williamson, A. Smola, and B. Schölkopf. Generalization performance of regularization networks and support vector machines via entropy numbers of compact operators. NeuroCOLT Technical Report NC-TR-98-019, Royal Holloway College, University of London, UK, 1998.

J. Yu, S. Vishwanathan, S. Gunter, and N. N. Schraudolph. A quasi-newton approach to nonsmooth convex optimization problems in machine learning. *JMLR*, 11:1145–1200, Mar 2010.

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

G. Zoutendijk. *Methods of feasible directions*. Elsevier, 1960.