

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/226743605>

Support Vector Machines – An Introduction

Chapter · May 2005

DOI: 10.1007/10984697_1

CITATIONS

297

READS

20,546

1 author:



Vojislav Kecman

Virginia Commonwealth University

129 PUBLICATIONS 4,654 CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:



Fast algorithms for nonlinear support vector machines and alike models [View project](#)



Fuzzy neural networks [View project](#)

Support Vector Machines – An Introduction

V. Kecman

The University of Auckland, School of Engineering,
Auckland, New Zealand

This is a book about learning from empirical data (i.e., examples, samples, measurements, records, patterns or observations) by applying support vector machines (SVMs) a.k.a. kernel machines. The basic aim of this introduction¹ is to give, as far as possible, a condensed (but systematic) presentation of a novel learning paradigm embodied in SVMs. Our focus will be on the constructive learning algorithms for both the classification (pattern recognition) and regression (function approximation) problems. Consequently, we will not go into all the subtleties and details of the statistical learning theory (SLT) and structural risk minimization (SRM) which are theoretical foundations for the learning algorithms presented below. Instead, a quadratic programming based learning leading to parsimonious SVMs will be presented in a gentle way – starting with linear separable problems, through the classification tasks having overlapped classes but still a linear separation boundary, beyond the linearity assumptions to the nonlinear separation boundary, and finally to the linear and nonlinear regression problems. The adjective “parsimonious” denotes an SVM with a small number of support vectors. The scarcity of the model results from a sophisticated learning that matches the model capacity to the data complexity ensuring a good performance on the future, previously unseen, data.

Same as the neural networks or similarly to them, SVMs possess the well-known ability of being universal approximators of any multivariate function to any desired degree of accuracy. Consequently, they are of particular interest for modeling the unknown, or partially known, highly nonlinear, complex systems, plants or processes. Also, at the very beginning, and just to be sure what the whole book is about, we should state clearly when there is no need for an application of SVMs’ model-building techniques. In short, whenever there exists a good and reliable analytical closed-form model (or it is possible to

¹This introduction strictly follows the School of Engineering of The University of Auckland Report 616. The right to use the material from this report is received with gratitude.

devise one) there is no need to resort to learning from empirical data by SVMs (or by any other type of a learning machine).

1 Basics of Learning from Data

SVMs have been developed in the reverse order to the development of neural networks (NNs). SVMs evolved from the sound theory to the implementation and experiments, while the NNs followed more heuristic path, from applications and extensive experimentation to the theory. It is interesting to note that the very strong theoretical background of SVMs did not make them widely appreciated at the beginning. The publication of the first papers by Vapnik, Chervonenkis and co-workers in 1964/65 went largely unnoticed till 1992. This was due to a widespread belief in the statistical and/or machine learning community that, despite being theoretically appealing, SVMs are neither suitable nor relevant for practical applications. They were taken seriously only when excellent results on practical learning benchmarks were achieved in digit recognition, computer vision and text categorization. Today, SVMs show better results than (or comparable outcomes to) NNs and other statistical models, on the most popular benchmark problems (see some results and comparisons in [3, 4, 12, 20]).

The learning problem setting for SVMs is as follows: there is some unknown and nonlinear dependency (mapping, function) $y = f(\mathbf{x})$ between some high-dimensional input vector $\mathbf{x}$ and scalar output y (or the vector output $\mathbf{y}$ as in the case of multiclass SVMs). There is no information about the underlying joint probability functions. Thus, one must perform a *distribution-free learning*. The only information available is a training data set $D = \{(\mathbf{x}_i, y_i) \in X \times Y\}$, $i = 1, l$, where l stands for the number of the training data pairs and is therefore equal to the size of the training data set D . Often, y_i is denoted as d_i , where d stands for a desired (target) value. Hence, SVMs belong to the supervised learning techniques.

Note that this problem is similar to the classic statistical inference. However, there are several very important differences between the approaches and assumptions in training SVMs and the ones in classic statistics and/or NNs modeling. Classic statistical inference is based on the following three fundamental assumptions:

1. Data can be modeled by a set of linear in parameter functions; this is a foundation of a parametric paradigm in learning from experimental data.
2. In the most of real-life problems, a stochastic component of data is the normal probability distribution law, that is, the underlying joint probability distribution is a Gaussian distribution.
3. Because of the second assumption, the induction paradigm for parameter estimation is the maximum likelihood method, which is reduced to the minimization of the sum-of-errors-squares cost function in most engineering applications.

All three assumptions on which the classic statistical paradigm relied turned out to be inappropriate for many contemporary real-life problems [35] because of the following facts:

1. Modern problems are high-dimensional, and if the underlying mapping is not very smooth the linear paradigm needs an exponentially increasing number of terms with an increasing dimensionality of the input space X (an increasing number of independent variables). This is known as “the curse of dimensionality”.
2. The underlying real-life data generation laws may typically be very far from the normal distribution and a model-builder must consider this difference in order to construct an effective learning algorithm.
3. From the first two points it follows that the maximum likelihood estimator (and consequently the sum-of-error-squares cost function) should be replaced by a new induction paradigm that is uniformly better, in order to model non-Gaussian distributions.

In addition to the three basic objectives above, the novel SVMs’ problem setting and inductive principle have been developed for standard contemporary data sets which are typically high-dimensional and sparse (meaning, the data sets contain small number of the training data pairs).

SVMs are the so-called “nonparametric” models. “Nonparametric” does not mean that the SVMs’ models do not have parameters at all. On the contrary, their “learning” (selection, identification, estimation, training or tuning) is the crucial issue here. However, unlike in classic statistical inference, the parameters are not predefined and their number depends on the training data used. In other words, parameters that define the capacity of the model are data-driven in such a way as to match the model capacity to data complexity. This is a basic paradigm of the structural risk minimization (SRM) introduced by Vapnik and Chervonenkis and their coworkers that led to the new learning algorithm. Namely, there are two basic constructive approaches possible in designing a model that will have a good generalization property [33, 35]:

1. choose an appropriate structure of the model (order of polynomials, number of HL neurons, number of rules in the fuzzy logic model) and, keeping the estimation error (a.k.a. confidence interval, a.k.a. variance of the model) fixed in this way, minimize the training error (i.e., empirical risk), or
2. keep the value of the training error (a.k.a. an approximation error, a.k.a. an empirical risk) fixed (equal to zero or equal to some acceptable level), and minimize the confidence interval.

Classic NNs implement the first approach (or some of its sophisticated variants) and SVMs implement the second strategy. In both cases the resulting model should resolve the trade-off between under-fitting and over-fitting the training data. The final model structure (its order) should ideally *match the learning machines capacity with training data complexity*. This important difference in two learning approaches comes from the minimization of different

Table 1. Basic Models and Their Error (Risk) Functionals

Multilayer Perceptron (NN)	Regularization Network (Radial Basis Functions Network)	Support Vector Machine
$R = \sum_{i=1}^l \underbrace{(d_i - f(\mathbf{x}_i, \mathbf{w}))^2}_{\text{Closeness to data}}$	$R = \sum_{i=1}^l \underbrace{(d_i - f(\mathbf{x}_i, \mathbf{w}))^2}_{\text{Closeness to data}} + \lambda \underbrace{\ \mathbf{P}f\ ^2}_{\text{Smoothness}}$	$R = \sum_{i=1}^l \underbrace{L\varepsilon}_{\text{Closeness to data}} + \underbrace{\Omega(l, h)}_{\text{capacity of a machine}}$

Closeness to data = training error, a.k.a. empirical risk

cost (error, loss) functionals. Table 1 tabulates the basic risk functionals applied in developing the three contemporary statistical models.

d_i stands for desired values, $\mathbf{w}$ is the weight vector subject to training, λ is a regularization parameter, $\mathbf{P}$ is a smoothness operator, L_ε is a loss function of SVMs', h is a VC dimension and Ω is a function bounding the capacity of the learning machine. In classification problems L_ε is typically 0–1 loss function, and in regression problems L_ε is the so-called Vapnik's ε -insensitivity loss (error) function

$$L_\varepsilon = |y - f(\mathbf{x}, \mathbf{w})|_\varepsilon = \begin{cases} 0, & \text{if } |y - f(\mathbf{x}, \mathbf{w})| \leq \varepsilon \\ |y - f(\mathbf{x}, \mathbf{w})| - \varepsilon, & \text{otherwise.} \end{cases} \quad (1)$$

where ε is a radius of a tube within which the regression function must lie, after the successful learning. (Note that for $\varepsilon = 0$, the interpolation of training data will be performed). It is interesting to note that [11] has shown that under some constraints the SV machine can also be derived from the framework of regularization theory rather than SLT and SRM. Thus, *unlike the classic adaptation algorithms (that work in the L_2 norm), SV machines represent novel learning techniques which perform SRM*. In this way, the SV machine creates a model with minimized VC dimension and when the VC dimension of the model is low, the expected probability of error is low as well. This means good performance on previously unseen data, i.e. a good generalization. This property is of particular interest because the model that generalizes well is a good model and not the model that performs well on training data pairs. Too good a performance on training data is also known as an extremely undesirable overfitting.

As it will be shown below, in the “simplest” pattern recognition tasks, support vector machines use a linear separating hyperplane to create a *classifier with a maximal margin*. In order to do that, the learning problem for the SV machine will be cast as a *constrained nonlinear optimization* problem. In this setting the cost function will be quadratic and the constraints linear (i.e., one will have to solve a classic *quadratic programming problem*).

In cases when given classes cannot be linearly separated in the original input space, the SV machine first (non-linearly) transforms the original input space into a higher dimensional *feature space*. This transformation can be achieved by using various nonlinear mappings; polynomial, sigmoidal as in multilayer perceptrons, RBF mappings having as the basis functions radially

symmetric functions such as Gaussians, or multiquadrics or different spline functions. After this nonlinear transformation step, the task of an SV machine in finding the linear optimal separating hyperplane in this feature space is “relatively trivial”. Namely, the optimization problem to solve in a feature space will be of the same kind as the calculation of a maximal margin separating hyperplane in the original input space for linearly separable classes. How, after the specific nonlinear transformation, nonlinearly separable problems in input space can become linearly separable problems in a feature space will be shown later.

In a probabilistic setting, there are three basic components in all learning-from-data tasks: A *generator* of random inputs $\mathbf{x}$, a *system* whose *training responses* $y(d)$ are used for training the learning machine, and a *learning machine* which, by using inputs $\mathbf{x}_i$ and system’s responses y_i , should learn (estimate, model) the unknown dependency between these two sets of variables defined by the weight vector $\mathbf{w}$ (Fig. 1). This figure shows the most common learning setting that some readers may have already seen in various

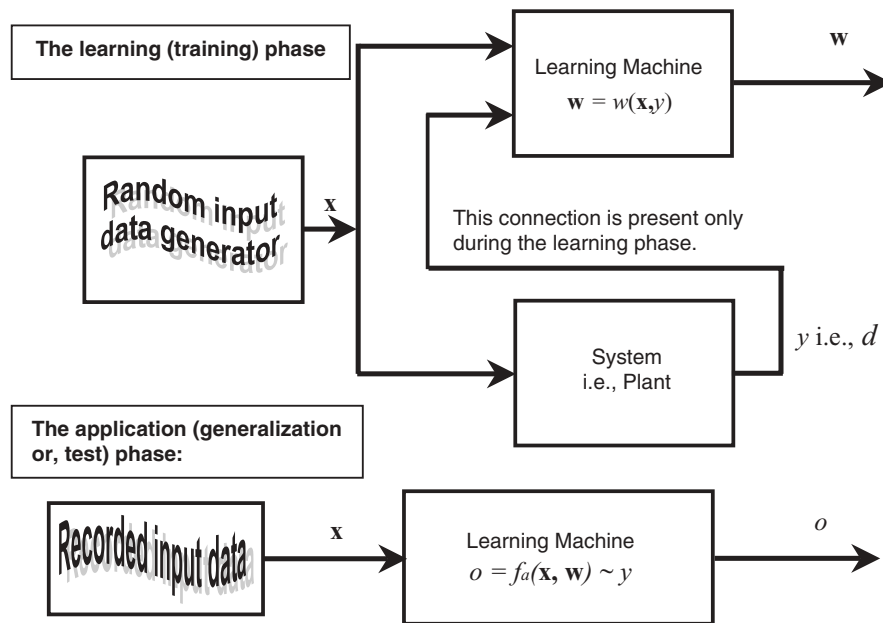


Fig. 1. A model of a learning machine (*top*) $\mathbf{w} = w(\mathbf{x}, y)$ that during the *training phase* (by observing inputs $\mathbf{x}_i$ to, and outputs y_i from, the system) estimates (learns, adjusts, trains, tunes) its parameters (weights) $\mathbf{w}$, and in this way learns mapping $y = f(\mathbf{x}, \mathbf{w})$ performed by the system. The use of $f_a(\mathbf{x}, \mathbf{w}) \sim y$ denotes that we will rarely try to *interpolate* training data pairs. We would rather seek an *approximating function* that can generalize well. After the training, at the *generalization* or *test phase*, the output from a machine $o = f_a(\mathbf{x}, \mathbf{w})$ is expected to be “a good” estimate of a system’s true response y

other fields – notably in statistics, NNs, control system identification and/or signal processing. During the (successful) training phase a learning machine should be able to find the relationship between an input space X and an output space Y , by using data D in regression tasks (or to find a function that separates data within the input space, in classification ones). The result of a learning process is an “approximating function” $f_a(\mathbf{x}, \mathbf{w})$, which in statistical literature is also known as, a *hypothesis* $f_a(\mathbf{x}, \mathbf{w})$. This function approximates the underlying (or true) dependency between the input and output in the case of regression, and the decision boundary, i.e., separation function, in a classification. The chosen hypothesis $f_a(\mathbf{x}, \mathbf{w})$ belongs to a *hypothesis space of functions* $H(f_a \in H)$, and it is a function that minimizes some *risk functional* $R(\mathbf{w})$.

It may be practical to remind the reader that under the general name “approximating function” we understand any mathematical structure that maps inputs $\mathbf{x}$ into outputs y . Hence, an “approximating function” may be: a multilayer perceptron NN, RBF network, SV machine, fuzzy model, Fourier truncated series or polynomial approximating function. Here we discuss SVMs. A set of parameters $\mathbf{w}$ is the very subject of learning and generally these parameters are called *weights*. These parameters may have different geometrical and/or physical meanings. Depending upon the hypothesis space of functions H we are working with, the parameters $\mathbf{w}$ are usually:

- the hidden and the output layer weights in multilayer perceptrons,
- the rules and the parameters (for the positions and shapes) of fuzzy subsets,
- the coefficients of a polynomial or Fourier series,
- the centers and (co)variances of Gaussian basis functions as well as the output layer weights of this RBF network,
- the support vector weights in SVMs.

There is another important class of functions in learning from examples tasks. A learning machine tries to capture an unknown *target function* $f_0(\mathbf{x})$ that is believed to belong to some target space T , or to a class T , that is also called a *concept class*. Note that we rarely know the target space T and that our learning machine generally does not belong to the same class of functions as an unknown target function $f_0(\mathbf{x})$. Typical examples of target spaces are continuous functions with s continuous derivatives in n variables; Sobolev spaces (comprising square integrable functions in n variables with s square integrable derivatives), band-limited functions, functions with integrable Fourier transforms, Boolean functions, etc. In the following, we will assume that the target space T is a space of differentiable functions. The basic problem we are facing stems from the fact that we know very little about the possible underlying function between the input and the output variables. All we have at our disposal is a training data set of labeled examples drawn by independently sampling an $(X \times Y)$ space according to some unknown probability distribution.

The learning-from-data problem is ill-posed. (This will be shown on Figs. 2 and 3 for a regression and classification examples respectively). The basic source of the ill-posedness of the problem is due to the infinite number of possible solutions to the learning problem. At this point, just for the sake of illustration, it is useful to remember that all functions that interpolate data points will result in a zero value for training error (empirical risk) as shown (in the case of regression) in Fig. 2. The figure shows a simple example of three-out-of-infinitely-many different interpolating functions of training data pairs sampled from a noiseless function $y = \sin(x)$.

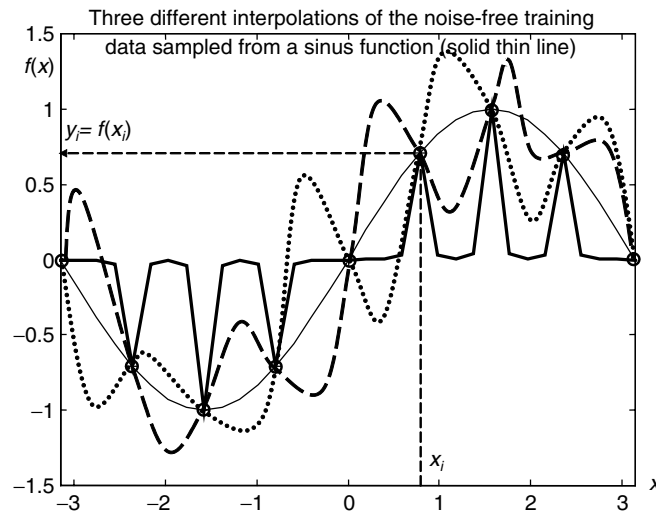


Fig. 2. Three-out-of-infinitely-many interpolating functions resulting in a training error equal to 0. However, a *thick solid*, *dashed* and *dotted lines* are bad models of a true function $y = \sin(x)$ (*thin dashed line*)

In Fig. 2, each interpolant results in a training error equal to zero, but at the same time, each one is a very bad model of the true underlying dependency between x and y , because all three functions perform very poorly outside the training inputs. In other words, none of these three particular interpolants can generalize well. However, not only interpolating functions can mislead. There are many other approximating functions (learning machines) that will minimize the empirical risk (approximation or training error) but not necessarily the generalization error (true, expected or guaranteed risk). This follows from the fact that a learning machine is trained by using some particular sample of the true underlying function and consequently it always produces biased approximating functions. These approximants depend necessarily on the specific training data pairs (i.e., the training sample) used.

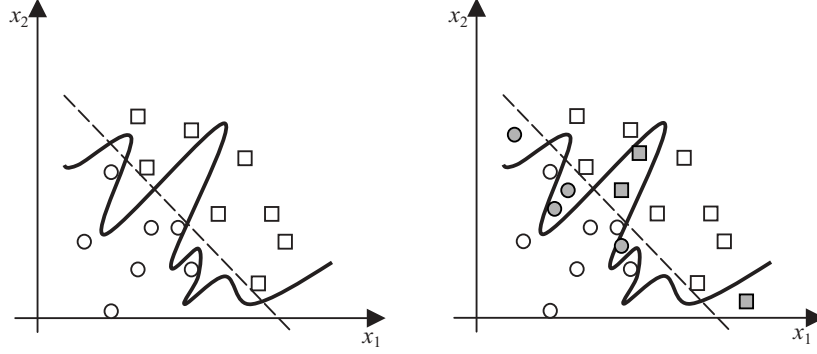


Fig. 3. Overfitting in the case of linearly separable classification problem. *Left:* The perfect classification of the training data (*empty circles and squares*) by both low order linear model (*dashed line*) and high order nonlinear one (*solid wiggly curve*). *Right:* Wrong classification of all the test data shown (*filled circles and squares*) by a high capacity model, but correct one by the simple linear separation boundary

Figure 3 shows an extremely simple classification example where the classes (represented by the empty training circles and squares) are linearly separable. However, in addition to a linear separation (dashed line) the learning was also performed by using a model of a high capacity (say, the one with Gaussian basis functions, or the one created by a high order polynomial, over the 2-dimensional input space) that produced a perfect separation boundary (empirical risk equals zero) too. However, such a model is overfitting the data and it will definitely perform very badly on, during the training unseen, test examples. Filled circles and squares in the right hand graph are all wrongly classified by the nonlinear model. Note that a simple linear separation boundary correctly classifies both the training and the test data.

A solution to this problem proposed in the framework of the SLT is restricting the hypothesis space H of approximating functions to a set smaller than that of the target function T while simultaneously controlling the flexibility (complexity) of these approximating functions. This is ensured by an introduction of a novel induction principle of the SRM and its algorithmic realization through the SV machine. The Structural Risk Minimization principle [31] tries to minimize an expected risk (the cost function) R comprising two terms as given in Table 1 for the SVMs $R = \Omega(l, h) + \sum_{i=1}^l L_{\varepsilon} = \Omega(l, h) + R_{\text{emp}}$ and it is based on the fact that for the classification learning problem with a probability of at least $1 - \eta$ the bound

$$R(\mathbf{w}_n) \leq \Omega\left(\frac{h}{l}, \frac{\ln(\eta)}{l}\right) + R_{\text{emp}}(\mathbf{w}_n), \quad (2a)$$

holds. The first term on the right hand side is named a VC confidence (confidence term or confidence interval) that is defined as

$$\Omega\left(\frac{h}{l}, \frac{\ln(\eta)}{l}\right) = \sqrt{\frac{h \left[\ln\left(\frac{2l}{h}\right) + 1 \right] - \ln\left(\frac{\eta}{4}\right)}{l}}. \quad (2b)$$

The parameter h is called the VC (Vapnik-Chervonenkis) dimension of a set of functions. It describes the capacity of a set of functions implemented in a learning machine. For binary classification h is the maximal number of points which can be separated (shattered) into two classes in all possible 2^h ways by using the functions of the learning machine.

An SV (learning) machine can be thought of as

- a set of functions implemented in an SVM,
- an induction principle and,
- an algorithmic procedure for implementing the induction principle on the given set of functions.

The notation for risks given above by using $R(\mathbf{w}_n)$ denotes that an expected risk is calculated over a set of functions $f_{an}(\mathbf{x}, \mathbf{w}_n)$ of increasing complexity. Different bounds can also be formulated in terms of other concepts such as *growth function* or *annealed VC entropy*. Bounds also differ for regression tasks. More detail can be found in ([33], as well as in [3]). However, the general characteristics of the dependence of the confidence interval on the number of training data l and on the VC dimension h is similar and given in Fig. 4.

Equations (2a) show that when the number of training data increases, i.e., for $l \rightarrow \infty$ (with other parameters fixed), an expected (true) risk $R(\mathbf{w}_n)$ is very close to empirical risk $R_{\text{emp}}(\mathbf{w}_n)$ because $\Omega \rightarrow 0$. On the other hand, when the probability $1 - \eta$ (also called a confidence level which should not be confused with the confidence term Ω) approaches 1, the generalization bound grows large, because in the case when $\eta \rightarrow 0$ (meaning that the confidence level $1 - \eta \rightarrow 1$), the value of $\Omega \rightarrow \infty$. This has an obvious intuitive interpretation [3] in that any learning machine (model, estimates) obtained from a finite number of training data cannot have an arbitrarily high confidence level. There is always a trade-off between the accuracy provided by bounds and the degree of confidence (in these bounds). Figure 4 also shows that the VC confidence interval increases with an increase in a VC dimension h for a fixed number of the training data pairs l .

The SRM is a novel inductive principle for learning from finite training data sets. It proved to be very useful when dealing with *small samples*. The basic idea of the SRM is to choose (from a large number of possibly candidate learning machines), a model of the right capacity to describe the given training data pairs. As mentioned, this can be done by restricting the hypothesis space H of approximating functions and simultaneously by controlling their flexibility (complexity). Thus, learning machines will be those parameterized models that, by increasing the number of parameters (typically called weights w_i here), form a nested structure in the following sense

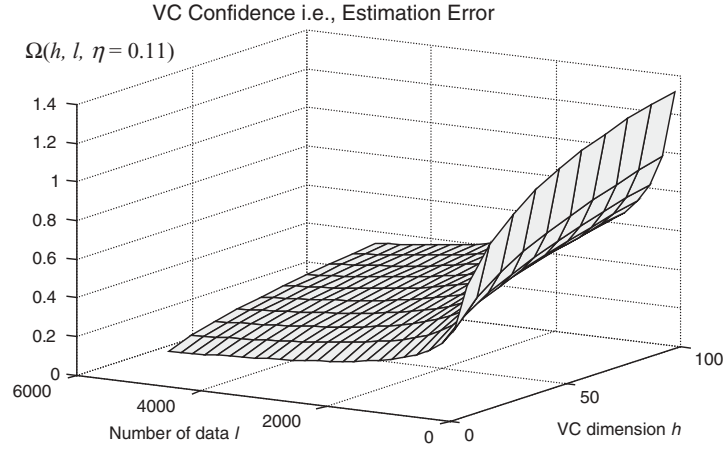


Fig. 4. The dependency of VC confidence $\Omega(h, l, \eta)$ on the number of training data l and the VC dimension h ($h < l$) for a fixed confidence level $1 - \eta = 1 - 0.11 = 0.89$

$$H_1 \subset H_2 \subset H_3 \subset \dots H_{n-1} \subset H_n \subset \dots \subset H \quad (3)$$

In such a nested set of functions, every function always contains a previous, less complex, function. Typically, H_n may be: a set of polynomials in one variable of degree n ; fuzzy logic model having n rules; multilayer perceptrons, or RBF network having n HL neurons, SVM structured over n support vectors. The goal of learning is one of a *subset selection* that matches training data complexity with approximating model capacity. In other words, a learning algorithm chooses an optimal polynomial degree or, an optimal number of HL neurons or, an optimal number of FL model rules, for a polynomial model or NN or FL model respectively. For learning machines linear in parameters, this complexity (expressed by the VC dimension) is given by the number of weights, i.e., by the number of “free parameters”. For approximating models nonlinear in parameters, the calculation of the VC dimension is often not an easy task. Nevertheless, even for these networks, by using simulation experiments, one can find a model of appropriate complexity.

2 Support Vector Machines in Classification and Regression

Below, we focus on the algorithm for implementing the SRM induction principle on the given set of functions. It implements the strategy mentioned previously – it keeps the training error fixed and minimizes the confidence interval. We first consider a “simple” example of linear decision rules (i.e., the separating functions will be hyperplanes) for binary classification (dichotomization) of linearly separable data. In such a problem, we are able to perfectly classify