МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО АТОМНОЙ ЭНЕРГИИ

РОССИЙСКАЯ АКАДЕМИЯ НАУК

РОССИЙСКАЯ АССОЦИАЦИЯ НЕЙРОИНФОРМАТИКИ

МОСКОВСКИЙ ИНЖЕНЕРНО-ФИЗИЧЕСКИЙ ИНСТИТУТ
(ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ)

ИНСТИТУТ ОПТИКО-НЕЙРОННЫХ ТЕХНОЛОГИЙ РАН

НАУЧНАЯ СЕССИЯ МИФИ–2007

НЕЙРОИНФОРМАТИКА–2007

**IX ВСЕРОССИЙСКАЯ
НАУЧНО-ТЕХНИЧЕСКАЯ
КОНФЕРЕНЦИЯ**

ЛЕКЦИИ
ПО НЕЙРОИНФОРМАТИКЕ

Часть 1

По материалам Школы-семинара
«Современные проблемы нейроинформатики»

Москва 2007

УДК 001(06)+004.032.26(06) Нейронные сети
ББК 72я5+32.818я5
М82

НАУЧНАЯ СЕССИЯ МИФИ–2007. IX ВСЕРОССИЙСКАЯ НАУЧНО-ТЕХНИЧЕСКАЯ КОНФЕРЕНЦИЯ «НЕЙРОИНФОРМАТИКА–2007»: ЛЕКЦИИ ПО НЕЙРОИНФОРМАТИКЕ. Часть 1. – М.: МИФИ, 2007. – 178 с.

В книге публикуются тексты лекций, прочитанных на Школе-семинаре «Современные проблемы нейроинформатики», проходившей 24–26 января 2007 года в МИФИ в рамках IX Всероссийской конференции «Нейроинформатика–2007».

Материалы лекций связаны с рядом проблем, актуальных для современного этапа развития нейроинформатики, включая ее взаимодействие с другими научно-техническими областями.

Ответственный редактор
Ю. В. Тюменцев, кандидат технических наук

ISBN 5–7262–0708–4

Содержание

УДК 001(06)+004.032.26(06) Нейронные сети

В. КЕЦМАН
Университет Окленда,
Новая Зеландия
E-mail: v.kecman@auckland.ac.nz
http://www.support-vector.ws

НОВЫЙ SVM-АЛГОРИТМ ДЛЯ СВЕРХБОЛЬШИХ НАБОРОВ ДАННЫХ

Аннотация

В лекции рассматривается новейший алгоритм обучения для машин опорных векторов (SVM), известных также под наименованием «машины ядерных функций» (kernel machines), при работе со сверхбольшими наборами данных (например, содержащими несколько миллионов обучающих пар). Вначале сравниваются нейронные сети и машины опорных векторов с точки зрения решения с их помощью задач классификации (распознавания образов) и регрессии (аппроксимации функций), после чего вводится алгоритм обучения для машин опорных векторов (SVM-алгоритм). Показано, что выбор значений весов в SVM-алгоритме приводит к задаче квадратичного программирования с ограничениями. В отличие от классических задач квадратичного программирования, матрицы Гессе, с которыми приходится иметь дело при реализации SVM-алгоритмов, обычно очень плотно заполнены ненулевыми элементами. Кроме, того размерность для таких матриц меняется с изменением числа обучающих пар данных. Это приводит к появлению матриц Гессе сверхбольших размерностей, что затрудняет решение задачи обучения. Для решения такого рода сверхбольших задач предлагается новый итерационный алгоритм, получивший наименование ISDA (Iterative Single Data Algorithm), основывающийся на последовательном использовании обучающих пар данных из имеющегося обучающего набора. Дается доказательство сходимости этого алгоритма, которое базируется на сходимости итеративного метода Гаусса–Зейделя для решения систем линейных уравнений. Приводятся результаты проверки работоспособности предлагаемого алгоритма на тестовых наборах данных. Алгоритм ISDA показал себя наиболее быстрым среди существующих в настоящее время при точном решении задач классификации и регрессии для сверхбольших наборов обучающих данных.

УДК 001(06)+004.032.26(06) Нейронные сети

VOJISLAV KECMAN
School of Engineering,
The University of Auckland
Private Bag 92019, Auckland, New Zealand
E-mail: v.kecman@auckland.ac.nz
http://www.support-vector.ws

NEW SUPPORT VECTOR MACHINES ALGORITHM FOR HUGE DATA SETS

Abstract

The seminar presents the newest learning algorithm for support vector machines (SVMs), a.k.a., kernel machines, when faced with huge data sets (say, millions of training data pairs). It starts with comparisons between the so-called neural networks (NNs) and SVMs for solving classification (pattern recognition) and regression (function approximation) tasks and then it introduces the SVMs learning algorithm. It shows how comes that learning the SVMs' weights means solving the quadratic programming (QP) problem with constraints. Unlike in classic QP problems, Hessian matrices involved in SVMs are usually extremely dense. In addition, their sizes scale with the number of training data pairs. This leads to huge Hessian matrices, and to intractable solutions. The new iterative single data algorithm (ISDA) is proposed for solving such huge problems. The proof of its convergence, based on the convergence of Gauss-Seidel iterative method for solving linear systems of equations, is given. Some comparisons on benchmark data sets are presented. ISDA implementation seems to be the quickest software for exact solving classification and regression tasks from huge training data sets problems at the moment.

Introduction

Today, we are surrounded by an ocean of all kind of experimental data (i. e., examples, samples, measurements, records, patterns, pictures, tunes, observations etc) produced by various sensors, cameras, microphones, pieces of software and/or other human made devices. The amount of data produced is enormous and ever increasing. The first obvious consequence of such a fact is — humans can't handle such massive quantity of data which are usually appearing in the numeric shape as the huge (rectangular or square) matrices. Typically, the number of their rows tells about the number of data pairs collected, and the number of columns represents the dimensionality of data. Thus, faced with the Giga- and Terabyte sized data files one has to develop new approaches, algorithms

and procedures. Few techniques for coping with huge data size problems are presented here. This explains the appearance of a wording "*huge data sets*" in the title of the seminar.

The direct consequence is that (instead of attempting to dive into the sea of hundreds of thousands or millions of high-dimensional data pairs) we have to develop other "machines" i. e., "devices" for analyzing, recognition in, and/or learning from, such huge data sets. The so-called "learning machine" is predominantly a piece of software that implements both the learning algorithm and the function (network, model) which parameters has to be determined by the learning part of the software. Next, it is worth of clarifying the fact that many authors tend to label similar (or even same) models, approaches and algorithms by different names. One is just destine to cope with concepts of data mining, knowledge discovery, neural networks, Bayesian networks, machine learning, pattern recognition, classification, regression, statistical learning, decision trees, decision making etc. All of them usually have a lot in common, and often they use same sets of techniques for adjusting, tuning, training or learning the parameters defining the models. The common object for all of them is a training data set. All the various approaches mentioned start with a set of data pairs $(\mathbf{x}_i, y_i)$ where $\mathbf{x}_i$ represents the input variables (causes, observations, records) and y_i denote the measured outputs (responses, labels).

This is a seminar on (machine) learning from empirical data by applying support vector machines (SVMs). We first show some similarities and differences between the SVMs and NNs and then we introduce the QP based learning for SVMs. The SVMs fall into the big group of supervised learning algorithms. This means that for each input vector (measurements) $\mathbf{x}_i$ there is always a known output value (label) y_i. Here, we do not present neither the semi-supervised learning algorithms (when only smaller part of inputs have corresponding labeled outputs), nor unsupervised methods (such as PCA or ICA, when there are no labeled desired outputs at all). The basic aim of this chapter 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 part* of the SVMs' 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 the theoretical foundations for learning algorithms presented below. The approach here seems more appropriate for the application oriented readers. The theoretically minded and interested reader may find an extensive presentation of both the SLT and

SRM in (*Vapnik* and *Chervonenkis*, 1989; *Vapnik*, 1995, 1998; *Cherkassky* and *Mulier*, 1998; *Cristianini* and *Shawe-Taylor*, 2001; *Kecman*, 2001; *Schölkopf* and *Smola* 2002). Instead of diving into such statistically colored theory, 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. Here, the adjective "parsimonious" denotes a SVM with a small number of support vectors ("hidden layer neurons"). The scarcity of the model results from a sophisticated, QP based, learning that matches the model capacity to data complexity ensuring a good generalization, i. e., a good performance of SVM on the future, previously, during the training 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 chapter is about, we should state clearly when there is no need for an application of SVMs' model-building techniques.

In short, whenever there exists an analytical closed-form model or, there is a knowledge making it possible to devise one, there is no need to resort to the learning from empirical data by SVMs (or by any other type of a learning machine).

Basics of learning from data by NNs and SVMs

SVMs have been developed in the reverse order to the development of neural networks. 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* and *Chervonenkis* (Vapnik and Chervonenkis, 1964, 1968) 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 numeral 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.

The learning problem setting for SVMs is as follows: there is some unknown nonlinear dependency (mapping, function) $y = f(\mathbf{x})$ between some high-dimensional input vector $\mathbf{x}$ and the 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 here. Thus, one must perform a *distribution-free learning*. The only information available is a training data set $D = \{(\mathbf{x}_i, y_i) \in X \times Y\}$, $i = 1, \ldots, l$, 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 (i.e., t_i), where $d(t)$ 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 (*Vapnik*, 1998) 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:

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

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

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

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

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

In Table 1, 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 SVMs' loss function, 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 (see more about it in the section "Regression by Support Vector Machines")

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

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 (*Girosi*, 1997) 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 present in NNs (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; it's very likely that they will have 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. Very good performance on training data usually leads to 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, sigmoid 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 a 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 an 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 supervised learning from data tasks: a *generator* of random inputs $\mathbf{x}$, a *system* whose *training responses* y (i.e., 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 (namely, $\mathbf{x}_i$ and y_i) defined by the weight vector $\mathbf{w}$(Fig. 1).

The figure shows the most common learning setting that some readers may have already seen in various other fields — notably in statistics, NNs, control system identification and/or in 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,

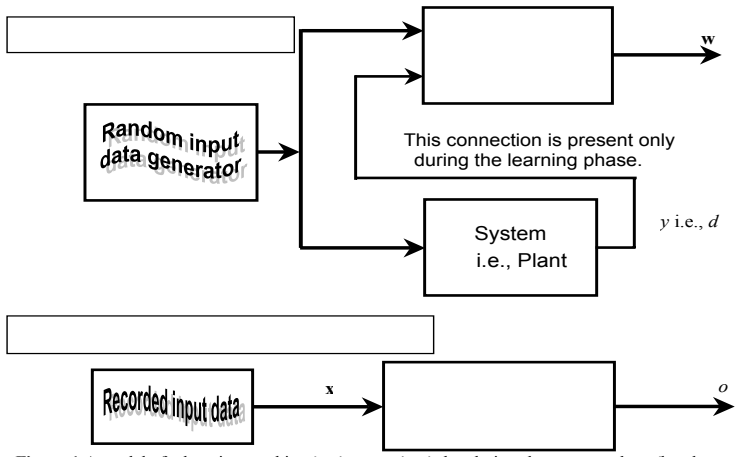

Figure 1. A model of a learning machine *(top)* $\mathbf{w} = w(\mathbf{x}, y)$ that during *the training phase* (by observing inputs $\mathbf{x}_i$ to, and outputs y_i from, the system) estimates (learns, adjusts, trains, tunes) its parameters (weights) $\mathbf{w}$, and in this way learns mapping $y = f(\mathbf{x}, \mathbf{w})$ 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.

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* $f_o(\mathbf{x})$ 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 $f_o(\mathbf{x})$. 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 $a(X \times Y)$ space according to some unknown probability distribution.

Now, we stop with general issues and concepts and we present the learning algorithms for SVMs, or we show how, from a training data, they can learn the unknown dependency.

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 data pairs, meaning that an empirical risk can be set to zero. It is the easiest classification problem and yet an excellent introduction of all relevant and important ideas underlying the SLT, SRM and SVM.

Our presentation will gradually increase in complexity. It will begin with a *Linear Maximal Margin Classifier for Linearly Separable Data* where there is no sample overlapping. Afterwards, we will allow some degree of overlapping of training data pairs. However, we will still try to separate classes by using linear hyperplanes. This will lead to the *Linear Soft Margin Classifier for Overlapping Classes*. In problems when linear decision hyperplanes are no longer feasible, the mapping of an input space into the so-called feature space (that "corresponds" to the HL in NN models) will take place resulting in the *Nonlinear Classifier*. Finally, in the subsection on *Regression by SV Machines* we introduce same approaches and techniques for solving regression (i.e., function approximation) problems.

Linear Maximal Margin Classifier for Linearly Separable Data

Consider the problem of binary classification or dichotomization. Training data are given as

$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_l, y_l), \ \mathbf{x} \in \Re^n, \ y \in \{+1, -1\}. \tag{2}$$

For reasons of visualization only, we will consider the case of a two-dimensional input space, i.e., $\mathbf{x} \in \Re^2$. Data are linearly separable and there are many different hyperplanes that can perform separation (Fig. 2)[1]. How to find "the best" one? The difficult part is that all we have at our disposal are sparse training data. There are many functions that can solve given pattern recognition (or functional approximation) tasks. In such a problem setting, the SLT (developed in the 1960s by *Vapnik* and *Chervonenkis*) shows that it is crucial to restrict the class of functions implemented by a learning machine to one with a complexity that is suitable for the amount of available training data.

In the case of a classification of linearly separable data, this idea is transformed into the following approach — among all the hyperplanes that minimize the training error (i.e., empirical risk) find the one with the largest margin. This is an intuitively acceptable approach. Just by looking at Fig. 2 we will find that the dashed separation line shown in the *right graph* seems to promise *probably* good classification while facing previously unseen data (meaning, in the generalization, i.e. test, phase). Or, at least, it seems to probably be better in

[1]Actually, for $\mathbf{x} \in \Re^2$, the separation is performed by "planes" $w_1 x_1 + w_2 x_2 + b = o$. In other words, the decision boundary, i.e., the separation line in input space is defined by the equation $w_1 x_1 + w_2 x_2 + b = 0$.

generalization than the dashed decision boundary having smaller margin shown in the left graph. This can also be expressed as that a classifier with smaller margin will have higher expected risk.

By using given training examples, during the learning stage, our machine finds parameters $\mathbf{w} = [w_1, w_2, \ldots, w_n]^T$ and b of a discriminant or decision function $d(\mathbf{x}, \mathbf{w}, b)$ given as

$$d(\mathbf{x}, \mathbf{w}, b) = \mathbf{w}^T\mathbf{x} + b = \sum_{i=1}^{n} w_i x_i + b, \tag{3}$$

where $\mathbf{x}, \mathbf{w} \in \Re^n$, and the scalar b is called *a bias*[2]. After the successful training stage, by using the weights obtained, the learning machine, given previously unseen pattern $\mathbf{x}_p$, produces output o according to an *indicator function* given as

$$i_F = o = \text{sign}(d(\mathbf{x}_p, \mathbf{w}, b)), \tag{4}$$

where o is the standard notation for the *output* from the learning machine. In other words, *the decision rule is*:

if $d(\mathbf{x}_p, \mathbf{w}, b) > 0$, the pattern $\mathbf{x}_p$ belongs to a class 1 (i.e., $o = y_1 = +1$), and

if $d(\mathbf{x}_p, \mathbf{w}, b) < 0$ the pattern $\mathbf{x}_p$ belongs to a class 2 (i.e., $o = y_2 = -1$).

The *indicator function* i_F given by (4) is a step-wise (i.e., a stairs-wise) function (see Figs 3 and 4). At the same time, the decision (or discriminant) function $d(\mathbf{x}_p, \mathbf{w}, b)$ is a hyperplane. Note also that both a decision hyperplane d and the indicator function i_F live in an $n+1$-dimensional space or they lie "over" a training pattern's n-dimensional input space. There is one more mathematical object in classification problems called a *separation boundary* that lives in the same n-dimensional space of input vectors $\mathbf{x}$. Separation boundary separates vectors $\mathbf{x}$ into two classes. Here, in cases of linearly separable data, the boundary is also a (separating) hyperplane but of a lower order than $d(\mathbf{x}_p, \mathbf{w}, b)$.

The decision (separation) *boundary* is an intersection of a decision *function* $d(\mathbf{x}, \mathbf{w}, b)$ and a space of input features. It is given by

$$d(\mathbf{x}, \mathbf{w}, b) = 0. \tag{5}$$

All these functions and relationships can be followed, for two-dimensional inputs $\mathbf{x}$, in Fig. 6. In this particular case, the decision boundary i.e., separating

[2]Note that the dashed separation lines in Fig. 2 represent the line that follows from $d(\mathbf{x}, \mathbf{w}, b) = 0$.

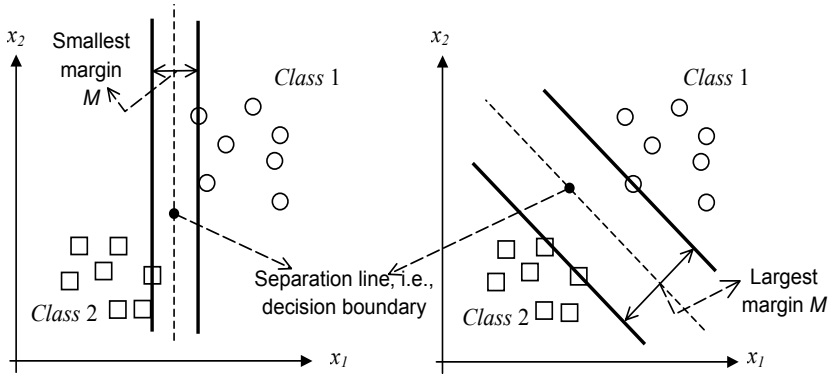

Figure 2. Two-out-of-many separating lines: a good one with a large margin *(right)* and a less acceptable separating line with a small margin, *(left)*.

(hyper)plane is actually a separating line in a $x_1 - x_2$ plane and, a decision function $d(\mathbf{x}, \mathbf{w}, b)$ is a plane over the 2-dimensional space of features, i.e., over a $x_1 - x_2$ plane.

In the case of 1-dimensional training patterns x (i.e., for 1-dimensional inputs x to the learning machine), decision function $d(\mathbf{x}, \mathbf{w}, b)$ is a straight line in an $x - y$ plane. An intersection of this line with an x-axis defines a point that is a separation boundary between two classes. This can be followed in Fig. 4. Before attempting to find an optimal separating hyperplane having the largest margin, we introduce the concept of the *canonical hyperplane.* We depict this concept with the help of the 1-dimensional example shown in Fig. 4. Not quite incidentally, the decision plane $d(\mathbf{x}, \mathbf{w}, b)$ shown in Fig. 6 is also a *canonical* plane. Namely, the values of d and of i_F are the same and both are equal to $|1|$ for the support vectors depicted by stars. At the same time, for all other training patterns $|d| > |i_F|$. In order to present a notion of this new concept of the canonical plane, first note that there are many hyperplanes that can correctly separate data. In Fig. 4 three different decision functions $d(\mathbf{x}, \mathbf{w}, b)$ are shown. There are infinitely many more. In fact, given $d(\mathbf{x}, \mathbf{w}, b)$, all functions $d(\mathbf{x}, k\mathbf{w}, kb)$, where k is a positive scalar, are correct decision functions too. Because parameters $(\mathbf{w}, b)$ describe the same separation hyperplane as parameters $(k\mathbf{w}, kb)$ there is a need to introduce the notion of a *canonical hyperplane*.

A hyperplane is in the canonical form with respect to training data $\mathbf{x} \in X$ if

$$\min_{x_i \in X} |\mathbf{w}^T \mathbf{x}_i + b| = 1. \tag{6}$$

The solid line $d(\mathbf{x}, \mathbf{w}, b) = -2x + 5$ in Fig. 4 fulfills (6) because *its minimal absolute value for the given six training patterns* belonging to two classes is 1. It achieves this value for two patterns, chosen as support vectors, namely for $x_3 = 2$, and $x_4 = 3$. For all other patterns, $|d| > 1$.

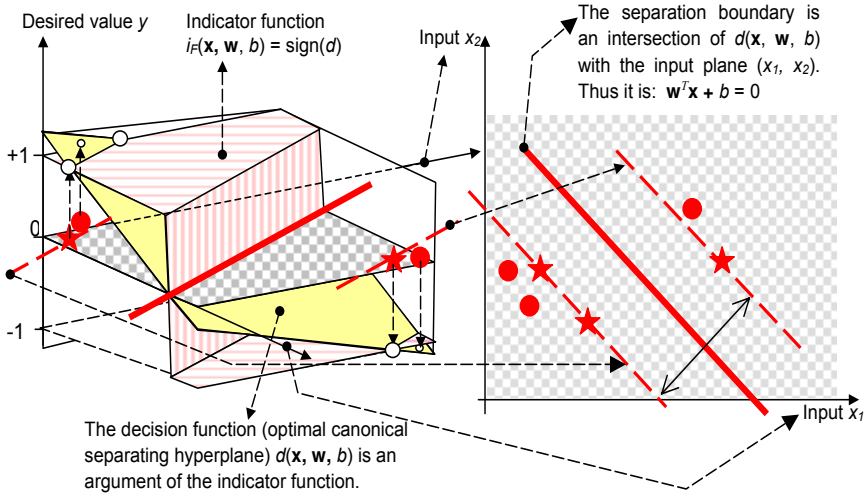

Figure 3. The definition of a decision (discriminant) *function* or hyperplane $d(\mathbf{x}, \mathbf{w}, b)$, a decision (separating) *boundary* $d(\mathbf{x}, \mathbf{w}, b) = 0$ and an indicator function $i_F = \text{sign}(d(\mathbf{x}_p, \mathbf{w}, b))$ which value represents a learning, or SV, machine's output o.

Note an interesting detail regarding the notion of a canonical hyperplane that is easily checked. There are many different hyperplanes (planes and straight lines for 2–D and 1–D problems in Figs 3 and 4 respectively) that have the same separation boundary (solid line and a dot in Figs 3 (right) and 4 respectively). At the same time there are far fewer hyperplanes that can be defined as canonical ones fulfilling (6). In Fig. 4, i.e., for a 1-dimensional input vector x, the canonical hyperplane is unique. This is not the case for training patterns of

higher dimension. Depending upon the configuration of class' elements, various canonical hyperplanes are possible.

Therefore, there is a need to define an *optimal* canonical hyperplane (OCSH) as a canonical hyperplane having a *maximal margin*. This search for a separating, maximal margin, canonical hyperplane is the ultimate learning goal in statistical learning theory underlying SV machines. Carefully note the adjectives used in the previous sentence. The hyperplane obtained from a limited training data must have a *maximal margin* because it will *probably* better classify new data. It must be in *canonical* form because this will ease the quest for significant patterns, here called support vectors. The canonical form of the hyperplane will also simplify the calculations. Finally, the resulting hyperplane must ultimately *separate* training patterns.

We avoid the derivation of an expression for the calculation of a distance (margin M) between the closest members from two classes for its simplicity here. The margin M can be derived by both the geometric and algebraic argument and is given as

$$M = \frac{2}{\|\mathbf{w}\|}. \tag{7}$$

This important result will have a great consequence for the constructive (i.e., learning) algorithm in a design of a maximal margin classifier. It will lead to solving a quadratic programming (QP) problem which will be shown shortly. Hence, the "good old" gradient learning in NNs will be replaced by solution of the QP problem here. This is the next important difference between the NNs and SVMs and follows from the implementation of SRM in designing SVMs, instead of a minimization of the sum of error squares, which is a standard cost function for NNs.

Equation (7) is a very interesting result showing that minimization of a norm of a hyperplane normal weight vector $\|\mathbf{w}\| = \sqrt{\mathbf{w}^T\mathbf{w}} = \sqrt{w_1^2 + w_2^2 + \cdots + w_n^2}$ leads to a maximization of a margin M. Because a minimization of $\sqrt{f}$ is equivalent to the minimization of f, the minimization of a norm $||\mathbf{w}||$ equals a minimization of $\mathbf{w}^T\mathbf{w} = \sum_{i=1}^{n} w_i^2 = w_1^2 + w_2^2 + \cdots + w_n^2$, and this leads to a maximization of a margin M. Hence, the learning problem is

$$\text{minimize } \frac{1}{2}\mathbf{w}^T\mathbf{w}, \tag{8a}$$

subject to constraints introduced and given in (8b) below[3].

[3] A multiplication of $\mathbf{w}^T\mathbf{w}$ by 0.5 is for numerical convenience only, and it doesn't change the solution.

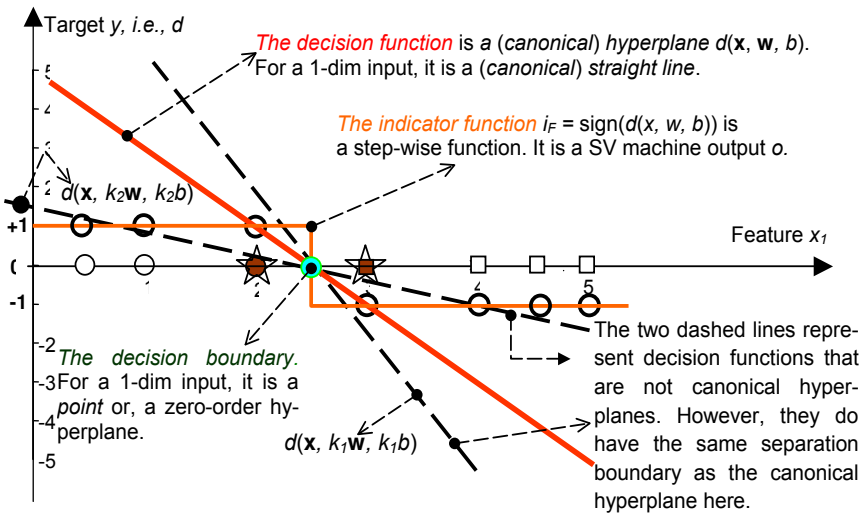

Figure 4. SV classification for 1-dimensional inputs by the linear decision function. Graphical presentation of a *canonical hyperplane*. For 1-dimensional inputs, it is actually a canonical straight line (depicted as a thick straight solid line) that passes through points $(+2, +1)$ and $(+3, -1)$ defined as the support vectors (stars). The two dashed lines are the two other decision hyperplanes (i.e., straight lines). The training input patterns $\{x_1 = 0.5,\ x_2 = 1,\ x_3 = 2\} \in Class\,1$ have a desired or target value (label) $y_1 = +1$. The inputs $\{x_4 = 3,\ x_5 = 4,\ x_6 = 4.5,\ x_7 = 5\} \in Class\,2$ have the label $y_2 = -1$.

Note that in the case of linearly separable classes empirical error equals zero ($R_{emp} = 0$ in (2a)) and minimization of $\mathbf{w}^T\mathbf{w}$ corresponds to a minimization of a confidence term Ω. The OCSH, i.e., a separating hyperplane with the largest margin defined by $M = 2/||\mathbf{w}||$, specifies *support vectors*, i.e., training data points closest to it, which satisfy $y_j[\mathbf{w}^T\mathbf{x}_j + b] \equiv 1,\ j = 1, N_{SV}$. For all the other (non-SVs data points) the OCSH satisfies inequalities $y_j[\mathbf{w}^T\mathbf{x}_j + b] > 1$. In other words, for all the data, OCSH should satisfy the following constraints

$$y_j[\mathbf{w}^T\mathbf{x}_j + b] > 1, \quad i = 1, l \tag{8b}$$

where l denotes a number of training data points, and N_{SV} stands for a number

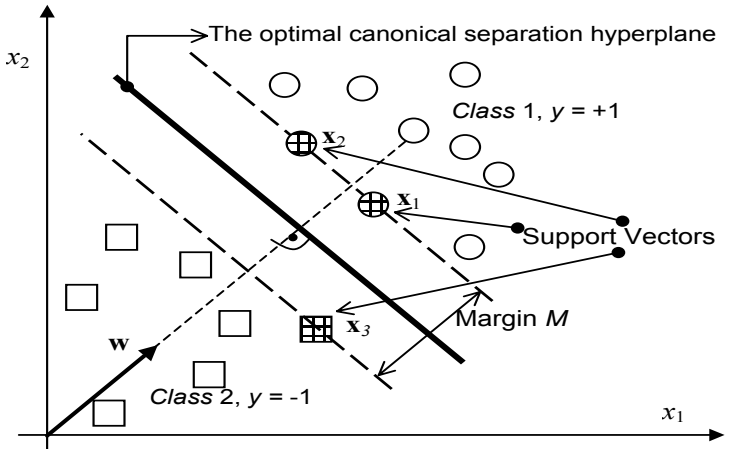

Figure 5. The optimal canonical separating hyperplane (OCSH) with the largest margin intersects halfway between the two classes. The points closest to it (satisfying $y_j|\mathbf{w}^T\mathbf{x}_j + b| = 1$, $j = 1, N_{SV}$) are *support vectors* and the OCSH satisfies $y_j|\mathbf{w}^T\mathbf{x}_j + b| \geqslant 1$, $j = 1, l$ (where l denotes the number of training data and N_{SV} stands for the number of SV). Three support vectors ($\mathbf{x}_1$ and $\mathbf{x}_2$ from class 1, and $\mathbf{x}_3$ from class 2) are the textured training data.

of SVs. The last equation can be easily checked visually in Figs 3 and 4 for 2-dimensional and 1-dimensional input vectors $\mathbf{x}$ respectively. Thus, in order to find the OCSH having a maximal margin, a learning machine should minimize $||\mathbf{w}||^2$ subject to the inequality constraints (8b). This is a *classic quadratic optimization problem with inequality constraints*. Such an optimization problem is solved by the *saddle point* of the Lagrange functional (Lagrangian)[4]

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{l} \alpha_i\{y_i[\mathbf{w}^T\mathbf{x}_i + b] - 1\} \tag{9}$$

where the α_i are Lagrange multipliers. The search for an optimal *saddle point*

[4]In forming the Lagrangian, for constraints of the form $f_i > 0$, the inequality constraints equations are multiplied by *nonnegative* Lagrange multipliers (i.e., $\alpha_i \geqslant 0$) and *subtracted* from the objective function.

$(\mathbf{w}_o, b_o, \boldsymbol{\alpha}_o)$ is necessary because Lagrangian L must be *minimized* with respect to $\mathbf{w}$ and b, and has to be *maximized* with respect to nonnegative α_i (i.e., $\alpha_i \geqslant 0$ should be found). This problem can be solved either in a *primal space* (which is the space of parameters $\mathbf{w}$ and b) or in a *dual space* (which is the space of Lagrange multipliers α_i). The second approach gives insightful results and we will consider the solution in a dual space below. In order to do that, we use Karush-Kuhn-Tucker (KKT) conditions for the optimum of a constrained function. In our case, both the objective function (9) and constraints (8b) are *convex* and KKT conditions are *necessary* and *sufficient* conditions for a maximum of (9). These conditions are: at the saddle point $(\mathbf{w}_o, b_o, \boldsymbol{\alpha}_o)$, derivatives of Lagrangian L with respect to primal variables should vanish which leads to,

$$\frac{\partial L}{\partial \mathbf{w}_o} = 0, \text{ i.e.,} \qquad \mathbf{w}_o = \sum_{i=1}^{l} \alpha_i y_i \mathbf{x}_i, \tag{10}$$

$$\frac{\partial L}{\partial b_o} = 0, \text{ i.e.,} \qquad \sum_{i=1}^{l} \alpha_i y_i = 0 \tag{11}$$

and the KKT complementarity conditions below (stating that at the solution point the products between dual variables and constraints equals zero) must also be satisfied,

$$\alpha_i \{ y_i [\mathbf{w}^T \mathbf{x}_i + b] - 1 \} = 0, \quad i = 1, l. \tag{12}$$

Substituting (10) and (11) into a *primal variables Lagrangian* $L(\mathbf{w}, b, \boldsymbol{\alpha})$ (9), we change to the *dual variables Lagrangian* $L_d(\alpha)$

$$L_d(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^T \mathbf{x}_j. \tag{13}$$

In order to find the optimal hyperplane, a dual Lagrangian $L_d(\boldsymbol{\alpha})$ has to be *maximized* with respect to nonnegative α_i (i.e., α_i must be in the nonnegative quadrant) and with respect to the equality constraint as follows

$$\alpha_i \geqslant 0, \quad i = 1, l, \tag{14a}$$

$$\sum_{i=1}^{l} \alpha_i y_i = 0. \tag{14b}$$

Note that the dual Lagrangian $L_d(\boldsymbol{\alpha})$ is expressed in terms of training data and depends *only* on the *scalar products* of input patterns $(\mathbf{x}_i^T\mathbf{x}_j)$. The dependency of $L_d(\boldsymbol{\alpha})$ on a scalar product of inputs will be very handy later when analyzing nonlinear decision boundaries and for general nonlinear regression. Note also that the number of unknown variables equals the number of training data l. After learning, the number of free parameters is equal to the number of SVs but it does not depend on the dimensionality of input space. Such a *standard quadratic optimization problem* can be expressed in a *matrix notation* and formulated as follows:

Maximize

$$L_d(\boldsymbol{\alpha}) = -0.5\boldsymbol{\alpha}^T\mathbf{H}\boldsymbol{\alpha} + \mathbf{f}^T\boldsymbol{\alpha}, \tag{15a}$$

subject to

$$\mathbf{y}^T\boldsymbol{\alpha} = 0, \tag{15b}$$

$$\alpha_i \geqslant 0, \quad i = 1, l \tag{15c}$$

where $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_l]^T$, $\mathbf{H}$ denotes the Hessian matrix ($H_{ij} = y_iy_j(\mathbf{x}_i\mathbf{x}_j) = y_iy_j\mathbf{x}_i^T\mathbf{x}_j$) of this problem[5], and $\mathbf{f}$ is an $(l, 1)$ unit vector $\mathbf{f} = \mathbf{1} = [1\,1\,\ldots\,1]^T$. Solutions α_{oi} of the dual optimization problem above determine the parameters $\mathbf{w}_o$ and b_o of the optimal hyperplane according to (10) and (12) as follows

$$\mathbf{w}_o = \sum_{i=1}^{l} \alpha_{oi}y_i\mathbf{x}_i, \tag{16a}$$

$$b_o = \frac{1}{N_{SV}}\sum_{s=1}^{N_{SV}}\left(\frac{1}{y_s} - \mathbf{x}_s^T\mathbf{w}_o\right) = \frac{1}{N_{SV}}\sum_{s=1}^{N_{SV}}(y_s - \mathbf{x}_s^T\mathbf{w}_o), \quad s = 1, N_{SV}. \tag{16b}$$

In deriving (16b) we used the fact that y can be either +1 or -1, and $1/y = y$. N_{SV} denotes the number of support vectors. There are two important observations about the calculation of $\mathbf{w}_o$. First, an optimal weight vector $\mathbf{w}_o$, is obtained in (16a) as a linear combination of the training data points and second, $\mathbf{w}_o$ (same as the bias term b_0) is calculated by using only the selected data points called *support vectors* (SVs). The fact that the summations in (16a) goes over all training data patterns (i.e., from 1 to l) is irrelevant because the Lagrange multipliers for all non-support vectors equal zero ($\alpha_{oi} = 0, \;\; i = N_{SV} + 1, l$).

[5]Note that maximization of (15a) equals a minimization of $L_d(\boldsymbol{\alpha}) = 0.5\boldsymbol{\alpha}^T\mathbf{H}\boldsymbol{\alpha} - \mathbf{f}^T\boldsymbol{\alpha}$, subject to the same constraints.

Finally, having calculated $\mathbf{w}_o$ and b_o we obtain a decision hyperplane $d(\mathbf{x})$ and an indicator function $i_F = o = \mathrm{sign}(d(\mathbf{x}))$ as given below

$$d(\mathbf{x}) = \sum_{i=1}^{l} w_{oi} x_i + b_o = \sum_{i=1}^{l} y_i \alpha_i \mathbf{x}^T \mathbf{x}_i + b_o\,, \quad i_F = o = \mathrm{sign}(d(\mathbf{x})). \quad (17)$$

Training data patterns having non-zero Lagrange multipliers are called *support vectors*. For linearly separable training data, all support vectors lie on the margin and they are generally just a small portion of all training data (typically, $N_{SV} \ll l$). Figs 3, 4 and 5 show the geometry of standard results for non-overlapping classes.

Before presenting applications of OCSH for both overlapping classes and classes having nonlinear decision boundaries, we will comment only on whether and how SV based linear classifiers actually implement the SRM principle. The more detailed presentation of this important property can be found in (*Kecman*, 2001; *Schölkopf* and *Smola* 2002)). First, it can be shown that an increase in margin reduces the number of points that can be shattered i.e., the increase in margin reduces the VC dimension, and this leads to the decrease of the SVM capacity. In short, by minimizing $||\mathbf{w}||$ (i.e., maximizing the margin) the SV machine training actually minimizes the VC dimension and consequently a generalization error (expected risk) at the same time. This is achieved by imposing a structure on the set of canonical hyperplanes and then, during the training, by choosing the one with a minimal VC dimension. A structure on the set of canonical hyperplanes is introduced by considering various hyperplanes having different $||\mathbf{w}||$. In other words, we analyze sets S_A such that $||\mathbf{w}|| \leqslant A$. Then, if $A_1 \leqslant A_2 \leqslant A_3 \leqslant \ldots \leqslant A_n$, we introduced a nested set $S_{A1} \subset S_{A2} \subset S_{A3} \subset \ldots \subset S_{An}$. Thus, if we impose the constraint $||\mathbf{w}|| \leqslant A$, then the canonical hyperplane cannot be closer than $1/A$ to any of the training points $\mathbf{x}_i$. *Vapnik* in (Vapnik, 1995) states that the VC dimension h of a set of canonical hyperplanes in $\Re^n$ such that $||\mathbf{w}|| \leqslant A$ is

$$H \leqslant \min[R^2 A^2, n] + 1, \quad (18)$$

where all the training data points (vectors) are enclosed by a sphere of the smallest radius R. Therefore, a small $||\mathbf{w}||$ results in a small h, and minimization of $||\mathbf{w}||$ is an implementation of the SRM principle. In other words, a minimization of the canonical hyperplane weight norm $||\mathbf{w}||$ minimizes the VC dimension according to (18). See also Fig. 4 that shows how the estimation

error, meaning the expected risk (because the empirical risk, due to the linear separability, equals zero) decreases with a decrease of a VC dimension. Finally, there is an interesting, simple and powerful result (*Vapnik*, 1995) connecting the generalization ability of learning machines and the number of support vectors. Once the support vectors have been found, we can calculate the bound on the expected probability of committing an error on a test example as follows

$$E_l[P(\text{error})] \leqslant \frac{E[\text{number of support vectors}]}{l}, \tag{19}$$

where E_l denotes expectation over all training data sets of size l. Note how easy it is to estimate this bound that is independent of the dimensionality of the input space. Therefore, an SV machine having a small number of support vectors will have good generalization ability even in a very high-dimensional space.

Example below shows the SVM's learning of the weights for a simple separable data problem in both the primal and the dual domain. The small number and low dimensionality of data pairs is used in order to show the optimization steps analytically and graphically. The same reasoning will be in the case of high dimensional and large training data sets but for them, one has to rely on computers and the insight in solution steps is necessarily lost.

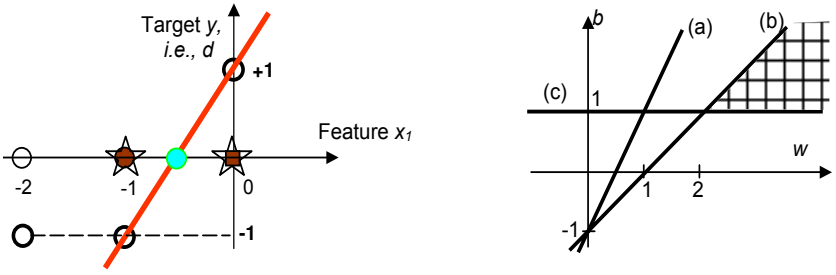

Figure 6. *Left:* Solving SVM classifier for 3 data shown. SVs are star data. *Right:* Solution space $w - b$.

***Example**: Design of an SVM classifier for 3 data shown in Fig. 6 above.*
First we solve the problem in the *primal domain*: From the constraints (8b) it

follows

$$2w - 1 \geqslant b, \qquad (a)$$

$$w - 1 \geqslant b, \qquad (b)$$

$$b \geqslant 1. \qquad (c)$$

The three straight lines corresponding to the equalities above are shown in Fig. 6 right. The textured area is a feasible domain for the weight w and bias b. Note that the area is not defined by the inequality (a), thus pointing to the fact that the point -1 is not a support vector. Points -1 and 0 define the textured area and they will be the supporting data for our decision function. The task is to minimize (8a), and this will be achieved by taking the value $w = 2$. Then, from (b), it follows that $b = 1$. Note that (a) must not be used for the calculation of the bias term b. Because both the cost function (8a) and the constraints (8b) are convex, the primal and the dual solution must produce same w and b. Dual solution follows from maximizing (13) subject to (14) as follows

$$L_d = \alpha_1 + \alpha_2 + \alpha_3 - \frac{1}{2}[\,\alpha_1\ \alpha_2\ \alpha_3] \begin{bmatrix} 4 & 2 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix},$$

such that

$$-\alpha_1 - \alpha_2 + \alpha_3 = 0,$$

$$\alpha_1 \geqslant 0, \quad \alpha_2 \geqslant 0, \quad \alpha_3 \geqslant 0,$$

The dual Lagrangian is obtained in terms of α_1 and α_2 after expressing α_3 from the equality constraint and it is given as $L_d = 2\alpha_1 + 2\alpha_2 - 0.5(4\alpha_1^2 + 4\alpha_1\alpha_2 + \alpha_2^2)$. L_d will have maximum for $\alpha_1 = 0$, and it follows that we have to find the maximum of $L_d = 2\alpha_2 - 0.5\alpha_2^2$ which will be at $\alpha_2 = 2$. Note that the Hessian matrix is extremely bad conditioned and if the QP problem is to be solved by computer $\mathbf{H}$ should be regularized first. From the equality constraint it follows that $\alpha_3 = 2$ too. Now, we can calculate the weight vector w and the bias b from (16a) and (16b) as follows,

$$w = \sum_{i=1}^{3} \alpha_i y_i \mathbf{x}_i = 0(-1)(-2) + 2(-1)(-1) + 2(1)0 = 2\,.$$

The bias can be calculated by using SVs only, meaning from either point -1 or point 0. Both result in same value as shown below

$$b = -1 - 2(-1) = 1, \ \text{ or } b = 1 - 2(0) = 1.$$

Linear Soft Margin Classifier for Overlapping Classes

The learning procedure presented above is valid for linearly separable data, meaning for training data sets without overlapping. Such problems are rare in practice. At the same time, there are many instances when linear separating hyperplanes can be good solutions even when data are overlapped (e.g., normally distributed classes having the same covariance matrices have a linear separation boundary). However, quadratic programming solutions as given above cannot be used in the case of overlapping because the constraints $y_i[\mathbf{w}^T\mathbf{x}_i + b] \geqslant 1$, $i = 1, l$ given by (8b) cannot be satisfied. In the case of an overlapping (see Fig 7), the overlapped data points cannot be correctly classified and for any misclassified training data point $\mathbf{x}_i$, the corresponding α_i will tend to infinity. This particular data point (by increasing the corresponding α_i value) attempts to exert a stronger influence on the decision boundary in order to be classified correctly. When the α_i value reaches the maximal bound, it can no longer increase its effect, and the corresponding point will stay misclassified. In such a situation, the algorithm introduced above chooses (almost) all training data points as support vectors. To find a classifier with a maximal margin, the algorithm presented in the section "Linear Maximal Margin Classifier for Linearly Separable Data" above, must be changed allowing some data to be unclassified. Better to say, we must leave some data on the "wrong" side of a decision boundary. In practice, we allow a *soft* margin and all data inside this margin (whether on the correct side of the separating line or on the wrong one) are neglected. The width of a soft margin can be controlled by a corresponding penalty parameter C (introduced below) that determines the trade-off between the training error and VC dimension of the model.

The question now is how to measure the degree of misclassification and how to incorporate such a measure into the hard margin learning algorithm given by equations (8). The simplest method would be to form the following learning problem

$$\text{minimize} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C(\text{number of misclassified data}), \tag{20}$$

where C is a penalty parameter, trading off the margin size (defined by $||\mathbf{w}||$, i.e., by $\mathbf{w}^T\mathbf{w}$) for the number of misclassified data points. Large C leads to small number of misclassifications, bigger $\mathbf{w}^T\mathbf{w}$ and consequently to the smaller margin and vice versa. Obviously taking $C = \infty$ requires that the number of misclassified data is zero and, in the case of an overlapping this is

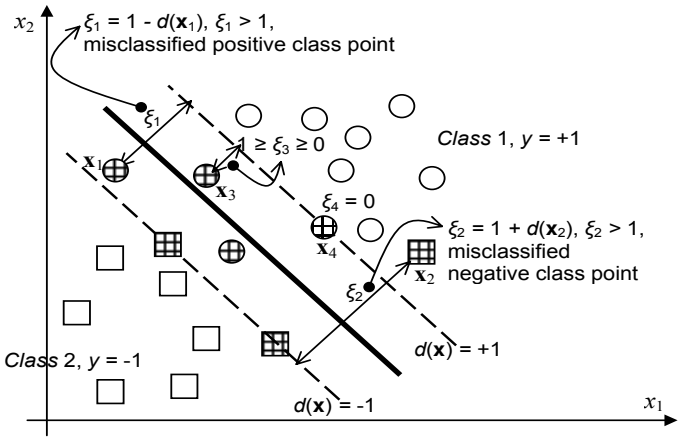

Figure 7. The soft decision boundary for a dichotomization problem with data overlapping. Separation line (*solid*), margins (*dashed*) and support vectors (textured training data points). 4 SVs in positive class (*circles*) and 3 SVs in negative class (*squares*). 2 misclassifications for positive class and 1 misclassification for negative class.

not possible. Hence, the problem may be feasible only for some value $C < \infty$.

However, the serious problem with (20) is that the error's counting can't be accommodated within the handy (meaning reliable, well understood and well developed) quadratic programming approach. Also, the counting only can't distinguish between huge (or disastrous) errors and close misses! The possible solution is to measure the distances ξ_i of the points crossing the margin from the corresponding margin and trade their sum for the margin size as given below

$$\text{minimize} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C(\text{sum of distances of the wrong side points}). \qquad (21)$$

In fact this is exactly how the problem of the data overlapping was solved in (*Cortes*, 1995; *Cortes* and *Vapnik*, 1995) — by generalizing the optimal "hard" margin algorithm. They introduced the nonnegative *slack variables* $\xi_i (i = 1, l)$ in the statement of the optimization problem for the overlapped data points.

Now, instead of fulfilling (8a) and (8b), the separating hyperplane must satisfy

$$\text{minimize} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}\xi_i, \tag{22a}$$

subject to

$$y_i[\mathbf{w}^T\mathbf{x}_i + b] \geqslant 1 - \xi_i, \quad i = 1, l,\ \xi_i \geqslant 0, \tag{22b}$$

i.e., subject to

$$\mathbf{w}^T\mathbf{x}_i + b \geqslant +1 - \xi_i, \quad \text{for } y_i = +1,\ \xi_i \geqslant 0, \tag{22c}$$

$$\mathbf{w}^T\mathbf{x}_i + b \geqslant -1 + \xi_i, \quad \text{for } y_i = -1,\ \xi_i \geqslant 0. \tag{22d}$$

Hence, for such a *generalized* optimal separating hyperplane, the functional to be minimized comprises an extra term accounting the cost of overlapping errors. In fact the cost function (22a) can be even more general as given below

$$\text{minimize} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}\xi_i^k, \tag{22e}$$

subject to same constraints. This is a convex programming problem that is usually solved only for $k = 1$ or $k = 2$, and such soft margin SVMs are dubbed *L1* and *L2* SVMs respectively. By choosing exponent $k = 1$, neither slack variables ξ_i nor their Lagrange multipliers β_i appear in a dual Lagrangian L_d. Same as for a linearly separable problem presented previously, for *L1* SVMs ($k = 1$) here, the solution to a quadratic programming problem (22), is given by the saddle point of the primal Lagrangian $L_p(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ shown below

$$\begin{aligned} L_p(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) &= \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}\xi_i - \\ -\sum_{i=1}^{l}\alpha_i\{y_i[\mathbf{w}^T\mathbf{x}_i + b] - 1 + \xi_i\} &- \sum_{i=1}^{l}\beta_i x_i, \quad \text{for } \textit{L1} \text{ SVM} \end{aligned} \tag{23}$$

where α_i and β_i are the Lagrange multipliers. Again, we should find an *optimal* saddle point $(\mathbf{w}_o, b_o, \boldsymbol{\xi}_o, \boldsymbol{\alpha}_o, \boldsymbol{\beta}_o)$ because the Lagrangian L_p has to be *minimized* with respect to $\mathbf{w}$, b and $\boldsymbol{\xi}$, and *maximized* with respect to nonnegative α_i and β_i. As before, this problem can be solved in either a *primal space* or *dual space* (which is the space of Lagrange multipliers α_i and β_i). Again, we consider a solution in a dual space as given below by using

- standard conditions for an optimum of a constrained function

$$\frac{\partial L}{\partial \mathbf{w}_o} = 0, \text{ i.e.,} \qquad \mathbf{w}_o = \sum_{i=1}^{l} \alpha_i y_i \mathbf{x}_i, \tag{24}$$

$$\frac{\partial L}{\partial b_o} = 0, \text{ i.e.,} \qquad \sum_{i=1}^{l} \alpha_i y_i = 0 \tag{25}$$

$$\frac{\partial L}{\partial \xi_{io}} = 0, \text{ i.e.,} \qquad \alpha_i + \beta_i = C, \tag{26}$$

- and the KKT complementarity conditions below,

$$\alpha_i \{ y_i [\mathbf{w}^T \mathbf{x}_i + b] - 1 + \xi_i \} = 0, \quad i = 1, l, \tag{27a}$$

$$\beta_i \xi_i = (C - \alpha_i) \xi_i = 0, \quad i = 1, l. \tag{27b}$$

At the optimal solution, due to the KKT conditions (27), the last two terms in the primal Lagrangian L_p given by (23) vanish and the *dual variables Lagrangian* $L_d(\boldsymbol{\alpha})$, for *L1* SVM, is not a function of β_i. In fact, it is same as the hard margin classifier's L_d given before and repeated here for the soft margin one,

$$L_d(\boldsymbol{\alpha}) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^T \mathbf{x}_j. \tag{28}$$

In order to find the optimal hyperplane, a dual Lagrangian $L_d(\boldsymbol{\alpha})$ has to be *maximized* with respect to nonnegative and (unlike before) smaller than or equal to C, α_i. In other words with

$$C \geqslant \alpha_i \geqslant 0, \quad i = 1, l, \tag{29a}$$

and under the constraint (25), i.e., under

$$\sum_{i=1}^{l} \alpha_i y_i = 0. \tag{29b}$$

Thus, the final quadratic optimization problem is practically same as for the separable case the only difference being in the modified bounds of the Lagrange multipliers α_i. The penalty parameter C, which is now the upper bound on α_i,

is determined by the user. The selection of a "good" or "proper" C is always done experimentally by using some cross-validation technique. Note that in the previous linearly separable case, without data overlapping, this upper bound $C = \infty$. We can also readily change to the matrix notation of the problem above as in equations (15). Most important of all is that the learning problem is expressed only in terms of unknown Lagrange multipliers α_i, and known inputs and outputs. Furthermore, optimization does not solely depend upon inputs $\mathbf{x}_i$ which can be of a very high (inclusive of an infinite) dimension, but it depends upon a scalar product of input vectors $\mathbf{x}_i$. It is this property we will use in the next section where we design SV machines that can create nonlinear separation boundaries. Finally, expressions for both a *decision function* $d(\mathbf{x})$ and an *indicator function* $i_F = \text{sign}(d(\mathbf{x}))$ for a soft margin classifier are same as for linearly separable classes and are also given by (17).

From (27) follows that there are only three possible solutions for α_i (see Fig. 7)

1. $\alpha_i = 0, \;\; \xi_i = 0 \rightarrow$ data point $\mathbf{x}_i$ is correctly classified.
2. $C > \alpha_i > 0 \rightarrow$ then, the two complementarity conditions must result result in $y_i[\mathbf{w}^T\mathbf{x}_i + b\,] - 1 + \xi_i = 0$, and $\xi_i = 0$. Thus, $y_i[\mathbf{w}^T\mathbf{x}_i + b\,] = 1$ and $\mathbf{x}_i$ is a support vector. The support vectors with $C \geqslant \alpha_i \geqslant 0$ are called *unbounded* or *free support vectors*. They lie on the two margins.
3. $\alpha_i = C \rightarrow$ then, $y_i[\mathbf{w}^T\mathbf{x}_i + b\,] - 1 + \xi_i = 0$, and $\xi_i \geqslant 0$, and $\mathbf{x}_i$ is a support vector. The support vectors with $\alpha_i = C$ are called *bounded support vectors*. They lie on the "wrong" side of the margin. For $1 > \xi_i \geqslant 0$, $\mathbf{x}_i$ is still correctly classified, and if $\xi_i \geqslant 1$, $\mathbf{x}_i$ is misclassified.

For *L2* SVM the second term in the cost function (22e) is quadratic, i.e., $C\sum_{i=1}^{l}\xi_i^2$, and this leads to changes in a dual optimization problem which is now,

$$L_d(\boldsymbol{\alpha}) = \sum_{i=1}^{l}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \Big(\mathbf{x}_i^T\mathbf{x}_j + \frac{\delta_{ij}}{C}\Big), \tag{30}$$

subject to

$$\alpha_i \geqslant 0, \quad i = 1, l, \tag{31a}$$

$$\sum_{i=1}^{l}\alpha_i y_i = 0. \tag{31b}$$

where, $\delta_{ij} = 1$ for $i = j$, and it is zero otherwise. Note the change in Hessian matrix elements given by second terms in (30), as well as that there is no upper

bound on α_i. The detailed analysis and comparisons of the $L1$ and $L2$ SVMs is presented in (*Abe*, 2004). We use the most popular $L1$ SVMs here, because they usually produce more sparse solutions, i.e., they create a decision function by using less SVs than the $L2$ SVMs.

The Nonlinear Classifier

The linear classifiers presented in two previous sections are very limited. Mostly, classes are not only overlapped but the genuine separation functions are nonlinear hypersurfaces. A nice and strong characteristic of the approach presented above is that it can be easily (and in a relatively straightforward manner) extended to create nonlinear decision boundaries. The motivation for such an extension is that an SV machine that can create a nonlinear decision hypersurface will be able to classify nonlinearly separable data. This will be achieved by considering a linear classifier in the so-called *feature space* that will be introduced shortly. A very simple example of a need for designing nonlinear models is given in Fig. 8 where the true separation boundary is quadratic. It is obvious that no errorless linear separating hyperplane can be found now. The best linear separation function shown as a dashed straight line would make six misclassifications (textured data points; 4 in the negative class and 2 in the positive one). Yet, if we use the nonlinear separation boundary we are able to separate two classes without any error. Generally, for n-dimensional input patterns, instead of a nonlinear curve, an SV machine will create a nonlinear separating hypersurface.

The basic idea in designing nonlinear SV machines is to map input vectors $\mathbf{x} \in \Re^n$ into vectors $\Phi(\mathbf{x})$ of a higher dimensional *feature space* F (where Φ represents mapping: $\Re^n \to \Re^f$), and to solve a linear classification problem in this feature space

$$\mathbf{x} \in \Re^n \to \Phi(\mathbf{x}) = [\phi_1(\mathbf{x})\ \phi_2(\mathbf{x})\ \ldots\ \phi_n(\mathbf{x})]^T \in \Re^f. \tag{32}$$

A mapping $\Phi(\mathbf{x})$ is chosen in advance, i.e., it is a fixed function. Note that an input space ($\mathbf{x}$-space) is spanned by components x_i of an input vector $\mathbf{x}$ and a feature space F (Φ-space) is spanned by components $\phi_i(\mathbf{x})$ of a vector $\Phi(\mathbf{x})$. By performing such a mapping, we hope that in a Φ-space, our learning algorithm will be able to linearly separate images of $\mathbf{x}$ by applying the linear SVM formulation presented above. (In fact, it can be shown that for a whole class of mappings the linear separation in a feature space is always possible. Such

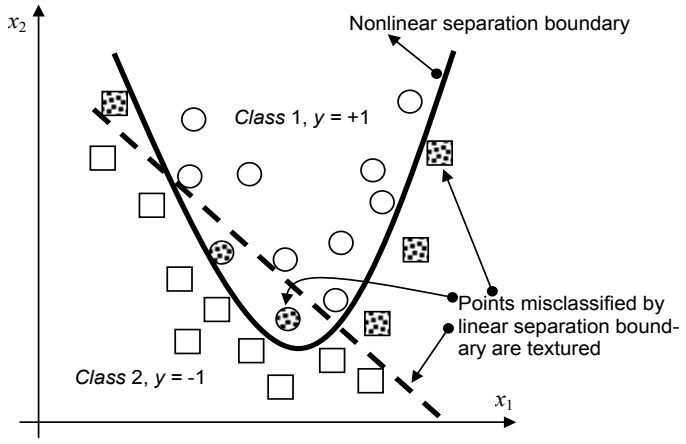

Figure 8. A nonlinear SVM without data overlapping. A true separation is a quadratic curve. The nonlinear separation line (*solid*), the linear one (*dashed*) and data points misclassified by the linear separation line (*the textured training data points*) are shown. There are 4 misclassified negative data and 2 misclassified positive ones. SVs are not shown.

mappings will correspond to the positive definite kernels that will be shown shortly). We also expect this approach to again lead to solving a quadratic optimization problem with similar constraints in a Φ-space. The solution for an indicator function

$$i_F(\mathbf{x}) = \operatorname{sign}(\mathbf{w}^T\Phi(\mathbf{x}) + b) = \operatorname{sign}\Big(\sum_{i=1}^{l} y_i\alpha_i\Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}) + b\Big),$$

which is a linear classifier in a feature space, will create a nonlinear separating hypersurface in the original input space given by (33) below. (Compare this solution with (17) and note the appearances of scalar products in both the original X-space and in the feature space F).

The equation for an $i_F(\mathbf{x})$ just given above can be rewritten in a "neural

networks" form as follows

$$i_F(\mathbf{x}) = \operatorname{sign}\Big(\sum_{i=1}^{l} y_i \alpha_i \Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}) + b\Big) = \\ = \operatorname{sign}\Big(\sum_{i=1}^{l} y_i \alpha_i k(\mathbf{x}_i, \mathbf{x}) + b\Big) = \operatorname{sign}\Big(\sum_{i=1}^{l} \nu_i k(\mathbf{x}_i, \mathbf{x}) + b\Big) \tag{33}$$

where v_i corresponds to the output layer weights of the "SVM's network" and $k(\mathbf{x}_i, \mathbf{x})$ denotes the value of the kernel function that will be introduced shortly [6]. Note the difference between the weight vector $\mathbf{w}$ which norm should be minimized and which is the vector of the same dimension as the feature space vector $\Phi(\mathbf{x})$ and the weightings $v_i = \alpha_i y_i$ that are scalar values composing the weight vector $\mathbf{v}$ which dimension equals the number of training data points l. The $(l - N_{\text{SVs}})$ of v_i components are equal to zero, and only N_{SVs} entries of $\mathbf{v}$ are nonzero elements.

A simple example below (Fig 9) should exemplify the idea of a nonlinear mapping to (usually) higher dimensional space and how it happens that the data become linearly separable in the F-space.

Consider solving the simplest 1-D classification problem given the input and the output (desired) values as follows: $\mathbf{x} = [-1 \;\; 0 \;\; 1]^T$ and $\mathbf{d} = \mathbf{y} = [-1 \;\; 1 \;\; -1]^T$. Here we choose the following mapping to the feature space: $\Phi(\mathbf{x}) = [\varphi_1(\mathbf{x}) \;\; \varphi_2(\mathbf{x}) \;\; \varphi_3(\mathbf{x})]^T = [x^2 \;\; \sqrt{2}x \;\; 1]^T)$

The mapping produces the following three points in the feature space (shown as the *rows* of the matrix $\mathbf{F}$ (F standing for features))

$$\mathbf{F} = \begin{bmatrix} 1 & -\sqrt{2} & 1 \\ 0 & 0 & 1 \\ 1 & \sqrt{2} & 1 \end{bmatrix}^T.$$

These three points are linearly separable by the plane $\varphi_3(\mathbf{x}) = 2\varphi_1(\mathbf{x})$ in a feature space as shown in Fig. 10. It is easy to show that the mapping obtained by $\Phi(\mathbf{x}) = [x^2 \;\; \sqrt{2}x \;\; 1]^T$ is a scalar product implementation of a quadratic kernel function $(\mathbf{x}_i^T \mathbf{x}_j + 1)^2 = k(\mathbf{x}_i, \mathbf{x}_j)$. In other words, $\Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}_j) = k(\mathbf{x}_i, \mathbf{x}_j)$. This equality will be introduced shortly.

There are two basic problems when mapping an input $\mathbf{x}$-space into higher order F-space:

[6] v_i equals $y_i \alpha_i$ in the classification case presented above and it is equal to $(\alpha_i - \alpha_i^*)$ in the regression problems.

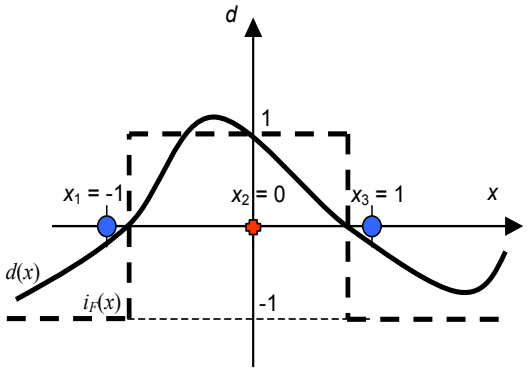

Figure 9. A nonlinear 1-dimensional classification problem. One possible solution is given by the decision function $d(x)$ (*solid curve*) i.e., by the corresponding indicator function defined as $i_F = \text{sign}(d(x))$ (*dashed stepwise function*).

1) the choice of mapping $\Phi(\mathbf{x})$ that should result in a "rich" class of decision hypersurfaces,
2) the calculation of the scalar product $\Phi^T(\mathbf{x})\Phi(\mathbf{x})$ that can be computationally very discouraging if the number of features f (i.e., dimensionality f of a feature space) is very large.

The second problem is connected with a phenomenon called the *"curse of dimensionality"*. For example, to construct a decision surface corresponding to a polynomial of degree *two* in an n-D input space, a dimensionality of a feature space $f = n(n+3)/2$. In other words, a feature space is spanned by f coordinates of the form

$$\begin{aligned} z_1 = x_1, \ldots, z_n = x_n \quad & (n \text{ coordinates}), \\ z_{n+1} = (x_1)^2, \ldots, z_{2n} = (x_n)^2 \quad & (\text{next } n \text{ coordinates}), \\ z_{2n+1} = x_1 x_2, \ldots, z_f = x_n x_{n-1} \quad & (n(n-1)/2 \text{ coordinates}), \end{aligned}$$

and the separating hyperplane created in this space, is a second-degree polynomial in the input space (*Vapnik*, 1998). Thus, constructing a polynomial of degree two only, in a 256-dimensional input space, leads to a dimensionality of

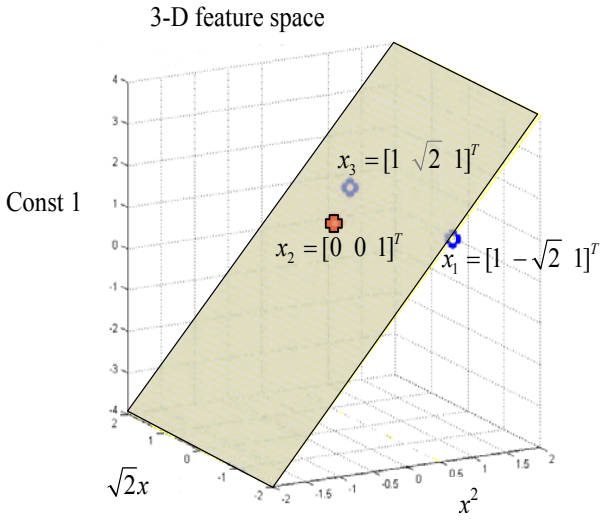

Figure 10. The three data points of a problem in Fig. 9 are linearly separable in the feature space (obtained by the mapping $\Phi(\mathbf{x}) = [\varphi_1(\mathbf{x}) \ \ \varphi_2(\mathbf{x}) \ \ \varphi_3(\mathbf{x})]^T = [x^2 \ \ \sqrt{2}x \ \ 1]^T$). The separation boundary is given as the plane $\varphi_3(\mathbf{x}) = 2\varphi_1(\mathbf{x})$ shown in the figure.

a feature space $f = 33,152$. Performing a scalar product operation with vectors of such, or higher, dimensions, is not a cheap computational task. The problems become serious (and fortunately only seemingly unsolvable) if we want to construct a polynomial of degree 4 or 5 in the same 256-dimensional space leading to the construction of a decision hyperplane in a billion-dimensional feature space.

This explosion in dimensionality can be avoided by noticing that in the quadratic optimization problem given by (13) and (28), as well as in the final expression for a classifier, *training data only appear in the form of scalar products* $\mathbf{x}_i^T\mathbf{x}_j$. These products will be replaced by scalar products

$$\Phi^T(\mathbf{x})\Phi(\mathbf{x}) = [\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \ldots, \phi_n(\mathbf{x})]^T[\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \ldots, \phi_n(\mathbf{x})]$$

in a feature space F, and the latter can be and will be expressed by using the *kernel function* $K(\mathbf{x}_i, \mathbf{x}_j) = \Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}_j)$.

Note that a *kernel function* $K(\mathbf{x}_i, \mathbf{x}_j)$ is a function in input space. Thus, the basic advantage in using kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$ is in avoiding performing a mapping $\Phi(\mathbf{x})$ at all. Instead, the required scalar products in a feature space $\Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}_j)$, are calculated directly by computing kernels $K(\mathbf{x}_i, \mathbf{x}_j)$ for given training data vectors in an input space. In this way, we bypass a possibly extremely high dimensionality of a feature space F. Thus, by using the chosen kernel $K(\mathbf{x}_i, \mathbf{x}_j)$, we can construct an SVM that operates in an infinite dimensional space (such a kernel function is a Gaussian kernel function given in table 2 below). In addition, as will be shown below, by applying kernels we do not even have to know what the actual mapping $\Phi(\mathbf{x})$ is. A kernel is a function K such that

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}_j). \tag{34}$$

There are many possible kernels, and the most popular ones are given in Table 2. All of them should fulfill the so-called Mercer's conditions. The Mercer's kernels belong to a set of *reproducing kernels.* For further details see (*Mercer*, 1909; *Aizerman* et al, 1964; *Smola* and *Schölkopf*, 1997; *Vapnik*, 1998; *Kecman*, 2001).

The simplest is a linear kernel defined as $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T\mathbf{x}_j$. Below we show a few more kernels.

Table 2. Popular Admissible Kernels

Kernel functions	**Type of classifier**
$K(\mathbf{x}, \mathbf{x}_i) = (\mathbf{x}^T\mathbf{x}_i)$	Linear, dot product, kernel, CPD
$K(\mathbf{x}, \mathbf{x}_i) = [(\mathbf{x}^T\mathbf{x}_i) + 1]^d$	Complete polynomial of degree *d,* PD
$K(\mathbf{x}, \mathbf{x}_i) = e^{-\frac{1}{2}[(\mathbf{x}-\mathbf{x}_i)^T\Sigma^{-1}(\mathbf{x}-\mathbf{x}_i)]}$	Gaussian RBF, PD
$K(\mathbf{x}, \mathbf{x}_i) = \tanh[(\mathbf{x}^T\mathbf{x}_i) + b]$*	Multilayer perceptron, CPD
$K(\mathbf{x}, \mathbf{x}_i) = \frac{1}{\sqrt{\|\mathbf{x}-\mathbf{x}_i\|^2 + \beta}}$	Inverse multiquadric function, PD

*only for certain values of *b,* (C)PD = (conditionally) positive definite

POLYNOMIAL KERNELS

Let $\mathbf{x} \in \Re^2$ i.e., $\mathbf{x} = [x_1 \ x_2]^T$, and if we choose $\Phi(\mathbf{x}) = [x_1^2 \ \sqrt{2}x_1x_2 \ x_2^2]^T$ (i.e., there is an $\Re^2 \to \Re^3$ mapping), then the dot product

$$\begin{aligned}\Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}_j) &= [x_{i1}^2 \ \sqrt{2}\,x_{i1}x_{i2} \ x_{i2}^2][x_{j1}^2 \ \sqrt{2}\,x_{j1}x_{j2} \ x_{j2}^2]^T = \\ &= [x_{i1}^2x_{j1}^2 \ \ 2\,x_{i1}x_2x_{j1}x_{j2} \ \ x_{i2}^2x_{j2}^2] = \\ &= (\mathbf{x}_i^T\mathbf{x}_j)^2 = K(\mathbf{x}_i, \mathbf{x}_j)\end{aligned}$$

or

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T\mathbf{x}_j)^2 = \Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}_j).$$

Note that in order to calculate the scalar product in a feature space $\Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}_j)$, we do not need to perform the mapping $\Phi(\mathbf{x}) = [x_1^2 \ \sqrt{2}\,x_1x_2 \ x_2^2]^T$ at all. Instead, we calculate this product directly in the input space by computing $(\mathbf{x}_i^T\mathbf{x}_j)^2$. This is very well known under the popular name of *the kernel trick.* Interestingly, note also that other mappings such as an

$\Re^2 \to \Re^3$ mapping given by $\Phi(\mathbf{x}) = [x_1^2 - x_2^2 \ \ 2x_1x_2 \ \ x_1^2 + x_2^2]$, or an
$\Re^2 \to \Re^4$ mapping given by $\Phi(\mathbf{x}) = [x_1^2 \ \ x_1x_2 \ \ x_1x_2 \ \ x_2^2]$,

also accomplish the same task as $(\mathbf{x}_i^T\mathbf{x}_j)^2$.

Now, assume the following mapping

$$\Phi(\mathbf{x}) = [1 \quad \sqrt{2}\,x_1 \quad \sqrt{2}\,x_2 \quad \sqrt{2}\,x_1x_2 \quad x_1^2 \quad x_1^2],$$

i.e., there is an $\Re^2 \to \Re^3$ mapping plus bias term as the constant 6^{th} dimension's value. Then the dot product in a feature space F is given as

$$\begin{aligned}\Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}_j) &= 1 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2} + 2x_{i1}x_{i2}x_{j1}x_{j2} + x_{i1}^2x_{j1}^2 + x_{i2}^2x_{j2}^2 = \\ &= 1 + 2(\mathbf{x}_i^T\mathbf{x}_j) + (\mathbf{x}_i^T\mathbf{x}_j)^2 = \\ &= (\mathbf{x}_i^T\mathbf{x}_j + 1)^2 = K(\mathbf{x}_i, \mathbf{x}_j)\end{aligned}$$

or

$$(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T\mathbf{x}_j + 1)^2 = \Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}_j).$$

Thus, the last mapping leads to the second order *complete* polynomial.

Many candidate functions can be applied to a convolution of an inner product (i.e., for kernel functions) $K(\mathbf{x}, \mathbf{x}_i)$ in an SV machine. Each of these functions constructs a different nonlinear decision hypersurface in an input space. In the first three rows, the table 2 shows the three most popular kernels in SVMs'

in use today, and the inverse multiquadrics one as an interesting and powerful kernel to be proven yet.

The positive definite (PD) kernels are the kernels which Gramm matrix $\mathbf{G}$ (a.k.a. Grammian, or a design matrix) calculated by using all the l training data points is positive definite (meaning all its eigenvalues are strictly positive, i.e., $\lambda_i > 0,\ i = 1, l$)

$$\mathbf{G} = \mathbf{K}(\mathbf{x}, \mathbf{x}_i) = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \cdots & k(\mathbf{x}_1, \mathbf{x}_l) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \cdots & k(\mathbf{x}_2, \mathbf{x}_l) \\ \vdots & \vdots & \vdots & \vdots \\ k(\mathbf{x}_l, \mathbf{x}_1) & k(\mathbf{x}_l, \mathbf{x}_2) & \cdots & k(\mathbf{x}_l, \mathbf{x}_l) \end{bmatrix} \tag{35}$$

The kernel matrix $\mathbf{G}$ is a symmetric one. Even more, any symmetric positive definite matrix can be regarded as a kernel matrix, that is — as an inner product matrix in some space.

Finally, we arrive at the point of presenting the learning in nonlinear classifiers (in which we are ultimately interested here). The learning algorithm for a nonlinear SV machine (classifier) follows from the design of an *optimal separating hyperplane* in a *feature space*. This is the same procedure as the construction of a "hard" (13) and "soft" (28) margin classifiers in an $\mathbf{x}$-space previously. In a $\Phi(\mathbf{x})$-space, the dual Lagrangian, given previously by (13) and (28), is now

$$L_d(\boldsymbol{\alpha}) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j \, \boldsymbol{\Phi}_i^T \boldsymbol{\Phi}_j, \tag{36}$$

and, according to (34), by using chosen kernels, we should maximize the following dual Lagrangian

$$L_d(\boldsymbol{\alpha}) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j \, K(\mathbf{x}, \mathbf{x}_i), \tag{37}$$

subject to

$$\alpha_i \geqslant 0, \quad i = 1, l, \quad \text{and} \quad \sum_{i=1}^{l} \alpha_i y_i = 0. \tag{37a}$$

In a more general case, because of a noise or due to generic class' features, there will be an overlapping of training data points. Nothing but constraints for

α_i change. Thus, the nonlinear "soft" margin classifier will be the solution of the quadratic optimization problem given by (37) subject to constraints

$$C \geqslant \alpha_i \geqslant 0, \quad i = 1, l \quad \text{and} \quad \sum_{i=1}^{l} \alpha_i y_i = 0. \tag{37b}$$

Again, the only difference to the separable nonlinear classifier is the upper bound C on the Lagrange multipliers α_i. In this way, we limit the influence of training data points that will remain on the "wrong" side of a separating nonlinear hypersurface. After the dual variables are calculated, the decision hypersurface $d(\mathbf{x})$ is determined by

$$d(\mathbf{x}) = \sum_{i=1}^{l} y_i \alpha_i K(\mathbf{x}, \mathbf{x}_i) + b = \sum_{i=1}^{l} \nu_i K(\mathbf{x}, \mathbf{x}_i) + b, \tag{38}$$

and the indicator function is

$$i_F(\mathbf{x}) = \text{sign}[d(\mathbf{x})] = \text{sign}\Big[\sum_{i=1}^{l} \nu_i K(\mathbf{x}, \mathbf{x}_i) + b\Big].$$

Note that the summation is not actually performed over all training data but rather over the support vectors, because only for them do the Lagrange multipliers differ from zero. The existence and calculation of a bias b is now not a direct procedure as it is for a linear hyperplane. Depending upon the applied kernel, the bias b can be implicitly part of the kernel function. If, for example, Gaussian RBF is chosen as a kernel, it can use a bias term as the $(f+1)^{st}$ feature in F-space with a constant output $= +1$, but not necessarily. In short, all PD kernels do not necessarily need an explicit bias term b, but b can be used. In section "On the Equality of Kernel AdaTron and Sequential Minimal Optimization and Alike Algorithms for Kernel Machines" we will develop new iterative learning algorithm for models having a bias term b[7]. Same as for the linear SVM, (37) can be written in a matrix notation as

maximize

$$L_d(\boldsymbol{\alpha}) = -0.5\boldsymbol{\alpha}^T \mathbf{H} \boldsymbol{\alpha} + \mathbf{f}^T \boldsymbol{\alpha}, \tag{39a}$$

subject to

$$\mathbf{y}^T \boldsymbol{\alpha} = 0,$$

[7] More on this can be found in (*Kecman, Huang*, and *Vogt*, 2005) as well as in the (*Vogt* and *Kecman*, 2005).

and

$$C \geqslant \alpha_i \geqslant 0, \quad i = 1, l. \tag{39c}$$

where $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_l]^T$, denotes the Hessian matrix

$$H_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

of this problem and $\mathbf{f}$ is an $(l, 1)$ unit vector $\mathbf{f} = \mathbf{1} = [1 \;\; 1 \;\; \ldots \;\; 1]^T$. Note that if $K(\mathbf{x}_i, \mathbf{x}_j)$ is the positive definite matrix, then so is the matrix $y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$ too.

The following 1-D example (just for the sake of graphical presentation) will show the creation of a linear decision function in a feature space and a corresponding nonlinear (quadratic) decision function in an input space.

Suppose we have 4 1-D data points given as $x_1 = 1$, $x_2 = 2$, $x_3 = 5$, $x_4 = 6$, with data at 1, 2, and 6 as class 1 and the data point at 5 as class 2, i.e., $y_1 = -1$, $y_2 = -1$, $y_3 = 1$, $y_4 = -1$. We use the polynomial kernel of degree 2, $K(x, y) = (xy + 1)^2$. C is set to 50, which is of lesser importance because the constraints will be not imposed in this example for maximal value for the dual variables alpha will be smaller than $C = 50$.

Case 1: Working with a bias term b as given in (38).

We first find $\alpha_i (i = 1, \ldots, 4)$ by solving dual problem (39) having a Hessian matrix

$$\mathbf{H} = \begin{bmatrix} 4 & 9 & -36 & 49 \\ 9 & 25 & -121 & 169 \\ -36 & -121 & 676 & -961 \\ 49 & 169 & -961 & 1369 \end{bmatrix}.$$

Alphas are $\alpha_1 = 0$, $\alpha_2 = 2.499999$, $\alpha_3 = 7.333333$, $\alpha_4 = 4.833333$ and the bias b will be found by using (16b), or by fulfilling the requirements that the values of a decision function at the support vectors should be the given y_i. The model (decision function) is given by

$$d(x) = \sum_{i=1}^{4} y_i \alpha_i K(x, x_i) + b = \sum_{i=1}^{4} \nu_i (x x_i + 1)^2 + b,$$

or by

$$d(x) = 2.499999(-1)(2x+1)^2 + 7.333333(1)(5x+1)^2 + 4.833333(-1)(6x+1)^2 + b,$$

$$d(x) = -0.666667x^2 + 5.333333x + b.$$

Bias b is determined from the requirement that at the SV points 2, 5 and 6, the outputs must be -1, 1 and -1 respectively. Hence, $b = -9$, resulting in the decision function

$$d(x) = -0.666667x^2 + 5.333333x + b.$$

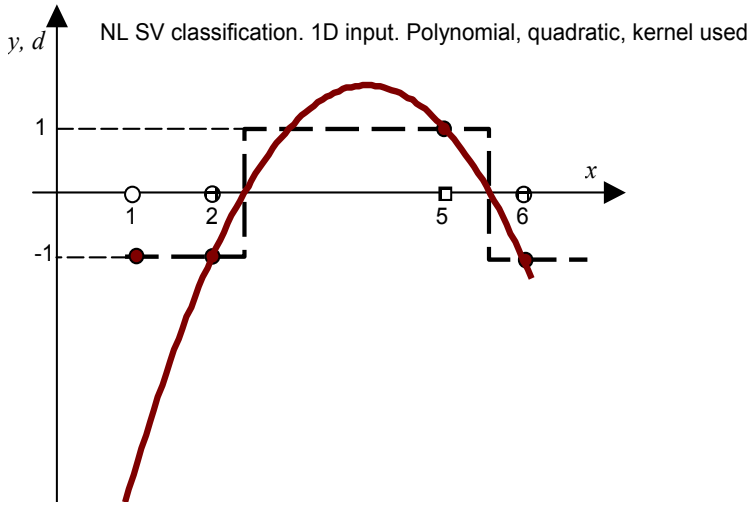

Figure 11. The nonlinear decision function (*solid*) and the indicator function (*dashed*) for 1-D overlapping data. By using a complete second order polynomial the model with and without a bias term b are same.

The nonlinear (quadratic) decision function and the indicator one are shown in Fig. 11. Note that in calculations above 6 decimal places have been used for alpha values. The calculation is numerically very sensitive, and working with fewer decimals can give very approximate or wrong results.

The complete polynomial kernel as used in the case 1, is *positive definite* and there is no need to use an explicit bias term b as presented above. Thus, one can use the same second order polynomial model without the bias term b. Note that in this particular case there is no equality constraint equation that originates from an equalization of the primal Lagrangian derivative in respect

to the bias term b to zero. Hence, we do not use (39b) while using a positive definite kernel without bias as it will be shown below in the case 2.

Case 2: Working without a bias term b.

Because we use the same second order polynomial kernel, the Hessian matrix $\mathbf{H}$ is same as in the case 1. The solution without the equality constraint for alphas is: $\alpha_1 = 0$, $\alpha_2 = 24.999999$, $\alpha_3 = 43.333333$, $\alpha_4 = 27.333333$. The model (decision function) is given by

$$d(x) = \sum_{i=1}^{4} y_i \alpha_i K(x, x_i) = \sum_{i=1}^{4} \nu_i (x x_i + 1)^2,$$

or by

$$d(x) = 2.499999(-1)(2x+1)^2 + 7.333333(1)(5x+1)^2 + 4.833333(-1)(6x+1)^2,$$

$$d(x) = -0.666667x^2 + 5.333333x + b.$$

Thus the nonlinear (quadratic) decision function and consequently the indicator function in the two particular cases are equal.

Example: *Nonlinear classifier for an Exclusive-Or (XOR) problem*

In the *next example* shown by Figs 12 and 13 we present all the important mathematical objects of a nonlinear SV classifier by using a classic XOR (*exclusive-or*) problem. The graphs show all the mathematical functions (objects) involved in a nonlinear classification. Namely, the nonlinear decision function $d(\mathbf{x})$, the NL indicator function $i_F(\mathbf{x})$, training data ($\mathbf{x}_i$), support vectors $(\mathbf{x}_{SV})_i$ and separation boundaries.

The same objects will be created in the cases when the input vector $\mathbf{x}$ is of a dimensionality $n > 2$, but the visualization in these cases is not possible. In such cases one talks about the decision hyperfunction (hypersurface) $d(\mathbf{x})$, indicator hyperfunction (hypersurface) $i_F(\mathbf{x})$, training data ($\mathbf{x}_i$), support vectors $(\mathbf{x}_{SV})_i$ and separation hyperboundaries (hypersurfaces).

Note the different character of a $d(\mathbf{x})$, $i_F(\mathbf{x})$ and separation boundaries in the two graphs given below. However, in both graphs all the data are correctly classified. Fig. 12 shows the resulting functions for the Gaussian kernel functions, while Fig. 13 presents the solution for a complete second order polynomial kernel. Below, we present the analytical derivation of the (saddle like) decision function in the later (polynomial kernel) case.

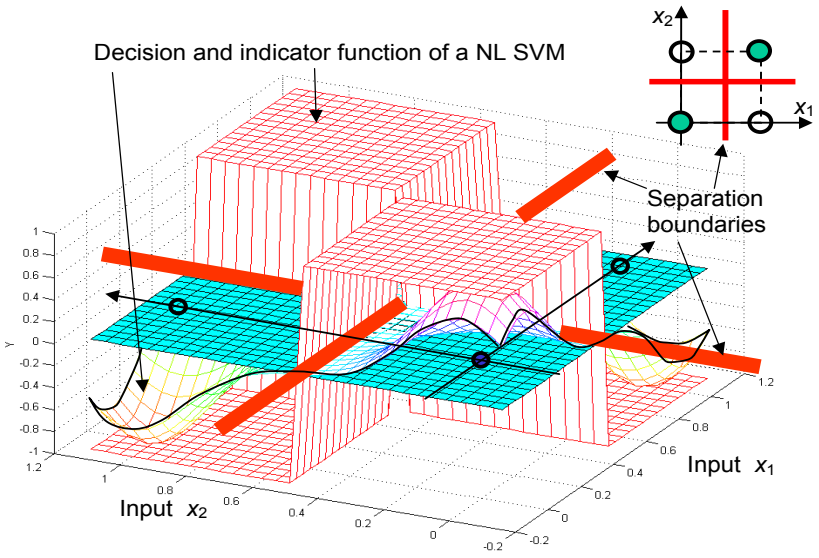

Figure 12. XOR problem. Kernel functions are the *2-D Gaussians* and they are not shown here. The nonlinear decision function, the nonlinear indicator function and the separation boundaries are shown. All four data are chosen as support vectors.

The analytic solution to the Fig. 13 for the *second order polynomial kernel* (i.e., for $(\mathbf{x}_i^T\mathbf{x}_j+1)^2=\Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}_j)$, where

$$\Phi(\mathbf{x})=[1\sqrt{2}\,x_1 \quad \sqrt{2}\,x_2 \quad \sqrt{2}\,x_1x_2 \quad x_1^2 \quad x_2^2],$$

no explicit bias and $C=\infty$) goes as follows. Inputs and desired outputs are,

$$\mathbf{x}=\begin{bmatrix}0 & 1 & 1 & 0\\ 0 & 1 & 0 & 1\end{bmatrix}^T, \qquad \mathbf{y}=\mathbf{d}=\begin{bmatrix}1 & 1 & -1 & -1\end{bmatrix}^T.$$

The dual Lagrangian (37) has the Hessian matrix

$$\mathbf{H}=\begin{bmatrix} 1 & 1 & -1 & -1\\ 1 & 9 & -4 & -4\\ -1 & -4 & 4 & 1\\ -1 & -4 & 1 & 4\end{bmatrix}.$$

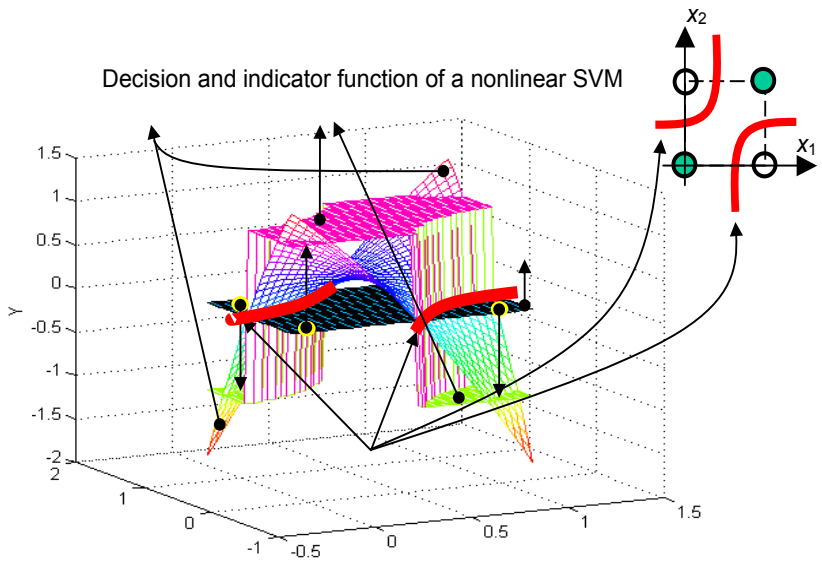

Figure 13. XOR problem. Kernel function is a *2-D polinomial*. The nonlinear decision function, the nonlinear indicator function and the separation boundaries are shown. All four data are support vectors.

The optimal solution can be obtained by taking the derivative of L_d with respect to dual variables $\alpha_i (i = 1, 4)$ and by solving the resulting linear system of equations taking into account the constraints. The solution to

$$\begin{array}{rcrcrcrcl} \alpha_1 & + & \alpha_2 & - & \alpha_3 & - & \alpha_4 & = & 1, \\ \alpha_1 & + & 9\alpha_2 & - & 4\alpha_3 & - & 4\alpha_4 & = & 1, \\ -\alpha_1 & - & 4\alpha_2 & + & 4\alpha_3 & + & \alpha_4 & = & 1, \\ -\alpha_1 & - & 4\alpha_2 & + & \alpha_3 & + & 4\alpha_4 & = & 1, \end{array}$$

subject to $\alpha_i > 0$, $(i = 1, 4)$, is $\alpha_1 = 4.3333$, $\alpha_2 = 2.0000$, $\alpha_3 = 2.6667$ and

$\alpha_4 = 2.6667$. The decision function in a 3-D space is

$$d(\mathbf{x}) = \sum_{i=1}^{4} y_i \alpha_i \Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}) =$$
$$= \big(4.3333\begin{bmatrix}1 & 0 & 0 & 0 & 0 & 0\end{bmatrix} + 2\begin{bmatrix}1 & \sqrt{2} & \sqrt{2} & \sqrt{2} & 1 & 1\end{bmatrix} -$$
$$-2.6667\begin{bmatrix}1 & \sqrt{2} & 0 & 0 & 1 & 0\end{bmatrix} - 2.6667\begin{bmatrix}1 & 0 & \sqrt{2} & 0 & 0 & 1\end{bmatrix}\big)\Phi(\mathbf{x}) =$$
$$= \begin{bmatrix}1 & -0.9429 & -0.9429 & 2.8284 & -0.6667 & -0.6667\end{bmatrix} \times$$
$$\times \begin{bmatrix}1 & \sqrt{2}\,x_1 & \sqrt{2}\,x_2 & \sqrt{2}\,x_1x_2 & x_1^2 & x_2^2\end{bmatrix}^T,$$

and finally

$$d(\mathbf{x}) = 1 - 1.3335x_1 - 1.3335x_2 + 4x_1x_2 - 0.6667x_1^2 - 0.6667x_2^2.$$

It is easy to check that the values of $d(\mathbf{x})$ for all the training inputs in $\mathbf{x}$ equal the desired values in $\mathbf{d}$. The $d(\mathbf{x})$ is the saddle-like function shown in Fig. 13.

Here we have shown the derivation of an expression for $d(\mathbf{x})$ by using explicitly a mapping Φ. Again, we do not have to know what mapping Φ is at all. By using *kernels in input space*, we calculate a *scalar product* required in a *(possibly high dimensional) feature space* and we avoid mapping $\Phi(\mathbf{x})$. This is known as kernel "trick". It can also be useful to remember that the way in which the kernel "trick" was applied in designing an SVM can be utilized in all other algorithms that depend on the scalar product (e.g., in principal component analysis or in the nearest neighbor procedure).

Regression by Support Vector Machines

In the regression, we estimate the functional dependence of the dependent (output) variable $y \in \Re$ on an n-dimensional input variable $\mathbf{x}$. Thus, unlike in pattern recognition problems (where the desired outputs y_i are discrete values e.g., Boolean) we deal with *real valued* functions and we model an $\Re^n$ to $\Re^1$ mapping here. Same as in the case of classification, this will be achieved by training the SVM model on a training data set first. Interestingly and importantly, a learning stage will end in the same shape of a dual Lagrangian as in classification, only difference being in a dimensionalities of the Hessian matrix and corresponding vectors which are of a double size now e.g., $\mathbf{H}$ is a $(2l, 2l)$ matrix.

Initially developed for solving classification problems, SV techniques can be successfully applied in regression, i.e., for a functional approximation problems (*Drucker* et al, (1997), *Vapnik* et al, (1997)). The general regression learning problem is set as follows — the learning machine is given l training data from which it attempts to learn the input-output relationship (dependency, mapping or function) $f(\mathbf{x})$. A training data set

$$D = \{[\mathbf{x}(i), y(i)] \in \Re^n \times \Re, \quad i = 1, \ldots, l\}$$

consists of l pairs $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_l, y_l)$, where the inputs $\mathbf{x}$ are n-dimensional vectors $\mathbf{x} \in \Re^n$ and system responses $y \in \Re$, are continuous values.

We introduce all the relevant and necessary concepts of SVMs' regression in a gentle way starting again with a *linear regression hyperplane* $f(\mathbf{x}, \mathbf{w})$ given as

$$f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x} + b. \tag{40}$$

In the case of SVM's regression, we measure the *error of approximation* instead of the margin used in classification. The most important difference in respect to classic regression is that we use a novel loss (error) functions here. This is the Vapnik's *linear loss function* with ε-insensitivity zone defined as

$$E(\mathbf{x}, y, f) = |y - f(\mathbf{x}, \mathbf{w})|_\varepsilon = \begin{cases} 0, & \text{if } |y - f(\mathbf{x}, \mathbf{w})| \leqslant \varepsilon, \\ |y - f(\mathbf{x}, \mathbf{w})| - \varepsilon, & \text{otherwise,} \end{cases} \tag{41a}$$

or as,

$$e(\mathbf{x}, y, f) = \max(0, |y - f(\mathbf{x}, \mathbf{w})| - \varepsilon). \tag{41b}$$

Thus, the loss is equal to 0 if the difference between the predicted $f(\mathbf{x}_i, \mathbf{w})$ and the measured value y_i is less than ε. Vapnik's ε-insensitivity loss function (41) defines an ε tube (Fig. 15). If the predicted value is within the tube the loss (error or cost) is zero. For all other predicted points outside the tube, the loss equals the magnitude of the difference between the predicted value and the radius ε of the tube.

The two classic error functions are: a square error, i.e., L_2 norm $(y - f)^2$, as well as an absolute error, i.e., L_1 norm, least modulus $|y - f|$ introduced by Yugoslav scientist *Rudjer Boskovic* in 1755 (*Eisenhart*, 1962). The latter error function is related to Huber's error function. An application of Huber's

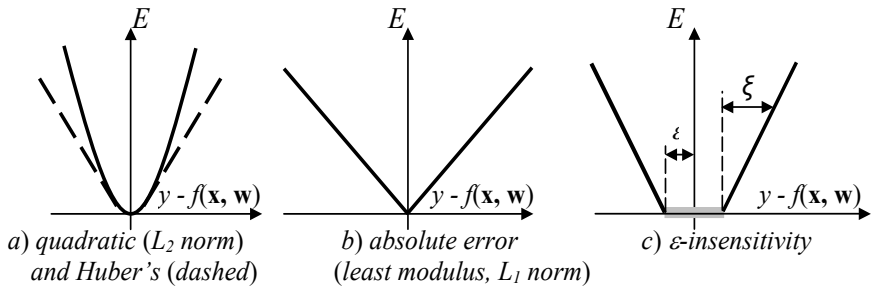

Figure 14. Loss (error) functions

error function results in a *robust regression*. It is the most reliable technique if nothing specific is known about the model of a noise. We do no present Huber's loss function here in analytic form. Instead, we show it by a dashed curve in Fig. 14a. In addition, Fig. 14 shows typical shapes of all mentioned error (loss) functions above.

Note that for $\varepsilon = 0$, Vapnik's loss function equals a least modulus function. Typical graph of a (nonlinear) regression problem as well as all relevant mathematical variables and objects required in, or resulted from, a learning unknown coefficients w_i are shown in Fig. 15.

We will formulate an SVM regression's algorithm for the linear case first and then, for the sake of a NL model design, we will apply mapping to a feature space, utilize the kernel "trick" and construct a nonlinear regression hypersurface. This is actually the same order of presentation as in classification tasks. Here, for the regression, we 'measure' the empirical error term R_{emp} by Vapnik's ε-insensitivity loss function given by (41) and shown in Fig. 14c (while the minimization of the confidence term Ω will be realized through a minimization of $\mathbf{w}^T\mathbf{w}$ again). The empirical risk is given as

$$R^{\varepsilon}_{emp}(\mathbf{w}, b) = \frac{1}{l}\sum_{i=1}^{l}\left|y_i - \mathbf{w}^T\mathbf{x}_i - b\right|_{\varepsilon}, \tag{42}$$

Fig. 16 shows two linear approximating functions as dashed lines inside an ε-tube having the same empirical risk R^{ε}_{emp} as the regression function $f(\mathbf{x}, \mathbf{w})$ on the training data.

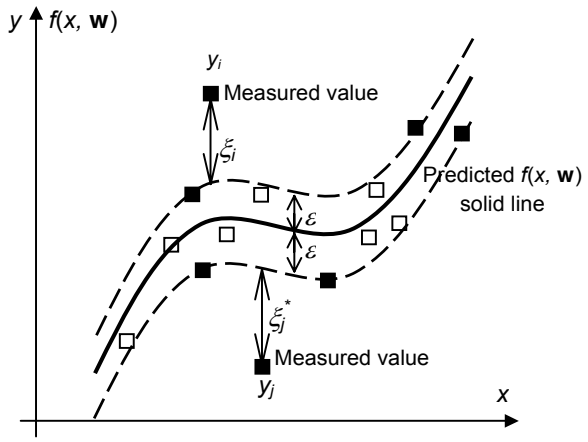

Figure 15. The parameters used in (1-D) support vector regression. Filled squares data ■ are support vectors, and the empty ones □ are not. Hence, SVs can appear only on the tube boundary or outside the tube.

As in classification, we try to minimize both the empirical risk R^{ε}_{emp} and $||\mathbf{w}||^2$ simultaneously. Thus, we construct a linear regression hyperplane

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

by minimizing

$$R = \frac{1}{2} ||\mathbf{w}||^2 + C \sum_{i=1}^{l} \left| y_i - f(\mathbf{x}_i, \mathbf{w}) \right|_{\varepsilon}, \tag{43}$$

Note that the last expression resembles the ridge regression scheme. However, we use Vapnik's ε-insensitivity loss function instead of a squared error now. From (41) and Fig. 15 it follows that for all training data outside an ε-tube,

$|y_i - f(\mathbf{x}, \mathbf{w})| - \varepsilon = \xi$ for data "above" an ε-tube, or
$|y_i - f(\mathbf{x}, \mathbf{w})| - \varepsilon = \xi^*$ for data "below" an ε-tube .

Thus, minimizing the risk R above equals the minimization of the following

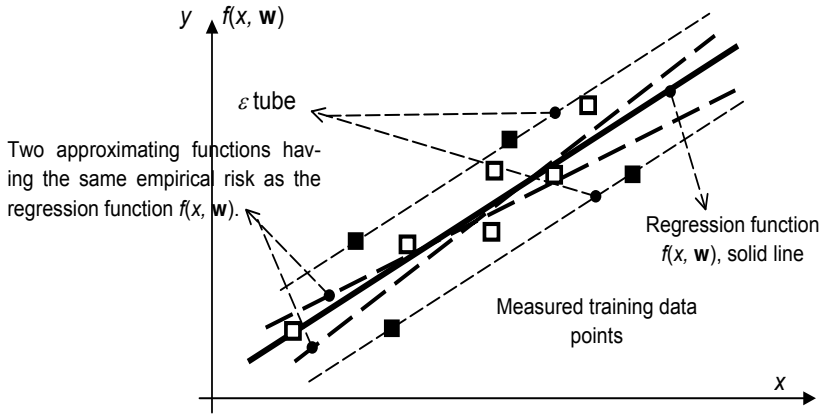

Figure 16. Two linear approximations inside an ε tube (*dashed lines*) have the same empirical risk R^{ε}_{emp} on the training data as the regression function (*solid line*).

risk

$$R_{\mathbf{w},\xi,\xi^*} = \Big[\frac{1}{2}\,||\mathbf{w}||^2 + C\Big(\sum_{i=1}^{l}\xi_i + \sum_{i=1}^{l}\xi_i^*\Big)\Big], \tag{44}$$

under constraints

$$y_i - \mathbf{w}^T\mathbf{x}_i - b \leqslant \varepsilon + \xi_i, \quad i = 1, l, \tag{45a}$$

$$\mathbf{w}^T\mathbf{x}_i + b - y_i \leqslant \varepsilon + \xi_i^*, \quad i = 1, l, \tag{45b}$$

$$\xi_i \geqslant 0, \quad \xi_i^* \geqslant 0, \quad i = 1, l, \tag{45c}$$

where ξ_i and ξ_i^* are slack variables shown in Fig. 15 for measurements "above" and "below" an ε-tube respectively. Both slack variables are positive values. Lagrange multipliers α_i and α_i^* (that will be introduced during the minimization below) related to the first two sets of inequalities above, will be nonzero values for training points "above" and "below" an ε-tube respectively. Because no training data can be on both sides of the tube, either α_i or α_i^* will be nonzero. For data points inside the tube, both multipliers will be equal to zero. Thus $\alpha_i\alpha_i^* = 0$.

Note also that the constant C that influences a trade-off between an approximation error and the weight vector norm $||\mathbf{w}||$ is a design parameter that is chosen by the user. An increase in C penalizes larger errors i.e., it forces ξ_i and ξ_i^* to be small. This leads to an approximation error decrease which is achieved only by increasing the weight vector norm $||\mathbf{w}||$. However, an increase in $||\mathbf{w}||$ increases the confidence term Ω and does not guarantee a small generalization performance of a model. Another design parameter which is chosen by the user is the required precision embodied in an ε value that defines the size of an ε-tube. The choice of ε value is easier than the choice of C and it is given as either maximally allowed or some given or desired percentage of the output values y_i (say, $\varepsilon = 0.1$ of the mean value of $\mathbf{y}$).

Similar to procedures applied in the SV classifiers' design, we solve the constrained optimization problem above by forming a *primal variables Lagrangian* as follows,

$$\begin{aligned} &L_p(\mathbf{w}, b, \xi_i, \xi_i^*, \alpha_i, \alpha_i^*, \beta_i, \beta_i^*) = \\ &= \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}\big(\xi_i + \xi_i^*\big) - \sum_{i=1}^{l}\big(\beta_i^*\xi_i^* + \beta_i\xi_i\big) - \\ &\quad - \sum_{i=1}^{l}\alpha_i\big[\mathbf{w}^T\mathbf{x}_i + b - y_i + \varepsilon + \xi_i\big] - \\ &\quad - \sum_{i=1}^{l}\alpha_i^*\big[\mathbf{w}^T\mathbf{x}_i + b - y_i + \varepsilon + \xi_i\big]. \end{aligned} \tag{46}$$

A primal variables Lagrangian $L_p(\mathbf{w}, b, \xi_i, \xi_i^*, \alpha_i, \alpha_i^*, \beta_i, \beta_i^*)$ has to be *minimized* with respect to primal variables $\mathbf{w}$, b, ξ_i and ξ_i^* and *maximized* with respect to nonnegative Lagrange multipliers α_i, α_i^*, β_i and β_i^*. Hence, the function has the saddle point at the optimal solution ($\mathbf{w}_o$, b_o, ξ_{io}, ξ_{io}^*) to the original problem. At the optimal solution the partial derivatives of L_p in respect to primal variables vanishes. Namely,

$$\frac{\partial L_p(\mathbf{w}_o, b_o, \xi_{io}, \xi_{io}^*, \alpha_i, \alpha_i^*, \beta_i, \beta_i^*)}{\partial \mathbf{w}} = \mathbf{w}_o - \sum_{i=1}^{l}\big(\alpha_i - \alpha_i^*\big)\mathbf{x}_i = 0, \tag{47}$$

$$\frac{\partial L_p(\mathbf{w}_o, b_o, \xi_{io}, \xi_{io}^*, \alpha_i, \alpha_i^*, \beta_i, \beta_i^*)}{\partial b} = \sum_{i=1}^{l}\big(\alpha_i - \alpha_i^*\big) = 0, \tag{48}$$

$$\frac{\partial L_p(\mathbf{w}_o, b_o, \xi_{io}, \xi^*_{io}, \alpha_i, \alpha^*_i, \beta_i, \beta^*_i)}{\partial \xi_i} = C - \alpha_i, \beta_i = 0, \tag{49}$$

$$\frac{\partial L_p(\mathbf{w}_o, b_o, \xi_{io}, \xi^*_{io}, \alpha_i, \alpha^*_i, \beta_i, \beta^*_i)}{\partial \xi^*_i} = C - \alpha^*_i - \beta^*_i = 0, \tag{50}$$

Substituting the KKT above into the primal L_p given in (46), we arrive at the problem of the *maximization of a dual variables Lagrangian* $L_d(\alpha, \alpha^*)$ below,

$$\begin{aligned} L_d(\alpha, \alpha^*) = &-\frac{1}{2}\sum_{i,j=1}^{l}(\alpha_i - \alpha^*_i)(\alpha_j - \alpha^*_j)\mathbf{x}_i^T\mathbf{x}_j - \\ &- \varepsilon\sum_{i=1}^{l}(\alpha_i + \alpha^*_i) + \sum_{i,j=1}^{l}(\alpha_i - \alpha^*_i)y_i = \\ = &-\frac{1}{2}\sum_{i,j=1}^{l}(\alpha_i - \alpha^*_i)(\alpha_j - \alpha^*_j)\mathbf{x}_i^T\mathbf{x}_j - \\ &- \sum_{i=1}^{l}(\varepsilon - y_i)\alpha_i - \sum_{i=1}^{l}(\varepsilon + y_i)\alpha^*_i \end{aligned} \tag{51}$$

subject to constraints

$$\sum_{i=1}^{l}\alpha^*_i = \sum_{i=1}^{l}\alpha_i \quad \text{or} \quad \sum_{i=1}^{l}(\alpha_i - \alpha^*_i) = 0, \tag{52a}$$

$$0 \leqslant \alpha_i \leqslant C, \quad i = 1, l\,, \tag{52b}$$

$$0 \leqslant \alpha^*_i \leqslant C, \quad i = 1, l\,, \tag{52c}$$

Note that the dual variables Lagrangian $L_d(\boldsymbol{\alpha}, \boldsymbol{\alpha}^*)$ is expressed in terms of Lagrange multipliers α_i and α^*_i only. However, the size of the problem, with respect to the size of an SV classifier design task, is doubled now. There are $2l$ unknown dual variables (l $\alpha_i - s$ and l $\alpha^*_i - s$) for a linear regression and the Hessian matrix $\mathbf{H}$ of the quadratic optimization problem in the case of regression is a $(2l, 2l)$ matrix. The *standard quadratic optimization problem* above can be expressed in a *matrix notation* and formulated as follows:

$$\text{minimize} \quad L_d(\boldsymbol{\alpha}) = 0.5\,\boldsymbol{\alpha}^T\mathbf{H}\boldsymbol{\alpha} + \mathbf{f}^T\boldsymbol{\alpha}, \tag{53}$$

subject to (52) where

$$\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_l, \alpha_1^*, \alpha_2^*, \ldots, \alpha_l^*]^T,$$

$$\mathbf{H} = \begin{bmatrix} \mathbf{G} & -\mathbf{G} \\ -\mathbf{G} & \mathbf{G} \end{bmatrix},$$

$\mathbf{G}$ is an (l, l) matrix with entries $G_{ij} = [\mathbf{x}_i^T \mathbf{x}_j]$ for a linear regression [8], and

$$\mathbf{f} = [\varepsilon - y_1, \varepsilon - y_2, \ldots, \varepsilon - y_l, \varepsilon + y_1, \varepsilon + y_2, \ldots, \varepsilon + y_l]^T.$$

Again, (53) is written in a form of some standard optimization routine that typically *minimizes* given objective function subject to same constraints (52).

The learning stage results in l Lagrange multiplier *pairs* (α_i, α_i^*). After the learning, the number of nonzero parameters α_i or α_i^* is equal to the number of SVs. However, this number does not depend on the dimensionality of input space and this is particularly important when working in very high dimensional spaces. Because at least one element of each pair $(\alpha_i, \alpha_i^*), \quad i = 1, l$, is zero, the product of α_i and α_i^* is always zero, i.e., $\alpha_i \alpha_i^* = 0$.

At the optimal solution the following *KKT complementarity conditions* must be fulfilled

$$\alpha_i \big(\mathbf{w}^T \mathbf{x}_i + b - y_i + \varepsilon + \xi_i\big) = 0, \tag{54}$$

$$\alpha_i^* \big(-\mathbf{w}^T \mathbf{x}_i - b + y_i + \varepsilon + \xi_i^*\big) = 0, \tag{55}$$

$$\beta_i \xi_i = (C - \alpha_i)\xi_i = 0, \tag{56}$$

$$\beta_i^* \xi_i^* = (C - \alpha_i^*)\xi_i^* = 0, \tag{57}$$

(56) states that for $0 < \alpha_i < C, \quad \xi_i = 0$ holds. Similarly, from (57) follows that for $0 < \alpha_i^* < C, \quad \xi_i^* = 0$ and, for $0 < \alpha_i < C, \quad \alpha_i^* < C$, from (54) and (55) follows,

$$\mathbf{w}^T \mathbf{x}_i + b - y_i + \varepsilon = 0, \tag{58}$$

$$-\mathbf{w}^T \mathbf{x}_i - b + y_i + \varepsilon = 0. \tag{59}$$

Thus, for all the data points fulfilling $y - f(\mathbf{x}) = +\varepsilon$, dual variables α_i must be between 0 and C, or $0 < \alpha_i < C$, and for the ones satisfying $y - f(\mathbf{x}) = -\varepsilon$, α_i^* take on values $0 < \alpha_i^* < C$. These data points are called the *free* (or

[8] Note that G_{ij}, as given above, is a badly conditioned matrix and we rather use $G_{ij} = [\mathbf{x}_i^T \mathbf{x}_j + 1]$ instead.

unbounded) support vectors. They allow computing the value of the bias term b as given below

$$b = y_i - \mathbf{w}^T\mathbf{x}_i - \varepsilon = 0, \quad \text{for } 0 < \alpha_i < C, \tag{60a}$$

$$b = y_i - \mathbf{w}^T\mathbf{x}_i + \varepsilon = 0, \quad \text{for } 0 < \alpha_i^* < C. \tag{60b}$$

The calculation of a bias term b is numerically very sensitive, and it is better to compute the bias b by averaging over all the *free* support vector data points.

The final observation follows from (56) and (57) and it tells that for all the data points outside the ε-tube, i.e., when both $\xi_i > 0$ and $\xi_i^* > 0$, both α_i and α_i^* equal C, i.e., $\alpha_i = C$ for the points above the tube and $\alpha_i^* = C$ for the points below it. These data are the so-called *bounded support vectors.* Also, for all the training data points within the tube, or when $|y - f(\mathbf{x})| < \varepsilon$, both α_i and α_i^* equal zero and they are neither the support vectors nor do they construct the decision function $f(\mathbf{x})$.

After calculation of Lagrange multipliers α_i and α_i^*, using (47) we can find an *optimal* (desired) weight vector of the *regression hyperplane* as

$$\mathbf{w}_o = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)\mathbf{x}_i. \tag{61}$$

The best regression hyperplane obtained is given by

$$f(\mathbf{x}, \mathbf{w}) = \mathbf{w}_o^T\mathbf{x} + b = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)\mathbf{x}_i^T\mathbf{x} + b. \tag{62}$$

More interesting, more common and the most challenging problem is to aim at solving the *nonlinear regression tasks*. A generalization to nonlinear regression is performed in the same way the nonlinear classifier is developed from the linear one, i.e., by carrying the mapping to the feature space, or by using kernel functions instead of performing the complete mapping which is usually of extremely high dimension (possibly even of an infinite dimension as it is the case with a Gaussian kernel function). Thus, the nonlinear regression function in an input space will be devised by considering a linear regression hyperplane in the *feature space*.

We use the same basic idea in designing SV machines for creating a *nonlinear regression function*. First, a mapping of input vectors $\mathbf{x} \in \Re^n$ into vectors $\Phi(\mathbf{x})$ of a higher dimensional *feature space* F (where Φ represents mapping:

$\Re^n \rightarrow \Re^f$) takes place and then, we solve a linear regression problem in this feature space. A mapping $\Phi(\mathbf{x})$ is again the chosen in advance, or fixed, function. Note that an input space ($\mathbf{x}$-space) is spanned by components x_i of an input vector $\mathbf{x}$ and a feature space F (Φ-space) is spanned by components $\phi_i(\mathbf{x})$ of a vector $\Phi(\mathbf{x})$. By performing such a mapping, we hope that in a Φ-space, our learning algorithm will be able to perform a linear regression hyperplane by applying the linear regression SVM formulation presented above. We also expect this approach to again lead to solving a quadratic optimization problem with inequality constraints in the feature space. The (linear in a feature space F) solution for the regression hyperplane $f = \mathbf{w}^T\Phi(\mathbf{x}) + b$, will create a nonlinear regressing hypersurface in the original input space. The most popular kernel functions are *polynomials* and *RBF* with *Gaussian kernels.* Both kernels are given in Table 2.

In the case of the *nonlinear regression*, the learning problem is again formulated as the maximization of a dual Lagrangian (53) with the Hessian matrix $\mathbf{H}$ structured in the same way as in a linear case, i.e. $\mathbf{H} = [\mathbf{G} \;\; -\mathbf{G}; -\mathbf{G} \;\; \mathbf{G}]$ but with the changed Grammian matrix $\mathbf{G}$ that is now given as

$$\mathbf{G} = \begin{bmatrix} G_{11} & \cdots & G_{1l} \\ \vdots & G_{ii} & \vdots \\ G_{l1} & \cdots & G_{ll} \end{bmatrix} \tag{63}$$

where the entries $G_{ij} = \Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}_j) = K(\mathbf{x}_i, \mathbf{x}_j),\ i, j = 1, l.$

After calculating Lagrange multiplier vectors $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}^*$, we can find an optimal weighting vector of the *kernels expansion* as

$$\mathbf{v}_o = \boldsymbol{\alpha} - \boldsymbol{\alpha}^*. \tag{64}$$

Note however the difference in respect to the linear regression where the expansion of a regression function is expressed by using the optimal weight vector $\mathbf{w}_o$. Here, in a NL SVMs' regression, the optimal weight vector $\mathbf{w}_o$ could be of infinite dimension (which is the case if the Gaussian kernel is used). Consequently, we neither calculate $\mathbf{w}_o$ nor we have to express it in a closed form at all. Instead, we create the best nonlinear regression function by using the weighting vector $\mathbf{v}_o$ and the kernel (Grammian) matrix $\mathbf{G}$ as follows,

$$f(\mathbf{x}, \mathbf{w}) = \mathbf{G}\mathbf{v}_o + b. \tag{65}$$

In fact, the last result follows from the very setting of *the learning (optimizing) stage in a feature space* where, in all the equations above from (45) to (62),

we replace $\mathbf{x}_i$ by the corresponding feature vector $\Phi(\mathbf{x}_i)$. This leads to the following changes:

- instead $G_{ij} = \mathbf{x}_i^T \mathbf{x}_j$ we get $G_{ij} = \Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}_j)$ and, by using the kernel function $K(\mathbf{x}_i, \mathbf{x}_j) = \Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}_j)$, it follows that $G_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$;
- similarly, (61) and (62) change as follows:

$$\mathbf{w}_o = \sum_{i=1}^{l} \big(\alpha_i - \alpha_i^*\big)\Phi(\mathbf{x}_i), \tag{66}$$

and,

$$\begin{aligned} f(\mathbf{x}, \mathbf{w}) = \mathbf{w}_o^T \Phi(\mathbf{x}) + b = \sum_{i=1}^{l} \big(\alpha_i - \alpha_i^*\big)\Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}) + b = \\ = \sum_{i=1}^{l} (\alpha_i^* - \alpha_i) K(\mathbf{x}_i \mathbf{x} + b. \end{aligned} \tag{67}$$

If the bias term b is explicitly used as in (65) then, for a NL SVMs' regression, it can be calculated from the upper SVs as,

$$\begin{aligned} b = y_i - \sum\nolimits_{j=1}^{N\ free\ upper\ SVs} \big(\alpha_i - \alpha_i^*\big)\Phi^T(\mathbf{x}_j)\Phi(\mathbf{x}_i) - \varepsilon = \\ = y_i - \sum\nolimits_{j=1}^{N\ free\ upper\ SVs} \big(\alpha_i - \alpha_i^*\big) K(\mathbf{x}_i, \mathbf{x}_j) - \varepsilon, \\ \text{for } \ 0 < \alpha_i < C, \end{aligned} \tag{68a}$$

or from the lower ones as,

$$\begin{aligned} b = y_i - \sum\nolimits_{j=1}^{N\ free\ lower\ SVs} \big(\alpha_i - \alpha_i^*\big)\Phi^T(\mathbf{x}_j)\Phi(\mathbf{x}_i) + \varepsilon = \\ = y_i - \sum\nolimits_{j=1}^{N\ free\ lower\ SVs} \big(\alpha_i - \alpha_i^*\big) K(\mathbf{x}_i, \mathbf{x}_j) + \varepsilon, \\ \text{for } \ 0 < \alpha_i^* < C, \end{aligned} \tag{68b}$$

Note that $\alpha_j^* = 0$ in (68a) and so is $\alpha_j = 0$ in (68b). Again, it is much better to calculate the bias term b by an averaging *over all* the *free* support vector data points.

There are a few learning parameters in constructing SV machines for regression. The three most relevant are the insensitivity zone ε, the penalty parameter

C (that determines the trade-off between the training error and VC dimension of the model), and the shape parameters of the kernel function (variances of a Gaussian kernel, order of the polynomial, or the shape parameters of the inverse multiquadrics kernel function). All three parameters' sets should be selected by the user. To this end, the most popular selection method is a cross-validation. Unlike in a classification, for not too noisy data (primarily without huge outliers), the penalty parameter C could be set to infinity and the modeling can be controlled by changing the insensitivity zone ε and shape parameters only.

The *example* below shows how an increase in an insensitivity zone ε has smoothing effects on modeling highly noise polluted data. Increase in ε means a reduction in requirements on the accuracy of an approximation. It decreases the number of SVs leading to higher data compression too. This can be readily followed in the lines and Fig. 17 below.

Example: *Nonlinear regression by SVMs*

The task here is to construct an SV machine for modeling measured data pairs. The underlying function (known to us but, not to the SVM) is a sinus function multiplied by the square one (i.e., $f(x) = x^2 \sin x$) and it is corrupted by 25% of normally distributed noise with a zero mean. Analyze the influence of an insensitivity zone ε on modeling quality and on a compression of data, meaning on the number of SVs.

Fig. 17 shows that for a very noisy data a decrease of an insensitivity zone ε (i.e., shrinking of the tube shown by dashed line) approximates the noisy data points more closely. The related more and more wiggly shape of the regression function can be achieved only by including more and more support vectors. However, being good on the noisy training data points easily leads to an overfitting. The cross-validation should help in finding correct ε value, resulting in a regression function that filters the noise out but not the true dependency and which, consequently, approximate the underlying function as close as possible.

The approximation functions shown in Fig. 17 are created by 9 and 18 weighted Gaussian basis functions for $\varepsilon = 1$ and $\varepsilon = 0.75$ respectively. These supporting functions are not shown in the figure. However, the way how the learning algorithm selects SVs is an interesting property of support vector machines and in Fig. 18 we also present the supporting Gaussian functions.

Note that the selected Gaussians lie in the dynamic area of the function in Fig. 18. Here, these areas are close to both the left hand and the right hand boundary. In the middle, the original function is pretty flat and there is no need to cover this part by supporting Gaussians. The learning algorithm realizes this

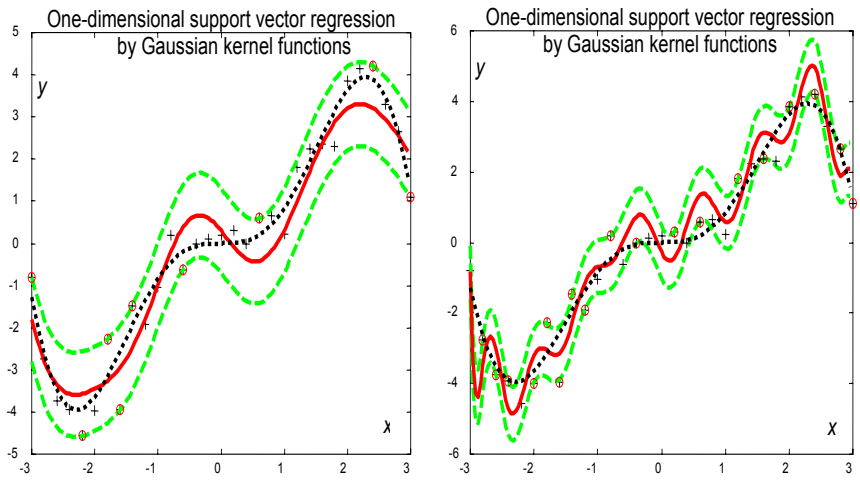

Figure 17. The influence of an insensitivity zone ε on the model performance. A nonlinear SVM creates a regression function f with Gaussian kernels and models a highly polluted (25% noise) function $x^2 sin(x)$ (*dotted*). 31 training data points (*plus signs*) are used. **Left:** $\varepsilon = 1$; 9 SVs are chosen (*encircled plus signs*). **Right:** $\varepsilon = 0.75$; the 18 chosen SVs produced a better approximation to noisy data and, consequently, there is the tendency of overfitting.

fact and simply, it does not select any training data point in this area as a support vector. Note also that the Gaussians are not weighted in Fig 18, and they all have the peak value of 1. The standard deviation of Gaussians is chosen in order to see Gaussian supporting functions better. Here, in Fig. 18, $\sigma = 0.6$. Such a choice is due the fact that for the larger σ values the basis functions are rather broad and flat above the domain shown. Thus, the supporting Gaussian functions are covering the whole domain as the broad umbrellas. For very big variances one wouldn't be able to distinguish them visually.

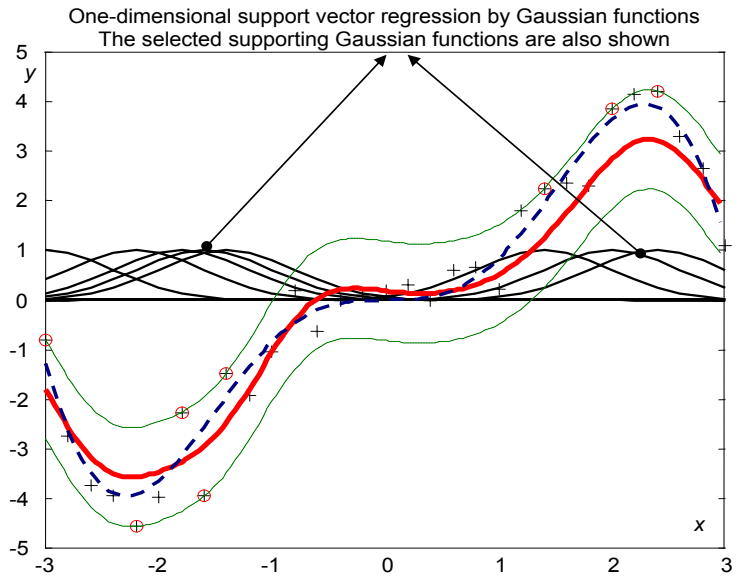

Figure 18. Regression function f created as the sum of 8 weighted Gaussian kernels. A standard deviation of Gaussian bells $\sigma = 0.6$. Original function (*dashed line*) is $x^2 \sin x$ and it is corrupted by 0.25% noise. 31 training data points are shown as plus signs. Data points selected as the SVs are encircled. The 8 selected supporting Gaussian functions are centered at these data points.

Implementation Issues

In both the classification and the regression the learning problem boils down to solving the QP problem subject to the so-called 'box-constraints and to the equality constraint in the case that a model with a bias term b is used. The SV training works almost perfectly for not too large data basis. However, when the number of data points is large (say $l > 2,000$) the QP problem becomes extremely difficult to solve with standard QP solvers and methods. For example, a classification training set of 50,000 examples amounts to a Hessian matrix $\mathbf{H}$ with 2.5×10^9 (2.5 billion) elements. Using an 8-byte floating-point representation we need 20,000 Megabytes = 20 Gigabytes of memory (*Osuna* et al, 1997).

This cannot be easily fit into memory of present standard computers, and this is the single basic disadvantage of the SVM method. There are three approaches that resolve the QP for large data sets. *Vapnik* in (*Vapnik*, 1995) proposed the *chunking method* that is the decomposition approach. Another *decomposition* approach is suggested in (*Osuna* et al, 1997). The sequential minimal optimization (*Platt*, 1997) algorithm is of different character and it seems to be an "error back propagation" for SVM learning. A systematic exposition of these various techniques is not given here, as all three would require a lot of space. However, the interested reader can find a description and discussion about the novel algorithms in (*Kecman, Huang*, and *Vogt*, 2005; *Vogt* and *Kecman*, 2005). The *Vogt* and *Kecman*'s chapter discusses the application of an *active set* algorithm in solving small to medium sized QP problems. For such data sets and when the high precision is required the active set approach in solving QP problems seems to be superior to other approaches (notably the interior point methods and SMO algorithm). The *Kecman, Huang,* and *Vogt*'s chapter introduces the efficient *iterative single data algorithm* (*ISDA*) for solving huge data sets (say more than 100,000 or 500,000 or over 1 million training data pairs). It seems that ISDA is the fastest algorithm at the moment for such large data sets (see the comparisons with SMO in (*Kecman, Huang* and *Vogt*, 2005)) still ensuring the convergence to the global minimum. This means that the ISDA provides the exact, and not the approximate, solution to the original dual problem. In the next section we will introduce the ISDA algorithm. As for now, let us conclude the presentation of the classic SVMs part by summarizing the basic constructive steps that lead to the SV machine.

A training and design of a support vector machine is an *iterative* algorithm and it involves the following steps:

a) define your problem as the classification or as the regression one;
b) preprocess your input data: select the most relevant features, scale the data between $[-1, 1]$, or to the ones having zero mean and variances equal to one, check for possible outliers (strange data points);
c) select the kernel function that determines the hypothesis space of the decision and regression function in the classification and regression problems respectively;
d) select the "shape", i.e., "smoothing" parameter of the kernel function (for example, polynomial degree for polynomials and variances of the Gaussian RBF kernels respectively);
e) choose the penalty factor C and, in the regression, select the desired accuracy by defining the insensitivity zone ε too;

f) solve the QP problem in l and $2l$ variables in the case of classification and regression problems respectively;
g) validate the model obtained on some previously (i.e., during the training) unseen test data, and if not pleased iterate between steps `d` (or, eventually `c`) and `g`.

The optimizing part `(f)` is computationally extremely demanding. First, the Hessian matrix $\mathbf{H}$ scales with the size of a data set — it is an (l, l) and an $(2l, 2l)$ matrix in classification and regression respectively. Second, unlike in classic original QP problems $\mathbf{H}$ is very dense matrix and it is usually badly conditioned requiring regularization before any numeric operation. Regularization means an addition of a small number to the diagonal elements of $\mathbf{H}$. Luckily, there are many reliable and fast QP solvers. A simple internet search will reveal many of them. Particularly, in addition to the classic ones such as MINOS or LOQO for example, there are many more free QP solvers designed specially for the SVMs. The most popular ones are — the LIBSVM, SVMlight, SVM Torch, mySVM and SVM Fu. All of them can be downloaded from their corresponding sites. Good educational software in MATLAB named LEARNSC, with very good graphic presentations of all relevant objects in a SVM modeling, can be downloaded from the author's book site [9] too.

Finally we mention that there are many alternative formulations and approaches to the QP based SVMs described above. Notably, they are the linear programming SVMs (*Mangasarian*, 1965; *Friess* and *Harrison*, 1998; *Smola*, et al, 1998; *Hadzic* and *Kecman*, 1999; *Graepel* et al, 1999; *Kecman* and *Hadzic*, 2000; *Kecman*, 2001; *Kecman, Arthanari, Hadzic*, 2001), υ-SVMs (*Schölkopf* and *Smola*, 2002) and least squares support vector machines (*Suykens* et al, 2002). Their description is far beyond this chapter and the curious readers are referred to references given above.

Below we introduce a novel Iterative Single Data Algorithm (ISDA) for resolving the problems coming from huge Hessian matrices in training SVMs. First, we show the equality of various approaches in learning from data and afterwards we present two variants of ISDA, namely one with the bias term b and the other without it.

The lines below are the shortened versions of two papers presented at the ESANN 2003 and 2004 which can be downloaded (together with few more contributions of the author) from the author's site [10].

[9] URL: www.support-vector.ws
[10] URL: http://www.support-vector.ws/html/publications.html

On the Equality of Kernel AdaTron and Sequential Minimal Optimization and Alike Algorithms for Kernel Machines

This section presents the equality of a kernel AdaTron (KA) method (originating from a gradient ascent learning approach) and sequential minimal optimization (SMO) learning algorithm (based on an analytic quadratic programming step) in designing the support vector machines (SVMs) having *positive definite kernels*. The conditions of the equality of two methods are established. The equality is valid for both the nonlinear classification and the nonlinear regression tasks, and it sheds a new light to these seemingly different learning approaches. The section also introduces other learning techniques related to the two mentioned approaches, such as the nonnegative conjugate gradient, classic Gauss-Seidel (GS) coordinate ascent procedure and its derivative known as the successive over-relaxation (SOR) algorithm as a viable and usually faster training algorithms for performing nonlinear classification and regression tasks. The convergence theorem for these related iterative algorithms is proven. Due to restricted space, the presentation is scarce, giving only the final expressions. More detailed derivations can be found in some papers and book chapter from the site given at the end of the previous section above. In addition, the ISDA software for solving huge SVMs' learning problems can be downloaded from appropriate site [11]. The site is accompanying the *Huang*, *Kecman* and *Kopriva*'s book which is in print at Springer Verlag.

Introduction

One of the mainstream research fields in learning from empirical data by support vector machines, and solving both the classification and the regression problems, is an implementation of the incremental learning schemes when the training data set is huge. Among several candidates that avoid the use of standard quadratic programming (QP) solvers, the two learning approaches which have recently got the attention are the KA (*Anlauf, Biehl*, 1989; *Friess, Cristianini, Campbell*, 1998; *Veropoulos*, 2001) and the SMO (*Platt*, 1998, 1999; *Vogt*, 2002). Due to its analytical foundation the SMO approach is particularly popular and at the moment the widest used, analyzed and still heavily developing algorithm. At the same time, the KA although providing similar results in solving classification problems (in terms of both the accuracy and the training

[11] URL: www.learning-from-data.com

computation time required) did not attract that many devotees. There are two basic reasons for that. First, until recently (*Veropoulos*, 2001) the KA seemed to be restricted to the classification problems only and second, it "lacked" the fleur of the strong theory (despite its beautiful "simplicity" and strong convergence proofs). The KA is based on a gradient ascent technique and this fact might have also distracted some researchers being aware of problems with gradient ascent approaches faced with possibly ill-conditioned kernel matrix.

Here we show when and why the recently developed algorithms for SMO using positive definite kernels or models *without a bias term* (Vogt, 2002), and the KA for both *classification* (*Friess, Cristianini, Campbell*, 1998) and *regression* (*Veropoulos*, 2001) are identical. Both the KA and the SMO algorithm attempt to solve the QP problem in the case of ***classification*** by *maximizing* the dual Lagrangian under the constraints as given in equations (37).

In the case of the ***nonlinear regression*** the learning problem is the maximization of a dual Lagrangian as given in equations (51) and (52). Note that (51) is given for a linear regression hypersurface. For the nonlinear regression the scalar product $\mathbf{x}_i^T \mathbf{x}_j$ must be replaced by the kernel function value $K(\mathbf{x}_i, \mathbf{x}_j)$.

The KA and SMO learning algorithms without-bias-term

It is known that *positive definite kernels* (such as the most popular and the most widely used RBF Gaussian kernels as well as the complete polynomial ones) do not require bias term (*Evgeniou, Pontil, Poggio*, 2000). Below, the KA and the SMO algorithms will be presented for such a fixed (i.e., no-) bias design problem and compared for the classification and regression cases. The equality of two learning schemes and resulting models will be established. Originally, in (*Platt*, 1998, 1999), the SMO *classification* algorithm was developed for solving the problem (1) including the constraints related to the bias b. In these early publications the case when bias b is fixed variable was also mentioned but the detailed analysis of a fixed bias update was not accomplished.

Incremental Learning in Classification

(a) Kernel AdaTron in classification. The classic AdaTron algorithm as given in (*Anlauf* and *Biehl*, 1989) is developed for linear classifier. The KA is a variant of the classic AdaTron algorithm in the feature space of SVMs (*Friess* et al., 1998). The KA algorithm solves the maximization of the dual

Lagrangian (37) by implementing the gradient ascent algorithm. The update $\Delta\alpha_i$ of the dual variables α_i is given as

$$\Delta\alpha_i = \eta\frac{\partial L_d}{\partial\alpha_i} = \eta\Big(1 - y_i\sum_{j=1}^{l}\alpha_j y_j K(\mathbf{x}_i,\mathbf{x}_j)\Big) = \eta\big(1 - y_i f_i\big), \tag{69a}$$

where f_i is the value of the decision function f at the point $\mathbf{x}_i$, i.e.,

$$f_i = \sum_{j=1}^{l}\alpha_j y_j K(\mathbf{x}_i,\mathbf{x}_j),$$

and y_i denotes the value of the desired target (or the class' label) which is either $+1$ or -1. The update of the dual variables α_i is given as

$$\alpha_i \leftarrow \min(\max(0,\alpha_i + \Delta\alpha_i), C), \quad (i = 1,\ldots,l). \tag{69b}$$

In other words, the dual variables α_i are clipped to zero if $(\alpha_i + \Delta\alpha_i) < 0$. In the case of the soft nonlinear classifier $(C < \infty)$, α_i are clipped between zero and C, $(0 \leqslant \alpha_i \leqslant C)$. The algorithm converges from any initial setting for the Lagrange multipliers α_i.

(b) SMO without-bias-term in classification. Recently (*Vogt*, 2002) derived the update rule for multipliers α_i that includes a detailed analysis of the Karush-Kuhn-Tucker (KKT) conditions for checking the optimality of the solution [12]. The following update rule for α_i for a no-bias SMO algorithm was proposed

$$\Delta\alpha_i = -\frac{y_i E_i}{K(\mathbf{x}_i,\mathbf{x}_i)} = -\frac{y_i f_i - 1}{K(\mathbf{x}_i,\mathbf{x}_i)} = \frac{1 - y_i f_i}{K(\mathbf{x}_i,\mathbf{x}_i)}, \tag{70}$$

where $E_i = f_i - y_i$ denotes the difference between the value of the decision function f at the point $\mathbf{x}_i$ and the desired target (label) y_i. Note the equality of (69a) and (70) when the learning rate in (69a) is chosen to be $\eta_i = 1/K(\mathbf{x}_i,\mathbf{x}_i)$. The important part of the SMO algorithm is to check the KKT conditions with precision τ (e.g., $\tau = 10^{-3}$) in each step. An update is performed only if

$$\begin{aligned} &\alpha_i < C \ \wedge\ y_i E_i < -\tau, \quad \text{or} \\ &\alpha_i > 0 \ \wedge\ y_i E_i > \tau. \end{aligned} \tag{70a}$$

[12] As referred above, a fixed bias update was only mentioned in Platt's papers.

After an update, the same clipping operation as in (69b) is performed

$$\alpha_i \leftarrow \min(\max(0, \alpha_i + \Delta\alpha_i), C), \quad (i = 1, \ldots, l). \tag{70b}$$

It is the nonlinear clipping operation in (69b) and in (70b) that strictly equals the KA and the SMO without-bias-term algorithm in solving nonlinear classification problems. This fact sheds new light on both algorithms. This equality is not that obvious in the case of a "classic" SMO algorithm with bias term due to the heuristics involved in the selection of active points which should ensure the largest increase of the dual Lagrangian L_d during the iterative optimization steps.

Incremental Learning in Regression. Similarly to the case of classification, there is a strict equality between the KA and the SMO algorithm when positive definite kernels are used for nonlinear regression.

(a) Kernel AdaTron in regression. The first extension of the Kernel AdaTron algorithm for regression is presented in (*Veropoulos*, 2001) as the following gradient ascent update rules for α_i and α_i^*

$$\begin{aligned} \Delta\alpha_i = \eta_i \frac{\partial L_d}{\partial \alpha_i} = \eta_i \Big(y_i - \varepsilon - \sum_{j=1}^{l} (\alpha_j - \alpha_j^*) K(\mathbf{x}_j, \mathbf{x}_i) \Big) = \\ = \eta_i \big(y_i - \varepsilon - f_i \big) = -\eta_i \big(E_i + \varepsilon \big), \end{aligned} \tag{71a}$$

$$\begin{aligned} \Delta\alpha_i^* = \eta_i \frac{\partial L_d}{\partial \alpha_i^*} = \eta_i \Big(-y_i - \varepsilon + \sum_{j=1}^{l} (\alpha_j - \alpha_j^*) K(\mathbf{x}_j, \mathbf{x}_i) \Big) = \\ = \eta_i \big(y_i - \varepsilon + f_i \big) = \eta_i \big(E_i - \varepsilon \big), \end{aligned} \tag{71b}$$

where y_i is the measured value for the input $\mathbf{x}_i$, ε is the prescribed insensitivity zone, and $E_i = f_i - y_i$ stands for the difference (an error) between the regression function f at the point $\mathbf{x}_i$ and the desired target value y_i at this point. The calculation of the gradient above does not take into account the geometric reality that no training data can be on both sides of the tube. In other words, it does not use the fact that either α_i or α_i^* or both will be nonzero, i.e., that $\alpha_i \alpha_i^* = 0$ must be fulfilled in each iteration step. Below we derive the gradients of the

dual Lagrangian L_d accounting for geometry. This new formulation of the KA algorithm strictly equals the SMO method and it is given as

$$\begin{aligned}\frac{\partial L_d}{\partial \alpha_i^*} &= -K(\mathbf{x}_i,\mathbf{x}_i)\,\alpha_i - \sum_{j=1,j\neq i}^{l}\big(\alpha_j-\alpha_j^*\big)K(\mathbf{x}_j,\mathbf{x}_i) + y_i - \varepsilon + \\ &\quad + K(\mathbf{x}_i,\mathbf{x}_i)\,\alpha_i - K(\mathbf{x}_i,\mathbf{x}_i)\,\alpha_i^* = \\ &= K(\mathbf{x}_i,\mathbf{x}_i)\,\alpha_i^* - \big(\alpha_i-\alpha_i^*\big)K(\mathbf{x}_i,\mathbf{x}_i) - \\ &\quad - \sum_{j=1,j\neq i}^{l}\big(\alpha_j-\alpha_j^*\big)K(\mathbf{x}_j,\mathbf{x}_i) + y_i - \varepsilon = \\ &= -K(\mathbf{x}_i,\mathbf{x}_i)\,\alpha_i^* + y_i - \varepsilon - f_i = -\big(K(\mathbf{x}_i,\mathbf{x}_i)\,\alpha_i^* + E_i + \varepsilon\big).\end{aligned} \tag{72a}$$

For the α_i^* multipliers, the value of the gradient is

$$\frac{\partial L_d}{\partial \alpha_i^*} = -K(\mathbf{x}_i,\mathbf{x}_i)\,\alpha_i + E_i - \varepsilon. \tag{72b}$$

The update value for α_i is now

$$\Delta\alpha_i = \eta_i\frac{\partial L_d}{\partial \alpha_i} = -\eta_i\big(K(\mathbf{x}_i,\mathbf{x}_i)\,\alpha_i^* + E_i + \varepsilon\big), \tag{73a}$$

$$\alpha_i \leftarrow \alpha_i + \Delta\alpha_i = \alpha_i + \eta_i\frac{\partial L_d}{\partial \alpha_i} = \alpha_i - \eta_i\big(K(\mathbf{x}_i,\mathbf{x}_i)\,\alpha_i^* + E_i + \varepsilon\big). \tag{73b}$$

For the learning rate $\eta_i = 1/K(\mathbf{x}_i,\mathbf{x}_i)$ the gradient ascent learning KA is defined as,

$$\alpha_i \leftarrow \alpha_i - \alpha_i^* - \frac{E_i+\varepsilon}{K(\mathbf{x}_i,\mathbf{x}_i)}, \tag{74a}$$

Similarly, the update rule for α_i^* is

$$\alpha_i^* \leftarrow \alpha_i^* - \alpha_i + \frac{E_i-\varepsilon}{K(\mathbf{x}_i,\mathbf{x}_i)}. \tag{74b}$$

Same as in the classification, α_i and α_i^* are clipped between zero and C,

$$\alpha_i \leftarrow \min(\max(0,\alpha_i),C), \quad i=1,\ldots,l, \tag{75a}$$

$$\alpha_i^* \leftarrow \min(\max(0,\alpha_i^*),C), \quad i=1,\ldots,l. \tag{75b}$$

(b) SMO without-bias-term in regression. The first algorithm for the SMO without-bias-term in regression (together with a detailed analysis of the KKT conditions for checking the optimality of the solution) is derived in (*Vogt*, 2002). The following learning rules for the Lagrange multipliers α_i and α_i^* updates were proposed

$$\alpha_i \leftarrow \alpha_i - \alpha_i^* - \frac{E_i + \varepsilon}{K(\mathbf{x}_i, \mathbf{x}_i)}, \tag{76a}$$

$$\alpha_i^* \leftarrow \alpha_i^* - \alpha_i + \frac{E_i - \varepsilon}{K(\mathbf{x}_i, \mathbf{x}_i)}. \tag{76b}$$

The equality of equations (74a, b) and (76a, b) is obvious when the learning rate, as presented above in (74a, b), is chosen to be $\eta_i = 1/K(\mathbf{x}_i, \mathbf{x}_i)$. Thus, in both the classification and the regression, the optimal learning rate is not necessarily equal for all training data pairs. For a Gaussian kernel, $\eta = 1$ is same for all data points, and for a complete n^{th} order polynomial each data point has different learning rate $\eta_i = 1/(\mathbf{x}_i^T \mathbf{x}_i + 1)^n$. Similar to classification, a joint update of α_i and α_i^* is performed only if the KKT conditions are violated by at least τ , i.e. if

$$\begin{aligned} &\alpha_i < C \ \wedge \ \varepsilon + E_i < -\tau, \quad \text{or} \\ &\alpha_i > 0 \ \wedge \ \varepsilon + E_i > \tau, \quad \text{or} \\ &\alpha_i^* < C \ \wedge \ \varepsilon - E_i < -\tau, \quad \text{or} \\ &\alpha_i^* > 0 \ \wedge \ \varepsilon - E_i > \tau. \end{aligned} \tag{77}$$

After the changes, the same clipping operations as defined in (11) are performed

$$\alpha_i \leftarrow \min(\max(0, \alpha_i), C), \quad i = 1, \ldots, l, \tag{78a}$$

$$\alpha_i^* \leftarrow \min(\max(0, \alpha_i^*), C), \quad i = 1, \ldots, l. \tag{78b}$$

The KA learning as formulated in this section and the SMO algorithm without-bias-term for solving regression tasks are strictly equal in terms of both the number of iterations required and the final values of the Lagrange multipliers. The equality is strict despite the fact that the implementation is slightly different. In every iteration step, namely, the KA algorithm updates both weights α_i and α_i^* without any checking whether the KKT conditions are fulfilled or not, while the SMO performs an update according to equations (77).

The Coordinate Ascent Based Learning for Nonlinear Classification and Regression Tasks — The Gauss-Seidel Algorithm

When positive definite kernels are used, the learning problem for both tasks is same. In a vector-matrix notation, in a dual space, the learning is defined as:

maximize

$$L_d(\alpha) = -0.5\alpha^T \mathbf{K}\alpha + \mathbf{f}^T\alpha, \tag{79}$$

such that

$$0 \leqslant \alpha_i \leqslant C, \quad (i = 1, \ldots, n), \tag{80}$$

where, in the classification $n = l$ and the matrix $\mathbf{K}$ is an (l, l) symmetric positive definite matrix, while in regression $n = 2l$ and $\mathbf{K}$ is a $(2l, 2l)$ symmetric semi-positive definite one. Note that the constraints (80) define a convex subspace over which the convex dual Lagrangian should be maximized. It is very well known that the vector $\boldsymbol{\alpha}$ may be looked at as the solution of a system of linear equations

$$\mathbf{K}\boldsymbol{\alpha} = \mathbf{f} \tag{81}$$

subject to the same constraints as given by (80).

Thus, it may seem natural to solve (81), subject to (80), by applying some of the well known and established techniques for solving a general linear system of equations. The size of training data set and the constraints (80) eliminate direct techniques. Hence, one has to resort to the *iterative approaches* in solving the problems above. There are three possible iterative avenues that can be followed. They are; the use of the Non-Negative Least Squares (NNLS) technique (*Lawson* and *Hanson*, 1974), application of the Non-Negative Conjugate Gradient (NNCG) method (*Hestenes*, 1980) and the implementation of Gauss-Seidel (GS) i.e., the related Successive Over-Relaxation technique (SOR). The first two methods, in their "classic" appearance, solve for the non-negative constraints only. Thus, they are not suitable in solving "soft" tasks, when penalty parameter $C < \infty$ is used, i.e., when there is an upper bound on maximal value of α_i. In the case of nonlinear regression, one can apply NNLS and NNCG by taking $C = \infty$ and compensating (i.e. smoothing or "softening" the solution) by increasing the sensitivity zone ε. The two methods (namely NNLS and NNCG) are not suitable for solving soft margin ($C < \infty$) classification problems in their present form, because there is no other parameter that can be used in "softening" the margin. Recently, the NNCG algorithm for solving (81) with box-constraints (80) is developed and presented in (*Huang, Kecman* and

Kopriva, 2006). However, due to the usually bad conditioned Hessian matrix $\mathbf{K}$, NNCG will not be used in the lines below.

Here we show how to extend the application of GS and SOR to both the nonlinear classification and to the nonlinear regression tasks. The Gauss-Seidel method solves (81) by using the i^{th} equation to update the i^{th} unknown doing it iteratively, i.e., starting in the k^{th} step with the first equation to compute the α_1^{k+1}, then the second equation is used to calculate the α_2^{k+1} by using new α_1^{k+1} and $\alpha_i^k \quad (i > 2)$ and so on. The iterative learning takes the following form,

$$\begin{aligned}\alpha_i^{k+1} &= \Big(f_i - \sum_{j=1}^{i-1} K_{ij}\alpha_j^{k+1} - \sum_{j=i+1}^{n} K_{ij}\alpha_j^k\Big)/K_{ii} = \\ &= \alpha_i^k - \frac{1}{K_{ii}}\Big(\sum_{j=1}^{i-1} K_{ij}\alpha_j^{k+1} + \sum_{j=i}^{n} K_{ij}\alpha_j^k - f_i\Big) = \\ &= \alpha_i^k + \frac{1}{K_{ii}}\left.\frac{\partial L_d}{\partial \alpha_i}\right|_{k+1},\end{aligned} \tag{82}$$

where we use the fact that the term within a second bracket (called the residual r_i in mathematics' references) is the i^{th} element of the gradient of a dual Lagrangian L_d given in (79) at the $(k+1)^{th}$ iteration step. The equation (82) above shows that GS method is a *coordinate* gradient ascent procedure as the KA and the SMO are. *The KA and SMO for positive definite kernels equal the GS!* Note that the optimal learning rate used in both the KA algorithm and in the SMO without-bias-term approach is exactly equal to the coefficient $1/K_{ii}$ in a GS method. Based on this equality, the convergence theorem for the KA, SMO and GS (i.e., SOR) in solving (81) subject to constraints (80) can be stated and proved as follows:

Theorem: For SVMs with positive definite kernels (while using them without the bias term b) the iterative learning algorithms KA i.e., SMO i.e., GS i.e., SOR, in solving nonlinear classification and regression tasks (81) subject to constraints (80), converge starting from any initial choice of $\boldsymbol{\alpha}_0$.

Proof: The proof is based on the very well known theorem of convergence of the GS method for symmetric positive definite matrices in solving (81) without constraints (*Ostrowski*, 1966). First note that for positive definite kernels, the matrix $\mathbf{K}$ created by terms $y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$ in the second sum in (1), and involved in solving classification problem, is also positive definite. In regression

tasks $\mathbf{K}$ is a symmetric positive semi-definite (meaning still convex) matrix, which after a mild regularization given as ($\mathbf{K} \leftarrow \mathbf{K} + \lambda\mathbf{I}$, $\lambda \sim 1e-12$) becomes positive definite one. (Note that the proof in the case of regression does not need regularization at all, but there is no space here to go into these details). Hence, the learning without constraints (80) converges, starting from any initial point $\boldsymbol{\alpha}_0$, and each point in an n-dimensional search space for multipliers α_i is a viable starting point ensuring a convergence of the algorithm to the maximum of a dual Lagrangian L_d. This, naturally, includes all the (starting) points within, or on a boundary of, any convex subspace of a search space ensuring the convergence of the algorithm to the maximum of a dual Lagrangian L_d over the given subspace. The constraints imposed by (80) preventing variables α_i to be negative or bigger than C, and implemented by the clipping operators above, define such a convex subspace. Thus, each "clipped" multiplier value α_i defines a new starting point of the algorithm guaranteeing the convergence to the maximum of L_d over the subspace defined by (80). For a convex constraining subspace such a constrained maximum is unique. ***Q.E.D.***

Due to the lack of the space we do not go into the discussion on the convergence rate here and we leave it to some other occasion. It should be only mentioned that both KA and SMO (i.e. GS and SOR) for positive definite kernels have been successfully applied for many problems (see references given here, as well as many other, benchmarking the mentioned methods on various data sets). Finally, let us just mention that the standard extension of the GS method is the method of successive over-relaxation that can reduce the number of iterations required by proper choice of relaxation parameter ω significantly. The SOR method uses the following updating rule

$$\begin{aligned} \alpha_i^{k+1} &= \alpha_i^k - \omega\frac{1}{K_{ii}}\Big(\sum_{j=1}^{i-1} K_{ij}\alpha_j^{k+1} + \sum_{j=i}^{n} K_{ij}\alpha_j^k - f_i\Big) = \\ &= \alpha_i^k + \omega\frac{1}{K_{ii}}\left.\frac{\partial L_d}{\partial \alpha_i}\right|_{k+1}, \end{aligned} \tag{83}$$

and similarly to the KA, SMO, and GS its convergence is guaranteed for $0 < \omega < 2$.

SVMs with a Bias Term b

Now, we discuss and present the use and calculation of the explicit bias term b in the support vector machines within the Iterative Single training Data learning Algorithm. The approach with a bias b can also be used for both nonlinear classification and nonlinear regression tasks. It is well known that for positive definite kernels there is no need for bias b (*Kecman*, 2001). We used this fact while developing ISDA in previous section. However, one can use the bias term b and this means implementing a different kernel. There is a report and a paper where this issue has been discussed. In (*Poggio* et al., 2001)

$$f(\mathbf{x}) = \sum_{j=1}^{l} w_j K(\mathbf{x}, \mathbf{x}_j) + b$$

and it was shown that $f(\mathbf{x})$ is a function resulting from a minimization of the functional shown below

$$I[f] = \sum_{j=1}^{l} V(y_j, f(\mathbf{x}_j)) + \lambda \big|\big| f \big|\big|^2_{K^*}, \tag{84}$$

where $K^* = K - a$ (for an appropriate constant a) and K is an original kernel function (more details can be found in the mentioned report). This means that by adding a constant term to a positive definite kernel function K, one obtains the solution to the functional $I[f]$ where K^* is a conditionally positive definite kernel. Interestingly, similar type of model was also presented in (*Mangasarian* and *Musicant*, 1999). However, their formulation is done for the classification problems only. They reformulated the optimization by adding the $b^2/2$ term to the cost function $||\mathbf{w}||^2/2$. This is equivalent to an addition of 1 to the original kernel matrix $\mathbf{K}$. As a result, they changed the original classification dual problem to the optimization of the following one

$$L_d(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j (K(\mathbf{x}_i, \mathbf{x}_j) + 1). \tag{85}$$

Iterative Single Data Algorithm for SVMs with Bias

In section “On the Equality of Kernel AdaTron and Sequential Minimal Optimization and Alike Algorithms for Kernel Machines” and for the SVMs models

when positive definite kernels are used without a bias term b, the learning algorithms for classification and regression (in a dual domain) were solved with box constraints only, originating from minimization of a primal Lagrangian in respect to the weights w_i. However, there remains an open question — how to apply the proposed ISDA for the SVMs that do use explicit bias term b. Such general nonlinear SVMs in classification and regression tasks are given below,

$$f(\mathbf{x}_i) = \sum_{j=1}^{l} y_j \alpha_j \Phi(\mathbf{x}_i)^T \Phi(\mathbf{x}_j) + b = \sum_{j=1}^{l} w_j K(\mathbf{x}_i, \mathbf{x}_j) + b, \tag{86a}$$

$$f(\mathbf{x}_i) = \sum_{j=1}^{l} (\alpha_j^* - \alpha_j) \Phi(\mathbf{x}_i)^T \Phi(\mathbf{x}_j) + b = \sum_{j=1}^{l} w_j K(\mathbf{x}_i, \mathbf{x}_j) + b, \tag{86b}$$

where $\Phi(\mathbf{x}_i)$ is the m-dimensional vector that maps n-dimensional input vector $\mathbf{x}$ into the feature space [13]. For each SVMs' model in (86), there is also one *equality constraint* originating from a minimization of the primal objective function in respect to the bias b as given below,

$$\sum_{i=1}^{l} \alpha_i y_i = 0, \tag{87a}$$

in a classification, and

$$\sum_{i=1}^{l} \alpha_i^* = \sum_{i=1}^{l} \alpha_i \tag{87b}$$

in a regression.

The motivation for developing the ISDA for the SVMs with an explicit bias term b originates from the fact that the use of an explicit bias term b *seems to lead to the SVMs with less support vectors*. This fact can often be very useful for both the data (information) compression and the speed of learning. Below, we present an iterative learning algorithm for the classification SVMs (86a) with an explicit bias b, subjected to the equality constraint (87a) [14]. The problem to

[13] Note that for a classification model in (86a), we usually take the sign of $f(\mathbf{x})$ but this is of lesser importance now.

[14] The same procedure is developed for the regression SVMs but due to the space constraints we do not go into these details here. However we give some relevant hints for the regression SVMs with bias b.

solve is,

$$\min \frac{1}{2}\mathbf{w}^T\mathbf{w} \tag{88a}$$

such that

$$y_i[\mathbf{w}^T\Phi(\mathbf{x}_i)+b] \geqslant 1, \quad i=1,\ldots,l\,, \tag{88b}$$

which can be transformed into its dual form by minimizing the primal Lagrangian

$$L_p(\mathbf{w},b,\boldsymbol{\alpha}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{l}\alpha_i\big\{y_i\big[\mathbf{w}^T\Phi(\mathbf{x}_i)+b\big]-1\big\} \tag{89}$$

in respect to $\mathbf{w}$ and b by using $\partial L_p/\partial \mathbf{w}_o = 0$ and $\partial L_p/\partial b = 0$, i.e. by exploiting

$$\mathbf{w} = \sum_{i=1}^{l}\alpha_i y_i \Phi(\mathbf{x}_i) \quad \text{and} \quad \sum_{i=1}^{l}\alpha_i y_i = 0. \tag{90}$$

The standard change to a dual problem is to substitute $\mathbf{w}$ from (90) into the primal Lagrangian and this leads to a dual Lagrangian problem below,

$$L_d(\alpha) = \sum_{i=1}^{l}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i,\mathbf{x}_j) - \sum_{i=1}^{l}\alpha_i y_i b \tag{91}$$

subject to the box constraints (92) and, in a standard SVMs formulation, also to the equality constraint (93) as given below

$$\alpha_i \geqslant 0, \quad i=1,\ldots,l \tag{92}$$

and

$$\sum_{i=1}^{l}\alpha_i y_i = 0\,. \tag{93}$$

There are *three major avenues* (procedures, algorithms) possible in solving the dual problem (91), (92) and (93).

The *first one* is the standard SVMs algorithm which imposes the equality constraint (93) during the optimization and in this way ensures that the solution never leaves a feasible region. In this case the last term in (91) vanishes. (Note that in a standard SMO iterative scheme for training SVMs the minimal number of training data points enforcing (93) and ensuring staying in a feasible region

is two). After the dual problem is solved, the bias term is calculated by using *unbounded* Lagrange multipliers α_i (*Kecman*, 2001; *Schölkopf, Smola*, 2002) as follows

$$b = \frac{1}{\#UnboundSVecs}\Big(\sum\nolimits_{i=1}^{\#UnboundSVecs}\big(y_i - \mathbf{w}^T\Phi(\mathbf{x}_i)\big)\Big). \qquad (94)$$

Below, we show *two more possible ways* how the ISDA works for the SVMs containing an explicit bias term too. In *the second method*, the cost function (88a) is augmented with the term $0.5kb^2$ (where $k \geqslant 0$). Note that this step is related to solving the dual problem by penalty method where a decrease in k leads to the stronger imposing of an equality constraint (see comments below). After forming the primal Lagrangian as well as using

$$\mathbf{w} = \sum_{i=1}^{l}\alpha_i y_i \Phi(\mathbf{x}_i) \quad \text{and} \quad b = \frac{1}{k}\sum_{i=1}^{l}\alpha_i y_i$$

(coming from $\partial L_p/\partial \mathbf{w}_o = 0$ and $\partial L_p/\partial b = 0$) one arrives at the dual problem not containing the explicit bias term b. Actually, the optimization of a dual Lagrangian is reformulated for the SVMs with a bias term b by applying "tiny" change only to the original matrix $\mathbf{K}$. For the *nonlinear classification* problems ISDA stands for an iterative solving of the following linear system

$$\mathbf{K}_k\,\boldsymbol{\alpha} = \mathbf{1}_l \qquad (95a)$$

such that

$$0 \leqslant \alpha_i \leqslant C, \quad i = 1, \ldots, l\,, \qquad (95b)$$

where

$$K_k(\mathbf{x}_i, \mathbf{x}_j) = y_i y_j\Big(K(\mathbf{x}_i, \mathbf{x}_j) + \frac{1}{k}\Big),$$

$\mathbf{1}_l$ is an l-dimensional unity vector and C is a penalty factor equal to infinity for a hard margin classifier. Note that during the updates of α_i, the bias term b must not be used because it is implicitly incorporated within the $\mathbf{K}_k$ matrix. Only after the solution vector α in (95) is found, the bias b should be calculated either by using *unbounded* Lagrange multipliers α_i as given in (94), or by implementing the equality constraint from $\partial L_p/\partial b = 0$ and given as

$$b = \frac{1}{k}\sum\nolimits_{j=1}^{\#SVecs}\alpha_i y_i. \qquad (96)$$

Note, however, that all the Lagrange multipliers, meaning both bounded (clipped to C) and unbounded (smaller than C) must be used in (96). Both equations, (94) and (96), result in the same value for the bias b. Thus, using the SVMs with an explicit bias term means that in the ISDA proposed above original kernel is changed, i.e., another kernel function is used. This means that the alpha values will be different for each k chosen, and so will be the value for b. However, the final SVM as given in (86) is produced by original kernels. Namely, $f(\mathbf{x})$ is obtained by adding the sum of weighted original kernel values and corresponding bias term b.

The second method presented above and aimed at an extending of the ISDA to the SVMs with a bias term b is related to the classic (quadratic) penalty methods for solving optimization problems with an equality constraint. Namely, the addition of $0.5kb^2$ to (88a) changes the last term of (91) to

$$\frac{1}{2k}\Big|\Big|\sum_{i=1}^{l}\alpha_i y_i\Big|\Big|_2^2$$

which is equivalent to applying a penalty parameter of $1/k$ to the L_2 norm of the equality constraint (93). As a result, for a large value of $1/k$, the solution will have a small L_2 norm of (93). In other words, as k approaches zero a bias b converges to the solution of the standard QP method that enforces the equality constraint. However, we do not use the ISDA with small parameter k values here, because the condition number of the matrix $\mathbf{K}_k$ increases as $1/k$ rises. Furthermore, the strict fulfilment of (93) may not be needed in obtaining a good SVM. Here, in classifying the MNIST data with Gaussian kernels, the value $k = 10$ proved to be a very good one justifying all the reasons for its introduction (fast learning, small number of support vectors and good generalization).

The third method in implementing the ISDA for SVMs with the bias term b is to work with original cost function (88a) and keep imposing the equality constraint during the iterations as suggested in (*Veropoulos*, 2001). The learning starts with $b = 0$ and after each epoch the bias b is updated by applying a secant method as follows

$$b^k = b^{k-1} - \omega^{k-1}\frac{b^{k-1} - b^{k-2}}{\omega^{k-1} - \omega^{k-2}} \tag{97}$$

where $\omega = \sum_{i=1}^{l}\alpha_i y_i$ represents the value of an equality constraint after each epoch. In the case of the regression SVMs, equation (97) is used by implement-

ing the corresponding regression's equality constraint, namely $\sum_{i=1}^{l}(\alpha_i - \alpha_i^*)$. This is different from (*Veropoulos*, 2001) where an iterative update after each data pair is proposed. In our SVMs regression experiments such an updating led to an unstable learning. Also, in an addition to changing expression for ω, both the $\mathbf{K}$ matrix, which is now $(2l, 2l)$ matrix, and the right hand side of (95a) which becomes $(2l, 1)$ vector, should be changed too and formed as given in (*Kecman, Vogt, Huang*, 2003).

Performance of an ISD Learning Algorithm and Comparisons

To measure the relative performance of different ISDAs, we ran all the algorithms with RBF Gaussian kernels on a MNIST dataset with 576-dimensional inputs (*Dong* et al, 2003), and compared the performance of our ISD algorithm with LIBSVM V2.4 (*Chang* et al, 2003) which is one of the fastest and the most popular SVM solvers at the moment based on the SMO type of an algorithm. The MNIST dataset consists of 60,000 training and 10,000 test data pairs. To make sure that the comparison is based purely on the nature of the algorithm rather than on the differences in implementation, our encoding of the algorithms are the same as LIBSVM's one in terms of caching strategy (LRU–Least Recent Used), data structure, heuristics for shrinking and stopping criterions. The only significant difference is that instead of two heuristic rules for selecting and updating two data points at each iteration step aiming at the maximal improvement of the dual objective function, our ISDA selects the worse KKT violator only and updates its α_i at each step.

Also, in order to speed up the LIBSVM's training process, we modified the original LIBSVM routine to perform faster by reducing the numbers of complete KKT checking without any deterioration of accuracy. All the routines were written and compiled in Visual C++ 6.0, and all simulations were run on a 2.4 GHz P4 processor PC with 1.5 Gigabyte of memory under the operating system Windows XP Professional. The shape parameter σ^2 of an RBF Gaussian kernel and the penalty factor C are set to be 0.3 and 10 (*Dong J.X.* et al, 2003). The stopping criterion τ and the size of the cache used are 0.01 and 250 Megabytes. The simulation results of different ISDA against both LIBSVM are presented in Tables 3 and 4, and in a Fig. 19. The first and the second column of the tables show the performance of the original and modified LIBSVM respectively. The last three columns show the results for single data point learning algorithms with various values of constant $1/k$ added to the kernel matrix in (95a). For $k = \infty$, ISDA is equivalent to the SVMs without bias term, and for $k = 1$,

Table 3. Simulation time for different algorithms

	LIBSVM original	**LIBSVM modified**	**Iterative single data algorithm (ISDA)** $k = 1$	$k = 10$	$k = \infty$
Class	Time(sec)	Time(sec)	Time(sec)	Time(sec)	Time(sec)
0	1606	885	800	**794**	1004
1	740	**465**	490	491	855
2	2377	1311	1398	**1181**	1296
3	2321	1307	1318	**1160**	1513
4	1997	1125	1206	**1028**	1235
5	2311	1289	1295	**1143**	1328
6	1474	818	808	**754**	1045
7	2027	1156	2137	**1026**	1250
8	2591	1499	1631	**1321**	1764
9	2255	1266	1410	**1185**	1651
Time Increase	**+95.3%**	**+10.3%**	**+23.9%**	**0**	**+28.3%**

Table 4. Number of support vectors for each algorithm

	LIBSVM original	**LIBSVM modified**	**Iterative single data algorithm (ISDA)** $k = 1$	$k = 10$	$k = \infty$
Class	# SV (BSV)	# SV (BSV)	# SV (BSV)	# SV (BSV)	# SV (BSV)
0	2172 (0)	2172 (0)	2162 (0)	2132 (0)	**2682** (0)
1	1440 (4)	1440 (4)	1429 (4)	1453 (4)	**2373** (4)
2	3055 (0)	3055 (0)	3047 (0)	3017 (0)	**3327** (0)
3	2902 (0)	2902 (0)	2888 (0)	2897 (0)	**3723** (0)
4	2641 (0)	2641 (0)	2623 (0)	2601 (0)	**3096** (0)
5	2900 (0)	2900 (0)	2884 (0)	2856 (0)	**3275** (0)
6	2055 (0)	2055 (0)	2042 (0)	2037 (0)	**2761** (0)
7	2651 (4)	2651 (4)	3315 (4)	2609 (4)	**3139** (4)
8	3222 (0)	3222 (0)	3267 (0)	3226 (0)	**4224** (0)
9	2702 (2)	2702 (2)	2733 (2)	2756 (2)	**3914** (2)
Average # of SV	**2574**	**2574**	**2639**	**2558**	**3151**

BSV = Bounded SVs

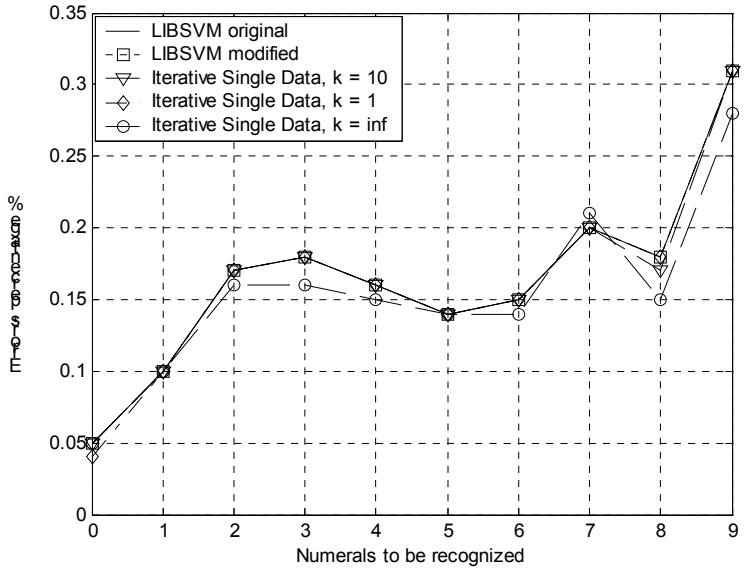

Figure 19. The percentage of errors on the test data

it is the same as the classification formulation proposed in (*Mangasarian* and *Musicant*, 1999).

Table 3 illustrates the running time for each algorithm. The ISDA with $k = 10$ was the quickest and required the shortest average time (T_{10}) to complete the training. The average time needed for the original LIBSVM is almost $2T_{10}$ and the average time for a modified version of LIBSVM is 10.3% bigger than T_{10}. This is contributed mostly to the simplicity of the ISD algorithm. One may think that the improvement achieved is minor, but it is important to consider the fact that approximately more than 50% of the CPU time is spent on the final checking of the KKT conditions in all simulations. During the checking, the algorithm must calculate the output of the model at each datum in order to evaluate the KKT violations. This process is unavoidable if one wants to ensure the solution's global convergence, i.e. that *all the data* do satisfy the KKT conditions with precision τ indeed. Therefore, the reduction of time spent

on iterations is approximately double the figures shown. Note that the ISDA slows down for $k < 10$ here. This is a consequence of the fact that with a decrease in k there is an increase of the condition number of a matrix $\mathbf{K}_k$, which leads to more iterations in solving (95). At the same time, implementing the no-bias SVMs, i.e., working with $k = \infty$, also slows the learning down due to an increase in the number of support vectors needed when working without bias b.

Table 4 presents the numbers of support vectors selected. For the ISDAs, the numbers reduce significantly when the explicit bias term b is included. One can compare the numbers of SVs for the case without the bias b ($k = \infty$) and the ones when an explicit bias b is used (cases with $k = 1$ and $k = 10$). Because identifying less support vectors speeds the overall training definitely up, the SVMs implementations with an explicit bias b are faster than the version without bias.

In terms of a generalization, or a performance on a test data set, all the algorithms had very similar results and this demonstrates that the ISDAs produce models that are as good as the standard QP, i.e., SMO based, algorithms. The percentages of the errors on the test data are shown in Fig. 19. Notice the extremely low error percentages on the test data sets for all numerals.

Conclusions

The seminar presents the basics of the standard SVMs models for solving the classification and regression problems first. Then, it also shows why and how, when positive definite kernels are used, the kernel AdaTron, sequential minimal optimization and Gauss-Seidel (i.e., successive over relaxation) algorithms are identical in their analytic form and numerical implementation. Till now, these facts were blurred mainly due to different pace in posing the learning problems and due to the "heavy" heuristics involved in an SMO implementation that shadowed an insight into the possible identity of the methods. It is shown that in the so-called no-bias SVMs, both the KA and the SMO procedure are the coordinate ascent based methods. Based on these equalities the novel ISDA for training SVMs is devised. Finally, due to the many ways how all the three algorithms (KA, SMO and GS i.e., SOR) can be implemented there may be some differences in their overall behavior. The introduction of the relaxation parameter $0 < \omega < 2$ will speed up the algorithm. The exact optimal value ω_{opt} is problem dependent.

Next, we demonstrate the use, the calculation and the effect of incorporating an explicit bias term b in the SVMs trained with the ISDA. The simulation results show that models generated by ISDAs (either with or without the bias term b) are as good as the standard QP (i.e., SMO) based algorithms in terms of a generalization performance. Moreover, ISDAs with an appropriate k value are faster than the standard SMO algorithms on large scale classification problems ($k = 10$ worked particularly well in all our simulations using Gaussian RBF kernels, however it may be that the "best" k value is problem dependent). This is due to both the simplicity of ISDAs and the decrease in the number of SVs chosen after an inclusion of an explicit bias b in the model. The simplicity of ISDAs is the consequence of the fact that the equality constraints (87) do not need to be fulfilled during the training stage. In this way, the *second order heuristics is avoided* during the iterations. Thus, the ISDA is an extremely good tool for solving large scale SVMs problems containing huge training data sets because it is faster than, and it delivers 'same' generalization results as, the other standard QP (SMO) based algorithms. The fact that an introduction of an explicit bias $b$means solving the problem with different kernel suggests that it may be hard to tell in advance for what kind of previously unknown multivariable decision (regression) function the models with bias b may perform better, or may be more suitable, than the ones without it. As it is often the case, the real experimental results, their comparisons and the new theoretical developments should probably be able to tell one day. As for the single data based learning approach presented here, the future work will focus on the development of even faster training algorithms.

References

1. *Abe, S.*, 2004. Support Vector Machines for Pattern Classification, Springer-Verlag, London.
2. *Aizerman, M.A., E.M. Braverman, and L.I. Rozonoer*, 1964. Theoretical foundations of the potential function method in pattern recognition learning // *Automation and Remote Control*, **25**, pp.821–837.
3. *Anlauf, J.K., Biehl, M.*, 1989. The AdaTron — an adaptive perceptron algorithm // *Europhysics Letters*, **10**(7), pp.687–692.
4. *Bartlett, P.L., A. Tewari*, 2004. Sparseness vs estimating conditional probabilities: Some asymptotic results // (submitted for a publication and taken from the P.L. Bartlett's site).

5. *Chang, C., Lin, C.*, 2003. LIBSVM: a library for support vector machines, Available at:
URL: http://www.csie.ntu.edu.tw/∼cjlin/libsvm/

6. *Cherkassky, V., F. Mulier*, 1998. Learning From Data: Concepts, Theory and Methods, John Wiley & Sons, New York, NY.

7. *Cortes, C.*, 1995. Prediction of Generalization Ability in Learning Machines. PhD Thesis, Department of Computer Science, University of Rochester, NY.

8. *Cortes, C., Vapnik, V.* 1995. Support Vector Networks // *Machine Learning*, **20**: 273–297.

9. *Cristianini, N., Shawe-Taylor, J.*, 2000, An introduction to Support Vector Machines and other kernel-based learning methods, Cambridge University Press, Cambridge, UK.

10. *Dong, X., Krzyzak, A., Suen, C.Y.*, 2003. A fast SVM training algorithm // *International Journal of Pattern Recognition and Artificial Intelligence*, vol. **17**, No.3, pp.367–384.

11. *Drucker, H., C.J.C. Burges, L. Kaufman, A. Smola, V. Vapnik*. 1997. Support vector regression machines // *Advances in Neural Information Processing Systems*, **9**, pp.155–161, MIT Press, Cambridge, MA.

12. *Eisenhart, C.*, 1962. Roger Joseph Boscovich and the Combination of Observationes // *Actes International Symposium on R. J. Boskovic*, pp.19–25, Belgrade–Zagreb–Ljubljana, YU.

13. *Evgeniou, T., Pontil, M., Poggio, T.*, 2000. Regularization networks and support vector machines // *Advances in Computational Mathematics*, **13**, pp.1–50.

14. *Friess, T., R.F. Harrison*, 1998. Linear programming support vectors machines for pattern classification and regression estimation and the set reduction algorithm, TR RR-706, University of Sheffield, Sheffield, UK.

15. *Friess, T.-T., Cristianini, N., Campbell, I.C.G.*, 1998. The Kernel-Adatron: a Fast and Simple Learning Procedure for Support Vector Machines // *In Shavlik, J.*, editor, *Proceedings of the 15th International Conference on Machine Learning*, Morgan Kaufmann, pp.188–196, San Francisco, CA.

16. *Girosi, F.*, 1997. An Equivalence Between Sparse Approximation and Support Vector Machines // *AI Memo* 1606, MIT.

17. *Graepel, T., R. Herbrich, B. Schölkopf, A. Smola, P. Bartlett, K.-R. Müller, K. Obermayer, R. Williamson*, 1999. Classification on proximity data with LP–machines // *Proc. of the* 9^{th} *Intl. Conf. on Artificial NN*, ICANN 99, Edinburgh, 7–10 Sept.

18. *Hadzic, I., V. Kecman*, 1999. Learning from Data by Linear Programming // *NZ Postgraduate Conference Proceedings*, Auckland, Dec. 15–16.

19. *Huang T.-M., Kecman V.*, 2004. Bias Term b in SVMs Again // *Proc. of the* 12^{th} *European Symposium on Artificial Neural Networks*, ESANN 2004, pp.441–448, Bruges, Belgium.

20. *Huang T.-M., V. Kecman, I. Kopriva*, 2006. Kernel Based Algorithms for Mining Huge Data Sets, Supervised, Semi-supervised, and Unsupervised Learning, Series "Computational Intelligence", Springer-Verlag, in print.

21. *Huang, T.-M.*, 2006. Large-Scale Support Vector Machines and Semi-Supervised Learning Algorithms, PhD thesis, The University of Auckland, School of Engineering, Auckland, NZ.

22. *Kecman, V., Arthanari T., Hadzic I.*, 2001 LP and QP Based Learning From Empirical Data // *IEEE Proceedings of IJCNN 2001*, Vol.4, pp.2451–2455, Washington, DC.

23. *Kecman, V.*, 2001. Learning and Soft Computing, Support Verctor machines, Neural Networks and Fuzzy Logic Models, The MIT Press, Cambridge, MA, the book's web site is:
URL: http://www.support-vector.ws

24. *Kecman, V., Hadzic I.*, 2000. Support Vectors Selection by Linear Programming // *Proceedings of the International Joint Conference on Neural Networks* (IJCNN 2000), Vol.5, pp.193–198, Como, Italy.

25. *Kecman, V., Vogt, M., Huang, T.-M.*, 2003. On the Equality of Kernel AdaTron and Sequential Minimal Optimization in Classification and Regression Tasks and Alike Algorithms for Kernel Machines // Proc. of ESANN 2003, 11^{th} *European Symposium on Artificial Neural Networks*, Bruges, Belgium, downloadable from
URL: http://www.support-vector.ws

26. *Kecman, V., T.M. Huang, M. Vogt, 2005*. Iterative Single Data Algorithm for Training Kernel Machines from Huge Data Sets: Theory and Performance, Chapter in a Springer-Verlag book, "Support Vector Machines: Theory and Applications", ed. *L. Wang*.

27. *Lawson, C. I., Hanson, R. J.*, 1974. Solving Least Squares Problems, Prentice-Hall, Englewood Cliffs, N.J.

28. *Mangasarian, O.L., 1965*. Linear and Nonlinear Separation of Patterns by Linear Programming // *Operations Research*, **13**, pp.444–452.

29. *Mangasarian, O.L., Musicant, D.R.*, 1999. Successive Overrelaxation for Support Vector Machines // *IEEE Trans. Neural Networks*, **11**(4), pp.1003–1008.

30. *Mercer, J.*, 1909. Functions of positive and negative type and their connection with the theory of integral equations // *Philos. Trans. Roy. Soc. London*, A 209:415{446}.

31. *Ostrowski, A.M.*, 1966. Solutions of Equations and Systems of Equations, 2nd ed., Academic Press, New York.

32. *Osuna, E., R. Freund, F. Girosi*. 1997. Support vector machines: Training and applications // *AI Memo* 1602, Massachusetts Institute of Technology, Cambridge, MA.

33. *Platt, J.C.* 1998. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report MSR-TR-98-14, Microsoft Research, 1998.

34. *Platt, J.C.*, 1999. Fast Training of Support Vector Machines using Sequential Minimal Optimization. Chapter 12 in Advances in Kernel Methods — Support Vector Learning, edited by *B.Schölkopf, C. Burges, A. Smola*, The MIT Press, Cambridge, MA.

35. *Poggio, T., Mukherjee, S., Rifkin, R., Rakhlin, A., Verri, A.*, 2001. *b*, CBCL Paper # 198/AI Memo #2001–011, Massachusetts Institute of Technology, Cambridge, MA.

36. *Schölkopf, B., Smola, A.*, 2002. Learning with Kernels — Support Vector Machines, Optimization, and Beyond, The MIT Press, Cambridge, MA.

37. *Smola, A., Schölkopf, B.* 1997. On a Kernel-based Method for Pattern Recognition, Regression, Approximation and Operator Inversion. GMD Technical Report No.1064, Berlin.

38. *Smola, A., T.T. Friess, B. Schölkopf*, 1998, Semiparametric Support Vector and Linear Programming Machines, NeuroCOLT2 Technical Report Series, NC2-TR-1998-024, also In: *Advances in Neural Information Processing Systems 11*

39. *Steinwart, I.*, 2003. Sparseness of support vector machines // *Journal of Machine Learning Research*, **4** (2003), pp.1071–1105.

40. Support Vector Machines Web Site:
URL: http://www.kernel-machines.org/

41. *Suykens, J.A.K., T. Van Gestel, J. De Brabanter, B. De Moor, J. Vandewalle*, 2002. Least Squares Support Vector Machines, World Scientific Pub. Co., Singapore.

42. *V. Vapnik and A. Chervonenkis*, 1964. A note on one class of perceptrons // *Automation and Remote Control*, **25**.

43. *Vapnik, V.N., A.Y. Chervonenkis*, 1968. On the uniform convergence of relative frequencies of events to their probabilities // *Doklady Akademii Nauk USSR*, **181**, (4) (In Russian).

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

45. *Vapnik, V.N., A.Y. Chervonenkis*, 1989. The necessary and sufficient conditions for the consistency of the method of empirical minimization [in Russian] // *Yearbook*

of the Academy of Sciences of the USSR on Recognition, Classification, and Forecasting, **2**, 217–249, Moscow, Nauka, (English transl.: The necessary and sufficient condititons for the consistency of the method of empirical minimization. Pattern Recognitio and Image Analysis, **1**, 284–305, 1991)

46. *Vapnik, V.N.*, 1995. The Nature of Statistical Learning Theory, Springer Verlag Inc, New York, NY.
47. *Vapnik, V., S. Golowich, A. Smola*. 1997. Support vector method for function approximation, regression estimation, and signal processing // In: *Advances in Neural Information Processing Systems* **9**, MIT Press, Cambridge, MA.
48. *Vapnik, V.N.*, 1998. Statistical Learning Theory, J.Wiley & Sons, Inc., New York, NY.
49. *Veropoulos, K.*, 2001. Machine Learning Approaches to Medical Decision Making, PhD Thesis, The University of Bristol, Bristol, UK.
50. *Vogt, M.*, 2002. SMO Algorithms for Support Vector Machines without Bias, Institute Report, IAT, TU Darmstadt, Darmstadt, Germany.
URL: http://w3.rt.e-technik.tu-darmstadt.de/∼vogt/
51. *Vogt, M., V. Kecman*, 2005. Active-Set Methods for Support Vector Machines, Chapter in a Springer-Verlag book, “Support Vector Machines: Theory and Applications”, ed. *L. Wang*.

Vojislav KECMAN, PhD, MSc, Dipl.-Ing., Associate Professor and a Head of Dynamics and Control Systems Group, Department of Mechanical Engineering, the University of Auckland, Auckland, New Zealand. He is an Associate Editor in IEEE Transaction on Neural Networks and a member of the Advisory Board of Interdisciplinary Journal of Information, Knowledge, and Management.
His newest research interests are machine learning from huge data sets in high dimensional spaces by support vector machines and/or neural networks; fuzzy logic systems; kernel machines; pattern recognition and/or multivariate function approximation; knowledge modelling and knowledge acquiring; bioinformatics; text categorization. Other research interests: neural networks based adaptive control systems; system dynamics modelling, simulation, identification and control; unified approach to the modelling of physically different systems; singularly perturbed control systems.

He has visited many European, North American and Pacific region universities, research institutions or industries and upon the invitations delivered seminars from the fields of research at Harvard University, MIT, University of California Santa Barbara, Rutgers University, TU Bremen, TU Dresden, TH Darmstadt, FH Heilbronn, Research Institute for Knowledge Systems in Maastricht, Melbourne University, The University of Auckland, Universities of Belgrade, Novi Sad and BanjaLuka.
Dr. Kecman has published more than 100 journal and conference papers, monographs, books or other bound publications.

Воислав КЕЦМАН, руководитель группы динамики и систем управления, доцент кафедры машиностроения Университета в Окленде, Новая Зеландия. Он является членом редакционной коллегии журнала IEEE Transaction on Neural Networks, а также членом консультативного совета журнала Interdisciplinary Journal of Information, Knowledge, and Management.
Его основные научные интересы лежат в настоящее время в таких областях, как обучение машин на сверхбольших наборах данных в многомерных пространствах с использованием машин опорных векторов и/или нейронных сетей; нечеткие системы; kernel machines; распознавание образов и/или аппроксимация функций многих переменных; модели знаний и извлечение знаний; биоинформатика; категоризация текста. К числу других областей научных интересов д-ра Кецмана относятся также нейросетевые адаптивные системы управления; моделирование, идентификация и управление для динамических систем; единый подход к моделированию систем различной физической природы; системы управления с сингулярными возмущениями.
Д-ра Кецмана приглашали для проведения семинаров по тематике его научных интересов многие университеты и исследовательские институты Европы, Северной Америки и Тихоокеанского региона.
Д-р Кецман является автором более 100 журнальных статей, докладов на конференциях и книг.