International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:12, 2007

Extended Least Squares LS–SVM

József Valyon and Gábor Horváth

***Abstract*—**Among neural models the Support Vector Machine (SVM) solutions are attracting increasing attention, mostly because they eliminate certain crucial questions involved by neural network construction. The main drawback of standard SVM is its high computational complexity, therefore recently a new technique, the Least Squares SVM (LS–SVM) has been introduced. In this paper we present an extended view of the Least Squares Support Vector Regression (LS–SVR), which enables us to develop new formulations and algorithms to this regression technique. Based on manipulating the linear equation set -which embodies all information about the regression in the learning process- some new methods are introduced to simplify the formulations, speed up the calculations and/or provide better results.

***Keywords*—**Function estimation, Least–Squares Support Vector Machines, Regression, System Modeling

I. Introduction

THIS paper focuses on the Least Squares version of SVM [1], the LS–SVM [2], whose main advantage is that it is computationally more efficient than the standard SVM method. In this case training requires the solution of a linear equation set instead of the long and computationally hard quadratic programming problem involved by the standard SVM. It must also be emphasized that LS–SVM is closely related to Gaussian processes and regularization networks in that the obtained linear systems are equivalent [2].

While the least squares version incorporates all training data in the network to produce the result, the traditional SVM selects some of them (the support vectors) that are important in the regression. The use of only a subset of all vectors is a desirable property of SVMs, because it provides additional information regarding the training data and concludes in a more effective solution formulating a smaller network. Similarly to the sparseness of a traditional SVM, sparse LS-SVM solutions can also be reached by applying pruning methods [3][4]. Unfortunately if this LS–SVM pruning is applied, the performance declines proportionally to the eliminated training samples, since the information (input-output relation) they described is lost. Another problem is that sparseness can be reached by iterative methods, which multiply the algorithmic complexity.

Moreover the training data is often corrupted by noise, which – if not handled properly – misleads the training. LS–SVMs should also be able to handle outliers (e.g. resulting from non–Gaussian noise). Another modification of the method, called weighted LS–SVM [2], is aimed at reducing the effects of this type of noise. The biggest problem is that pruning and weighting – although their goals do not rule out each other – cannot be used at the same time, because they work in opposition.

This paper introduces a generalized approach that enables us to accomplish both goals by allowing a more universal construction and solution of the LS–SVM equation set. This generalized approach is based on a simple geometric interpretation of the problem represented in the kernel space.

The LS–SVM method is capable of solving both classification and regression problems. The classification approach is easier to understand and more historic. However, the present study concerns regression, therefore only this is introduced in the sequel, along with the most common extensions (pruning, weighting, and a fixed size version). It must be emphasized that all the methods can be applied to classification as well.

This paper is organized as follows. Before going into the details the main and distinguishing features of the basic procedures are summarized is Section II. Section III presents the geometric interpretation in the kernel space and summarizes the main idea behind the propositions. Section IV contains the details:

- It shows how the kernel space dimension can be reduced using a so called *partial reduction* technique that results in an overdetermined equation set and consequently a sparse solution. (IV. A.).
- It proposes some possible ways to solve this equation set (IV. B.).
- It presents a practical way to construct the overdetermined equation set (IV. C).
- It gives the algorithmic complexity of the method (IV. D.).

Section V. contains some experimental results, while conclusions are drawn in section VI.

II. A brief overview of the LS-SVM method

Only a brief outline of the LS-SVM method is presented, a detailed description can be found in refs. [2]-[4], [6].

Given the $\{\mathbf{x}_i, d_i\}_{i=1}^N$ training data set, where $\mathbf{x}_i \in \Re^p$ represents a p–dimensional input vector and $d_i = y_i + z_i$,

Manuscript received July 20, 2005.

J. Valyon is with Budapest University of Technology and Economics-Department of Measurement and Information Systems, Budapest, Hungary, H-1521, pf. 91. (phone +36 1 463-2057; fax +36 1 463-4112; e-mail: valyon@mit.bme.hu).

G. Horváth (e-mail: horvath@mit.bme.hu).

This work was partly sponsored by National Fund for Scientific Research (OTKA) under contract T 046771.

$d_i \in \Re$ is a scalar measured output, which represents the y_i system output disturbed by some z_i noise. Our goal is to construct an $y = f(\mathbf{x})$ function, which represents the dependence of the output d_i on the input $\mathbf{x}_i$. Let's define the form of this function as formulated below:

$$y = \sum_{i=1}^{h} w_i \varphi_i(\mathbf{x}) + b = \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}) + b, \quad (1)$$

$$\mathbf{w} = [w_1, w_2, ..., w_h]^T, \quad \boldsymbol{\varphi} = [\varphi_1, \varphi_2, ..., \varphi_h]^T.$$

The $\boldsymbol{\varphi}(.) : \Re^p \to \Re^h$ is a mostly non-linear function, which maps the data into a higher, possibly infinite dimensional feature space. The main difference from the standard SVM is in the constraints defined by the training samples [1]. The optimization problem and the inequality constraints are replaced by the following equations ($i = 1,..., N$):

$$\min_{\mathbf{w},b,e} J_p(\mathbf{w}, e) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\frac{1}{2}\sum_{i=1}^{N} e_i^2 \quad (2)$$

with constraints: $d_i = \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}_i) + b + e_i$.

The first term forces a smooth solution, while the second one minimizes the training errors ($C \in \Re^+$ is the trade–off parameter between the two terms). From this, a Lagrangian is formed

$$L(w, b, e; \alpha) = J_p(w, e) - \sum_{i=1}^{N} \alpha_i \{\mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}_i) + b + e_i - d_i\}, \quad (3)$$

where the α_i parameters are the Lagrange multipliers. The solution concludes in a constrained optimization, where the conditions for optimality are the followings:

$$\begin{aligned}
&\frac{\partial L}{\partial w} = 0 \quad \to \quad \mathbf{w} = \sum_{i=1}^{N} \alpha_i \boldsymbol{\varphi}(\mathbf{x}_i) \\
&\frac{\partial L}{\partial b} = 0 \quad \to \quad \sum_{i=1}^{N} \alpha_i = 0 \\
&\frac{\partial L}{\partial e_i} = 0 \quad \to \quad \alpha_i = C\, e_i \qquad\qquad i = 1,..., N \\
&\frac{\partial L}{\partial \alpha_i} = 0 \quad \to \quad \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}_i) + b + e_i - d_i = 0 \quad i = 1,..., N
\end{aligned} \quad (4)$$

This leads to the following overall solution:

$$\left[\begin{array}{c|c} 0 & \vec{\mathbf{1}}^{\mathbf{T}} \\ \hline \vec{\mathbf{1}} & \boldsymbol{\Omega} + C^{-1}\mathbf{I} \end{array}\right] \left[\begin{array}{c} b \\ \hline \boldsymbol{\alpha} \end{array}\right] = \left[\begin{array}{c} 0 \\ \hline \mathbf{d} \end{array}\right] \quad (5)$$

where $\mathbf{d} = [d_1, d_2, ..., d_N]$., $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, .., \alpha_N]$, $\vec{\mathbf{1}} = [1, ..., 1]$ and , $\Omega_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\varphi}^T(\mathbf{x}_i)\boldsymbol{\varphi}(\mathbf{x}_j)$. Here $K(\mathbf{x}_i, \mathbf{x}_j)$ is the kernel function, and $\boldsymbol{\Omega}$ is the kernel matrix. Throughout this paper the RBF like kernel function is used, but other kernels can also be applied [2].

For a given input $\mathbf{x}$ the response of an LS-SVM is a weighted sum of N kernel functions, where the center parameters of these kernel functions are determined by the training input vectors $\mathbf{x}_i$:

$$y(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i K(\mathbf{x}, \mathbf{x}_i) + b \quad (6)$$

It is important to emphasize that according to (4) the α_i weights are proportional to the e_i errors in the training points: $\alpha_i = Ce_i$. The following iterative methods are based on this property of the LS–SVM.

A. LS–SVM pruning

A sparse LS-SVM can be obtained by applying a pruning method, which eliminates some training samples based on the sorted $|\alpha_i|$ spectrum [2]-[4]. The α_i values reflect the importance of the training samples, therefore by eliminating some training samples, represented by the smallest values from this $|\alpha_i|$ spectrum, the complexity of the result can be reduced. The most irrelevant points are left out, by iteratively leaving out the least significant ones. These are the ones corresponding to the smallest $|\alpha_i|$ values.

In a classical SVM sparseness is achieved by the use of an ε–insensitive loss function, where errors smaller than ε are ignored (e.g. ε–insensitive loss function). The pruning of LS-SVM reduces the difference between the classical and the least squares SVMs, because the omission of some data points implicitly corresponds to creating an ε–insensitive zone [5].

The described method leads to a sparse model, but some questions arise: How many neurons are needed in the final model? How many iterations it should take to reach the final model? Another problem is that a usually large linear system must be solved in all iterations. The pruning is especially important if the number of training vectors is large. In this case however, the iterative method is not very effective.

B. Fixed LS–SVM

Another solution to this problem is called Fixed Size LS–SVM [2], which uses an entropy based iterative method to select a predefined (fixed) size subset of the training samples as support vectors. After determining this optimal subset – in the sense of the defined entropy measure – a final LS–SVM network is trained.

C. Weighted LS–SVM

Weighted LS–SVM [2] addresses the problem of noisy data – like outliers in a dataset –, by using a weighting factor in the calculation based on the error variables determined from a previous – first unweighted – solution. The method uses a bottom–up approach by starting from a standard solution, and calculating one or more weighted LS–SVM networks based on the previous result. The weighting is designed such that the results improve in view of robust statistics. Large e_i –s mean small weights and vice versa.

A common property of the described methods is that they are all iterative, where every step is based on the result of an LS-SVM solution. This means that the entire large problem must be solved at least once, and a relatively large one in every further iteration step. Another drawback is that pruning and weighting cannot be easily combined, because the

methods favor contradictory types of points. While pruning drops the training points belonging to small α_i-s, the weighted LS–SVM increases the effects of these points

The proposed Generalized LS-SVM method leads to a sparse solution, automatically answers the questions and solves the problem described above.

III. The Main Idea

One of the most important features of a support vector classifier or regressor is that they apply two consecutive mappings. The first one maps the training data from the input space into a higher-dimensional feature space, then by using the kernel trick the feature space representation is mapped into a kernel space. The feature space is defined by the $\boldsymbol{\varphi}(\mathbf{x}) = [\varphi_1(\mathbf{x}), \varphi_2(\mathbf{x}), ..., \varphi_h(\mathbf{x})]^T$ non-linear function set, while the kernel space is defined by the kernel functions $K(\mathbf{x}_i, \mathbf{x}_j)$ obtained using inner product as $K(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\varphi}^T(\mathbf{x}_i)\boldsymbol{\varphi}(\mathbf{x}_j)$. The main reason of introducing these mappings is to linearize the problem: the solution of a problem in these spaces will be linear even if in the original representation only a non-linear solution could have been obtained [5]. In the kernel space, a hyperplane is fitted on the training samples, so the task is to determine the free parameters α_i-s and b of this hyperplane (see Eq. (6).

The main ideas introduced in this paper work in the kernel space. As one of the main goals of this paper is to get a sparse LS-SVM, we should define what sparseness means in the kernel space. In general sparseness means that instead of using all training samples, only a subset, namely the support vectors are used. In the kernel space the degree of sparseness is determined by the dimension of the hyperplane. If it is less than the number of training samples (this is the case in a traditional SVM) the solution will be sparse, while if the dimensionality of the hyperplane equals to the number of all training samples no sparseness is obtained. One major goal of this paper is to propose a new way of dimension-reduction without degrading the quality of an LS-SVM. Another feature of the proposed approach is that it allows many different linear fitting strategies (even a non-linear fit) in the kernel space, so the result will be not only sparse, but a simpler formulation, noise reduction and further reduction of algorithmic complexity can also be achieved while the quality is maintained. The unique feature of the propositions is that they start directly from the kernel space formulation.

When an LS-SVM is constructed from N training samples:

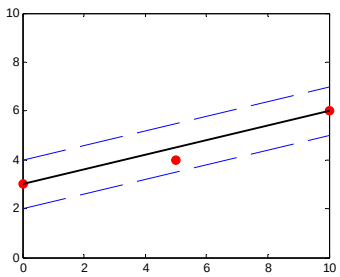
1. The training samples are mapped to an N+1 dimensional space, where N dimensions are defined by the kernel functions and one by the desired output.
2. In the N+1-dimensional space an N-dimensional hyperplane is fitted on the mapped samples. The free parameters of the hyperplane are determined by the N mapped training points, and one additional constraint ($\sum_{i=1}^{N} \alpha_i = 0$). For the sake of generalization and to avoid overfitting the accuracy of the fit can be adjusted through regularization. To trade off between training error and a smooth solution the C regularization parameter is used (see Eq. (2)), which is the same for all samples, and can be considered as a predefined, intentional error term in the kernel space fitting.

For a new sample $\mathbf{x}$ the response of the networks is determined by Eq. (6). This response is a point in the hyperplane and we expect that it will be close to the desired output. When a sparse solution is looked for only a subset of the training points are used to determine of the kernel space and the hyperplane and the dimensions of both the kernel space and the hyperplane are reduced. However, because of this reduction some training points may be far from the hyperplane, the accuracy of the mapping decreases. The main problem is to reduce the dimension without decreasing the accuracy.

In dimension reduction some questions arise: How many – and which – dimensions are needed in the kernel space? In a more definite form: can we use less than N dimensions, and how can the necessary dimensions be selected? What is a good value of C, or more generally, how should the hyperplane be placed in the kernel space?

Having fewer dimensions in the kernel space result in a sparse solution, while it means that we have more than the required number of points for determining the hyperplane in the kernel space. Having more points than dimensions in the kernel space an overdetermined equation set is obtained and this allows us to optimize the linear fit. The dimensionality of the kernel space is high enough, if samples (not used in determining this space) fall close to this plane after mapping. This is illustrated in a simple example in Fig. 1. Here the training points determine a hyperplane in the three-dimensional kernel space. In this kernel space all training points fit exactly the hyperplane. Reducing the dimension of this hyperplane we should select which training points will be used to define the two-dimensional hyperplane. At the same time in general in this reduced kernel space there will be some error in the mapping of the not used training points. Defining a tolerance interval one can decide if the dimension reduction can be allowed or not.

a.)



b.)

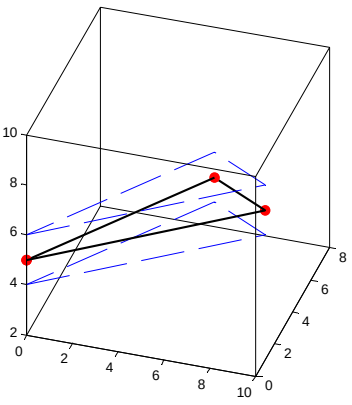


Fig. 1 The image of training samples in a kernel space of different dimensions. Using all three samples as support vectors (kernel centers), a three-dimensional kernel can space guarantees exact fit for the samples. The dashed lines represent a zone in which errors can be accepted (corresponding to the ε-insensitivity of SVM)

IV. The Proposed Methods

This section proposes some modifications and extensions to the standard LS–SVM. Their main purpose is to gain control over the model complexity and to improve the quality of the results.

A. Using an overdetermined equation set

If the training set consists of N samples, then our original linear equation set (see eq. 5) will have $(N+1)$ unknowns, the α_i-s, $(N+1)$ equations and $(N+1)^2$ multipliers. These factors are mainly the values of the $K(\mathbf{x}_i, \mathbf{x}_j)$ kernel function calculated for every combination of the training inputs. The cardinality of the training set therefore determines the size of the kernel matrix, which plays a major part in the solution, as algorithmic complexity; the complexity of the result etc. depends on this. It is easy to see that, in order to reduce complexity, this matrix has to be manipulated. Let's take a closer look at the linear equation set of eq. 5..

$$\left[\begin{array}{c|c} 0 & \vec{\mathbf{1}}^{\mathbf{T}} \\ \hline \vec{\mathbf{1}} & \mathbf{\Omega} + C^{-1}\mathbf{I} \end{array}\right]\left[\begin{array}{c} b \\ \hline \boldsymbol{\alpha} \end{array}\right] = \left[\begin{array}{c} 0 \\ \hline \mathbf{d} \end{array}\right] \tag{7}$$

The first row means:

$$\sum_{i=1}^{N} \alpha_i = 0 \tag{8}$$

and the j-th row stands for the:

$$b + \alpha_1 K(\mathbf{x}_j, \mathbf{x}_1) + \ldots + \alpha_k \left[K(\mathbf{x}_j, \mathbf{x}_j) + C^{-1}\right] + \ldots + \alpha_N K(\mathbf{x}_j, \mathbf{x}_N) = d_j \tag{9}$$

condition.

To reduce the equation set, columns and/or rows may be omitted.

If the k-th column is left out, then the corresponding α_k is also deleted, therefore the resulting model will be smaller. The condition of eq. 8. automatically adapts, since the remaining α-s will still add up to zero.

If the j-th row is deleted, then the condition defined by the $(\mathbf{x}_j, d_j)$ training sample is lost, because the j-th equation is removed.

The most important component of the main matrix is the $\mathbf{\Omega}$ kernel matrix; its elements are the results of the kernel function for pairs of training inputs:

$$\Omega_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j) \tag{10}$$

To reduce the size of $\mathbf{\Omega}$ some training samples should be omitted. Each column of the kernel matrix represents an additive term in the final solution, with a kernel function centered on the corresponding $\mathbf{x}_i$ input. The rows however, represent the input–output relations, described by the training points. The solution (α–weighting) is determined to satisfy these. It can be seen that the network size is determined by the number of columns, which -in order to reach sparseness- must be reduced. The following reduction techniques can be used on the kernel matrix (the names of these techniques are introduced here for easier discussion):

Traditional full reduction

A training sample $(\mathbf{x}_k, d_k)$ is fully omitted, therefore both the column and the row corresponding to this sample are eliminated. In this case however reduction also means that the knowledge represented by the numerous other samples are lost. The next equation demonstrates how the equation changes by fully omitting some training points. The deleted elements are colored grey.

$$\left[\begin{array}{c|ccc} 0 & \vec{\mathbf{1}} & & \\ \hline \vec{\mathbf{1}} & \Omega_{11}+\frac{1}{C} & \Omega_{12} & \cdots \; \Omega_{1N} \\ & \Omega_{21} & \Omega_{12}+\frac{1}{C} & \cdots \; \Omega_{2N} \\ & \vdots & \vdots & \vdots \\ & \Omega_{(N-1)1} & \Omega_{(N-1)2} & \cdots \; \Omega_{(N-1)N} \\ & \Omega_{N1} & \Omega_{N2} & \cdots \; \Omega_{NN}+\frac{1}{C} \end{array}\right]\left[\begin{array}{c} b \\ \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{array}\right] = \left[\begin{array}{c} 0 \\ d_1 \\ d_2 \\ \vdots \\ d_N \end{array}\right] \tag{11}$$

This is exactly what traditional LS–SVM pruning does since it iteratively omits some training points. The information embodied in these points is entirely lost.

To avoid this information loss, one may use the technique referred here as partial reduction.

The proposed partial reduction

In partial reduction, the omission of a training sample $(\mathbf{x}_k, d_k)$ means that only the corresponding column is eliminated, while the row -which defines an input-output relation- is kept. Eliminating the k-th column reduces the model complexity, while keeping the k-th row means that the weighted sum of that row should still meat the d_k regression goal (as closely as possible).

By selecting some (e.g. M, $M < N$) vectors as "*support vectors*", the number of α_i variables are reduced, resulting in

more equations than unknowns. The effect of partial reduction is shown on the next equation, where the removed elements are colored grey.

$$\left[\begin{array}{c|ccc} 0 & \multicolumn{3}{c}{\vec{1}} \\ \hline \vec{1}^T & \Omega_{11}+\frac{1}{C} & \Omega_{12} & \cdots & \Omega_{1N} \\ & \Omega_{21} & \Omega_{12}+\frac{1}{C} & \cdots & \Omega_{2N} \\ & \vdots & \vdots & \ddots & \vdots \\ & \Omega_{(N-1)1} & \Omega_{(N-1)2} & \cdots & \Omega_{(N-1)N} \\ & \Omega_{N1} & \Omega_{N2} & \cdots & \Omega_{NN}+\frac{1}{C} \end{array}\right] \left[\begin{array}{c} b \\ \hline \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{array}\right] = \left[\begin{array}{c} 0 \\ \hline d_1 \\ d_2 \\ \vdots \\ d_N \end{array}\right] \tag{12}$$

This proposition resembles to the basis of the Reduced Support Vector Machines (RSVM) introduced for standard SVM classification in [13].

For further discussions, let's simplify the notations of our main equation as follows:

$$\mathbf{A} = \left[\begin{array}{c|c} 0 & \vec{\mathbf{1}}^{\mathbf{T}} \\ \hline \vec{\mathbf{1}} & \mathbf{\Omega} + C^{-1}\mathbf{I} \end{array}\right], \ \mathbf{u} = \begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix}, \ \mathbf{v} = \begin{bmatrix} 0 \\ \mathbf{d} \end{bmatrix}. \tag{13}$$

The omission of columns with keeping the rows means that the network size is reduced; still all the known constraints are taken into consideration. This is the key concept of keeping the quality, while the equation set is simplified.

It is important to mention that the hyperparameter C is not necessarily needed in case of partial reduction, because as it will be seen later, the overdermined system means that errors inherently are expected at the samples. The original kernel matrix formulation –including the regularization term *C*- is used to show, how our proposition reduces the original formulation, but it can be left out from the formulas entirely. This corresponds to removing the second term from eq. 2., leaving that e_i is optimized through the constraints.

The deleted columns can be selected many ways e.g. randomly, or by using the method proposed in the sequel.

B. Solving the overdetermined system

It is easy to see that partial reduction leads to a sparse solution, but having an overdetermined equation set, has several other advantages. By having more equations than unknowns, we have means to analyze this information set. The solution of this equation set corresponds to a linear fitting problem, where we have to fit an *M*+1-dimensional linear hyperplane on the points defined by the *N* rows of the matrix. Since *N* >> *M*+1, this can be done several ways.

The residual for the *i*-th data point e_i is defined as the difference between the observed response value, the desired response d_i and the fitted response value y_i, and is identified as the error associated with the data. In the geometric interpretation, the residual is the distance of the data sample from the fitted hyperplane.

$$e_i = d_i - y_i \tag{14}$$

The solutions differ in the way they calculate the accumulated error – which is then minimized – from the residuals. The optimal solution depends on the statistical properties of the dataset. (The term statistical here does not necessarily mean a large number of samples, but it means "*more than one*" which is the case in the original formulations.) Some possible solutions:

- Linear least squares
- Weighted linear least squares
  - Custom weighting
  - Robust methods
    - Least absolute residuals (LAR)
    - Bisquare weights
- Interpolation techniques (Linear, Polynomial, Spline)
- Nonlinear fitting

It is important to emphasize that the proposed partial reduction is essential, since it allows us to have more samples than dimensions in the kernel space, which allows us to optimize further in this space.

1) Linear least squares

Usually there are two important assumptions that are made about the noise (*z*):

- The error exists only on the output.
- The errors are random and follow a normal (Gaussian) distribution with zero mean and constant variance σ^2.

In this case we minimize the summed square of the residuals:

$$S = \sum_{i=1}^{N} e_i^2 = \sum_{i=1}^{N} (d_i - y_i)^2 \tag{15}$$

The solution of equation (14) can be formulated as

$$\mathbf{A}^T\mathbf{A}\mathbf{u} = \mathbf{A}^T\mathbf{v}. \tag{16}$$

The modified matrix $\mathbf{A}$ has $(N+1)$ rows and $(M+1)$ columns. After the matrix multiplications the results are obtained from a reduced equation set, incorporating $\mathbf{A}^T\mathbf{A}$, which is of size $(M+1)\times(M+1)$ only. Our proposition, to use partial reduction along with the linear least squares solution has already been presented [7][8], where we named this method LS^2-SVM since it gives the least squares solution of a least squares method.

2) Weighted methods

If the assumption that the random errors have constant variance does not hold, weighted least squares regression may be used. Instead of leveling the errors statistically, it is assumed that the weights used in the fitting represent the differing quality of data samples. The error term is:

$$S = \sum_{i=1}^{N} w_i e_i^2 = \sum_{i=1}^{N} w_i (d_i - y_i)^2 \tag{17}$$

The weighted solution can be formulated as:

$$\mathbf{A}^T\mathbf{W}\mathbf{A}\mathbf{u} = \mathbf{A}^T\mathbf{W}\mathbf{v}. \tag{18}$$

where $\mathbf{W}$ weight matrix is given by the diagonal elements of the w_i weights. The weights are used to adjust the amount of influence each data point has on the estimated linear fit to an appropriate level. This formulation is exactly the same that was reached by Suykens in the Weighted LS-SVM but the way it is derived differs greatly. Suykens introduces different regularization parameters (*C*-s) for the samples, which leads to the same result. In the method proposed here the weights can be calculated from the statistical properties of the points in the

kernel space. Another important difference is that the proposed weighted solution is also *sparse*, so the weighting can be determined from the distribution of many points.

- CUSTOM WEIGHTING - this method can be used if one had *a priori* knowledge about the quality of the samples. If so, weights can be defined, to determine how much each learning sample influences the fit. Samples known to have less noise are expected to fit more, than low-quality ones.
  The weights should transform the response variances to a constant value. If the variances of the data are known, the weights are given by:
  $$w_i = 1/\sigma_i^2 \quad (19)$$
- LEAST ABSOLUTE RESIDUALS (LAR) – this method minimizes the absolute difference of residuals, and not the squared differences. This means that extreme values have less influence on the fit.
- BISQUARE WEIGHTS – a method that minimizes a weighted sum of squares, where the weight of each data point depends on its distance from the fitted line. The farther away is the point, the less weight it gets. This method fits the hyperplane to the bulk of the data with the least squares approach, while it minimizes the effect of outliers. More details on robust regression can be found in [9][10].

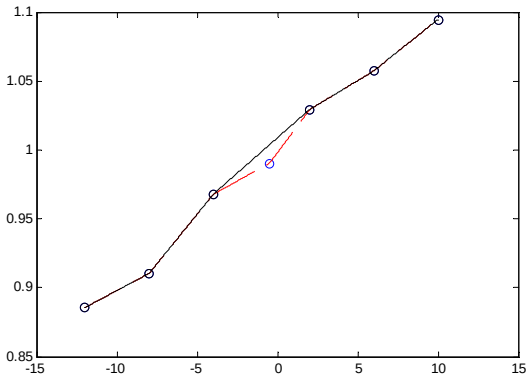


Fig. 2 Linear interpolation and incremental learning

From the above described methods, in our experiments, we will illustrate the bisquare weights method, which can be effectively used in case of outliers. The reason for this is that the effect of this solution is very straightforward and it is very easy to verify the results - namely that the influence of an outlier is reduced.

3) Interpolation techniques

The approximation of a complicated function by a simple function is closely related to interpolation. Interpolation techniques (linear, polynomial, spline), are mentioned here because they allow us to have an exact answer for the training samples even in case of a sparse solution. This may be important in certain cases, e. g. if there is no noise, but a sparse solution is required. Another advantage of these methods is that they can be local, in a sense that a new sample can be inserted, without recalculating the whole system. For example in case of linear interpolation, a new training sample can be inserted with only local effect. Linear interpolation corresponds to having many different weighting sets on the output, which depends on the input (or more directly the corresponding point in the kernel space). Nonlinear fitting means the same, but in that case, the weights change continuously.

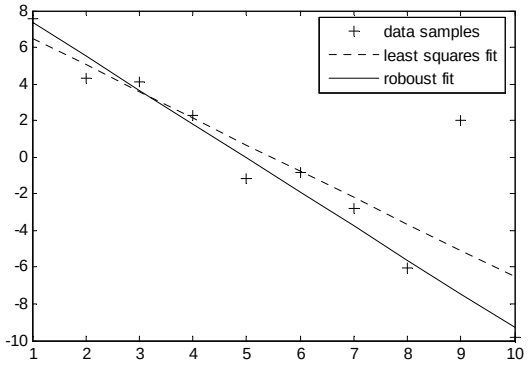


Fig. 3 The least squares and the robust (bisquare) fitting in two dimensions

4) Nonlinear fitting

All methods shown earlier apply a linear regression in the kernel space. Generally, our goal is to map our data to a higher dimensional (kernel) space, where it can be approximated (or separated - in case of classification) linearly. If the dimensionality of the kernel space is not large enough, the data cannot be fit by a linear hyper plane. This can be a result of being "too sparse", which may be the result of an extensive reduction. In this case the data can be fitted more accurately, by a nonlinear in the kernel space. The approximation results can still be calculated, but in this case it will be a result of a nonlinear function of the kernels, instead of the –linear– weighted sum, shown in eq. 7.

This possibility is only mentioned to give a more general feel of the method. The nonlinear fitting is not discussed further, since many questions arise about the applicability, effect, and use of this solution.

C. Selecting support vectors

Standard SVM automatically selects the support vectors. To achieve sparseness by partial reduction, the linear equation set has to be reduced in such a way that the solution of this reduced (overdetermined) problem is the closest to what the original solution would be.

As the matrix is formed from columns we can select a linearly independent subset of column vectors and omit all others, which can be formed as linear combinations of the selected ones. This can be done by finding a "basis" (the quote indicates that this basis is only true under certain conditions defined later) of the coefficient matrix, because the basis is by definition the smallest set of vectors that can solve the

problem.

The basic idea of doing a feature selection in the kernel space is not new. The nonlinear principal component analysis technique, the Kernel PCA uses a similar idea [11]. A basis selection from the kernel matrix has been shown in [12].

This reduced input set (the support vectors) is (are) selected automatically by finding a "basis" of the $\mathbf{\Omega}$ (or the $\mathbf{\Omega} + C^{-1}\mathbf{I}$) matrix. A slight modification of the common mathematical method, used for bringing the matrix to the reduced row echelon form, can be utilized to find a set of vectors that are linearly independent. The linear dependence discussed here, does not mean exact linear dependence, because the method uses an adjustable tolerance value when determining the "resemblance" (parallelism) of the column vectors. The use of this tolerance value is essential, because none of the columns of the coefficient matrix will likely be exactly dependent (parallel).

The reduction is achieved as a part of transforming the $\mathbf{A}^T$ matrix into reduced row echelon form (Gauss-Jordan elimination with partial pivoting) [14][15]. The algorithm uses elementary row operations:

- Interchange of two rows.
- Multiply one row by a nonzero number.
- Add a multiple of one row to a different row.

The algorithm goes as follows:

1. Work along the main diagonal of the matrix starting at row one, column one (*i*-row index, *j*-column index).
2. Determine the largest element *p* in column *j* with row index $i \geq j$.
3. If $p \leq \varepsilon$' (where ε' is the tolerance parameter) then zero out the elements in the *j*-th column with index $i \geq j$; else remember the column index (*j*) because we found a basis vector (support vector). If necessary move the row, to have the pivot element in the diagonal and divide the row with the pivot element *p*. Subtract the right amount of this row from all rows below this element, to make their entries in the *j*-th column zero.
4. Step forward to the next diagonal element ($i=i+1$, $j=j+1$). Go to step 2.

This method returns a list of the column vectors which are linearly independent form the others considering tolerance *ε'*.

The tolerance (*ε'*) can be related to the *ε* parameter of the standard SVM, because it has similar effects. The larger the tolerance, the fewer vectors the algorithm will select. If the tolerance is chosen too small, than a lot of vectors will seem to be independent, resulting in a larger network. As stated earlier the standard SVM's sparseness is due to the *ε*-insensitive margin, which allows the samples falling inside this boundary to be neglected. According to this, it may not be very surprising to find that an additional parameter is needed to achieve sparseness in LS–SVM, and this parameter corresponds to the one, which was originally left when changing from the SVM to the standard least squares solution.

D. Complexity issues

This section deals with the algorithmic issues of the described solutions. LS–SVM training requires the solution of a linear equation set. In case of N training vectors this may be solved using the LU decomposition in $1/3\,N^3 + N^2$ steps, each with one multiplication and one addition. If the training set comprises N points, than the actual size of the equation set is $N+1$, but to keep the formulas simple the effect of the one additional row is neglected. The reduced row echelon form of a matrix can be reached in about N^2 steps. The proposed "support vector" selection is made based on the result of this transformation. Let's assume that the reduction method leads to M selected vectors, and partial reduction is used. In case of calculating the *linear least squares* solution, the calculation of $\mathbf{A}^T\mathbf{A}$ and $\mathbf{A}^T\mathbf{v}$ (defined in (15)) requires M^2N and MN steps respectively. Solving this new equation set costs $1/3\,M^3 + M^2$ steps. So the total cost of the proposed algorithm adds up to: $N^2 + M^2N + MN + 1/3\,M^3 + M^2$. If $M << N$ this means a smaller complexity compared to that of traditional LS–SVM. It is important to mention that even if there is no algorithmic gain, or it is rather small, this calculation provides a sparse solution, with a good performance. If –in order to reach sparseness– the iterative pruning algorithm is applied to the traditional LS–SVM, than an equation set –slowly decreasing in size– must be solved in every step, which multiplies the complexity, whilst the errors may grow.

V. Experiments

The next figures show the results for a *sinc*(x) regression. The training set contains 50 data samples burdened with Gaussian noise.

Fig. 4. demonstrates that by using partial reduction along with the described support vector selection method can lead to almost the same quality result, but much less neurons. The equation set is solved by the linear least-squares method.

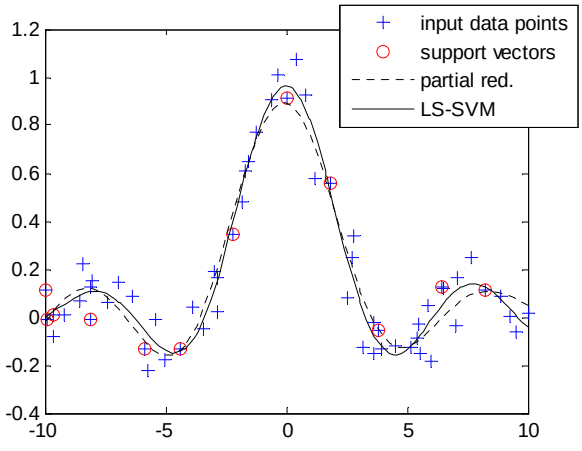


Fig. 4 The continuous black line plots the result for a LS^2-SVM using

the SV selection method described. The dashed line is the original LS-SVM. ($MSE_{partial\ red.}= 4.6*10^{-3}$, $MSE_{LS\text{-}SVM}= 1.5*10^{-3}$)

It is important to mention that the result of the LS^2-SVM is based on a sparse network, containing 12 neurons, which were marked as support vectors. If the number of training samples is very high for the problem complexity, than the gain in the network size can be very large.

The method works for multi-dimensional functions as well (Fig. 5.) The number of training samples is 2500, while the final network consists of only 63 neurons.

a.)

b.)

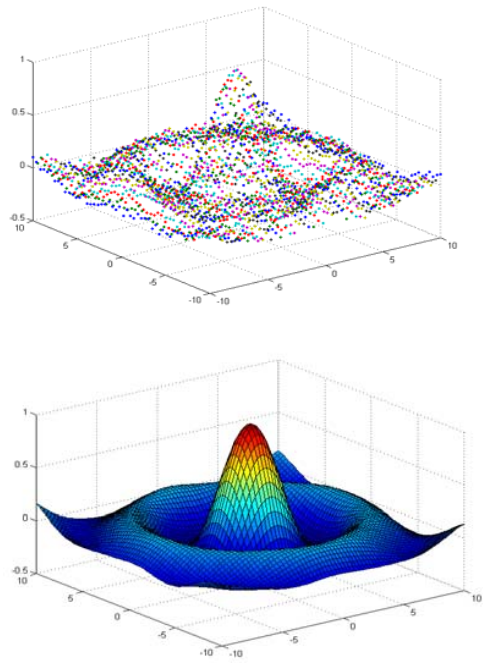


Fig. 5 Approximation of a two-dimensional sinc function. a.) The plotted training samples and b.) the result of the partially reduced LS–SVM, where the support vectors were selected by the proposed method

The dimensionality of the function only affects the calculation of $K(\mathbf{x}_i, \mathbf{x}_j)$, but nothing in the rest of the method. Therefore the described process works irrespectively of dimensionality. It is also independent from the kernel function, since after calculating the kernel matrix, the proposed methods can be applied, without any change.

In the last two experiments (Fig. 4. and Fig. 5.), the complexity of the solution has been reduced greatly, while the mean squared error stayed in the same order of magnitude.

We have proposed that in certain situations alternative solutions of the overdetermined system may be useful.

Figure 6. shows the results of custom weighting. We have 60 samples burdened with Gaussian noise, where the σ of the noise is known for all samples. It can be seen that the effect of noise is reduced. The original LS-SVM is plotted, because the weighted LS-SVM would give almost the same results as the partially reduced solution, but in this case we have a sparse solution.

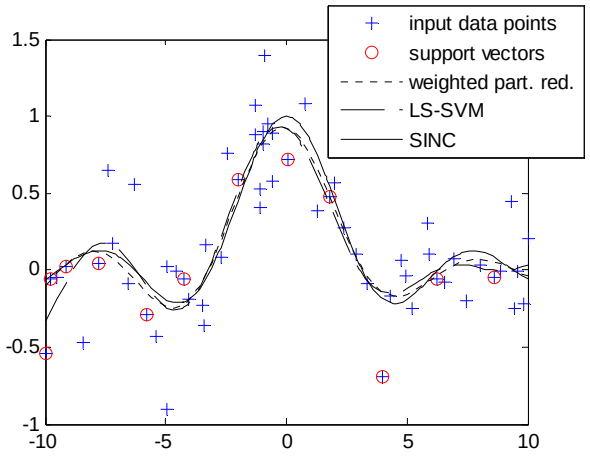


Fig. 6 Custom weighting is applied with partial reduction. ($MSE_{weighted\ part.\ red.}= 2*10^{-3}$, $MSE_{LS\text{-}SVM}= 6*10^{-3}$)

The following experiment (Fig. 7.) shows the same problem as Fig. 4, but in this case a few data points are corrupted to provide outliers.

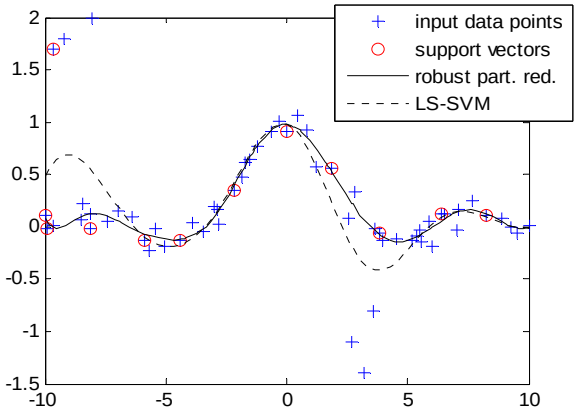


Fig. 7 The continuous black line plots the result for a partially reduced LS-SVM solved by the bisquare weights method. ($MSE_{robust\ part.\ red.}= 2.3*10^{-3}$, $MSE_{LS\text{-}SVM}= 4.6*10^{-3}$)

It can be seen that by using a robust fitting method in the kernel space, the effect of the outliers was successfully reduced. Depending on the properties of noise, or on our *a priori* knowledge, the other fitting methods can also be used successfully.

VI. Conclusion

In this paper an extended view of the least squares support vector machine was presented. The basic idea is that the number of vectors chosen to be centers of kernels, and the number of constraints can be different, which leads to an overdetermined equation set. By achieving this equation set, we have means to analyze, and weight the importance of the constraints defined by the learning samples. This is especially important, to deal with non Gaussian noise.

There describe methods lead to two important results:

- A *sparse* LS-SVM solution
- Further means to *analyze* the problem and *optimize* the solution.

REFERENCES

[1] V. Vapnik, "The Nature of Statistical Learning Theory", New–York: Springer–Verlag., 1995
[2] J. A. K. Suykens, V. T. Gestel, J. De Brabanter, B. De Moor, J. Vandewalle, "Least Squares Support Vector Machines", World Scientific, 2002
[3] J. A. K. Suykens, L. Lukas, and J. Vandewalle, "Sparse approximation using least squares support vector machines", *IEEE International Symposium on Circuits and Systems ISCAS'2000*, 2000
[4] J. A. K. Suykens, L. Lukas, and J. Vandewalle, "Sparse least squares support vector machine classifiers", *ESANN'2000 European Symposium on Artificial Neural Networks*, 2000, pp. 37–42.
[5] J.A.K. Suykens, J. De Brabanter, L. Lukas, and J. Vandewalle, "Weighted least squares support vector machines: robustness and sparse approximation", *Neurocomputing*, 2002. pp. 85-105
[6] B. Schölkopf and A. Smola: Learning with Kernels. Support Vector Machines, Regularization, Optimization, and Beyond. The MIT Press, Cambridge, MA, 2002.
[7] J. Valyon and G. Horváth, „A generalized LS–SVM", SYSID'2003 Rotterdam, 2003, pp. 827-832.
[8] J. Valyon and G. Horváth, „A Sparse Least Squares Support Vector Machine Classifier", Proceedings of the International Joint Conference on Neural Networks IJCNN 2004, 2004, pp. 543-548.
[9] P. W. Holland, and R. E. Welsch, "Robust Regression Using Iteratively Reweighted Least-Squares," *Communications in Statistics: Theory and Methods*, *A6*, 1977, pp. 813-827.
[10] P. J. Huber, *Robust Statistics*, Wiley, 1981.
[11] B. Schölkopf, S. Mika, C.J.C. Burges, P. Knirsch, K.-R. Müller, G. Rätsch, and A. Smola, „Input space vs. feature space in kernel-based methods". *IEEE Transactions on Neural Networks*, 1999, 10(5), pp. 1000–1017.
[12] G. Baudat and F. Anouar, "Kernel-based methods and function approximation". In *International Joint Conference on Neural Networks*, pages 1244–1249, Washington DC, 2001. July 15–19.
[13] Yuh – Jye Lee and O. L. Mangasarian, "RSVM: Reduced Support Vector Machines", *Proceedings of the First SIAM International Conference on Data Mining*, Chicago, 2001. April 5–7.
[14] W. H. Press, S. A. Teukolsky, W. T. Wetterling and B. P. Flannery , "Numerical Recipes in C", Cambridge University Press, Books On-Line, Available: www.nr.com, 2002
[15] H. Golub and Charles F. Van Loan, Matrix Computations", Gene Johns Hopkins University Press, 1989