International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:7, 2007

A Robust LS-SVM Regression

József Valyon, and Gábor Horváth

Abstract—In comparison to the original SVM, which involves a quadratic programming task; LS–SVM simplifies the required computation, but unfortunately the sparseness of standard SVM is lost. Another problem is that LS-SVM is only optimal if the training samples are corrupted by Gaussian noise. In Least Squares SVM (LS–SVM), the nonlinear solution is obtained, by first mapping the input vector to a high dimensional kernel space in a nonlinear fashion, where the solution is calculated from a linear equation set. In this paper a geometric view of the kernel space is introduced, which enables us to develop a new formulation to achieve a sparse and robust estimate.

Keywords—Support Vector Machines, Least Squares Support Vector Machines, Regression, Sparse approximation.

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.

The method effectively reduces the algorithmic complexity, however for really large problems, comprising a very large number of training samples, even this least-squares solution can become highly memory and time consuming.

Whereas 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 sparseness of traditional SVM can also be reached with LS–SVM by applying a pruning method [3][4]. Unfortunately if the traditional LS–SVM pruning method 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 this iterative method multiplies the algorithmic complexity.

The training data is often corrupted by noise, which –if not handled properly– misleads the training. Another modification of the method, called weighted LS–SVM [1][5], is aimed at reducing the effects of non-Gaussian noise (e.g. outliers).

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. The generalized approach presented in this paper enables us to accomplish both goals by allowing a more universal construction and solution of the LS–SVM equation set.

This paper proposes a geometric view of the kernel space and the linear solution that is based on the mapped training samples.

In the LS-SVM solution, the training samples are mapped to a kernel space, where a hyperplane is fitted on these points. In this case all the training samples are used to achieve a result, which consequently isn't sparse. To trade off between training error and a smooth solution a regularization parameter is used, which is the same for all samples, and can be considered as a predefined, intentional error term in the kernel space fitting.

Our proposition is to use a kernel space of smaller dimensionality, which means that by mapping all training samples, the hyperplane can be fitted many ways, since there are more equations than unknown. This means that the position of the hyperplane and consequently the training errors –the distances from this plane– can be automatically determined according to the distribution of many mapped points.

The LS–SVM method is capable of solving both classification and regression problems. The classification approach is easier to understand and more historic. The present study concerns regression, but it must be emphasized that all presented methods can be applied to classification as well.

This paper is organized as follows. Before going into the details LS–SVM, and its extensions of *pruning* and *weighting* are summarized in Section II. Section III provides a geometric interpretation of the kernel space and summarizes the main idea behind the propositions. Section IV contains the details of the solution: It shows how partial reduction is used to achieve an overdetermined equation set, and proposes robust solutions for this.

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

II. A Brief Introduction to the LS-SVM Method

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 $d_i = y_i + z_i$, $d_i \in \Re$ is a scalar measured output, which represents the y_i system output corrupted by some z_i noise. Our goal is to

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

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).

construct an $y = f(\mathbf{x})$ function, which represents the dependence of the output y_i on the input $\mathbf{x}_i$. Let's define the form of this function as formulated below:

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

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

The $\boldsymbol{\varphi}(.): \Re^p \rightarrow \Re^h$ 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 that LS-SVM involves equality constraints instead of inequality ones and works with a least squares cost function [1]. The optimization problem and the inequality constraints are defined by the following equations ($i = 1,...,N$):

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

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

The $C \in \Re^+$ is the trade–off parameter between a smoother solution, and training errors. From this, a Lagrangian is formed

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

The solution concludes in a constrained optimization, where the conditions for optimality lead to the following overall solution:

$$\begin{bmatrix} 0 & \vec{\mathbf{1}}^{\mathbf{T}} \\ \vec{\mathbf{1}} & \boldsymbol{\Omega} + C^{-1}\mathbf{I} \end{bmatrix} \begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{d} \end{bmatrix},$$

$$d = [d_1, d_2, ..., d_N]., \ \boldsymbol{\alpha} = [\alpha_1, \alpha_2, .., \alpha_N], \tag{5}$$

$$\vec{\mathbf{1}} = [1, ..., 1],\ \Omega_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\varphi}^T(\mathbf{x}_i)\boldsymbol{\varphi}(\mathbf{x}_j),$$

where $K(\mathbf{x}_i, \mathbf{x}_j)$ is the kernel function, and $\boldsymbol{\Omega}$ is the kernel matrix. The result is:

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

A detailed description of LS-SVM can be found in refs. [2]-[5].

LS-SVM pruning - One of the main drawbacks of the least–squares solution is that, it is not sparse, because –unlike the original SVM– it incorporates all training vectors in the result. In order to get a sparse solution, a pruning method must be used. Since the $|\alpha_i|$ support values are proportional to the errors at data points:

$$\alpha_i = Ce_i, \tag{7}$$

the irrelevant points are left out, by iteratively leaving out the least significant vectors. These are the ones corresponding to the smallest $|\alpha_i|$ values.

In the case of the classical SVM sparseness is achieved by the use of such loss functions, where errors smaller than ε are ignored (e.g. ε-insensitive loss function). This method reduces the difference between SVM and LS–SVM, because the omission of some data points implicitly corresponds to creating an ε-insensitive zone [1].

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 each iteration. Pruning is especially important if the number of training vectors is large. In this case however, the iterative method is not very effective.

Weighted LS-SVM - This method 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 an unweighted– solution. The method uses a bottom-up approach by starting from a standard solution, and calculating one or more weighted LS–SVM based on the previous result. The weighted LS–SVM is formed as:

$$\begin{bmatrix} 0 & \vec{\mathbf{1}}^{\mathbf{T}} \\ \vec{\mathbf{1}} & \boldsymbol{\Omega} + \mathbf{V}_C \end{bmatrix} \begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{d} \end{bmatrix} \tag{8}$$

where

$$\mathbf{V}_C = diag\left\{ \frac{1}{Cv_1}, ..., \frac{1}{Cv_N} \right\}. \tag{9}$$

The v_i weighting is designed such, that the results improve in view of robust statistics. Large e_i-s mean a small weight 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 learning. 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.

III. The Main Idea

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

1. The samples are mapped to an N+1 dimensional kernel space, where N dimensions are defined by the kernel functions and one is the desired output.
2. A hyperplane is fitted on these mapped samples. The hyperplane is determined by N mapped points, and one additional constraint ($\sum_{i=1}^{N} \alpha_i = 0$).

The approximated answer for a new sample results from mapping it into N dimensions, and calculating the corresponding point on the fitted hyperplane (dimension N+1). Therefore in case of this solution, we expect that when a new sample is transformed to this kernel space, the desired output will be close to this hyperplane. For the sake of generalization and to avoid overfitting the accuracy of the fit can be adjusted through regularization (the C hyperparameter).

Some questions still need answers: How many –and which–

dimensions are needed in the kernel space? How good is my approximation, for a specific mapping? What is a good value of *C*, or more generally, how should the hyperplane be placed in the kernel space?

Our proposition is to take control of the problem in the kernel space by:

- controlling the dimensionality of this space (see IV. A.),
- finding a better linear hyperplane in the kernel space (see IV. B.),
- choosing an appropriate kernel space (IV. C.).

Having fewer dimensions in the kernel space, results in a sparse solution, while at the same time it increases the number of mapped points that can be used to determine the linear fit. Having more points than dimensions in the kernel space allows us to optimize the linear fit. The dimensionality of the kernel space high is enough, if samples (not used in determining this space) fall close to this plane after mapping (see Fig. 1.).

a.)

b.)

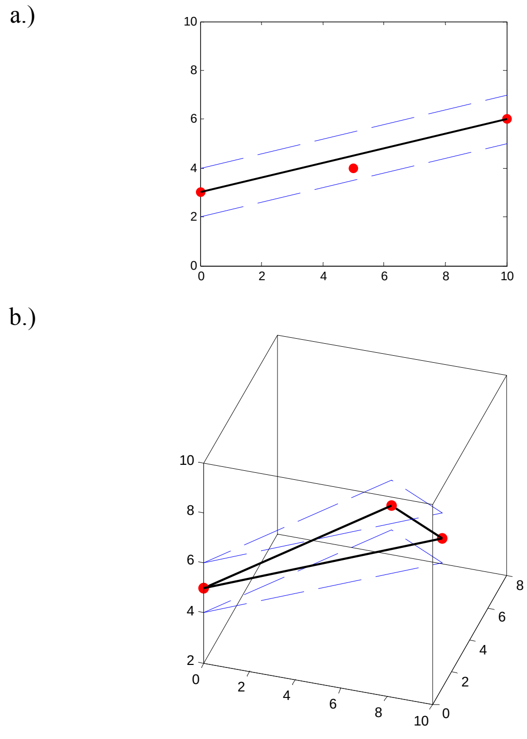

Fig. 1 The image of training samples in a kernel space of different dimensions. Using all three samples as support vectors (kernel centers), a three dimensional kernel can space guarantee exact fit for the samples. The dashed lines represent a zone in which errors can be accepted (corresponding to the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

IV. The Proposed Methods

This section proposes some modifications and extensions to the standard LS–SVM. Their purpose is to gain control over network size, to reduce 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 and *b*, $(N+1)$ 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 input pairs. 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.

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 $\sum_{i=1}^{N}\alpha_i=0$ condition 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. It can be seen that in order to reach sparseness the number of columns 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. This is exactly the case in traditional LS–SVM pruning since pruning 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 also 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.

$$\left[\begin{array}{c|cccc} 0 & & \vec{1} & & \\ \hline \vec{1}^T & \Omega_{00}+\frac{1}{C} & \Omega_{01} & \cdots & \Omega_{0N} \\ & \Omega_{10} & \Omega_{11}+\frac{1}{C} & \cdots & \Omega_{1N} \\ & \vdots & \vdots & \ddots & \vdots \\ & \Omega_{(N-1)0} & \Omega_{(N-1)1} & \cdots & \Omega_{(N-1)N} \\ & \Omega_{N0} & \Omega_{N1} & \cdots & \Omega_{NN}+\frac{1}{C} \end{array}\right] \left[\begin{array}{c} b \\ \hline \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_N \end{array}\right] = \left[\begin{array}{c} 0 \\ \hline d_0 \\ d_1 \\ \vdots \\ d_N \end{array}\right] \tag{11}$$

This proposition resembles to the basis of the Reduced Support Vector Machines (RSVM) introduced for standard SVM classification in [6].

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

$$\mathbf{A} = \left[\begin{array}{c|c} 0 & \vec{\mathbf{1}}^{\mathbf{T}} \\ \hline \vec{\mathbf{1}} & \mathbf{\Omega}+C^{-1}\mathbf{I} \end{array}\right],\ \mathbf{u} = \left[\begin{array}{c} b \\ \hline \boldsymbol{\alpha} \end{array}\right],\ \mathbf{v} = \left[\begin{array}{c} 0 \\ \hline \mathbf{d} \end{array}\right]. \tag{12}$$

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. As it will be seen later, the overdetermined system means that errors are inherently expected at the samples. C is used to show how our proposition reduces the original formulation, but it can be left out from the formulas entirely.

The dimensionality of the $\mathbf{x}_i$ input vectors 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 the input dimensionality. It is also independent from the kernel function, since after calculating the kernel matrix, the proposed methods can be applied without any change.

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 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 corresponds to the e_i error, which is defined as the difference between the observed desired response value d_i and the fitted response value y_i.

$$e_i = d_i - y_i \tag{13}$$

The solutions differ in the way they calculate the accumulated error (residuals), which is then minimized. 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 (for Gaussian noise LS^2–SVM)
- ♦ Weighted linear least squares
  - Custom weighting
  - Robust bisquare weights method

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 optimizing 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:

$$S = \sum_{i=1}^{N} e_i^2 = \sum_{i=1}^{N} (d_i - y_i)^2 \tag{14}$$

The solution of equation (12) can be formulated as

$$\mathbf{A}^T\mathbf{A}\mathbf{u} = \mathbf{A}^T\mathbf{v} \tag{15}$$

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 have already been presented in [7] and [8], where we named this method LS^2–SVM, since it gives the least squares solution of a least squares SVM 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:

$$S = \sum_{i=1}^{N} w_i e_i^2 = \sum_{i=1}^{N} w_i (d_i - y_i)^2 \tag{16}$$

The weighted solution can be formulated as:

$$\mathbf{A}^T\mathbf{W}\mathbf{A}\mathbf{u} = \mathbf{A}^T\mathbf{W}\mathbf{v}\,. \tag{17}$$

where the $\mathbf{W}$ weight matrix is:

$$\mathbf{W} = diag\{w_1, \dots, w_N\}\,. \tag{18}$$

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 [1] but the way it is derived differs greatly. Suykens introduces different regularization parameters (Cv_i-s) for the samples, while in the proposed method the weights are introduced in the linear fitting method. The most important difference, however is that the use of partial reduction leads to an overdetermined system, so the weights can be calculated from the statistical properties of the points (distribution of many points) in the kernel space. Another important difference is that the proposed weighted solution is also *sparse*.

- 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:
  $$w_i = 1/\sigma_i^2 \,. \tag{19}$$
- 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 (Fig. 2.). More details on robust regression can be found in [9][10].

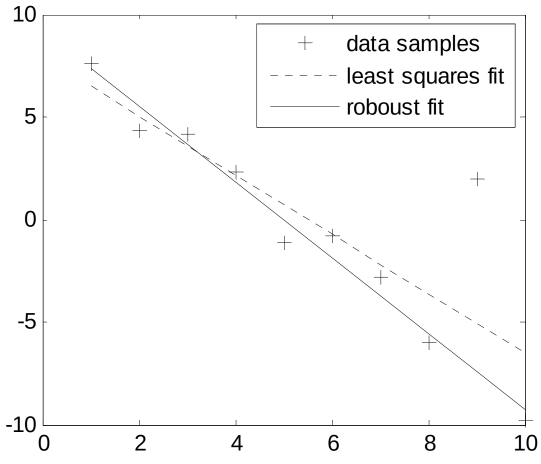

Fig. 2 The least squares and the roboust (bisquare) fitting in two dimensions

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 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, using a slight modification of Gauss-Jordan elimination with partial pivoting [11]. This method returns a list of the column vectors which are linearly independent form the others considering a 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 zone, 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.

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 [12]. A basis selection from the kernel matrix has been shown in [13].

V. EXPERIMENTS

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

Fig. 3 shows the results of custom weighting. We have 60 samples with additive 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.

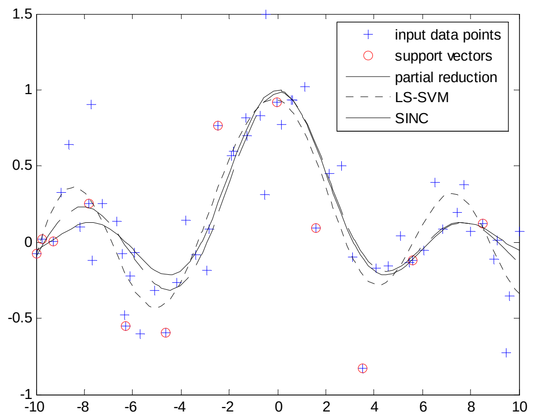

Fig. 3 Custom weighting is applied with partial reduction (The

LS–SVM is not weighted.)

The following experiment (Fig. 4) shows the same problem as Fig. 3, but in this case a few data points are corrupted to provide outliers.

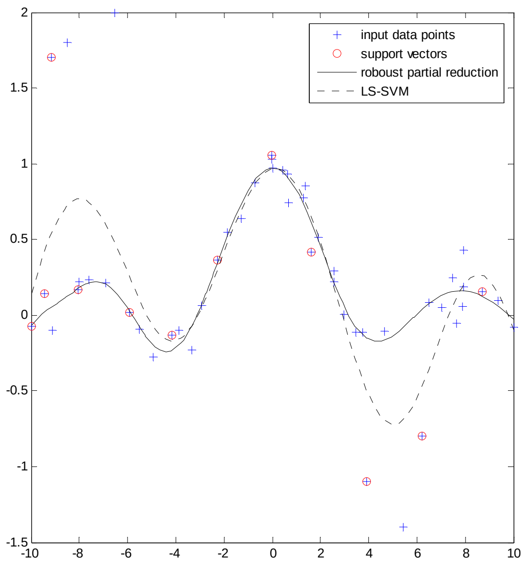

Fig. 4 The continuous black line plots the result for a partially reduced LS-SVM solved by the bisquare weights method. The dashed line is the original LS–SVM

It can be seen that by using a robust bisquare fitting, the effect of the outliers was successfully reduced.

It is important to mention that the result of the LS^2–SVM is sparse, consisting of only 12 support vectors. If the number of training samples is very high for the problem complexity, than the gain in the network size can be rather large.

VI. Conclusion

In this paper a geometric view and a generalized formulation of the least squares support vector machine was presented. The basic idea is that by reducing the dimensionality of the kernel space, the hyperplane fitted to the mapped training samples can be optimized according to their distribution. This is especially important, to deal with non Gaussian noise.

The described solution achieves two important results simultaneously:

- ♦ a *sparse* LS–SVM solution,
- ♦ the effect of noise is reduced.

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] 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.

[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] Holland, P. W., and R. E. Welsch, "Robust Regression Using Iteratively Reweighted Least-Squares," *Communications in Statistics: Theory and Methods*, *A6*, 1977, pp. 813-827.

[10] Huber, P. J., *Robust Statistics*, Wiley, 1981.

[11] 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

[12] 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.

[13] 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.

[14] H. Golub and Charles F. Van Loan, Matrix Computations”, Gene Johns Hopkins University Press, 1989.