Least Squares Support Vector Machines for Data Mining

JÓZSEF VALYON, GÁBOR HORVÁTH

Budapest University of Technology and Economics,
Department of Measurement and Information Systems

{valyon, horvath}@mit.bme.hu

Keywords: Support Vector Machines, Least Squares Support Vector Machines, function approximation, time series prediction

Due to the extensive use of databases and data warehouses; it is very easy to collect large quantities of data from any aspect of life. The analysis of these data may lead to many useful or interesting conclusions. Data mining is a collection of methods and algorithms that can effectively be used to discover implicit, previously unknown, hidden trends, relationships, patterns in masses of data. Classical data mining uses many methods from different fields, like the foundations of linear algebra, graph theory, database theory, machine learning and artificial intelligence. In this paper, we address the problem of black-box modeling, where the model is built based on the analysis of input-output data. These problems usually cannot be addressed analytically, so they require the use of soft computing methods. We focus on the use of a special type of Neural Network (NN), the least squares version of Support Vector Machines (SVMs), the LS-SVM. We also present an extension of this method, the LS^2-SVM, which enables us to deal with large quantities of data. Using this method we present solutions for function approximation, time series analysis, and time series prediction.

1. Introduction

Most of the real-life problems concern very complex systems, whose exact properties, inside functioning and operation are unknown. Such problems are often found in the field of medical research and diagnosis, in industrial and economic problems, etc. During the operation of such systems a large number of input and output data samples may be collected, which can be processed to extract knowledge or to create a model of the system. This is called black-box modeling.

The goal of data mining is to extract implicit, previously unknown and useful information from large data sets accumulated in databases or data warehouses.

The tools most commonly used in data mining contain many different soft computing techniques, inclusive of neural networks [1],[2]. Neural networks can be effectively used for nonlinear function approximation (regression) problems. This paper deals with some special types of networks, namely Support Vector Machines [3],[4]. The main idea of support vector machines is to map the complex nonlinear primal problem – or to be exact, the data samples – with the use of nonlinear transformations into a higher dimensional space, where a linear solution can be found.

The main advantage of this method is that it guarantees an upper limit on the generalization error of the resulting model. Another important property of the method is that the training algorithm seeks to minimize the model size and creates a sparse model. This is a trade-off problem between model complexity and the approximation error that can be is controlled by a hyper parameter.

The biggest problem with the traditional SVM is its high algorithmic complexity and memory requirement, caused by the use of a quadratic programming. This shuts out large datasets and data mining. Many different solutions have been introduced to overcome this problem. These are mostly iterative solutions, breaking down the large optimization problem into a series of smaller tasks. The different “chunking” algorithms differ in the way they decompose the problem [5-7].

Another possibility to use the Least Squares Support Vector Machine, where the algorithmic problems are tackled by replacing quadratic programming by a simple matrix inversion. The price paid for this simplicity is the loss of sparseness, thus all samples are embodied in the resulting – and, therefore, large – model.

Data mining problems incorporate very large data sets; therefore it is extremely important that – in contrast to traditional LS-SVM – the size of the resulting model must be independent of the training set size.

In the sequel, we propose some modifications for the LS-SVM that allows us to control the network size, while at the same time they simplify the formulations, speed up the calculations and/or provide better results.

In the field of black-box modeling, there are two general problem types that must be discussed: (a) *function approximation* (regression), and in case of dynamic systems (b) *time series prediction*.

Function approximation

The available data set is analyzed to find, and determine mathematical relations among them. In case of N p-dimensional samples one must find how one of the variables (which is mostly the known output) depends on the p-1 others (or any subset of them). In this case there is no ordering (sequence – e.g. time) in the data, the system is expected to be static, thus the output depends only on the actual inputs.

Time series prediction

It is assumed, that the system has a memory (e.g. it contains feedback connections), so it is dynamic. This means that in time series prediction problems, the input and output data must have an ordering and the model constructed should represent the dynamics of the process. This way a proper model can continue a data series by predicting the future values.

This problem can be generalized, since the prediction may be done along any variable (not only the time). Most of the real life systems are dynamic, where the output depends not only on the inputs, but the current state of the system. This case can also be handled as a regression (function approximation), but the input variables are extended to include earlier inputs and/or outputs.

Section 2 shows a common way to convert a time series prediction problem to a function approximation one. Section 3 describes the LS-SVM regression, and the proposed LS-LS-SVM (the LS^2-SVM). Section 4 of this paper contains some experiments and then finally the conclusions are drawn.

2. Creating a data set for dynamic problems

In a time series prediction problem, a series of output values changing in time must be predicted, based on some earlier values and sometimes some other inputs. In cases like this, the approximating model must also have some dynamics. The easiest way to achieve this is to take a (usually nonlinear) static model and extend it with some dynamic components (e.g. delays or feedback paths). Probably the most common solution for adding external dynamic components is the use of tapped delay lines, as shown below.

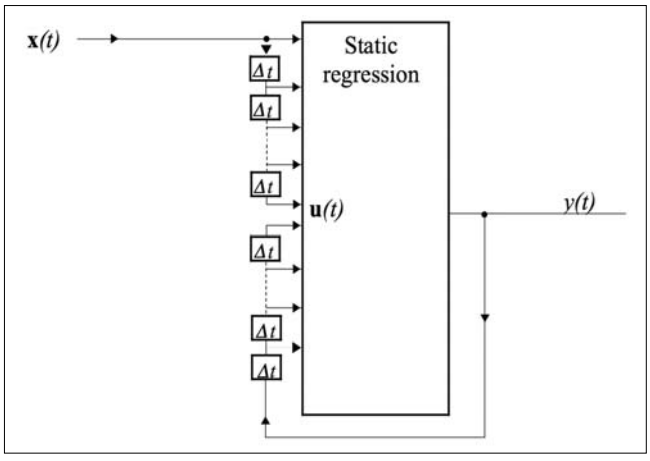


Figure 1. A static system that is made dynamic by delays

As it can be seen in the figure, the static system can be made dynamic, by extending the inputs with delays. The N dimensional $\mathbf{x}$ vectors are expanded to $N+K$ dimensions, by incorporating certain past input and output values in the new model input.

$$\mathbf{u}^T(t) = [\, x_1(t), x_1(t - T_{11}), \ldots, x_1(t - T_{K_1 1}), \ldots, x_i(t), x_i(t - T_{1i}), \ldots, x_i(t - T_{K_i i}), \ldots, y(t - T_{1Y}), \ldots, y(t - T_{K_Y Y})\,] \quad (1)$$

Where

$x_i(t)$ is the i-th input in the t-th time step,
$x_i(t - T_{1i})$ is the i-th input in the t-T_{1i} time step,
$[T_{1i} \ldots T_{K_i i}]$ is the collection of delays (i=1...N) for the i-th input,
K_i is the number of delays for the i-th input,
$\mathbf{u}^T(t)$ is the N+K dimensional vector at time step t,
$[T_{1Y} \ldots T_{K_Y Y}]$ is the collection of delays for the y output,
K_i is the number of delays for the y output,
y is the output.

The method described above is very general, since it allows different delays for all inputs and the output. In practice this is usually simplified:

– by the use of the same delays for all input components,
– by using a sliding window (e.g. the last n samples) instead of custom delays,

With the use of this extended input vector, the originally dynamic problem can be handled by a static regression. After training in the recall phase the model's output is also calculated based on an extended input vector. This means that the result must be calculated iteratively, since the previous results may be needed as an input to predict the next output.

3. Function approximation

There are a large number of different methods for function approximation. These methods can be organized by many different aspects, but based on the field of use and the implementation, the following characterization is emphasized:

- Linear regression
- Non-linear regression

This categorization is especially important, because the kernel based methods discussed here transform the nonlinear regression problem to address it in a linear manner as a much easier linear regression.

The problem is the following in both cases:

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 $d_i \in \Re$ is the scalar target output, our goal is to construct a $y = f(\mathbf{x})$ function, which represents the dependence of the output d_i on the input $\mathbf{x}_i$.

Besides the well known analytic methods (linear/polynomial regression, splines etc.), the function approximation problem can also be solved by neural networks [1],[2]. The described LS-SVM can be considered as a special kind of neural solution [8].

As mentioned earlier, Support Vector Machines use nonlinear transformations to transform the input sam-

ples to a higher dimensional space, where a linear solution is constructed. Whilst doing this, the SVM selects a subset of the samples – the support vectors – as relevant for the solution and discards the rest. This property is called sparseness [9].

In order to simplify the calculations, the LS-SVM sacrifices this, therefore its model is often unnecessarily large. In this paper a reduction method is proposed, which enables us to achieve a sparse solution, while at the same time the algorithmic complexity is further reduced [10],[11].

3.1. LS-SVM regression

The Least Squares SVM (LS-SVM) is a modification of Vapnik's traditional Support Vector Machines (SVMs) [3]. In this formulation the solution is obtained by solving a linear set of equations, instead of solving a quadratic programming problem involved by standard SVM. The main advantage of this is that algorithmic complexity is reduced.

The solution is formed $y(\mathbf{x}) = \mathbf{w}^T \varphi(\mathbf{x}_i) + b$,

where $\varphi(.): \Re^n \rightarrow \Re^{n_h}$ is a mostly non-linear function, which maps the input data into a higher – possibly infinite – dimensional feature space. The data is transformed into a higher dimensional space, where the solution is calculated according to the (**w** and *b*) parameters resulting from the linear solution of the training. To minimize the generalization error, an additional constraint – the minimization of the length of **w**, that is $w^T w$ – is introduced.

The optimization problem and the inequality constraints are defined by the following equations (i = 1,...,N):

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

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

The $C \in \Re^+$ is the trade-off parameter between a smoother solution, and training errors. From this, a Lagrangian is formed (3):

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

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

$$\begin{bmatrix} 0 & \vec{\mathbf{1}}^{\mathbf{T}} \\ \vec{\mathbf{1}} & \boldsymbol{\Omega} + C^{-1}\mathbf{I} \end{bmatrix} \begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{d} \end{bmatrix},$$
$$d = [d_1, d_2, ..., d_N]., \ \boldsymbol{\alpha} = [\alpha_1, \alpha_2, .., \alpha_N] \,, \qquad (4)$$
$$\vec{\mathbf{1}} = [1, ..., 1] \,, \ \Omega_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j) = \varphi^T(\mathbf{x}_i)\varphi(\mathbf{x}_j)$$

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

The result is:

$$y = \sum_{i=1}^{N} \alpha_i K(\mathbf{x}, \mathbf{x}_i) + b \qquad (5)$$

Where α_k and b come from the solution of Eq. 5. It can be seen that the achieved solution is linear but the nonlinear mapping is replaced by a new $K(.,x_i)$ kernel function, which is obtained as the dot product of the φ(.)-s.

The training of a support vector machine is a series of mathematical calculations, but the equation used for determining the machine's answer for a given input represents similar calculations as those of a one hidden layer neural network. Although in practice SVMs are rarely formulated as actual networks, this neural interpretation is important, because it provides an easier discussion framework than the purely mathematical point of view. *Figure 2* illustrates the neural interpretation of an SVM.

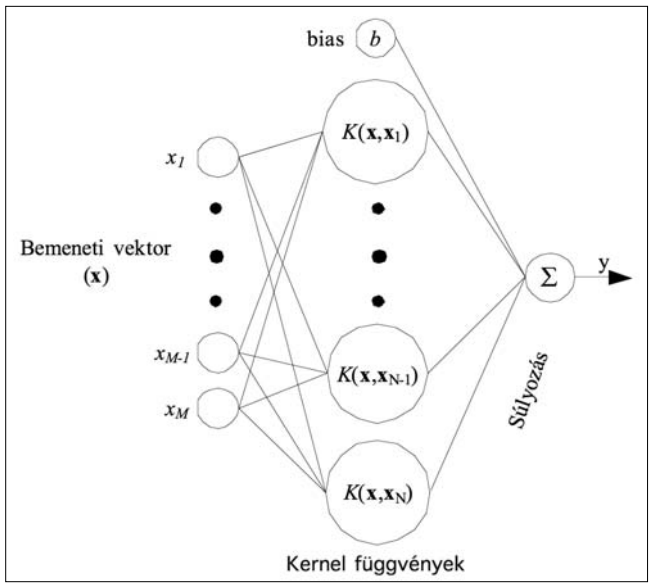


Figure. 2. A neural network that corresponds to an SVM

The input is an *M*-dimensional vector. The hidden layer implements the mapping from the input space into the kernel space, where the number of hidden neurons equals to the number of selected training samples, the support vectors. In this interpretation, the network size of a standard SVM – because of the sparseness of this solution – is determined by the number of support vectors, which is usually much smaller than the number of all training samples (in case of LS-SVM the number of neurons equals to the number of training samples *N*).

The response of the network (y) is the weighted sum of the hidden neurons' outputs, where the α_i weights are the calculated Lagrange multipliers.

3.2. LS²-SVM regression

The main goal of an LS²-SVM is to reduce model complexity, to reduce the number of neurons in the hidden layer. As a side effect, the algorithmic complexity of the required calculation is also reduced.

The starting point of the new method is Eq. 4 which is modified by the following two steps: *(i)* The *first step* reformulates the LS-SVM solution to use only a subset of the training samples as "support vectors'. We also

show that the resulting overdetermined system can still be solved. (ii) In the *second step* an automatic "support vector" selection method is proposed.

Using an overdetermined equation set

If the training set consists of N samples, then our original linear equation set will have (N+1) unknowns, the α_i-s and b, (N+1) equations and (N+1)^2 coefficients. These factors are mainly the values of the $K(\mathbf{x}_i,\mathbf{x}_j)$ i,j = 1,...,N 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.

Let's take a closer look at the linear equation set describing the regression problem. The first row means:

$$\sum_{k=1}^{N}\alpha_k = 0\,, \tag{6}$$

and the j-th row stands for the:

$$b+\alpha_1 K(\mathbf{x}_j,\mathbf{x}_1)+\ldots+\alpha_k\left[K(\mathbf{x}_j,\mathbf{x}_k)+C^{-1}\mathbf{I}\right]+\ldots$$
$$\ldots+\alpha_N K(\mathbf{x}_j,\mathbf{x}_N)=d_j \tag{7}$$

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 $\sum_{i=1}^{N}\alpha_i = 0$ 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 $\mathbf{\Omega}+C^{-1}\mathbf{I}$ kernel matrix; its elements are the results of the kernel function for pairs of training inputs ($\Omega_{i,j}$= $K(\mathbf{x}_i,\mathbf{x}_j)$).To reduce the number of elements in $\mathbf{\Omega}$, one column, one row, or both (a column and a corresponding row) may be eliminated. The rows, however, represent the constraints (input-output relations) that the solution must satisfy. The following two reduction techniques can be used on the regularized $\mathbf{\Omega}+C^{-1}\mathbf{I}$ matrix:

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. The next equation demonstrates how the equation changes by fully omitting some training points. The deleted elements are colored grey.

$$\begin{bmatrix} 0 & \vec{1} \\ \vec{1} & \mathbf{\Omega}+C^{-1}\mathbf{I} \end{bmatrix}\begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix}=\begin{bmatrix} 0 \\ \mathbf{d} \end{bmatrix} \tag{8}$$

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.

Proposed partial reduction – a training sample $(\mathbf{x}_j,d_j)$ is only partially omitted, by eliminating the corresponding j-th column, but keeping the j-th row. The corresponding input-output relation is still in effect, which means that the weighted sum of that row should still meet the d_j (regression) goal, as closely as possible.

By selecting some (e.g. M, $M < N$) vectors as *"support vectors"*, the number of columns is reduced, resulting in more equations than unknowns. The effect of this reduction is shown in the next equation, where the removed elements are colored grey.

$$\begin{bmatrix} 0 & \vec{1} \\ \vec{1}^T & \mathbf{\Omega}+C^{-1}\mathbf{I} \end{bmatrix}\begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix}=\begin{bmatrix} 0 \\ \mathbf{d} \end{bmatrix} \tag{9}$$

As a consequence of partial reduction, our equation set becomes overdetermined, which can be solved as a linear least-squares problem, consisting of only (M+1)x(N+1) coefficients. Let's simplify the notations of our main equation as follows:

$$\mathbf{A}=\begin{bmatrix} 0 & \vec{1}^{\mathrm{T}} \\ \vec{1} & \mathbf{\Omega}+C^{-1}\mathbf{I} \end{bmatrix},\quad \mathbf{u}=\begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix},\quad \mathbf{v}=\begin{bmatrix} 0 \\ \mathbf{d} \end{bmatrix}. \tag{10}$$

then the solution will be

$$\mathbf{u}=\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{v}\,. \tag{11}$$

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.

The modified, reduced LS-SVM equation set is solved in a least squares sense, therefore we call this method Least Squares LS-SVM or shortly LS^2-SVM. This proposition resembles to the basis of the Reduced Support Vector Machines (RSVM) introduced for standard SVM in [12]. The RSVM also selects a subset of the samples as possible delegates to be support vectors, but the selection method, the solution applied and the purpose of this reduction differs from the propositions presented. Since SVM is inherently sparse, the purpose of this selection is to reduce the algorithmic complexity, while our main goal is to achieve a sparse LS-SVM.

Selecting support vectors

To achieve sparseness by the above described partial reduction, the linear equation set has to be redu-

ced 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 tolerance parameter indirectly controls the number of resulting basis vectors (M). This number does not really depend on the number of training samples (N), but only on the problem, since M only depends on the number of linearly independent columns. In practice it means that if the complexity of a problem requires M neurons, then no matter how many training samples are presented, the size of the resulting network does not change.

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 [13],[14].

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 a tolerance ε'.

The problem of choosing a proper ε' resembles the selection of Vapnik's SVM hyper-parameters, like of C, ε and the kernel parameters. One possibility is to use cross-validation. With a larger tolerance value, we can achieve smaller networks, but consequently the error of the estimation grows.

It is easy to see that the selection of the ε' tolerance is a trade-off problem between network size and performance. It is also important to emphasize that by using 0 tolerance, LS^2-SVM and LS-SVM are equivalent, since all of the input samples will be kept by the described selection method.

4. Experiments

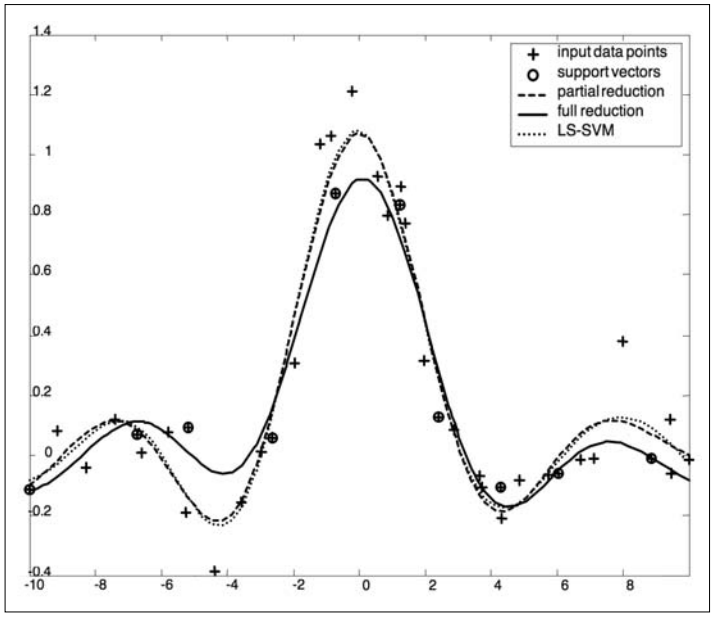
The results will be demonstrated on the most commonly used benchmark problem in the literature of LS-SVM. Most of the experiments where done with the *sinc(x)* function in the [-10,10] domain. The kernel is Gaussian like (RBF kernel), where $\sigma = \pi$. The tolerance (ε') is set to 0.2 and C =100.

The samples are corrupted by additive Gaussian output noise of zero mean and standard deviation of 0.1.

First the effects of partial reduction are examined. This is extended with the automatic selection method. The same problem is used to compare the described methods to the original solutions (LS-SVM and pruned LS-SVM), but a more complex time series prediction problem – the Mackey-Glass chaotic time series prediction problem – is also described.

To compare the reduction methods first an extremely simple support vector selection method is applied: every forth input is chosen from the 40 samples (the 40 sample points are randomly selected from the domain). *Figure 3* shows the result of the partial- and the full reduction plotted together, along with the original – unreduced – LS-SVM.

Figure 3.
The different reduction methods plotted together



It can be seen that the partial reduction gave the same quality result as the original LS-SVM, while in this case the complexity is reduced to its one fourth. The fully reduced solution is only influenced by the "support vectors", which can be easily seen on the figure. In this case the resulting function is burdened with much more error.

The original unreduced LS-SVM almost exactly covers the partial reductions' dotted line ($MSE_{partial\ red.}$: $1.04x10^{-3}$, $MSE_{full\ red.}$: $6.43x10^{-3}$, $MSE_{LS\text{-}SVM}$: $1.44x10^{-3}$).

The "support vectors" may be selected automatically, by the use of the proposed selection method. *Figure 4* shows a solution that is based on the automatically selected support vector set.

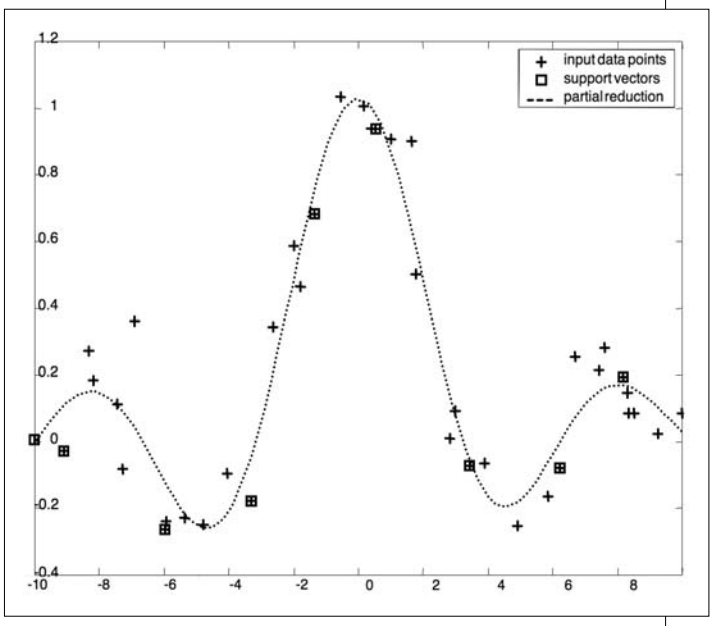


Figure 4.
A partially reduced LS-SVM, where the support vectors were selected by the proposed method (ε'= 0.2)

Since 40 samples were provided, the original network would have 40 nonlinear neurons, while the reduced net incorporates only 9. An even more appealing property of the proposed solution is that the cardinality of the support vector set is indeed independent from the number of training samples. If the problem can be solved with *N* neurons in the hidden layer, then no matter how many inputs are presented, the network size should not change.

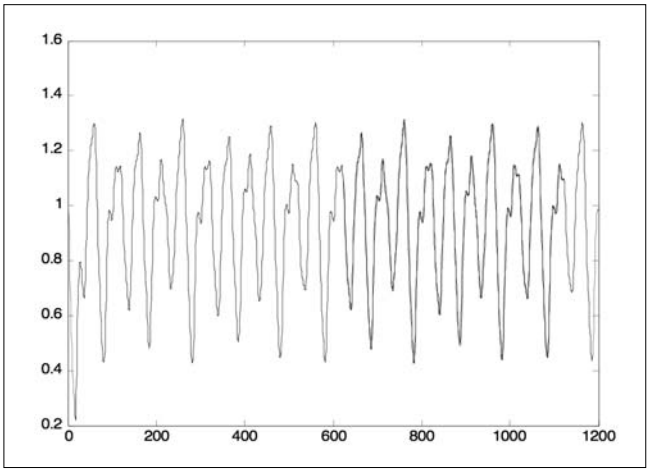


Figure 5.
An LS-SVM approximation of the Mackey-Glass time series prediction problem

The *Table* shows the number of "support vectors" calculated by the algorithm for different training set sizes of exactly the same problem (*sinc*(*x*) with matching noise etc. parameters). The mean square errors tested for 100 noise-free samples are also shown. It can be seen that by increasing the number of training samples, the error decreases – as expected –, but the network size does not change significantly.

The next figure *(Figure 5)* shows a solution to the widely used Mackey-Glass time series prediction problem. In the prediction we have used the [-6,-12,-18,-24] delays, thus the **x**(*t*) value of the *t*-th time instant is approximated by four past values (in Mackey-Glass process, there is no input – the output only depends on the past values). In this experiment the training is done by using 500 training samples.

This experiment shows that the above/described solution along with LS-SVM regression is applicable to solve time series prediction problems.

5. Summary

In this paper, the possibilities of data mining applications of LS-SVMs are investigated. We have shown the original formulation of LS-SVM and proposed some

Table:
The number of support vectors, and the mean squared error calculated for different training set sizes of the same problem using the proposed methods (the tolerance was set to 0.25)

Number of training samples *(This would also be the network size in case of standardLS-SVM)*	Number of "support vectors" *(Network size using the proposed reduction method)*	The mean square error *(The approximation error for the proposed method)*
40	8	1.890×10^{-3}
80	9	0.877×10^{-3}
800	9	0.155×10^{-3}
1600	9	0.029×10^{-3}

modifications to extend it, in order to achieve a small, sparse model, even in the case of large datasets. This paper also shows a simple method to solve a time series prediction problem through simple static regression, which can be solved by analytic methods, or by the neural model (e. g. the described LS-SVM).

The described methods can be used for a wide range of problems; therefore they provide an efficient tool for a large number of research or real-life problems.

References

[1] Horváth G. (ed.), "Neural Networks and their Applications", Műegyetem kiadó, Budapest, 1998. (in Hungarian)

[2] S. Haykin, "Neural networks. A comprehensive foundation", Prentice Hall, N. J., 1999.

[3] V. Vapnik, "The Nature of Statistical Learning Theory", New York: Springer Verlag, 1995.

[4] E. Osuna, R. Freund, F. Girosi, "Support vector machines: Training and applications", Technical Report AIM-1602, MIT A.I. Lab., 1996.

[5] C. J. C. Burges, B. Schölkopf, "Improving the accuracy and speed of support vector learning machines", In: M. Mozer, M. Jordan, T. Petsche (ed.), Advances in Neural Information Processing Systems 9, pp.375–381., Cambridge, MA, MIT Press, 1997.

[6] E. Osuna, R. Freund, F. Girosi, "An improved training algorithm for support vector machines" In: J. Principe, L. Gile, N. Morgan, E. Wilson (ed.), Neural Networks for Signal Processing VII – Proceedings of the 1997 IEEE Workshop, pp.276–285, New York, 1997.

[7] Thorsten Joachims, "Making Large-Scale SVM Learning Practical", Advances in Kernel Methods-Support Vector Learning', MIT Press, Cambridge, USA, 1998.

[8] J.A.K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, J. Vandewalle, "Least Squares Support Vector Machines", 2002. World Scientific, Singapore, (ISBN 981-238-151-1)

[9] F. Girosi, "An equivalence between sparse approximation and support vector machines," Neural Computation, 10(6), pp.1455–1480., 1998.

[10] J. Valyon, G. Horváth, "A generalized LS-SVM", SYSID'2003 Rotterdam, 2003, pp.827–832.

[11] J. Valyon, G. Horváth, "A Sparse Least Squares Support Vector Machine Classifier", Proceedings of the International Joint Conference on Neural Networks IJCNN 2004, pp.543–548.

[12] Yuh-Jye Lee, Olvi L. Mangasarian, "RSVM: Reduced support vector machines", Proc. of the First SIAM International Conference on Data Mining, Chicago, April 5-7, 2001.

[13] W. H. Press, S. A. Teukolsky, W. T. Wetterling, B. P. Flannery, "Numerical Recipes in C", Cambridge University Press, Books On-Line, www.nr.com, 2002.

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