Predicting Time Series with Support Vector Machines

K.-R. Müller[1], A. J. Smola[1], G. Rätsch[1], B. Schölkopf[2], J. Kohlmorgen[1], V. Vapnik[3]

[1] GMD FIRST, Rudower Chaussee 5, 12489 Berlin, Germany
[2] Max-Planck-Institut f. biol. Kybernetik, Spemannstr. 38, 72076 Tübingen, Germany
[3] AT&T Research, 101 Crawfords Corner Rd, PO 3030, Holmdel 07733, NJ, USA

Abstract. Support Vector Machines are used for time series prediction and compared to radial basis function networks. We make use of two different cost functions for Support Vectors: training with (i) an ϵ insensitive loss and (ii) Huber's robust loss function and discuss how to choose the regularization parameters in these models. Two applications are considered: data from (a) a noisy (normal and uniform noise) Mackey Glass equation and (b) the Santa Fe competition (set D). In both cases Support Vector Machines show an excellent performance. In case (b) the Support Vector approach improves the best known result on the benchmark by a factor of 37%.

1 Introduction

Support Vector Machines have become a subject of intensive study (see e.g. [3, 13]). They have been applied successfully to classification tasks as OCR [13, 10] and more recently also to regression [4, 14].
In this contribution we use Support Vector Machines in the field of time series prediction and we find that they show an excellent performance.
In the following sections we will give a brief introduction to support vector regression (SVR) and we discuss the use of different types of loss functions. The experimental section considers a comparison of radial basis function (RBF) networks with adaptive centers and variances and SVR. Both approaches show similarly excellent performance with an advantage for SVR in the high noise regime for Mackey Glass data. For benchmark data from the Santa Fe Competition (data set D) we get the best result achieved so far which is 37% better than the winning approach during the competition [15] and still 29% better than our previous result [8]. A brief discussion concludes the paper.

2 Support Vector Regression

In SVR the basic idea is to map the data $\mathbf{x}$ into a high dimensional feature space $\mathcal{F}$ via a nonlinear mapping Φ and to do linear regression in this space (cf. [3, 13])

$$f(\mathbf{x}) = (\omega \cdot \Phi(\mathbf{x})) + b \quad \text{with } \Phi : \mathbb{R}^n \to \mathcal{F},\ \omega \in \mathcal{F}, \tag{1}$$

where b is a threshold. Thus, *linear* regression in a *high* dimensional (feature) space corresponds to *nonlinear* regression in the *low* dimensional input space $\mathbb{R}^n$. Note that the dot product in Eq.(1) between $\omega \cdot \Phi(\mathbf{x})$ *would have* to be computed in this high dimensional space (which is usually intractable), if we were not able to use a trick – described in the following – that finally leaves us with dot products that can be implicitly expressed in the low dimensional input space $\mathbb{R}^n$. Since Φ is fixed, we determine ω from the data by minimizing the sum of the empirical risk $R_{emp}[f]$ and a complexity term $||\omega||^2$, which enforces *flatness* in feature space

$$R_{reg}[f] = R_{emp}[f] + \lambda||\omega||^2 = \sum_{i=1}^{l} \mathcal{C}(f(\mathbf{x}_i) - y_i) + \lambda||\omega||^2, \tag{2}$$

where l denotes the sample size, $\mathcal{C}(.)$ is a cost function and λ is a regularization constant. For a large set of cost functions, eq. (2) can be minimized by solving a quadratic programming problem, which is *uniquely* solvable. It can be shown that the vector ω can be written in terms of the data points

$$\omega = \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) \Phi(\mathbf{x}_i) \tag{3}$$

with α_i, α_i^* being the solution of the aforementioned quadratic programming problem [13]. α_i, α_i^* have an intuitive interpretation (see Fig. 1b) as forces pushing and pulling the estimate $f(\mathbf{x}_i)$ towards the measurements y_i. Taking (3) and (1) into account, we are able to rewrite the whole problem in terms of dot products in the *low* dimensional input space (a concept introduced in [1])

$$f(\mathbf{x}) = \sum_{i=1}^{l} (\alpha_i - \alpha_i^*)(\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x})) + b = \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) k(\mathbf{x}_i, \mathbf{x}) + b. \tag{4}$$

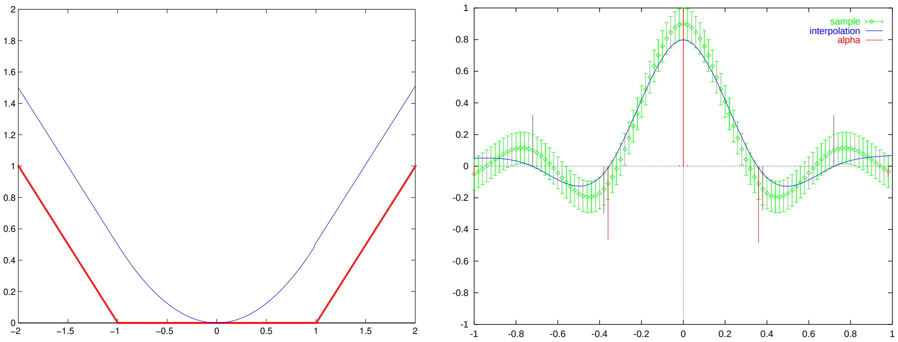
In Eq.(4) we introduced a kernel function $k(\mathbf{x}_i, \mathbf{x}_j) = (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j))$. It can be shown that any symmetric kernel function k satisfying Mercer's condition corresponds to a dot product in some feature space (see [3] for details). A common kernel is e.g. a RBF kernel $k(x, y) = \exp(-||\mathbf{x} - \mathbf{y}||^2/(2\sigma^2))$.

Vapnik's ϵ-insensitive Loss Function. For this special cost function the Lagrange multipliers α_i, α_i^* are sparse, i.e. they result in non-zero values after the optimization (2) only if they are on the boundary (see Fig. 1b), which means that they fulfill the Karush-Kuhn-Tucker conditions (for more details see [13, 11]). The ϵ–insensitive cost function is given by

$$\mathcal{C}(f(\mathbf{x}) - y) = \begin{cases} |f(\mathbf{x}) - y| - \epsilon \text{ for } & |f(\mathbf{x}) - y| \geq \epsilon \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

(cf. Fig. 1a); the respective quadratic programming problem is defined as

$$\text{minimize} \quad \frac{1}{2} \sum_{i,j=1}^{l} (\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j) k(\mathbf{x}_i, \mathbf{x}_j) - \sum_{i=1}^{l} \alpha_i^*(y_i - \epsilon) - \alpha_i(y_i + \epsilon)$$



(a) (b)

Fig. 1. (a) ϵ-insensitive and Huber's loss for $\epsilon = 1$. (b) The shown regression (kernel: B-splines [11]) of the sinc function is the flattest within the ϵ tube around the data. α, α^* are drawn as positive and negative forces respectively. All points on the margin, where $f(\mathbf{x}_i) - y_i = \epsilon \operatorname{sign}(\alpha_i - \alpha_i^*)$, are used for the computation of b.

$$\text{subject to } \sum_{i=1}^{l} \alpha_i - \alpha_i^* = 0, \quad \alpha_i, \alpha_i^* \in [0, \frac{1}{\lambda}]. \tag{6}$$

Huber's Loss Function. Other cost functions like the robust loss function in the sense of [5] can also be utilized (cf. Fig. 1a) [11]. This cost function has the advantage of not introducing additional bias (like the ϵ-insensitive one does), at the expense, however, of sacrificing sparsity in the coefficients α_i, α_i^*.

$$\mathcal{C}(f(\mathbf{x}) - y) = \begin{cases} \epsilon|f(\mathbf{x}) - y| - \frac{\epsilon^2}{2} & \text{for } |f(\mathbf{x}) - y| \geq \epsilon \\ \frac{1}{2}(f(\mathbf{x}) - y)^2 & \text{otherwise} \end{cases} \tag{7}$$

The corresponding quadratic programming problem takes the following form

$$\begin{aligned} &\text{minimize} \quad \frac{1}{2} \sum_{i,j=1}^{l} (\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j) k(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) y_i + \frac{1}{2\lambda}(\alpha_i^2 + {\alpha_i^*}^2) \\ &\text{subject to} \quad \sum_{i=1}^{l} \alpha_i - \alpha_i^* = 0, \quad \alpha_i, \alpha_i^* \in [0, \frac{\epsilon}{\lambda}]. \end{aligned} \tag{8}$$

How to compute the threshold b**?** Eqs. (6) and (8) show how to compute the variables α_k, α_k^*. For the proper choice of b however one has to make more direct use of the Karush-Kuhn-Tucker conditions that lead to the quadratic programming problems stated above. The key idea is to pick those values α_k, α_k^* for which the prediction error $\delta_k = f(\mathbf{x}_k) - y_k$ can be determined uniquely. In the ϵ-insensitive case this means picking the points $\mathbf{x}_k$ on the margin, i.e. α_k or α_k^* in the open interval $(0, \frac{1}{\lambda})$ as here we know the exact value $\delta_k = \epsilon \operatorname{sign}(\alpha_k - \alpha_k^*)$ of the prediction error. Already one $\mathbf{x}_k$ would be sufficient but for stability purposes it is recommended to take the average over all points on the margin $\mathbf{x}_k$ with

$b = \text{average}_k\{\delta_k + y_k - \sum_i(\alpha_i - \alpha_i^*)k(\mathbf{x}_i, \mathbf{x}_k)\}$. For the Huber case b is computed along the same lines with $\delta_k = \lambda(\alpha_k - \alpha_k^*)$ for α_k or $\alpha_k^* \in [0, \frac{\epsilon}{\lambda})$, i.e. for points where the the quadratic part of the cost function is active.

noise	normal				uniform					
SNR	22.15%		44.3%		6.2%		12.4%		18.6%	
test error	1S	100S	1S	100S	1S	100S	1S	100S	1S	100S
ϵ-insensitive	0.017	0.218	0.040	0.335	0.006	0.028	0.012	0.070	0.017	0.142
Huber	0.017	0.209	0.040	0.339	0.008	0.041	0.014	0.065	0.019	0.226
RBF	0.018	0.109	0.044	0.266	0.009	0.062	0.014	0.083	0.028	0.282

experiment	ϵ-ins.	Huber	RBF	ZH [15]	PKM [8]
full set	0.0639	0.0653	0.0677	0.0665	-
segmented set	0.0418	0.0425	0.0569	-	0.0596

Table 1. First Table: 1S denotes the 1-step prediction error (RMS) on the test set. 100S is the 100-step iterated autonomous prediction. "level" is the ratio between the standard deviation of the respective noise and the underlying time series. Second Table: Comparison of 25 step iterated predictions (root mean squared errors) on Data set D. "-" denotes: no prediction available. "Full set" means, that the full training set of set D was used, whereas "segmented set" means that a prior segmentation according to [8] was done as preprocessing.

3 Experiments

The RBF nets used in the experiments are based on the the method of Moody and Darken [7]. However, we do not only adjust the output weights by back-propagation (on squared loss with regularization), but also the RBF centers and variances. In this way, the networks fine-tune themselves to the data after the clustering step, yet of course overfitting has to be avoided (cf. [2]).
We fix the following experimental setup for our comparison: (a) RBF nets and (b) SVR are trained using a simple cross validation technique. We stop training the RBF networks at the minimum of the one step prediction error measured on a validation set. For SVR the parameters (λ, ϵ) are also determined at the minimum of the one step prediction error on the validation set. Other methods, e.g. bootstrap can also be used to assess λ and ϵ. Note, for SVR we distinguish between a training with Huber loss and ϵ-insensitive loss and use RBF kernels with $k(x, y) = \exp(-||\mathbf{x} - \mathbf{y}||^2/(2\sigma^2))$ and $\sigma^2 = 0.75$.

Mackey Glass Equation. Our first application is a high-dimensional chaotic system generated by the Mackey-Glass delay differential equation

$$\frac{dx(t)}{dt} = -0.1x(t) + \frac{0.2x(t-t_d)}{1 + x(t-t_d)^{10}}, \quad (9)$$

with delay $t_d = 17$. Eq. (9) was originally introduced as a model of blood cell regulation [6] and became quite common as artificial forecasting benchmark. After integrating (9), we added noise to the time series. We obtained training

(1000 patterns) and validation (194 patterns) sets using an embedding dimension $m = 6$ and a step size $\tau = 6$ ($\mathbf{x}_t = (x_t, x_{t-\tau}, \ldots, x_{t-(m-1)\tau})^T$). The test set (1000 patterns) is noiseless to measure the true prediction error. We conducted experiments for different signal to noise ratios (SNR) using Gaussian and uniform noise (Table 1). RBF networks and SVR achieve similar results for normal noise. It is to be expected that the method using the proper loss function (squared loss) wins for gaussian noise, so we would actually expect the RBF nets to perform best followed by SVR trained with Huber loss, which is for large ϵ close to the squared loss and finally followed by SVR using an ϵ-insensitive loss. Table 1 confirms this intuition largely. For uniform noise the whole scenario should be reversed, since ϵ-insensitive loss is the more appropriate noise model (cf. [5]). This is again confirmed in the experiment. The use of a validation set to assess the proper parameters λ and ϵ, however, is suboptimal and so the low resolution with which the (λ, ϵ) space is scanned is partly responsible for table entries that do not match the above intuition.

Data Set D from the Santa Fe Competition. Data set D from the Santa Fe competition is artificial data generated from a nine-dimensional periodically driven dissipative dynamical system with an asymmetrical four-well potential and a drift on the parameters [12]. As embedding 20 consecutive points were used. Since the time series is non-stationary, we first segment into regimes of approximately stationary dynamics with competing predictors [8]. We use only the subset for training (327 patterns) which was tagged by the predictor responsible for the data points at the end of the full training set. This allows us to train the RBF networks and the SVR on quasi stationary data and we avoid to predict the average over all dynamical modes hidden in the full training set (see also [8] for further discussion), however at the same time we are left with a rather small training set requiring careful regularization. As in the previous section we use a validation set (50 patterns) to determine the stopping point and (λ, ϵ) respectively. Table 1 shows that our 25 step iterated prediction of the SVR is 37% better than the one achieved by Zhang et al. [15], who assumed a stationary model. It is still 29% better than our previous result [8] that used the same preprocessing as above and simple RBF nets with non-adaptive centers and variances. The results obtained, if we train on the full (non-stationary) training set (without prior segmentation) are inferior, as expected, however ϵ-insensitive SVR is still better than the previous results on the full set.

4 Discussion and Outlook

The paper showed the performance of SVR in comparison to tuned RBF networks. For data from the Mackey-Glass equation we could observe that also for SVR it pays to choose the *proper* loss function for the respective noise model. In both SVR cases training consisted in solving a – uniquely solvable – *quadratic optimization* problem, unlike the RBF network training, which requires complex non-linear optimization with the danger of getting stuck in local minima. Note that a stable prediction is a difficult problem since the noise level applied to

the chaotic dynamics was rather high. For the data set D benchmark we obtained excellent results for SVR – 37% above the best result achieved during the Santa Fe competition [15]. Clearly, this remarkable difference is mostly due to the segmentation used as preprocessing step to get stationary data [8], nevertheless still 29% improvement remain compared to a previous result using the same preprocessing step [8]. This underlines that we need to consider possible non-stationarities or even mixings in the time series *before* the actual prediction, for which we used SVR or RBF nets (see also [8, 9] for discussion).
Our experiments show that SVR methods work particularly well if the data is *sparse* (i.e. we have little data in a high dimensional space). This is due to their good inherent regularization properties.
Several things remain: determining the proper parameters λ and ϵ is still suboptimal and computationally intensive (if not clumsy). Both, some improved theoretical bounds and/or a simple heuristics to choose them would enhance the usability of SVR, since (λ, ϵ) are powerful means for *regularization* and *adaptation to the noise in the data*. Bootstrap methods or methods using a validation set are only a first step.
Acknowledgement: A.S. and G.R. are supported by the DFG (# Ja 379/51).

References

1. M. A. Aizerman, E. M. Braverman, and L. I. Rozonoér (1964), Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837.
2. C.M. Bishop (1995) , Neural networks for pattern recognition, Oxford U. Press.
3. B. E. Boser, I .M. Guyon, and V. N. Vapnik. (1992), A training algorithm for optimal margin classifiers. In D. Haussler, editor, Proc. of COLT'92, 144.
4. H. Drucker, C. Burges, L. Kaufman, A. Smola, V. Vapnik (1997), Linear support vector regression machines, NIPS'96.
5. P. J. Huber (1972), Robust statistics: a review. *Ann. Statist.*, 43:1041.
6. M. C. Mackey and L. Glass (1977), *Science*, 197:287–289.
7. J. Moody and C. Darken (1989), *Neural Computation*, 1(2):281–294.
8. K. Pawelzik, J. Kohlmorgen, and K.-R. Müller (1996), Annealed competition of experts for a segmentation and classification of switching dynamics. *Neural Computation*, 8(2):342–358.
9. K. Pawelzik, K.-R. Müller, J. Kohlmorgen (1996), Prediction of Mixtures, in ICANN '96, LNCS 1112, Springer Berlin, 127-132 and GMD Tech.Rep. 1069.
10. B. Schölkopf, C. Burges, V. Vapnik (1995), Extracting support data for a given task. In: U. M. Fayyad and R. Uthurusamy (eds.), Proceedings, First International Conference on Knowledge Discovery & Data Mining, AAAI Press, Menlo Park, CA.
11. A. J. Smola, B. Schölkopf (1997) On a kernel–based method for pattern recognition, regression, approximation and operator inversion. *Algorithmica* to appear and GMD Tech.Rep. 1064.
12. A.S. Weigend, N.A. Gershenfeld (Eds.) (1994), *Time Series Prediction: Forecasting the Future and Understanding the Past*, Addison-Wesley.
13. V. Vapnik (1995), *The Nature of Statistical Learning Theory*. Springer NY.
14. V. Vapnik, S. Golowich, A. Smola (1997), Support vector method for function approximation, regression estimation, and signal processing, NIPS'96.
15. X. Zhang, J. Hutchinson (1994). Simple architectures on fast machines: practical issues in nonlinear time series prediction in [12].