Towards Scalable Support Vector Machines using Squashing

Dmitry Pavlov
Info. and Comp. Science
University of California, Irvine
Irvine, CA, 92612
pavlovd@ics.uci.edu

Darya Chudova
Info. and Comp. Science
University of California, Irvine
Irvine, CA, 92612
dchudova@ics.uci.edu

Padhraic Smyth
Info. and Comp. Science
University of California, Irvine
Irvine, CA, 92612
smyth@ics.uci.edu

ABSTRACT

Support vector machines (SVMs) provide classification models with strong theoretical foundations as well as excellent empirical performance on a variety of applications. One of the major drawbacks of SVMs is the necessity to solve a large-scale quadratic programming problem. This paper combines likelihood-based squashing with a probabilistic formulation of SVMs, enabling fast training on squashed data sets. We reduce the problem of training the SVMs on the weighted "squashed" data to a quadratic programming problem and show that it can be solved using Platt's sequential minimal optimization (SMO) algorithm. We compare performance of the SMO algorithm on the squashed and the full data, as well as on simple random and boosted samples of the data. Experiments on a number of datasets show that squashing allows one to speed-up training, decrease memory requirements, and obtain parameter estimates close to that of the full data. More importantly, squashing produces close to optimal classification accuracies.

Categories and Subject Descriptors

H.2.8 [**Information Systems**]: Database Management—*Database Applications,Data Mining*; I.2.6 [**Computing Methodologies**]: Artificial Intelligence—*Learning*

General Terms

support vector machines (SVMs), scalability, squashing, boosting

1. INTRODUCTION

Following the pioneering work of Vladimir Vapnik [15], support vector machines (SVMs) are steadily gaining popularity in the machine learning community. SVMs have been proven to exhibit several attractive theoretical properties including maximum margin separation between the classes. In addition, SVMs have been empirically shown to outperform conventional classifiers on a variety of benchmarks [13]. However, the applicability of SVMs to large datasets is limited because of the high computational cost involved in solving quadratic programming problem arising in their training.

Various training algorithms have been proposed to speed up the training, including chunking [13], Osuna's decomposition method [9], and Sequential Minimal Optimization (SMO) [11]. Although these algorithms accelerate the training, they do not scale well with the size of the training data.

Another approach to reducing the computational cost is to use approximation methods. The simplest method is sampling from the original dataset and using the sample to train the SVM. An extension of this idea was explored by Pavlov et. al. [10] who reported results on the application of boosting to training SVMs. Boost-SMO algorithm trains a sequence of SVM classifiers on samples of data so that each subsequent classifier concentrates mostly on the errors made by the previous ones [12, 10]. While this method is not optimal in general, it allows for fast training of SVMs, has substantially lower memory cost and yields performance close to that of SMO on the full data.

In this paper we suggest another method for scaling SVMs up based on *squashing*. DuMouchel et. al. [5] and Madigan et. al. [7] recently introduced squashing as a technique that allows one to scale a dataset down while preserving its statistical properties. In particular, likelihood-based squashing [7] assumes a particular statistical model and tries to preserve the behavior of the likelihood function of the original data (referred to as the "mother-data") in the neighborhood of the maximum likelihood solution. To apply likelihood-based squashing to SVMs it is necessary to have a probabilistic interpretation (see, e.g.,[6, 16]). We use an interpretation of the SVM training procedure [14] as a problem of finding maximum aposteriori values for the parameters of the SVM.

We show that the probabilistic interpretation of SVM training in conjunction with likelihood-based squashing allows one to scale the training up. We reduce the problem of training the SVMs on the weighted "squashed" data to a quadratic programming problem and show that it can be solved using the SMO algorithm. We compare the performance of the SMO algorithm on the squashed and the full data, as well as on simple random and boosted samples of the data. Experiments on a number of datasets show that squashing allows one to speed-up training, decrease memory requirements, and obtain parameter estimates close to that of the full data. More importantly, squashing produces classification accuracy that is close to optimal. Note that although our results can be directly generalized to large-scale datasets that do not fit in the main memory, we are using moderately sized data that resides in memory in our experiments.

The rest of the paper is organized as follows. We give a brief overview of SVMs in section 2. We then describe squashing and its application to SVMs in more detail in section 3. Section 4 describes results of empirical evaluation of the algorithms on 4 datasets. In section 5 we draw conclusions and discuss possible directions for the future work.

2. SVMS: A BRIEF OVERVIEW

We give a brief overview of linear SVMs here. Suppose that we have a total of N labeled patterns in the d-dimensional space $D = \{(\mathbf{x}_i, y_i)\}$ belonging to two classes, so that labels y_i are either -1 or 1.In linear SVMs the line separating the classes is sought in the form

$$y = <\mathbf{w}, \mathbf{x}> + b, \tag{1}$$

where $\mathbf{w}$ is the normal vector and b is the intercept of the hyperplane. The problem of finding the maximum margin separation between the classes is equivalent to the minimization of $\|\mathbf{w}\|^2_{L_2}$ with the additional constraints $y_i(<\mathbf{w}, \mathbf{x}_i> + b) \geq 1$ to ensure that all the patterns in the training set are classified correctly [4]. However, in most cases perfect separation is impossible and we need to trade errors on the individual training patterns for the maximum margin. The optimization problem becomes

$$\min_{\mathbf{w},b} E = \min_{\mathbf{w},b}\{\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i\}, \tag{2}$$

subject to

$$y_i(<\mathbf{w}, \mathbf{x}_i> + b) \geq 1 - \xi_i, \tag{3}$$

where C is a tradeoff constant and ξ_i are the non-negative errors. Thus, learning the parameters of the SVM is reduced to a linearly constrained quadratic programming problem. The standard SMO algorithm solves the dual of problem 2 and 3 by decomposing it into the smaller problems that can be solved analytically. SMO is provably optimal [8].

Several authors (e.g., [16, 14]) have shown that the right side of Equation 2 can be treated as a log-posterior on the parameters $\mathbf{w}$ and b, with the first term corresponding to a prior on $\mathbf{w}$ and the second one to the likelihood. The prior on $\mathbf{w}$ will have a normal distribution with zero mean:

$$p(\mathbf{w}) \sim \exp(-\|\mathbf{w}\|^2), \tag{4}$$

while the log-likelihood of the data will (up to a constant additive factor independent of $\mathbf{w}$ and b) be proportional to:

$$l^f(\mathbf{w}, b) \propto \sum_{i=1}^{N} I(1 - y_i[<\mathbf{w}, \mathbf{x}_i> + b]), \tag{5}$$

where $I(z)$ is z times the indicator function of z.

By solving the dual optimization problem via SMO one essentially finds the maximum aposteriori values for $\mathbf{w}$ and b. We will use this probabilistic interpretation of the training procedure to develop a *squashing* procedure for SVMs.

3. SQUASHING FOR SVMS

To illustrate the idea of squashing let us assume that we have selected a probabilistic model $p(z|\theta)$ for the data D and our objective is to find the maximum likelihood estimate θ_{ML} for the vector of parameters θ:

$$\theta_{ML} = \arg\max_\theta \sum_{i=1}^{N} \log p((\mathbf{x}_i, y_i)|\theta). \tag{6}$$

Further, suppose that the terms in equation 6 can be grouped (or clustered) so that the likelihoods of the individual points within each group do not vary significantly in the neighborhood of θ_{ML}. The squashing procedure eliminates points that contribute similar amounts to the likelihood by replacing each cluster with a single point placed at its center of mass and a weight equal to the number of points in the cluster. The maximum likelihood estimate can now be found from expression 7 which approximates equation 6:

$$\theta_{ML} \simeq \arg\max_\theta \sum_{c=1}^{N_c} \beta_c \log p((\mathbf{x}_c, y_c)^{sq}|\theta). \tag{7}$$

Here N_c is the number of clusters, β_c is the weight (or equivalently the number of points in the original cluster c) and $(\mathbf{x}_c, y_c)^{sq}$ is the squashed data point placed at the center of the cluster c. Note that we typically look for at least a 50-100 times reduction in the amount of data so that conventional algorithms can be applied to the squashed data directly.

Notice that clustering at this step should not be computationally more intensive than solving the original maximum likelihood problem on the mother-data. In the formulation given in [7] squashing makes only two passes through the mother-data. It evaluates the likelihoods of the training points for certain choices of θ and forms the so-called *likelihood profiles* in the first pass. After this N_c points are randomly selected to form initial centers of clusters. Finally, on the second pass each training point is assigned to the cluster whose center's likelihood profile is closest to the likelihood profile of the data point.

Important design issues for the squashing algorithm include the choice of the number and location of the points in the parameter space to evaluate the likelihoods at, and the number of squashed data points to ensure a sufficiently good approximation. Various *factorial designs* [3] in the parameter space are suggested in [7]. While this method is universal, as we noted above, for SVMs we can sample the values of $\mathbf{w}$ from the prior distribution in equation 4. The choice of the intercept term b can be made from the condition that the hyperplane goes through the cloud of training points. We evaluate the likelihood of each training point for the fixed pair of $\mathbf{w}$ and b and repeat the selection procedure L times, where L is the length of likelihood profile. Our experiments show that for SVMs the performance of the squash-SMO method is almost insensitive to the choice of L as long as L is at least on the order of 10^2.

Suppose that we have performed squashing of the original data set using the likelihood defined in equation 5. We now need to classify the squashed dataset $D^{sq} = \{(\mathbf{x}_c, y_c)^{sq}\}$ containing N_c weighted points. An expression for the likelihood of the squashed dataset (up to a constant additive factor independent of $\mathbf{w}$ and b) is proportional to

$$l^{sq}(\mathbf{w}, b) \propto \sum_{c=1}^{N_c} \beta_c I(1 - {y_c}^{sq}[< \mathbf{w}, {\mathbf{x}_c}^{sq} > +b]), \qquad (8)$$

since the squashed dataset contains β_c points coinciding with $(\mathbf{x}_c, y_c)^{sq}$. Going back to the non-probabilistic formulation we will have the following optimization problem for the squashed data:

$$\min_{\mathbf{w},b} E^{sq} = \min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C^{sq} \sum_{i=1}^{N_c} \beta_i \xi_i, \qquad (9)$$

with the constraints

$${y_i}^{sq}(< \mathbf{w}, {\mathbf{x}_i}^{sq} > +b) \geq 1 - \xi_i, \qquad (10)$$

where the slack variables ξ_i are non-negative as before. It is easy to see that the substitution of slack variables $\eta_i = \beta_i \xi_i$ transforms the objective function for the squashed problem into exactly the same form as we had for the mother-data, while the inequalities for constraints will have slack variables η_i divided by the corresponding weights. Thus, the only difference between the dual problems for the mother- and the squashed data will be in the allowable regions for the Lagrange multipliers: the i^{th} multiplier will be bounded by $C^{sq}\beta_i$ in the squashed data as opposed to C in the mother-data. Once the SMO code is updated to account for the new bounds on Lagrange multipliers, it can be used directly on the weighed data from squashing.

4. EXPERIMENTS

We have run experiments on four datasets, one of which is synthetic and the rest are publicly available at either the UCI machine learning [2] or UCI KDD repositories [1]. We evaluated the performance of the full-SMO (SMO on the full training data), srs-SMO (SMO on a simple random sample), squash-SMO (SMO on the squashed data) and boost-SMO (SMO on the boosted samples) with respect to the misclassification rate, the time to learn the parameters of the model and the memory used during the training.

We performed all experiments on Sun Solaris 2.7 UNIX workstation with two 168MHz Sparc CPUs and 256Mb of main memory. We averaged results for squash-SMO, boost-SMO and srs-SMO over 100 runs to account for variability in the data resulting from sampling. Our implementation of the SMO algorithm used caching [11] to store frequently used dot products.

4.1 Results on Synthetic Data

Synthetic data was obtained by sampling from the specially designed piece-wise gaussian distributions in 2D. Each distribution is composed of two parts as follows. Let $g_1(x, y)$ and $g_2(x, y)$ be two gaussian distributions, with the equal means $\mu = (\mu_x, \mu_y) = (-0.75, 0)$ and diagonal covariance matrices Σ^1 and Σ^2, such that $(\Sigma^1)_{11} = 1$, $(\Sigma^2)_{11} = 3$, $(\Sigma^1)_{22} = (\Sigma^2)_{22} = 9$. Let the probability density for one of the classes be $g_1(x, y)$ for $x < \mu_x$ and $g_2(x, y)$ for $x > \mu_x$ so that there is a discontinuity at $x = \mu_x$. Another distribution is a mirror symmetrical image of the first one relative to the line $x = 0$. We generated 5000 examples of each class for both training and test sets. This example was specifically tailored to have high density of data from both classes in the neighborhood of the optimal decision boundary $x = 0$.

Recall that we seek the boundary in the form $w_1x+w_2y+b = 0$, here $\mathbf{w} = (w_1, w_2)$ is a slope vector and b is an intercept. Table 1 illustrates how dramatically different was the performance of squash-SMO from srs-SMO on this dataset. In the first column we report the average relative error in parameter estimates over 100 runs, where the error is defined as follows:

$$E^2 = \frac{\|\mathbf{w_f} - \mathbf{w_s}\|^2 + (b_f - b_s)^2}{\|w_f\|^2 + {b_f}^2}. \qquad (11)$$

Here $(\mathbf{w_f}, b_f)$ are the parameters estimated by the SMO algorithm on the full data and $(\mathbf{w_s}, b_s)$ are obtained from either the squashed or sampled data. Squash-SMO obtains parameter estimates that are almost 4 times more accurate than srs-SMO. The next 3 columns contain the standard deviation of the individual estimates for w_1, w_2 and b in 100 runs of srs-SMO and squash-SMO. Notice that the parameters estimated by the squash-SMO are 5 times more stable than estimates obtained by srs-SMO.

Table 1: Average Relative Error and St. Dev. in Parameter Estimates on Synthetic Dataset.

	E	σ_{w_1}	σ_{w_2}	σ_b
Srs-SMO	0.467	0.52	0.47	0.38
Squash-SMO	0.124	0.1	0.09	0.08

We analyzed the structure of the errors made by different models and concluded that srs-SMO made most errors in the neighborhood of the optimal decision boundary. Since Squash-SMO was able to obtain more stable and accurate parameter estimates, it exhibited much better performance in that region.

Table 2 shows the error on the test set for various models on the synthetic data. Clearly, srs-SMO provides the worst improvement over the baseline model, full-SMO is the best with an error close to the Bayes error rate, and squash-SMO is a close runner-up.

Table 2: Test Set Error Rates On Synthetic Dataset.

Baseline	50%
1% srs-SMO	41%
1% squash-SMO	34.04%
1% boost-SMO	34.84%
full-SMO	33.54%

4.2 Results on Benchmark Data

The public datasets were “The Microsoft Anonymous Web” and the ”Forest Cover Type” datasets available at UCI KDD archive [1] and Adult dataset available at UCI machine learning repository [2]. The classification task for the *Web Data*, as we pose it, is to predict whether a user will visit the most popular site S based on his/her visiting pattern of all other sites. This dataset contains binary data and average record has only 3% of its attributes set to 1. For *Adult data* the task is to predict if the income of a person is greater than $50K based on several census parameters, such as age, education, marital status and so forth. Finally, for the Forest data the task is to distinguish between several forest cover types based on the information about the location of the place, type of soil, distance to the water etc. For this dataset we experimented with 2 most populated classes for which the misclassification rate is the greatest: Spruce-Fir and Lodgepole Pine. The parameters of all datasets are summarized in Table 3.

Table 3: Characteristics of the Data Sets.

Parameter	Web	Adult	Forest
$N_{Attributes}$	294	14	54
Attributes	Binary	Categ.	Categ.
Training set	10057(+)	7508(+)	56404(+)
	21875(-)	22654(-)	42480(-)
Test set	1561(+)	3700(+)	226897(+)
	3325(-)	11360(-)	169360(-)

Note that one of the distinct advantages of the methods that use sampled or squashed data is that they allow to do cross-validation to optimize the parameters of the SVM classifier. Since m-fold cross-validation will increase the computational cost of training by a factor of m, using it for the full-SMO is out of question. In our experiments we used 5-fold cross-validation to adjust the value of the regularization constant C for the squash-SMO, srs-SMO and boost-SMO. For the full-SMO C was set to 1 in all runs.

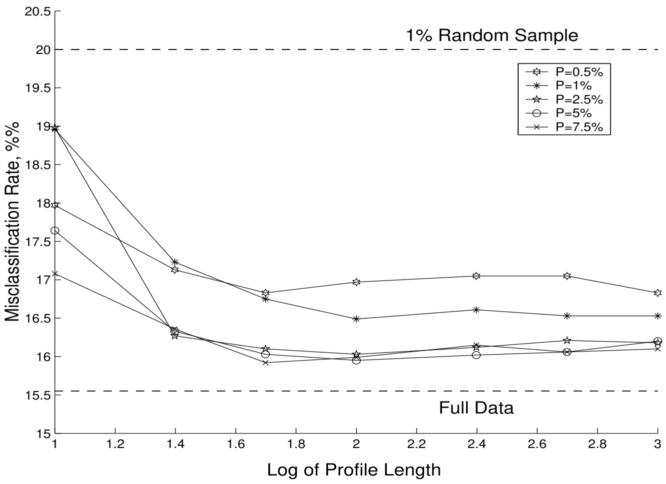

Figure 1: Misclassification Rate Vs Likelihood Profile Length on the Adult dataset.

We analyzed the dependence of the SVM misclassification rate on the length of the likelihood profile and the number N_c of squashed points. In Figure 1 the parameter P refers to the squashing ratio, i.e., the number of points in the squashed data relative to the mother-data. As P increases from 0.5% to 7.5% we see a gradual decrease in the misclassification rate so that it approaches the misclassification rate of SVMs on the full data. It is also apparent that for this data (Adult dataset) likelihood profiles of size 50-100 will yield the best rate. Similar results were obtained on the Web data. In all experiments described below we were using a fixed length of the likelihood profile set to 100.

Table 4: % Decrease from Baseline Error Rate.

	Web	Adult	Forest
Baseline Error	0.32	0.25	0.43
1% srs-SMO,%	10	18.7	40
1% squash-SMO,%	20	31.8	46.7
1% boost-SMO,%	20.5	33.2	45.6
full-SMO,%	22.5	36.8	48.1

Table 4 shows the misclassification rate on the test set for various datasets and algorithms. The first row, baseline error, shows the misclassification rate of the trivial prediction that classifies all examples into the most populated class. The rest of the rows show what percentage of decrease in error relative to baseline a particular model is able to provide. Notice that srs-SMO algorithm is the worst and full-SMO algorithm is the best. The difference in misclassification rate between the two extremes ranges from 8% to 18%. It is remarkable that performance of both squash-SMO and boost-SMO is much closer to full-SMO than performance of srs-SMO.

Table 5: CPU Time to Learn Parameters of Various Models for Various Datasets. For squash-SMO (srs-SMO) the first number is the time to do squashing (sampling) and the second is the time to do SMO on the squashed (sampled) data.

	Web	Adult	Forest
1% srs-SMO	1.5 + 0.2	1.5 + 0.2	13 + 2
1% squash-SMO	361.7 + 0.2	132.4 + 0.2	682.9 + 2
1% boost-SMO	10	7.5	85.3
full-SMO	4018	1151	1225

Table 5 reports timing results for various algorithms. Srs-SMO turns out to be the fastest with times to carry out SMO below 2 seconds. The runner up is boost-SMO with times ranging from 7.5 for the Adult data to 85 seconds for the forest data. For both srs-SMO and squash-SMO the time to perform sampling or squashing dominates over the time to do SMO and it takes much longer to do squashing than sampling. The full-SMO algorithm is the slowest, being on average 10-100 times slower than the rest of the algorithms. While the optimality is a significant advantage of the full-SMO, its time and memory requirements leave much to be desired even for the modestly sized problems that we considered here. As an illustration, in our experiments we only allowed caching of up to 9 million dot products. When this upper limit was reached the cache was flushed. For all non-synthetic datasets this upper limit was reached and the program overall took from 80 to 120Mb of memory during training (the training and test data resided in main memory as well). Note that in SMO we could trade memory for speed if we did not allow caching dot products, but in this case the training will be even slower.

For the really large datasets squash-SMO might outperform boost-SMO since the latter needs a) to store the current boosted sample and the distribution over all training examples and b) multiple passes through the dataset to update the distribution on examples and sample from it. The Squash-SMO on the other hand will only need 2 full passes over the data as we discussed before during the squashing phase and the mother-data will not be used at all during SVM training.

Yet another advantage of squash-SMO over boost-SMO is a better model interpretability that may be useful for exploratory data analysis and visualization.

5. CONCLUSIONS AND FUTURE WORK

We describe how the use of squashing makes the training of SVMs applicable to large datasets. Comparison with the optimal full-SMO algorithm shows that both squash-SMO and boost-SMO are able to reach near-optimal performance with much lower time and memory requirements. On the other hand, the simplest and fastest standard random sampling SMO on average has a considerably higher misclassification rate. We conclude that statistical techniques based on the ideas of sampling (boost-SMO) and likelihood preservation (squash-SMO) promise high computational advantages for large datasets over the standard SMO algorithm.

Both squash- and boost-SMO allow one to use cross-validation to tune parameters of the SVM, such as regularization constant C or the width of the kernel in non-linear SVMs. Such tuning is usually impossible on the full data due to the high computational cost.

Although the performance of boost-SMO and squash-SMO is similar on the benchmark problems, we outlined when and why one algorithm should be preferred over another. In particular, squash-SMO offers a better interpretability of the model and can be expected to run faster than SMO on the datasets that do not reside in the main memory. For the datasets that fit in the main memory we saw that boost-SMO is faster than squash-SMO.

We see several possibilities to extend this work. The gains offered by squash-SMO and boost-SMO might be even more significant for non-linear SVMs that take much longer to train. As we outlined above squashing itself takes considerable time. For the high-dimensional data one might need to increase the profile length to get acceptable estimates for parameters of the model. This will result in a time increase and performance degradation. Thus, it would be useful to study the conditions under which squashing will work well on the high dimensional data. An interesting theoretical question is whether it is possible to give a bound on the distance between the parameter estimates obtained by the full-SMO and squash-SMO.

6. ACKNOWLEDGEMENTS

The research described in this paper was supported in part by NSF CAREER award IRI-9703120, NSF award IIS-9813584, the NIST Advanced Technology Program and KLA-Tencor, the US Department of Energy and Lawrence Livermore Laboratory, HNC Software Inc., Microsoft Research, and SmithKline Beecham Research. We would also like to thank Jianchang Mao for making his SMO code available to us.

7. REFERENCES

[1] S. Bay. UCI KDD archive, http://kdd.ics.uci.edu.

[2] C. Blake and C. Merz. UCI repository of machine learning databases, http://www.ics.uci.edu/~mlearn.

[3] G. Box, W. Hunter, and J. Hunter. *Statistics for Experimenters: An Introduction to Design, Data Analysis, and Model Building*. John Wiley & Sons, New York, 1978.

[4] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, pages 273–297, 1997.

[5] W. DuMouchel, C. Volinsky, T. Johnson, C. Cortes, and D. Pregibon. Squashing flat files flatter. 1999.

[6] T. S. Jaakola and D. Haussler. Probabilistic kernel regression methods; http://www.ai.mit.edu/~tommi/publications/.

[7] D. Madigan, N. Raghavan, W. DuMouchel, M. Nason, C. Posse, and G. Ridgeway. Likelihood-based data squashing: A modeling approach to instance construction; http://www.stat.washington.edu/greg/docs/lds.ps.

[8] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. In *Proc. of CVPR 1997*.

[9] E. Osuna, R. Freund, and F. Girosi. An improved training algorithm for support vector machines. In *Proc. of IEEE Workshop on Neural Networks for Signal Processing*, pages 276–285. 1997.

[10] D. Pavlov, J. Mao, and B. Dom. Scaling-up support vector machines using boosting algorithm. In *Proc. of ICPR 2000 (to appear)*.

[11] J. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods – Support Vector Learning*, pages 185–208. 1999.

[12] R. E. Schapire. An introduction to boosting algorithm. In *Proc. of IJCAI 1999*.

[13] B. Scholkopf, C. Burges, and A. Smola (eds). *Advances in Kernel Methods – Support Vector Learning*. MIT Press, Cambridge, MA, 1999.

[14] P. Sollich. Probabilistic interpretation and bayesian methods for support vector machines. In *Proc. of ICANN 1999*, pages 91–96, 1999.

[15] V. N. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, New York, 1998.

[16] G. Wahba. Support vector machines, reproducing kernel hilbert spaces and the randomized GACV, ftp://ftp.stat.wisc.edu/pub/wahba/nips97.ps.gz. University of Wisconsin Statistics Dept. TR 984, 1997.