Uncertainty Handling in Model Selection for Support Vector Machines

Tobias Glasmachers and Christian Igel

Institut für Neuroinformatik, Ruhr-Universität Bochum, Germany
{Tobias.Glasmachers,Christian.Igel}@neuroinformatik.rub.de

Abstract. We consider evolutionary model selection for support vector machines. Hold-out set-based objective functions are natural model selection criteria, and we introduce a symmetrization of the standard cross-validation approach. We propose the covariance matrix adaptation evolution strategy (CMA-ES) with uncertainty handling for optimizing the new randomized objective function. Our results show that this search strategy avoids premature convergence and results in improved classification accuracy compared to strategies without uncertainty handling.

1 Introduction

Support vector machines (SVMSs) are powerful algorithms for supervised learning, especially for binary classification [1, 2]. However, their performance crucially depends on appropriate model selection, that is, the choice of the right kernel and the right regularization parameter. If a parametrized family of kernel functions is considered, model selection reduces to real-valued optimization. We propose evolution strategies (ES) for solving the resulting optimization problem (see [3, 4] and references therein), in particular if the model selection criterion is not differentiable and using grid-search is not possible because of the dimensionality.

Cross-validation is regularly applied as a model selection criterion to estimate the quality of a parameter vector (i.e., as a fitness function). We argue that the cross-validation procedure suffers from its fixed partition of the available data into training and validation sets. Especially for small datasets this has a considerable influence on the objective function and the locations of its minima. Therefore we propose to average over all possible dataset partitions to increase reliability. The resulting fitness function is only of theoretical interest because of the complexity of its computation. We avoid this computational problem by sampling, but at the cost of introducing uncertainty. Another advantage of this averaging is that the performance measure gets more fine-grained when using the 0-1-loss and therefore provides additional information for the search algorithm.

The aim of the present paper is to assess the effects of noise introduced into the SVM model selection due to this sampling. We apply the highly efficient covariance matrix adaptation ES (CMA-ES) to the minimization of the new fitness function [5, 6]. Recently, a simple and efficient uncertainty handling

mechanism has been proposed for the CMA-ES [7, 8], which we employ to handle the uncertainty in our model selection criterion.

The paper is organized as follows. In the next section we introduce the CMA-ES and the noise-handling technique. In Section 3 we briefly review SVMs for binary classification. Then we motivate our model selection objective function. An empirical evaluation is presented in Section 5, and we conclude with a short summary.

2 Handling Uncertainty in Evolution Strategies

We first briefly describe the CMA-ES [5, 6, 8] and then its extension to adaptive reevaluation for noisy optimization proposed in [7, 8].

CMA-ES. In each generation of the CMA-ES λ offspring are generated. Their fitness is evaluated and the $\mu = \lfloor \lambda/2 \rfloor$ best form the next parent population. In each iteration, the kth offspring $\boldsymbol{x}_k \in \mathbb{R}^n$ is created by multi-variate Gaussian mutation and weighted global intermediate recombination: $\boldsymbol{x}_k = \langle \boldsymbol{x}_{\text{parents}} \rangle_{\boldsymbol{w}} + \sigma \boldsymbol{z}_k$, where $\boldsymbol{z}_k \sim \mathrm{N}(\boldsymbol{0}, \boldsymbol{C})$ and $\langle \boldsymbol{x}_{\text{parents}} \rangle_{\boldsymbol{w}} = \sum_{i=1}^{\mu} w_i \boldsymbol{x}_{i\text{th-best-parent}}$ ($\boldsymbol{w}_i \propto \ln(\mu + 1) - \ln(i)$, $\|\boldsymbol{w}\|_1 = 1$). The CMA-ES is a variable metric algorithm adapting both the n-dimensional covariance matrix $\boldsymbol{C}$ of the normal mutation distribution as well as the *global step size* $\sigma \in \mathbb{R}^+$. In the basic algorithm, a low-pass filtered *evolution path* $\boldsymbol{p}$ of successful (i.e., selected) steps is stored, $\boldsymbol{p} \leftarrow \eta_1 \boldsymbol{p} + \eta_2 \, (\langle \boldsymbol{x}_{\text{new parents}} \rangle - \langle \boldsymbol{x}_{\text{old parents}} \rangle)$, and $\boldsymbol{C}$ is changed to make steps $\boldsymbol{p}$ more likely: $\boldsymbol{C} \leftarrow \eta_3 \boldsymbol{C} + \eta_4 \boldsymbol{p}\boldsymbol{p}^T$ (this rank-one update of $\boldsymbol{C}$ is augmented by a rank-μ update, see [6]). The variables $\eta_1, \ldots, \eta_4$ denote fixed learning rates and normalization constants set to default values [8]. The global step size σ is adapted on a faster timescale. It is increased if the selected steps are larger and/or more correlated than expected and decreased if they are smaller and/or more anticorrelated than expected. The highly efficient use of information and the fast adaptation of σ and $\boldsymbol{C}$ makes the CMA-ES one of the best direct search algorithms for real-valued optimization [9].

Uncertainty Handling. Evolution algorithms are well suited for optimization in noisy environments, see [10] for a general overview and [11] for a book on ESs for noisy optimization. The population-based approach, the averaging in the recombination process, and the rank-based, non-elitist selection are inherent features that make the CMA-ES less vulnerable to noise. However, if the signal to noise ratio is too small, special uncertainty handling is required. Here we use a slightly simplified version of the uncertainty handling proposed in [7, 8]. It is called *UH-CMA-ES* and relies on adaptive reevaluation of solutions.

Because the selection process is rank-based, we only care about noise if it changes the ranking of offspring. In our scenario, individuals can be reevaluated and computing the mean or median of several evaluations reduces the noise level. However, the signal to noise ratio changes in the course of evolution. Because every fitness evaluation is time consuming, we implement a strategy that adapts

the number of evaluations per individual such that individuals are not evaluated too often, but still often enough so that the fitness values can guide the optimization.

We use an algorithm to detect the effective noise by monitoring the stability of the ranking of the offspring. Following [7, 8], we consider a population $\mathcal{L}$ composed of two copies of the current offspring population (i.e., each offspring is contained twice in $\mathcal{L}$) and reevaluate λ_{reev} of them. Then we sort $\mathcal{L}$ twice using the new and the old fitness values (f_i^{new} and f_i^{old}, $i = 1 \dots, 2\lambda$), respectively, and determine the ranks $\text{rank}(f_i^{\text{new}})$ and $\text{rank}(f_i^{\text{old}})$, respectively, of each reevaluated individual x_i. Then we compute the *rank change*

$$\Delta_i = |\,\text{rank}(f_i^{\text{new}}) - \text{rank}(f_i^{\text{old}})| - 1 \ .$$

The *uncertainty level* s is now defined by

$$s = \frac{1}{\lambda_{\text{reev}}} \sum_{i, x_i \text{was reevaluated}} \big(2\Delta_i - \Delta_\theta^{\text{lim}}(\text{rank}(f_i^{\text{new}}) - \mathbb{I}\{f_i^{\text{new}} > f_i^{\text{old}}\}) - \Delta_\theta^{\text{lim}}(\text{rank}(f_i^{\text{old}}) - \mathbb{I}\{f_i^{\text{old}} > f_i^{\text{new}}\})\big) \ .$$

The indicator function $\mathbb{I}$ is one if its argument is true and zero otherwise. The parameter $\theta \in [0, 1]$ (here set to 0.2) controls the level of noise we tolerate and $\Delta_\theta^{\text{lim}}(r)$ denotes the $\theta \times 50$ percentile of the possible rank changes (given by the $2\lambda - 1$ values $|1 - r|, |2 - r|, \dots, |2\lambda - 1 - r|$) when having the original rank r.

If $s > 0$ we increase the number of evaluations in the computation of a fitness value by a factor of α. Otherwise we decrease the number of evaluations by $1/\alpha$.

The reevaluation is done before the environmental selection in the standard CMA-ES, which uses the median of the fitness values of the reevaluated individuals for ranking. The additional fitness evaluations increase the computational costs per generation. However, we reevaluate on average only $\overline{\lambda}_{\text{reev}} = \max(\lambda/10, 2)$ individuals in each generation.

3 Support Vector Machines

Support vector machines are considered state-of-the art in machine learning for pattern recognition, in particular for binary classification [1, 2].

In supervised learning we consider an input space X and an output space $Y = \{-1, 1\}$. The learning is driven by sample data $S = \{(x_1, y_1), \dots, (x_\ell, y_\ell)\}$ with $x_i \in X$ and labels $y_i \in Y$ for $1 \le i \le \ell$ drawn independently from some fixed unknown distribution p over $X \times Y$. The goal of binary classification is to infer from S a hypothesis $h : X \to Y$ minimizing the expected loss $R_p(h) = \int_{X \times Y} L(y, h(x)) \, \mathrm{d}p(x, y)$ corresponding to the generalization error. We consider the 0-1-loss given by $L(y, h(x)) = (-h(x)y + 1)/2$ (i.e., the classification error).

Support vector machines transfer the input data to a feature space and perform linear classification in that space. Given a positive semi-definite kernel function $k : X \times X \to \mathbb{R}$ ($\forall x, x' \in X, \ \forall c_1, \dots, c_m \in \mathbb{R} : \ \sum_{i,j=1}^m c_i c_j k(x, x') \ge 0$),

we consider the feature space $\mathcal{H}_k = \text{span}\{k(x,\cdot) \,|\, x \in X\}$ and the function class $\mathcal{H}_k^b = \{f(x) = g(x) + b \,|\, g \in \mathcal{H}_k, b \in \mathbb{R}\}$. We classify according to the sign of a function $f \in \mathcal{H}_k^b$. The decision boundary induced by f is a hyperplane in $\mathcal{H}_k$.

Then the hypothesis generated by a 1-Norm Soft Margin SVM corresponds to a solution of

$$\underset{f \in \mathcal{H}_k^b}{\text{minimize}} \quad \frac{1}{\ell} \sum_{i=1}^{\ell} L_{\text{hinge}}(y_i, f(x_i)) + \frac{\gamma_\ell}{2} \|f\|_k^2$$

where $\gamma_\ell = (\ell C)^{-1}$ and the (semi-)norm $\|.\|_k$ is inherited from $\mathcal{H}_k$ to $\mathcal{H}_k^b$. The loss function is given by $L_{\text{hinge}}(y, f(x)) = \max(0, 1 - yf(x))$. That is, we do not only penalize if a pattern x is classified wrongly (i.e., $yf(x) < 0$), but also if the pattern is too close to the separating hyperplane in the sense that $f(x)$ does not meet the functional target margin (i.e., $|f(x)| < 1$). The parameter $C > 0$ controls the trade-off between the optimization goals of reducing the empirical loss measured by L_{hinge} and the complexity of the hypothesis measured by $\|.\|_k$.

The most frequently used kernel (for $X \subset \mathbb{R}^n$) is the radial Gaussian kernel $k(x, x') = \exp(-\gamma \|x - x'\|^2)$ with a single bandwidth parameter $\gamma > 0$. A standard extension is the Gaussian kernel with feature scaling $k(x, x') = \exp\left(-\sum_{i=1}^n \gamma_i (x_i - x_i')^2\right)$, which is also known as automatic relevance detection (ARD) kernel and has as many degrees of freedom as the input space has dimensions. The regularization parameter C and the parameters of the kernel function are called hyperparameters. Their proper selection is the model selection problem for SVMs.

4 A Fitness Function for SVM Model Selection

In this section we introduce a natural objective function for SVM model selection. Because the objective function is impractical to compute we propose a randomized variant, which allows to trade-off accuracy and time to compute the objective function value.

4.1 Hold-Out Sets and Cross-Validation

The probably most simple type of objective function for model selection is the error on a hold-out set. Assume we use a fraction of, for example, 1/5 of the training data as a hold-out set. Then we train a learning machine on the remaining 4/5 of the data and compute the fraction of errors on the hold-out set. This error measure is an unbiased estimate of the generalization error of a machine trained on a dataset of size $(4/5)\ell$ sampled i.i.d. from the data generating distribution p. Usually the optimal parameters for this machine will be quite good for a machine trained on the whole dataset of size ℓ, such that this objective function seems to be a simple and appropriate criterion for model selection. But it turns out that the hold-out error has a high variance, in the sense that it strongly depends on the particular partition of the dataset into training and

validation sets. This effect is very pronounced for small datasets and for small hold-out sets. On the other hand, the larger the hold-out set the smaller becomes the remaining training set, and this in turn imposes a larger bias in the estimation of the generalization error, because fewer examples are used for training. Furthermore, the asymmetry between the roles of the reduced training set and the hold-out set is dissatisfactory.

Cross-validation is a simple procedure which improves on these points. However, the partition of the data into training and hold-out sets remains arbitrary. In a k-fold cross-validation procedure the data S are split into k disjoint subsets $S_1, \ldots, S_k$ of roughly equal size. Then for each $i \in \{1, \ldots, k\}$ a machine is trained on the dataset $S \setminus S_i$ and the error E_i on the corresponding hold-out set S_i is computed. Finally the total error $\sum_{i=1}^{k} E_i$ is the so-called k-fold cross-validation error. In this procedure the underlying loss function is evaluated exactly once on each training example. In the machine learning literature, common values for the parameter k range from three to ten, and the choice $k = 5$ can be considered a default value [12].

Compared to the simple hold-out error the variance of the generalization error estimate is reduced, but because the data used in the different partitions are of course not independent the reduction of the variance is not by a factor of $\frac{1}{k}$. In general the cost for the computation of the cross-validation error is k times the cost of the computation of the hold-out error. Especially for small datasets the cross-validation error can heavily depend on the partition $S_1, \ldots, S_k$ of the dataset and thus has a relatively high variance w.r.t. the choice of the partition. This is an unsatisfactory situation as there is no such thing as a canonical data partition.

4.2 Bootstrapping

Conceptually it is straight-forward to avoid the problem of having an a priori fixed partition of the data. We define the set $J_n = \left\{ I \subset \{1, \ldots, \ell\} \;\middle|\; |I| = n \right\}$ of index subsets of size n, leading to the objective function

$$\bar{B} = \frac{1}{|J_n|} \sum_{I \in J_n} \left(\sum_{i \in I^C} L(y_i, h_I(x_i)) \right)$$

where L is a loss function and h_I is the hypothesis constructed from the data indexed by I. Each summand of this objective function computes the hold-out error of the hold-out set $I^C := \{1, \ldots, \ell\} \setminus I$, evaluated on the hypothesis h_I. In the style of the k-fold cross-validation procedure we can choose $n = \lfloor \frac{k-1}{k} \ell \rfloor$, and, as usual, we consider $k = 5$ as the default.

This objective function has the conceptual advantage to be completely symmetric w.r.t. the partition of the dataset into training and hold-out set and computes the probably best possible estimate of the generalization error of a machine trained on n examples. It has the disadvantage that the set J_n grows according to $|J_n| = \binom{\ell}{n}$ which is clearly computationally intractable.

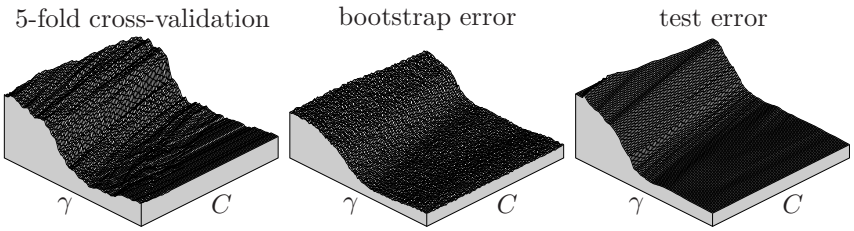

Fig. 1. The plots show the error landscapes for an instance of the `chess` problem with $\ell = 200$ training points (see Section 5) with the radial Gaussian kernel over an equidistant grid on the logarithmic scale for C and γ. Here, the bootstrap error was approximated by averaging over 100 i.i.d. drawn dataset partitions $I \in J_n$ per grid point.

Therefore, we introduce the random variable

$$\hat{B} : J_n \to \mathbb{R} \qquad I \mapsto \sum_{i \in I^C} L(y_i, h_I(x_i))$$

with the uniform distribution for $I \in J_n$. For each fixed index set I we get the hold-out error function $\hat{B}(I)$, which is a function of the SVM hyperparameters. We write $\hat{B}$ for the randomized objective function that picks a random $I \in J_n$ for each of its evaluations. It clearly holds $\mathbb{E}[\hat{B}] = \bar{B}$. This way we are able to avoid a systematic bias resulting from a fixed partition of the data like in the cross-validation procedure, at the cost of a randomized objective function. We will refer to this objective function as the bootstrapping error and use it with the standard 0-1-loss (counting misclassified test patterns).

For the minimization of $\bar{B}$ based on evaluations of $\hat{B}$ we need a search strategy that can deal with a non-differentiable and noisy objective function. This randomized objective function takes as long to evaluate as the simple hold-out error, but we have to be prepared for relatively long optimization runs due to the need for the search algorithm to handle the uncertainty in the objective function evaluations, for example when it is necessary to compute statistics over many evaluation in order to obtain sufficiently reliable information.

Of course there are a lot of possible criteria in between the randomized hold-out error $\hat{B}$, standard cross-validation, and full bootstrapping. We can choose a subset of J_n of considerably smaller size (which should be as symmetric as possible), or take the mean over a few index sets sampled from J_n. The most straight-forward example is to randomly pick a new partition of the dataset for each evaluation of the cross-validation error, which requires a search strategy that can deal with uncertain function evaluations as needed for the minimization of $\hat{B}$. We could even define a (weighted) mean over all choices of n. This leads to a large number of deterministic or randomized objective functions, but for the sake of clarity and for conceptual reasons we stick to the basic randomized hold-out error $\hat{B}$.

The plots in Fig. 1 illustrate the difference between cross-validation and bootstrap error. It is obvious that minimization of the newly proposed bootstrap

error gives much more reliable results than cross-validation with a fixed partition of the data.

5 Experimental Evaluation

The focus of our experiments is to assess the effect of uncertainty handling via self-adaptation in the CMA-ES in combination with our new model selection criterion. As discussed above, the UH-CMA-ES algorithm decides automatically how many averages it computes to make its fitness evaluations sufficiently reliable. We compare this strategy to the standard variant of the CMA-ES. Because the standard CMA-ES has no special uncertainty handling, we incorporate the averaging into the objective function simply by computing the statistics over a fixed number of realizations in each evaluation. Thus, we ask for the differences of averaging at the level of the objective function or the search algorithm. Furthermore, we demonstrate that the standard cross-validation procedure indeed suffers from its fixed partition of the data.

5.1 Setup

We consider four experimental setups. The first and most naïve strategy (referred to as `CMA-1`$\times$) is to apply the standard CMA-ES to the randomized bootstrapping objective function $\hat{B}$, ignoring its uncertainty. A second strategy (`CMA-5`$\times$) is to use a fixed average of k evaluations of $\hat{B}$ as a fitness function for the CMA-ES. In the style of cross-validation we use $k = 5$ evaluations in our experiments.[1] The third strategy in the comparison, `CMA-CV`, is 5-fold cross-validation without uncertainty, that is, using a random but fixed partitioning of the data. Again, the CMA-ES is used to minimize the resulting error function. We compare these strategies to the UH-CMA-ES applied to the $\hat{B}$ objective function.

As a proof of concept, the experiments are carried out on four benchmark datasets. In the *chess board problem* we consider the input space $X = [0, 4)^2$ and sample x from the uniform distribution on X. Then we assign a label according to the fixed rule $y = (-1)^{\sum_{i=1}^{2} \lfloor x_i \rfloor}$. This rule assigns labels according to the colors of the fields of a chess board of size 4×4 [13]. This distribution will be referred to as the `chess` problem. We use the radial Gaussian kernel for this problem. The next task is called *sparse coordinate problem* (`sparse` problem for short). To generate a sample we draw $n \in \{1, \ldots, 6\}$ uniformly at random and set $S = \{1, \ldots, 20\} \setminus \{n\}$. For $n \leq 3$ we assign the positive label $y = +1$, and set $y = -1$ otherwise. Then we randomly remove four more elements from the set S. The final representation is chosen to be $x \in \mathbb{R}^{20}$ with $x_i = 0$ if $x \in S$ and $x_i = 1$ otherwise. We apply the Gaussian ARD kernel to this problem. It should identify the first six coordinates as highly discriminative, while the remaining coordinates

[1] We could instead use any other number of averages, or use a fixed rule how the number of averages changes over time. However, any such strategy requires problem specific knowledge. Because we aim at a general and automated solution we will assume such expert knowledge to be not available in this study.

provide no useful information. In addition to the artificial distributions `chess` and `sparse`, we consider the benchmark problems `banana` and `image` from the benchmark collection introduced by [14] and apply SVM classifiers with radial Gaussian kernels to these problems.

The search space for the evolutionary algorithms is a low-dimensional vector space, and we use the parameterization $\log(C)$ and $\log(\gamma)$ (or $\log(\gamma_i)$ for the Gaussian ARD kernel) of regularization and kernel parameters. This allows for unconstrained optimization. All parameters of the CMA-ES are set to default values [8], the initial global step size is set to $\sigma = 1$.

Each strategy is given $100,000$ evaluations of $\hat{B}$. This relatively high (in practice presumably too high) number of evaluations is chosen because it is sufficient for the CMA-ES without uncertainty handling to converge. To generate reliable results, we conducted 1000 trials for all experiments and evaluated the classification performance of the resulting machines with a Mann-Whitney U-test. Of course, this requires that the trials are statistically independent. Due to a lack of data this is impossible to ensure on standard benchmark datasets, because there is no alternative to re-using the same data in each trial. The possibility to sample arbitrary amounts of data and thus to ensure statistical independence is the main motivation for the consideration of artificial test problems such as `chess` and `sparse`. All four methods in the comparison are reasonable strategies for SVM model selection. Therefore we expect the differences to be small. Further-

	`chess`							`sparse`						
method	q_{25}	q_{50}	q_{75}	(1)	(2)	(3)	(4)	q_{25}	q_{50}	q_{75}	(1)	(2)	(3)	(4)
(1) `CMA-1×`	0.149	0.168	0.187	—	$\ggg$	$\gg$	$\ggg$	0.257	0.274	0.295	—	$\ggg$	$\ggg$	$\ggg$
(2) `CMA-5×`	0.146	0.162	0.181	$\lll$	—	0.13	$\gg$	0.250	0.262	0.281	$\lll$	—	0.44	$\ggg$
(3) `CMA-CV`	0.147	0.163	0.183	$\ll$	0.87	—	$\ggg$	0.251	0.263	0.278	$\lll$	0.56	—	$\ggg$
(4) `UH-CMA`	0.143	0.159	0.178	$\lll$	$\ll$	$\lll$	—	0.249	0.258	0.274	$\lll$	$\lll$	$\lll$	—
	`banana`							`image`						
method	q_{25}	q_{50}	q_{75}	(1)	(2)	(3)	(4)	q_{25}	q_{50}	q_{75}	(1)	(2)	(3)	(4)
(1) `CMA-1×`	0.126	0.138	0.158	—	$\ggg$	$\ggg$	$\ggg$	0.119	0.133	0.154	—	$\ggg$	$\ggg$	$\ggg$
(2) `CMA-5×`	0.124	0.135	0.149	$\lll$	—	0.62	$>$	0.116	0.129	0.144	$\lll$	—	0.52	0.38
(3) `CMA-CV`	0.124	0.134	0.149	$\lll$	0.38	—	$>$	0.116	0.129	0.144	$\lll$	0.48	—	0.34
(4) `UH-CMA`	0.123	0.132	0.147	$\lll$	$<$	$<$	—	0.116	0.129	0.145	$\lll$	0.62	0.66	—

Table 1. Absolute and relative performance of the classifiers resulting from the parameters found by the different strategies. The errors are given as 25%, 50%, and 75% quantiles over 1000 trials. The comparison matrix on the right uses the symbols $<$, $\ll$, and $\lll$ to indicate that the method in this row performs significantly better than the method in this column with significance levels 0.05, 0.01, and 0.001, respectively. A one-sided Mann-Whitney U-Test (also known as Wilcoxon rank sum test) is used for the comparison. Analogously, the symbols $>$, $\gg$, and $\ggg$ indicate that the method in that row performs significantly worse with the corresponding significance level. If the differences are not significant at a level of at least 0.05 the significance level is reported. Note that for the fixed size datasets `banana` and `image` the trials are not independent, such that the "true" significance levels are in general worse.

more, the fitness function and the function used to judge the final parameters differ. The objective function of model selection is, of course, the generalization error, which is estimated by the test error. In contrast, we have no alternative to fitness functions computable from the available training data. This difference between the functions used for training and for evaluation is an additional source of perturbations in the results. These two reasons make clear that we need a relatively large number of trials in order to obtain statistically significant results. We used training datasets of size $\ell = 100$. For the artificial problems we sampled test sets of size $100,000$, giving extremely reliable estimates of the generalization error. For the fixed size benchmark problems we used the remaining examples for testing, which amounts to $5,000$ test examples for the `banana` benchmark and $2,210$ for the `image` problem.

5.2 Results and Discussion

The results are summarized in Table 1. The significant differences between the methods clearly indicate that averaging over several evaluations of $\hat{B}$ improves the solution. The CMA-ES profits from automatic uncertainty handling. The UH-CMA-ES method performs clearly best, although it evaluates only a comparatively small number of search points. For the SVM model selection problem, and in particular for the fine-tuning of the SVM hyperparameters, the reliable evaluation of a small number of candidates turns out to be more successful than the cheap but unreliable evaluation of a large number of search points. Furthermore, the experiments indicate that the robust estimation of the generalization error requires a large number of averages over simple hold-out error evaluations.

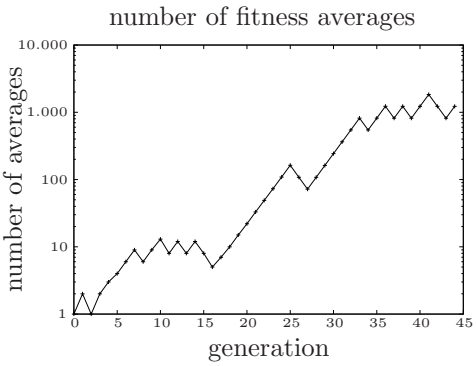

Fig. 2. Typical evolution of the number of averages over the generations of the UH-CMA-ES. Usually only few generations can be evaluated with a limit of 100,000 evaluations of $\hat{B}$.

The plot in Fig. 2 clearly reveals that there is no uniformly best number of averages for all search points, but that the number of averages grows to large numbers if needed. This result is not surprising. Of course, the closer the CMA-ES comes to a local optimum, the worse gets the signal-to-noise ratio. This drives the UH-CMA-ES algorithm to large numbers of averages in late generations. The algorithm very quickly identifies the region of well-generalizing classifiers, and then gradually switches over to fine-tuning of the hyperparameters which requires a large sample per individual.

In our experiments, the bootstrap error $\bar{B}$ is clearly superior to the cross-validation error if the uncertainty of $\hat{B}$ is handled properly.

6 Conclusion

We applied the CMA-ES with and without uncertainty handling mechanism to the problem of model selection for SVMs. As a new model selection criterion, we proposed the minimization of the bootstrapping error $\bar{B}$ based on evaluations of its estimate $\hat{B}$. There are good arguments to prefer this objective function over standard cross-validation. Our experiments support these theoretical considerations and show the advantage of automatic uncertainty handling for this problem. The small overhead of the uncertainty handling for the re-evaluation of some individuals is clearly justified by the resulting improvement in performance.

References

1. Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning **20** (1995) 273–297
2. Evgeniou, T., Pontil, M., Poggio, T.: Regularization networks and support vector machines. Advances in Computational Mathematics **13** (2000) 1–50
3. Friedrichs, F., Igel, C.: Evolutionary Tuning of Multiple SVM Parameters. Neurocomputing **64** (2005) 107–117
4. Mersch, B., Glasmachers, T., Meinicke, P., Igel, C.: Evolutionary Optimization of Sequence Kernels for Detection of Bacterial Gene Starts. International Journal of Neural Systems **17** (2007) 369–381 Selected paper of ICANN 2006.
5. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation **9** (2001) 159–195
6. Hansen, N., Müller, S.D., Koumoutsakos, P.: Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). Evolutionary Computation **11** (2003) 1–18
7. Hansen, N., Niederberger, A.S.P., Guzzella, L., Koumoutsakos, P.: Evolutionary optimization of feedback controllers for thermoacoustic instabilities. In Morrison, J.F., Birch, D.M., Lavoie, P., eds.: IUTAM Symposium on Flow Control and MEMS, Springer-Verlag (2008)
8. Hansen, N., Niederberger, A.S.P., Guzzella, L., Koumoutsakos, P.: A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. IEEE Transactions on Evolutionary Computation (2008) In press.
9. Beyer, H.G.: Evolution strategies. Scholarpedia **2** (2007) 1965
10. Jin, Y., Branke, J.: Evolutionary optimization in uncertain environments—a survey. IEEE Transactions on Evolutionary Computation **9** (2005) 303–317
11. Arnold, D.V.: Noisy Optimization With Evolution Strategies. Kluwer Academic Publishers (2002)
12. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer-Verlag (2001)
13. Glasmachers, T., Igel, C.: Gradient-based Adaptation of General Gaussian Kernels. Neural Computation **17** (2005) 2099–2105
14. Rätsch, G., Onoda, T., Müller, K.R.: Soft Margins for AdaBoost. Machine Learning **42** (2001) 287–320