# Efficient tuning in supervised machine learning

Koch, P.

# Chapter 5

# Improving the efficiency of tuning

In this chapter we describe approaches, which can diminish the computation time of parameter tuning. As tuning can be very expensive for complex ML tasks, we want to pay more attention to the effectiveness of the methods. In general we analyze three different approaches for saving computation time, that is

1. Tuning with limited budgets
2. Efficient sampling during the optimization
3. Automatically setting the number of repeated objective function evaluations within SPO, by using optimal computing budget allocation (OCBA) [39]

While the first and third approach aim at saving computation time by the optimization procedure, the second approach is proposed for diminishing the computation time of the objective function. Nevertheless all three approaches can be efficient solutions towards practical use of parameter tuning.

We give a brief summary about tuning with limited budgets in Sec. 5.1. As a new investigation proposed in this thesis we analyze the effect of sub-sampling during tuning on the classification accuracy in Sec. 5.2. In Sec. 5.3 we introduce an approach based on the optimal computing budget allocation (OCBA) to reduce the number of function evaluations.

## 5.1 Tuning with limited budgets

The tuning of hyperparameters and pre-processing parameters can be expensive in terms of required computation time. Usually, feature selection approaches, as the wrapper approaches described in Sec. 2.2.2 or other tuning runs, need many model evaluations. As a consequence, tuning is seldom performed in practice or parameters are just set by hand.

An often used strategy to limit the computational budget is to limit the evaluations for the optimization procedure. In Sec. 4.3.2 we performed a series of experiments on the acid concentration task. It showed that the budget has an effect on the tuning results. For

SPOT, better results were obtained, when the budget was increased. The opposite was the case with CMA-ES, where the error on the test data suddenly became worse compared with lower budgets. The number of necessary iterations for a tuning task has to be given for each problem. In general, limiting the budget is not recommendable, since this usually impedes the optimizer to converge too early. Nevertheless computation time can be saved using model-assisted optimization, like efficient global optimization (EGO). Here only a few parameter sets are evaluated on the objective function, but the main part of the optimization is performed on a surrogate model, or response surface methodology (RSM), of the real objective function. Konen *et al.* [155] showed, that using these strategies very restrictive budgets like 50–200 evaluations for more than six parameters are necessary to calculate competitive parameter sets.

## 5.2 Efficient sampling for tuning experiments

Although Computational Intelligence (CI) methods can generate good and robust solutions for global optimization problems, it is known that they sometimes require a large number of function evaluations. Unfortunately, ML tasks are usually very expensive to evaluate and quick solutions are requested by the users. In this section we investigate how computation time can be saved in order to make CI methods more applicative for ML tasks.

### 5.2.1 Research questions

We propose the following hypotheses:

**Q1** Are the results of parameter tuning more subject to noise, when smaller fractions $X$ of the training data are used?

**Q2** Are the optimal design points found by an efficient optimization strategy nevertheless competitive (as long as the training set size is not *too* small)?

The question **Q1** aims at analyzing the variance of the tuning results, when the training set size is decreased. If **Q1** can be answered affirmatively, we should be able to measure this effect directly by an increased variance of the prediction error. If we can answer **Q2** affirmatively, considerable computation speedups in tuning ML models should be possible, without a too high decrease in generalization performance.

### 5.2.2 Related work

Previous work in analyzing the effects of the chosen sample size has been mainly done in fields like statistics and machine learning. A good overview of different strategies for data sampling can be found in Cochran [44]. A description of resampling methods for optimization in data mining has been given by Bischl *et al.* [18]. Raudis and Jain [202] and Jain and Zongker [125] discuss the influence of sample sizes on the results of feature

**Table 5.1:** Datasets used for the training set size experiments.

| Dataset | Records | Min. Training Size | Max. Training Size | Validation and Test Size | Number of Parameters |
|---|---|---|---|---|---|
| Sonar | 208 | 8 | 124 | 41 | 2 |
| Glass | 214 | 8 | 128 | 42 | 2 |
| Liver | 345 | 13 | 207 | 69 | 2 |
| Ionosphere | 351 | 14 | 210 | 70 | 2 |
| Pima | 768 | 30 | 460 | 153 | 2 |
| AppAcid | 4400 | 176 | 2640 | 880 | 12 |
| DMC-2007 | 50000 | 2000 | 30000 | 10000 | 7 |

selection, which is in a certain way related to the parameter optimization task in this section. In statistics, sample sizes are frequently discussed in terms of statistical studies like significance tests [164]. In machine learning, sampling strategies like the bootstrap [66] have been well analyzed and are often applied in practice. However, to our knowledge no study exists where the size of the underlying training data is diminished during parameter tuning. The analysis in this chapter is based on the work of Koch *et al.* [149].

## 5.2.3 Experimental analysis

In this section we describe the experimental setup and the methodology of our empirical study.

### 5.2.3.1 Datasets

All experiments presented in this study were performed using datasets from the UCI repository [84], which can be regarded as the simpler datasets, the DMC 2007 data from the Data Mining Cup 2007 [33], and the real-world dataset for predicing acid concentrations which has been introduced in section 4.3.2. In Tab. 5.1 an overview about the datasets and their sizes is given. We present the minimal training set size for each dataset, and also the sizes of the test and validation sets. Each model is evaluated a) during tuning on the validation data and b) after tuning on the independent test data. The sizes for the test and validation sets are equal and stay constant all the time at 20% of the total dataset size. In Tab. 5.1 these sizes are given as *Validation and Test Size*.

Every time a fixed set of 20% of the patterns was set aside prior to tuning for testing purposes. From the remaining 80% of the data (subset $D_{train}$), we use a fraction $X$ from $X_{min} = 5\%$ to $X_{max} = 75\%$ for training and a fraction of 25% for validation (which is 20% of all data). See Tab. 5.2 for illustration. At the end of tuning a best design point is returned. Using this best design point, we ran a final "full" training with all data in $D_{train}$ (80%) and evaluated the trained model on the test set (20%). Since the training, validation and test sets were drawn at random we repeated all of our experiments ten times.

**Table 5.2:** Splitting of data. We use fractions from 4% to 60% of the data for model training, 20% for validation during the tuning and the remaining 20% for independent testing.

| Training $\longleftrightarrow$ | not used | Validation | Test |
|---|---|---|---|

While a first benchmark of tuning algorithms has been performed by Konen *et al.* [155], it remained unclear, whether the results also hold for smaller training set sizes. Now we also compare the sequential parameter optimization toolbox (SPOT) as a tuning algorithm with a Latin hypercube design (LHD) as a simple, but robust sampling heuristic. LHD is based on the following procedure: We chose random and roughly equally distributed design points from the region of interest and evaluated the design three times. Again, the best point is taken for the 'full' training as above.

#### 5.2.3.2 Tuning setup

SPOT can be controlled through various settings, which have to be adapted slightly for each task. We briefly present our settings for the experimental study here (Tab. 5.3). With $150$ function evaluations for the UCI experiments we chose a rather large number of evaluations compared to the other (real-world) datasets. Our aim was to achieve a good and clear convergence to an optimum. For this reason we considered to analyze simpler datasets first, since complex datasets require much more time for such experiments. Nevertheless we also set a number of $200$ function evaluations for AppAcid, which proved to be a good and sufficient number in one of our last studies [155]. It has to be noted that the dimensionality of the parameter space for AppAcid is $12$, while it is only $2$ for the UCI benchmarks.

**Table 5.3:** SPOT Configuration

| Setting | UCI | AppAcid |
|---|---|---|
| function evaluations | 150 | 200 |
| initial design size | 10 | 24 |
| sequential design points | 3 | 3 |
| sequential design repeats | $\{1, 3\}$ | 2 |

As region of interest (RoI) for the UCI datasets we set quite large ranges (as we have enough evaluations for these benchmarks). The range of $\gamma$ is in $[0.0, 1.0]$ and the range of cost $C$ in $[0.0, 10.0]$. For the other applications (AppAcid, DMC2007) we relied on the same settings as in our previous experiments, see [155] for more information.

#### 5.2.3.3 Resampling

First of all, $k$ subsets $D_1, \ldots, D_k \subset D_{train}$ of the complete training data $D_{train}$ lead to models $M_1, \ldots, M_k$ which are presumably not equal when the training data subsets are not

equal, although hyperparameters $\gamma$ and $C$ are identical (on the same training data SVM is deterministic). Different strategies are possible for sampling the training subsets during tuning:

(A) Choose a subset $D_1 \subseteq D_{train}$ *once* and use it for the whole parameter optimization process. We call this option *parameter tuning without resampling*.

(B) In each call $i$ of the objective function, choose a new subset $D_i \subseteq D_{train}$ for training. If a design point is evaluated repeatedly, e.g., $n$ times, we choose different subsets $D_1, D_2, \ldots, D_n$, train models for them and use an aggregation function (here: the mean) to return an objective function value. We call this option *parameter tuning with resampling*.

#### 5.2.3.4 Results

In Fig. 5.1 and Fig. 5.2 we present boxplots of the SPOT hyperparameter tuning for the Sonar and AppAcid datasets respectively. We distinguish between the error achieved on the validation data when using only a smaller training set fraction $X$ (Error VALI Set) and the independent test set error when a complete re-training with the optimal parameter setting and the complete training data is performed (Error TEST Set). Note that the results on the test set can be better than the error obtained on the validation set (the tuning result), which is caused by using the complete training data when measuring the error on the test set. The results shown in Fig. 5.1 and Fig. 5.2 were optimized using SPOT with sampling strategy (B) and a total number of $3$ repeated evaluations for each design point. The validation error in both plots is clearly increasing if the training set size $X$ is reduced from $X = 75\%$ to $X = 5\%$. However, the same does not necessarily hold for the test data. While the mean errors and their variances are large for small training fractions, they are roughly constant and small for all $X \geq 15\%$.

This is a promising result, as it may allow us to use only a sub-sample of the complete data during tuning. As we can see in Tab. 5.4, good tuning results with a very small number of training data (here $X = 10\%$) were observed as well for all other datasets. If we take the best parameters from such a tuning process and re-train *once* with the complete training data, we obtain results on independent test data which are competitive with a much more time-consuming "full" tuning.

We observed, that the prediction accuracies for models trained on different data are very noisy. Thus, the chosen sampling strategy (see Sec. 5.2.3.3) does have an impact on the final results. A comparison of the sampling strategies (A) and (B) showed that we can achieve the most stable results using strategy (B). While it is easier to obtain a good parameter setting for a specific training/validation set with strategy (A), this does not hold for sampling strategy (B). When strategy (A) is used, the optimization always uses some training data which is never changed during the optimization. With deterministic learning algorithms like

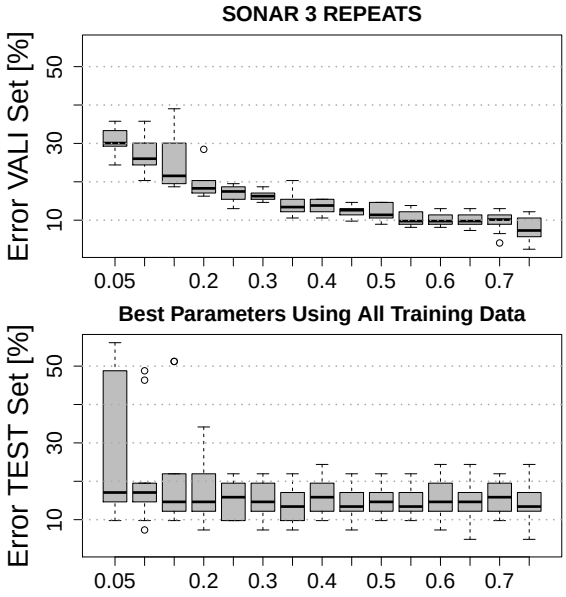


**Figure 5.1:** Tuning results with SPOT for Sonar dataset using 3 repeats. The $x$-axis shows the training set fraction $X$.

SVM, this leads to a deterministic optimization problem, which means to minimize the error always on the same validation set. In strategy (B) instead, the training and validation sets *change in each iteration*. This has an impact on the learned prediction model, and also on the error on the validation set (the training set and validation sets of two consecutive iterations are not comparable). While this strategy seems to be disadvantageous at first, it has to be noted, that we force the learning algorithm indirectly to be good generalizing. With strategy (B) the parameter setting must perform well on different validation sets, otherwise the setting will be discarded soon by the optimization procedure. For this reason, this strategy is not prone to overfitting as is strategy (A), although it might be valuable to get more stable results by testing parameter settings multiple times, avoiding to discard a good setting by mistake. The least is known from design of experiments (DoE) [76], and is achieved by integrating replicates, meaning that each parameter setting is evaluated multiple times. Another option is to introduce noise estimating methods, which are unfortunately only available for some surrogate models, e.g., Kriging and the re-interpolation method by Forrester *et al.* [82].

Due to the high noise levels usually occurring in ML we recommend a number of repeated evaluations greater than $1$ (from now on we always set the repeats to $3$ in this study). We think that repeated evaluations are an important factor to circumvent wrong decisions of the optimizer operating in a noisy environment.

Regarding the comparison of SPOT and LHD we show in Fig. 5.3 that SPOT usually

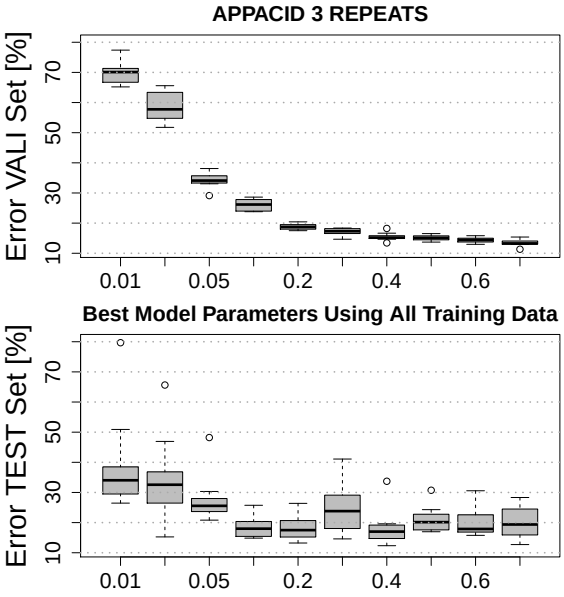


**Figure 5.2:** Tuning results with SPOT for AppAcid dataset using 3 repeats. The $x$-axis shows the training set fraction $X$.

yields better parameter settings when trained with a fraction of the data, especially for $X \leq 10\%$. Results degrade for very low training set sizes (1% or 2%). They are however competitive for all $X \geq 10\%$ (SPOT) and $X \geq 20\%$ (LHS): the tuning results are as good as with a tuning on the "full" training data and the models generalize well on unseen data.

#### 5.2.3.5 Computation times

The characteristics of the computation times for different training set sizes are shown in Fig. 5.4 for the AppAcid dataset. Note that the curve which appears to be linear here can have a different appearance for other datasets. The roughly linear slope has its origin in the model building process for the acid concentration task. This process includes several other operators beneath SVM training, e.g., pre-processing (Principal Component Analysis and feature selection). In other cases the pure SVM computation time might increase quadratically with the number of training samples, leading to even larger computation time savings.

## 5.3 Optimal computing budget allocation

Chen *et al.* [39] developed a strategy for estimating the optimal budget for repeated design point evaluations. In EGO this strategy was proposed together with the SPOT environment in [9]. The principle in DoE is, that design points are repeatedly evaluated. There are some advantages and disadvantages for performing more than one evaluation of the same design

**Table 5.4:** Test-set error rates (mean and standard deviation of ten runs) for all datasets investigated. We show results when using small and full training data during tuning, after re-training once with the best parameters found. In case of DMC-2007 we show relative gain instead of error rate.

| Dataset | $X = 10\%$ median (std.dev.) | $X = 75\%$ median (std.dev.) |
|---|---|---|
| Sonar | 17.07 (14.3) | 13.41 (5.5) |
| Glass | 28.57 (7.7) | 28.57 (7.6) |
| Liver | 36.23 (4.3) | 31.88 (6.5) |
| Ionosphere | 5.71 (2.4) | 5.71 (2.4) |
| Pima | 23.20 (2.5) | 23.20 (3.9) |
| AppAcid | 17.99 (3.3) | 19.38 (5.5) |
| DMC-2007 | 14.73 (2.0) | 15.66 (1.0) |

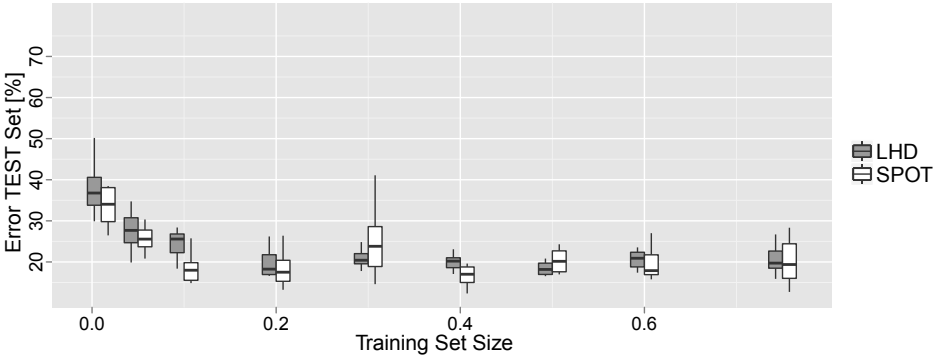


**Figure 5.3:** Comparison of SPOT and LHD algorithms on AppAcid.

point. Replications are useful, when the underlying objective function is noisy. This is usually the case in parameter tuning for machine learning model selection. Replications can help to get more robust surrogate models, especially after the initial design has been built. As an example in some engineering applications researchers need to perform analyses of the process before further steps are undertaken. However, other researchers think it is not necessary to perform repeated evaluations. Another solution is to evaluate more sequential points instead of spending evaluations in the beginning of the search. This approach rather aims at handling the noise by the surrogate model. This can be done using the re-interpolation technique by Forrester (cf. Sec. 2.5.5), where model uncertainties are considered by deploying an uncertainty band for the evaluated design points. It is difficult to give a clear indication when to use repeats and when not. Chen *et al.* [39] proposed a technique for finding the best number of replications for designs called optimal computing budget allocation (OCBA).

Bartz-Beielstein *et al.* [10] integrated OCBA in the SPO toolbox and analyzed it on five noisy test problems. They showed, that SPO using OCBA could outperform three

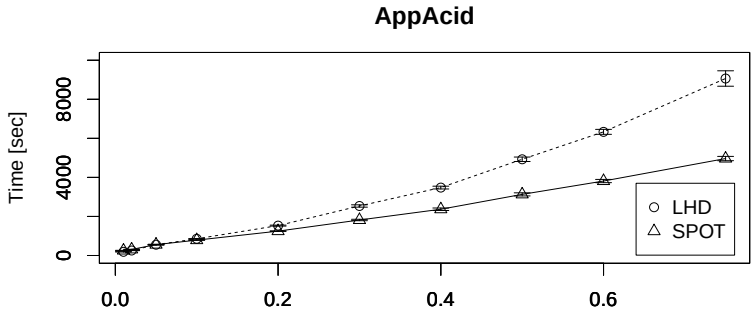


**Figure 5.4:** Computation time on the acid concentration task as a function of training set size $X$.

other optimization methods, including CMA-ES [107], Simplex search [185] and Simulated Annealing [143]. Also the robustness of SPO was increased by incorporating OCBA, leading to less outliers in the tuning results.

## 5.4 Conclusions

We showed that tuning with a low training set size very often leads to good results: an astonishing small fraction of the training set (10–15%) was sufficient to find with high probability a good model when performing one "full" training with the best design point parameters. The resampling strategy (B) (see Sec. 5.2.3.3) might be a crucial ingredient for this success with small training samples, but further research is needed to justify this claim.[1]

In the same way Koch *et al.* [149] investigated seven different datasets with sizes from 208 to 50,000 records. For bigger datasets like the acid concentration tasks large speedups (factor six to ten) are possible as Fig. 5.4 shows. Summarizing the results, we can conclude that both research questions **Q1** and **Q2** can be affirmed for the datasets used in this study. In the future we plan to search strategies for selecting the right sample sizes adaptively as a dynamic parameter during tuning.

[1] We note that Daelemans *et al.* [53] got negative results with a small training sample, but only on a specific word disambiguation task and without resampling.