The handle http://hdl.handle.net/1887/22055 holds various files of this Leiden University dissertation.

**Author:** Koch, Patrick  
**Title:** Efficient tuning in supervised machine learning  
**Issue Date:** 2013-10-29
Chapter 7

Efficient multi-criteria optimization in machine learning

In the previous chapters we showed that finding good parameters is essential for training a precise classifier. Up to now, this problem has been defined as a single-criteria optimization (SCO) problem, where the classification error or any other quality indicator was minimized. Systematic global optimization was used to optimize ML processes in [148, 149, 155]. Here, SPO [8] with Kriging surrogate models performed best for problems with very restrictive budgets.

However, in most cases the user is not only interested in high-quality prediction models, but also wants that model training and parameter tuning is performed in a reasonable amount of time. These objectives are usually conflicting, leading to the concept of multi-criteria optimization (MCO). Nowadays, model-assisted optimization algorithms are also available for multi-criteria optimization problems [69, 144, 194, 259]. Up to now this paradigm has received little attention in the ML community, and only few work has been spent to optimize the mentioned objectives at the same time.

One reason for this is in the nature of ML parameter tuning, where the evaluated design points are usually noisy. What is the source of noise in ML tasks? Besides the normal noise of the ML data, it is mainly related to the randomness in the selection of training, validation and test data which lead to different results, even for deterministic ML models. A surrogate model has to cope with such noisy responses. The noise in the quality response will be increased if we seek for solutions with low runtime, which is usually connected with smaller training set samples.

Summarizing, the typical challenges of noisy MCO can be formulated as follows: (a) finding a good approximation of the Pareto front, (b) coping with the noise and (c) finding a good approximation with very restricted budgets of function evaluations.

We aim at applying multi-criteria model-assisted optimization variants to solve multi-criteria ML tasks.
For this reason we define research questions in this chapter as follows:

Q1 Is multi-criteria optimization with surrogate modeling (Kriging) possible in the presence of strong noise?

Q2 Is it also possible when the budget for function evaluations (i.e., ML training runs) is very restricted?

Q3 Is it necessary to dampen the noise by averaging over repeated function evaluations, at the price of fewer allowed infills under the given budget?

Q4 Are the multi-criteria model-assisted optimization approaches better in finding good approximation sets than traditional design of experiments (DoE) techniques, e.g., LHS? Are there significant differences between the different model-assisted optimization approaches?

The chapter is structured as follows: in Sec. 7.1 we highlight related approaches. A general introduction of MCO is given in Sec. 7.2.1. In Sec. 7.3 we describe the setup of the study for efficient multi-criteria ML experiments. The experimental results are discussed in Sec. 7.4 and we give a conclusion in Sec. 7.5.

7.1 Related work

Because most supervised ML models like Support Vector Machines (SVMs) [246, 52, 215] are sensitive to their hyperparameter settings, an optimization is required until an optimal behaviour of the models can be guaranteed. For single-criteria ML problems we already discussed related work in Sec. 4.1 and showed that a tuning using SPO performs better on a set of benchmark problems than other state-of-the-art optimization heuristics (see Sec. 4.3). On the basis of these results Koch and Konen [149] discovered that tuning with small fractions of the available training data can lead to good parameter settings (see Sec. 5.1). But any a priori setting of the training set size without special problem knowledge remained virtually impossible. A solution to this issue is to explore a set of solutions, representing alternatives between small and large training set sizes, so that a suitable size is finally delivered to the user.

In ML research, MCO was firstly proposed by Liu and Kadirkamanathan [167]. They optimized a radial basis function network, where two objectives functions were considered to optimize the differences between the real non-linear system and the non-linear model, and another function to emphasize simpler models. Freitas [85] and Jin and Sendhoff [130] give comprehensive reviews about the employment of multi-criteria algorithms in ML. Jin [128] advocates to use Pareto-based approaches, covering both supervised and unsupervised learning.
In these approaches the authors used evolutionary multi-objective algorithms (EMOA). The required number of real function evaluations is considerably higher for these algorithms which can be problematic, because the computation time is usually very limited. Instead Knowles and Nakayama [146] discuss the use of surrogate-modeling techniques for multi-criteria optimization problems. The first EMOA using surrogate models was proposed by Giannakoglou et al. [91]. A popular variant of EGO for multi-criteria optimization was given by Knowles [144], the Pareto-EGO (Par-EGO). Later, EGO approaches using the hypervolume as infill criterion were introduced, e.g., the SMS-EGO by Ponweiser et al. [194], or the hypervolume-based EI criterion by Emmerich et al. [69], also referred to as $S$-metric based Expected Improvement (SExl) [249]. Another approach based on decomposition is the MOEA/D by Zhang et al. [260]. Recently, Zaefferer et al. [258] compared SMS-EMOA, a well-known solver without surrogate modeling, and four cutting-edge multi-criteria solvers with surrogate modeling (SMS-EGO, SExl-EGO, MSPOT and MEI-SPOT) on an optimization task without noise. An overview about the properties of multi-criteria EGO variants was given by Wagner et al. [249].

There is a large body of work on noisy Kriging-based optimization (NKO) for single-criteria optimization tasks [82, 119, 191, 247]. Picheny et al. [192] give a comprehensive overview and perform a benchmark of several NKO-approaches on a variety of well-known hard optimization problems with a steerable amount of additive noise. Compared to this, only few work has been done in the area of NKO for multi-criteria optimization tasks. An exception can be found in Knowles et al. [145], but to our knowledge there exists no related work of multi-criteria NKO in ML.

7.2 Methods

In this section we describe the general multi-criteria optimization task and give a short overview about existing algorithms solving such problems.

7.2.1 Multi-criteria optimization

In multi-criteria optimization, multiple objective functions are optimized in parallel:

$$\min_{\vec{x} \in S} (f_1(\vec{x}), f_2(\vec{x}), \ldots, f_m(\vec{x}))$$  \hspace{1cm} (7.1)

Note that the individual objective functions $f_1, \ldots, f_m$ can be conflicting, so that the sets of the optimal solutions for the functions cannot be unique. We say a point $\vec{x}_1$ dominates a point $\vec{x}_2$, if all functions values of $\vec{x}_1$ are smaller than or equal to those of the corresponding objective function values of $\vec{x}_2$, and at least one objective function value of $\vec{x}_1$ is strictly better:

$$\vec{x}_1 \prec \vec{x}_2$$  \hspace{1cm} (7.2)
A point $x^*$ is considered to be Pareto-optimal, if there does not exist any feasible point $\bar{x}^{(d)} \in S$ which would decrease one objective function value without a simultaneous increase in any other objective function value.

In the last years set-based approaches have been established to approximate the Pareto-optimal front [262], and here especially evolutionary multi-objective algorithms (EMOA) have become popular methods in practice. The advantage of set-based approaches is that a group of points distributed over the Pareto front is approximated within a single run of the algorithm, while in single-solution approaches only a single trade-off between the objectives is returned in the end.

### 7.2.2 Model-assisted multi-criteria optimization

The most important weakness of the usage of evolutionary algorithms is probably that they suffer from a large number of function calls required for converging to good regions of the search space. Especially in multi-criteria optimization this is a clear disadvantage and can result in prohibitive computation times. A frequently shared paradigm in this case is to use surrogate-functions which model the real fitness landscape and can help in performing optimization runs with much fewer real function evaluations [146].

In MCO, surrogate models must be learned for each objective. This set of surrogate models is then used for the prediction of promising new design points using EI definitions for multiple criteria [249].

The following EGO methods are popular in MCO:

**ParEGO** (Pareto optimization EGO by Knowles [144]): This was the first approach to use model-assisted EGO techniques in MCO. In each iteration a different random weight vector is used to combine the normalized objectives to a scalar function (scalarization). For each previously visited point this scalar function is calculated and a surrogate model is fitted. On this surrogate model the expected improvement (EI) criterion defined in Sec. 2.5.4.2 is used to determine the next infill point.

**SMS-EGO** ($S$-Metric Selection EGO by Ponweiser et al. [194]): A hypervolume-based infill criterion where the important part of the improvement function is based on the hypervolume increase due to a potential solution.

**SExI-EGO** ($S$-Metric Expected Improvement EGO by Emmerich et al. [69]): This is another hypervolume-based infill criterion where for each point with objective values and predicted variances given by the surrogate models the exact computation of the expected improvement in the hypervolume is done.

The hypervolume indicator or the $S$-metric is defined as the Lebesgue measure of the subspace that is dominated by the approximation set. To make this measure finite, the
7.3. Experimental analysis

subspace is cut from above by a reference point \( r \). The reference point is chosen in such a way that all points in the approximation set dominate it. The \( S \)-metric is also called a scalar indicator for measuring improvements in an approximation set to the Pareto front.

Since the \( S \)-metric indicates good theoretical properties in [249], we consider both the \( S \)-metric Selection-EGO (SMS-EGO) and the \( S \)-Metric Expected Improvement-EGO (SExI-EGO) as model-assisted algorithms for our experiments on multi-criteria optimization of ML models.

We compare these two EGO-approaches with Latin hypercube sampling (LHS) as a baseline. For LHS the full budget of real function evaluations are spent to build competing optimal LHS in the region of interest. The set of non-dominated solutions among the LHS points is determined and its hypervolume calculated.

7.2.3 Noise handling

Model evaluations in ML are usually subject to noise. Although learning algorithms like SVMs are deterministic, their training depends on the (random) order of feeding examples which can lead to different model accuracies. This can cause wrong decisions during tuning, because the returned objective function values are unstable and have variances significantly larger than zero.

Replications have been introduced in Sec. 2.5.5.1. We investigate below whether replicates have a beneficial effect on the overall quality. As another alternative the re-interpolation procedure by Forrester et al. [82] can be used. We compare replications and re-interpolation for multi-criteria tuning in ML.

7.3 Experimental analysis

We performed an experimental study using two objectives \( f_1 \) and \( f_2 \) for parameter tuning in ML. For the first objective we used the classification gain and for the second objective we measured the training time of the learning algorithm. It is very likely that these objectives are conflicting: the runtime increases, when more data is used for the learning. However, the classification result should improve in this case.

Different variants of model-assisted multi-criteria algorithms are analyzed to measure the performance of the EGO approach. All algorithms are implemented in Matlab version 7.12.0.635 (R2011a) and always Kriging is used together with different EI criteria based on the hypervolume. The learning itself was implemented in R using the TDMR framework [154].

7.3.1 Experimental setup

As initial design we always used a LHS consisting of \( n_{init} \) parameter design points, which were evaluated on the real objective function. We spent \( n_{init} = 20 \) evaluations for the Sonar dataset from the UCI library [84] and \( n_{init} = 50 \) evaluations on the acid concentration problem (AppAcid) introduced in Sec. 4.3.2. For Sonar 130 sequential steps were performed,
while we spent 150 evaluations for the AppAcid problem. The maximum number of surrogate model evaluations was restricted to 50,000 evaluations. As an additional stopping criterion we forced the algorithms to stop when the objective function value doesn’t give more than $\epsilon = 10^{-7}$ improvement. This restriction was necessary, because although evaluations on the surrogate model are usually cheap, some EGO implementations tend to calculate the dominated hypervolume very often, which can be time-consuming also for small objective dimensions $m$.

For the ML, we used SVM as learning algorithm and the available data were split prior to tuning in 20% test data and 80% for training and validation. The SVM implementation was taken from the e1071 R package. As kernel for the SVM we chose the radial basis kernel. As shown earlier this learning algorithm is sensitive to the parameter settings of the kernel parameter $\gamma$. For the Sonar benchmark we tuned $\gamma$ together with the parameter $trnFrac$ as another tuning parameter, having a somehow comparable setup as in [149] (see Tab. 7.1). The tuning parameter $trnFrac$ defines the fraction of the training patterns related to the size of the training and validation set. The patterns not considered for training were used for validation purposes. We did not tune the SVM regularization parameter $C$ because in earlier experiments the model was rather insensitive to its settings.

**Table 7.1:** Region of interest (RoI) for Sonar tuning experiments. See main text in Sec. 7.3.1 for explanation of parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Lower</th>
<th>Upper</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\gamma$</td>
<td>0.01</td>
<td>0.7</td>
<td>float</td>
</tr>
<tr>
<td>$trnFrac$</td>
<td>0.05</td>
<td>0.7</td>
<td>float</td>
</tr>
</tbody>
</table>

For the more difficult AppAcid problem, we tuned seven parameters. Detailed information about the region of interest and the parameters for AppAcid can be found in Tab. 7.2. We employed two objectives for the problems consisting of $f_1 = \text{gain on test set}$ and $f_2 = \text{training time}$. The parameter $trnFrac$ should directly affect objective $f_2$, while the other parameters should mainly affect function $f_1$. The region of interest (RoI) was chosen according to settings from preliminary runs and the settings used in [155]. Since the parameters partly affect functions $f_1$ and $f_2$, it is of interest, whether the optimization methods are capable to learn the parameter relevances.

All experiments include the run of a multi-criteria optimizer (SMS-EGO, SExl-EGO or LHS), where in each iteration the learning algorithm is called. In each call the learning algorithm takes as input a subset of the data for building a prediction model (both for Sonar and AppAcid) and the hyperparameters set by the optimizer. For each experiment, ten runs were performed with different randomly drawn test and training-validation data samples to get statistically sound results.
Table 7.2: RoI for AppAcid tuning experiments. Parameters $\gamma$ and $trnFrac$ have the same meaning as in the Sonar task, see Sec. 7.3.1. Parameter $xperc$ controls the feature selection, the algorithm selects a number of the most important features to capture a fraction $xperc$ of the overall importance. The parameters $CUTOFFi, i = 1, 2, 3, 4$ control the weight for each class.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Lower</th>
<th>Upper</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\gamma$</td>
<td>0.001</td>
<td>0.8</td>
<td>float</td>
</tr>
<tr>
<td>$trnFrac$</td>
<td>0.2</td>
<td>0.7</td>
<td>float</td>
</tr>
<tr>
<td>$xperc$</td>
<td>0.050</td>
<td>1.0</td>
<td>float</td>
</tr>
<tr>
<td>$CUTOFF1$</td>
<td>0.010</td>
<td>0.4</td>
<td>float</td>
</tr>
<tr>
<td>$CUTOFF2$</td>
<td>0.010</td>
<td>0.4</td>
<td>float</td>
</tr>
<tr>
<td>$CUTOFF3$</td>
<td>0.010</td>
<td>0.4</td>
<td>float</td>
</tr>
<tr>
<td>$CUTOFF4$</td>
<td>0.010</td>
<td>0.4</td>
<td>float</td>
</tr>
</tbody>
</table>

7.3.2 Benchmarking

Benchmarking multi-criteria optimization algorithms usually includes the distance of the approximation set to the real Pareto front and the spread of the points over the front. Since the real Pareto front is unknown in our case, we try to introduce a fair benchmarking which nevertheless respects both performance measures. At first we summarize all results by taking the union of the approximation sets from all 10 runs, and selecting its non-dominated subset. We call this resulting set the reference set. The distance of the non-dominated sets of each run and this reference set then describes the approximation quality. In the best case an algorithm would produce exactly this reference set, so that the total distance would become zero. Now we only have to define how the distance between the single algorithm runs and the reference set are measured. Since the hypervolume and other quality indicators are controversially discussed in MCO, we additionally incorporate the $R^2$ indicator by Hansen and Jaszkiewicz [103].

For a given approximation set $A$ and a set of arbitrary utility functions $U$, the $R^2$ indicator is defined as follows:

$$R^2(A, U) := -\frac{1}{|U|} \sum_{u \in U} \max_{a \in A} u(a)$$  \hspace{1cm} (7.3)

As utility function we used the weighted Tchebycheff distance, but in fact also other metrics (e.g., the Minkowski distance) could be used here:

$$u(z) = u_\lambda(\vec{z}) = -\max_{j \in \{1, \ldots, m\}} \lambda_j |z^*_j - z_j|$$  \hspace{1cm} (7.4)

with $z^*$ the utopian point (a point that can never be approximated in the objective space) and $\vec{\lambda} = (\lambda_1, \ldots, \lambda_m)^T$ a weight vector for the objective functions. Different, randomly chosen $\vec{\lambda}$ constitute the utility function set $U$. In our experiments we sampled $U$ uniformly at random from the interval $[0, 1]$ as proposed by Hansen and Jaszkiewicz [103]. The size of $U$ is heuristically defined as follows:
\[ |U| = \binom{s + m - 1}{m - 1} \tag{7.5} \]

with
\[
s = \begin{cases} 
500 & \text{if } m = 2 \\
30 & m = 3 \\
12 & m = 4 \\
8 & m = 5 \\
3 & \text{otherwise}
\end{cases} \tag{7.6}
\]

Other performance indicators are not considered here, but with similar advantageous properties the averaged Hausdorff distance [216] to the reference attainment surface could be taken.

### 7.3.3 Sonar

Because Sonar is a small dataset with only 208 records [84], a parameter tuning can be performed quickly to test the performance of tuning algorithms. For simplicity we performed a tuning with only two parameters (SVM kernel parameter \( \gamma \) and parameter \( \text{trnFrac} \) for the size of the training set within the tuning).

In Fig. 7.1 we show the solution space of ten independent runs of SMS-EGO on the Sonar dataset. We spent 150 evaluations for the tuning. Input parameters were \( \gamma \) and \( \text{trnFrac} \) with the RoI settings as described in Tab. 7.1. In the upper plot (\( r = 1 \): one evaluation for each sequential design point) we used the re-interpolation procedure by Forrester et al. [82] with the SMS infill criterion. It is possible, that the solutions with a gain value better than \(-90\%\) are biased due to the random resampling. Nevertheless for SMS-EGO with \( r = 1 \) the approximation quality appears to be better than in the lower plot (\( r = 3 \): three evaluations for each sequential design point). Using replicates with a reduced total number of iterations at first results in worse approximation, compared with having more exploration through spending only a single evaluation for each design point.

To highlight the amount of noise of the ML problem (we term the observed noise of the problem empirical variance), we analyzed the variances of the obtained parameter settings. Therefore we re-evaluated each solution obtained from the SMS-EGO runs ten times, each time using a new random sample for the training set. We show the results of the empirical variance for the two objectives in Fig. 7.2, where the plot of Fig. 7.1 is extended with the variances obtained in the re-evaluation procedure.

We also investigated the quality of the surrogate model predictions. In Fig. 7.3 we compare the obtained solutions during tuning with ten repeated evaluations on the real objective function. In the optimal case all points would lie on the diagonal of the plot.
However, a perfect fit is unrealistic, because a certain amount of noise is always present in the data. But as can be seen from the plot, most of the solutions tend to be close to the diagonal, with some outliers.

In a final comparison all non-dominated solutions of all variants were re-evaluated ten times after the budget was fully spent (here: 150 trainings and evaluations of the Sonar dataset). This was done to remove biases in the non-dominated sets. Then a reference set for all algorithms was determined, consisting of the non-dominated solutions of all algorithms and runs of the re-evaluation. The differences of the obtained quality indicator values for $R^2$ and dominated hypervolume and this reference set are shown in Fig. 7.4. Small values indicate good performance of the algorithm, while large differences to the reference set indicate poor performances.

It can be seen from the boxplots that all EGO variants have smaller variances than the LHS solutions. In a t-test we obtained that SExl-EGO with three evaluations ($r=3$) is statistically significant better than LHS. All other variants did not show significant results, but as the boxplots reveal, they are more stable than the simple LHS baseline approach. Another thing to note is, that although sometimes LHS performs good, this will be in
Figure 7.2: Objective space plots showing the empirical variances for the two objectives $-\text{Gain}$ and $\text{Runtime}$. The error bars represent the standard deviations from 10 repeated evaluations (re-training on a new training sample) of a certain parameter point. The gain is obtained on an independent test set. Shown are the variances for Sonar and AppAcid. The variances are bigger for Sonar since the number of records in a sample is smaller.

Figure 7.3: Tuning result of SMS-EGO on the Sonar dataset for the objective $\text{gain}$. The plot shows the results seen by the tuning algorithm on the $x$-axis and of a new evaluation, where each design point was evaluated again ten times (yielding the empirical gain). Left: ten runs of SMS-EGO with a single evaluation for each design point ($R^2 = 0.67$). Right: ten runs of SMS-EGO, but with three evaluations for each design point in the sequential part of the algorithm ($R^2 = 0.77$). The red diagonal line denotes possible situations where the empirical gain would be equal to the gain during tuning.
Figure 7.4: \( R^2 \) and HV quality indicators for ten independent runs on task Sonar. The non-dominated solutions obtained after 150 function evaluations were evaluated again ten times, each time using different training/validation splittings to remove biased evaluations. For each quality indicator the mean value of these re-evaluations was calculated. Then we determined the differences between the reference set (set of non-dominated solutions of all runs) and the mean quality indicator values. Thus values of zero would be optimal. We compare the EGO variants both with one function evaluation \((r = 1)\) and three function evaluations \((r = 3)\). Additionally, SMS-EGO was run without re-interpolation and a single evaluation for each design point (SMS-EGO without RI).

The behaviour of SMS-EGO, SExl-EGO and LHS during the optimization process is shown as a line plot in Fig. 7.5. It can be seen from the plot that the EGO variants achieve a very fast decrease of the difference to the attainment surface, especially in the first 40 iterations. Although LHS does not include a sequential process, we plot it as a line here. The line for LHS is obtained by evaluating the quality indicators, when 51, ..., 150 LHS points are considered for the approximation set. It can be concluded that LHS converges more slowly to the reference set compared with SMS-EGO and SExl-EGO.
Figure 7.5: Sonar: Mean HV and $R^2$ development of ten runs of SExI-EGO, SMS-EGO and LHS. The plot shows that the EGO variants approximate quicker to good HV and $R^2$ approximations compared with LHS. The initial design of size 20 is not plotted for the variants, but we show the situation directly after the initial evaluations.

7.3.4 AppAcid

AppAcid is a dataset for classifying spectrography measurements of a biogas plant (cf. Sec 4.3.2). In our earlier SCO tuning with model-assisted optimization techniques (e.g., Sec. 4.3 or [155]) we obtained a performance in mean classification accuracy of up to 88%.

It was our goal to show the effect of balancing runtime and prediction accuracy. Our approach shows an interesting behaviour of the runtime, because SCO in ML is known to be very time-consuming when state-of-the-art models are considered. It can be seen from Fig. 7.6 that the multi-criteria optimization has no negative effect on the overall model accuracies. As in the single-criteria task before, the optimized SVM classifiers reached a high mean classification accuracy of up to 88%. Interestingly the plot also shows that high computation times do not necessarily lead to good prediction models in terms of accuracy. With wrong settings for $xperc$, the percentage of selected features, even a high runtime can lead to disappointing low gain as the crosses around $Gain = 0.2$ in Fig. 7.6 show.

Similar to the Sonar task, we re-evaluated the non-dominated sets of each run ten times for all variants considered in this study. The differences of the re-evaluated points and the corresponding reference set are shown in Fig. 7.8. For the AppAcid task 200 ML trainings were spent, since more parameters had to be tuned compared with the Sonar task. The different EGO variants are discussed in more detail in the following Sec. 7.4.
Figure 7.6: Plot of objectives for AppAcid. The mean classification accuracy (gain) is plotted on the $x$-axis and the runtime in seconds on the $y$-axis. Note, that the negative value for the gain is taken, because all objectives are being minimized.

Figure 7.7: AppAcid: Mean HV and R2 development of ten runs of SExI-EGO, SMS-EGO and LHS. The plot shows that the EGO variants approximate quicker to good HV and R2 approximations compared with LHS. The initial design of size 50 is not plotted for the variants, but we show the situation directly after the initial evaluations.
7.4 Discussion

In this section we give a discussion for the multi-criteria tuning using model-assisted optimization algorithms. In Sec. 7.4.1 we review the noise handling of model-assisted MCO and in Sec. 7.4.2 we criticize the runtimes required for the multi-criteria tuning algorithms.

7.4.1 Noise handling

In machine learning it is difficult to obtain a good and reliable estimator of the real generalization error of a prediction model. In our MCO experiments we evaluated the models using the holdout set error (the classification error obtained on a validation set). Although this gives a better estimate than just to consider the training error (which would rather lead to overfitting issues), the objective function values are still subject to high noise levels, because they depend on the randomly sampled training data. Therefore we need to respect special characteristics of ML evaluations, especially highlighting the noisy observations here.

We compared three variants to handle noise in optimization under restricted budgets:

(a) the re-interpolation (RI) method by Forrester et al. [82], one evaluation of each design point ($r = 1$),

(b) replicates: repeated evaluations of each design point without RI ($r = 3$, this allows only one third of the infills), and

(c) no RI and no replicates ($r = 1$).

The goal is to avoid a too early convergence of the surrogate models to misleading solutions caused by noise.

Variant (a) tries to avoid selecting biased solutions by an internal estimation of the noise. Variant (b) instead explicitly performs re-evaluations to obtain more stable results during the optimization and to give better estimates for the surrogate model predictions. Variant (c) was added as baseline comparison for the special noise handling approaches.

In our experiments, the variant using one evaluation ($r = 1$) and noise estimation within the Kriging procedure (re-interpolation) performed worse once on task Sonar (SExI-EGO using three evaluations performed better than SExI-EGO ($r = 3$), Fig. 7.4). Also, SMS-EGO ($r = 3$) performed better on task AppAcid (compared with SMS-EGO ($r = 1$), Fig. 7.8). But it has to be noted, that in both cases the significance level of the t-test did not reach the 95% confidence level for both quality indicators R2 and dominated HV. This means that we cannot give evidence for using $r = 3$ rather than $r = 1$ on any of these tasks or vice versa.

On task AppAcid all EGO variants are significantly better than LHS (t-tests with $p < 0.029$) for the R2 indicator, with the exception of SExI-EGO ($r = 3$). Interestingly, the behaviour for different quality indicators is not necessarily the same, which can be seen...
from the result for SExl-EGO ($r=1$) on task AppAcid. In this boxplot a very large box was obtained for indicator HV, while the box for the R2 indicator is rather small, leading to a significant better result of the EGO approach (t-test for R2 with $p = 0.029$, while the p-value for HV is 0.977).

Very high noise levels were observed for the Sonar task, leading to rather unstable results. As a positive result in Fig. 7.4 the EGO variants reveal smaller boxes, indicating a higher robustness of the optimization compared to LHS. However, in our statistical test (t-test, confidence level 95%) only SExl-EGO ($r = 3$) was significantly better than LHS. The good result of SExl-EGO ($r = 3$) can be reasoned by the usage of repeated evaluations, when the task is very noisy. Instead when the parameter space becomes larger (e.g., task AppAcid), more exploration is required, and the algorithms using replicates do not perform as good (Fig. 7.8). It is also an important achievement that all EGO variants result in more stable quality indicator measures as can be derived from the smaller box sizes in the boxplot in Fig. 7.4.

A point of discussion is the setting of the reference point for the hypervolume indicator. We noticed that this point plays an important role, because it can lead to a perturbation of the results. This most likely occurs, when the reference point is set too close to the points in the approximation set. As a consequence some points do not contribute much to the hypervolume indicator, resulting in a too pessimistic evaluation of the approximation set. Therefore special care must been given to the corner solutions. Such solutions will not contribute to the dominated hypervolume if the reference point is set too close. For this reason we always set the reference point

$$r = \left( \max_{\vec{x} \in N} f_1(\vec{x}) + C_1, \max_{\vec{x} \in N} f_2(\vec{x}) + C_2 \right) \quad (7.7)$$

where $N$ denotes the set of non-dominated solutions from the reference set and $C_i$ are additional constants for each objective. Here we chose $C_i = 1$. For the R2 indicator we set an utopian point as $(-1, 0)^T$ which is guaranteed to dominate every real solution.

Summarizing we find no significant difference between the noise-dampening variant with $r = 3$ and the other variant $r = 1$ which has more noise but also allows more infills.\(^1\)

RI deals successfully with the observed noise at the ($r = 1$)-level. SMS-EGO with RI performs always slightly better than SMS-EGO without RI, the difference are however not significant in our current experiments. It is matter of future research to reveal whether RI makes a difference compared with simple interpolating Kriging strategies without special noise handling.

\(^1\)As an interesting side remark we note that we had a significant difference in the pre-final results: Directly after optimization, the ($r = 1$)-variants seemed significantly better. But this turned out to be only a statistically irrelevant fluctuation of the ($r = 1$)-solutions: After the ten-fold re-evaluation the difference was mainly gone. This emphasizes the importance of sound evaluation procedures in the presence of noise.
Efficient multi-criteria optimization in machine learning

Figure 7.8: R2 and HV quality indicators for ten independent runs on task AppAcid. The non-dominated solutions obtained after 200 function evaluations were evaluated again ten times, each time using different training/validation splittings to remove biased evaluations. For each quality indicator the mean value of these re-evaluations was calculated. Then we determined the differences between the reference set (set of non-dominated solutions of all runs) and the mean quality indicator values. Thus values of zero would be optimal. We compare the EGO variants both with one function evaluation (r=1) and three function evaluations (r=3). Additionally, SMS-EGO was run without re-interpolation and a single evaluation for each design point (SMS-EGO w/o RI).

7.4.2 Parameter space dimension

It is interesting to note the influence of the parameter space dimension on the relative performance of LHS against the multi criteria EGO variants. In the Sonar case (dimension 2) LHS is very close to the EGO results (Fig. 7.4). This is not too surprising, because we spent a large number of 150 function evaluations for the Sonar task. The line plot in Fig. 7.5 shows that the EGO variants could obtain good approximations of the reference set after 50 iterations. This indicates that EGO would also produce good results with less function evaluations, and therefore would probably outperform the LHS. After a sufficient number of function evaluations finally all algorithms achieve a comparable attainment of the reference set (here, after 150 iterations).

In contrast to this, the AppAcid task (dimension 7) shows that LHS cannot keep pace with EGO, because, as Fig. 7.7 shows, the multi criteria EGO indicators now drop much faster: even at iteration 100 (50 initial design + one third of 150), the HV-indicators for both EGO variants are already better than LHS at iteration 200.
7.4.3 Single- and multi-criteria optimization compared

If we perform multi-criteria optimization, is there a large price to pay with respect to the individual goals? If we consider only one objective *gain*, is the best gain delivered from MCO considerably lower than the gain from SCO which concentrates only on this objective? How does the *gain* develop in a restricted-budget optimization?

To answer these questions we performed ten SCO-runs for task AppAcid with the same parameters to optimize, the same RoI and the same budget (50 initial design points, 150 sequential infills) as in the MCO runs. As SCO tuning algorithm we chose the well-known sequential parameter optimization (SPO) [12] within the TDMR-framework [154]. SPO as well uses Krigeing as surrogate model, but in this case only for single-criteria optimization of the objective *gain*. We show in Fig. 7.9 the average of ten runs. It is interesting to note that the MCO-gain is only slightly below the SCO-gain which is quite surprising because MCO operates in an objective space bigger by one dimension.\(^2\) But at the same time MCO gives the user more information with respect to the tradeoff between runtime and quality (*gain*).

\(^2\)We note that an earlier SCO-experiment [155] on task AppAcid with method SVM and a slightly different RoI had a mean gain of 86.1%, which is on the same level as the current MCO-results.
7.4.4 EGO runtimes

A disadvantage of hypervolume-based EGO are the high computation times, which can exceed the computation times of the objective function evaluations. E.g., the total runtime of the AppAcid optimization was 15.8 hours with SMS-EGO, while the runtime of SExl-EGO was 143.6 hours (almost 6 days) on an Intel Xeon E5530 CPU running at 2.40GHz. This result was obtained with all settings as described before, having 200 evaluations in total, an initial design size of 50, and one evaluation per design point ($r = 1$). Because the largest runtime for the function evaluations is below one minute, we can conclude that the real objective function cannot be the source for the long runtime. Hence, the optimization on the surrogate model must be responsible for the high computation times. The reasons for this are twofold. At first the Kriging procedure badly scales with increasing parameter sizes. Today, parameter spaces of 20 are challenging. But since we only have 7 tunable parameters, this cannot be the reason. Second, the hypervolume indicator is calculated very often. While the calculation of the hypervolume is NP-hard [29], this bares a real disadvantage for the model-assisted MCO approach. The reason why SExl-EGO has almost a ten times higher runtime, is caused by the underlying implementation. We are aware of the fact that simple modifications of the algorithm can lead to better runtimes. Nevertheless one has to note, that the MCO approach only makes sense when the runtime of the EGO is significantly lower than the runtime of the learning algorithm. In our benchmark this was not the case, but it might become important for really large datasets.

7.5 Conclusions

Summarizing the results, we showed that multi-criteria optimization can help to offer the user a variety of possible solutions to select from. In the earlier SCO approach the selection was restricted to use the best generalizing model delivered by the optimization procedure. Now, with the multi-criteria approach presented in this chapter, the user is able to select the best models spending a certain limited time budget for the training process.

We applied two hypervolume-based EGO variants, namely SMS-EGO and SExl-EGO, for the first time to noisy ML tasks. It was found that they operate well, even in the presence of considerable noise affirming research question Q1. Both algorithms deliver comparable quality in terms of the covered hypervolume and the R2-indicator. SMS-EGO in its current implementation is considerably faster than SExl-EGO.

Both MCO-approaches use Kriging surrogate models to cope with heavily restricted budgets (Q2). Those Kriging models have to work robustly under the presence of noise. Noise is inevitable in most ML tasks due to the variations of randomly drawn training samples and the characteristics of the data. We found it to be crucial to use the re-interpolation technique to cope with the noise confirming Q3. Plain Kriging models without re-interpolation (boxplot SMS-EGO without RI in Fig. 7.8) perform worse, because they tend to overfit the noise.
The re-interpolation could deal well with the noise in our benchmark. An additional noise reduction by aggregating repeated evaluations can be necessary, when high noise levels occur. For larger dimensions, repeated evaluations diminish the exploration of the search space, deteriorating the approximation quality due to a reduced number of infill steps.

On the MCO tasks, both Kriging-based EGO techniques performed better than the baseline LHS approach (Q4). The advantage increases with the number of dimensions in the parameter space: the difference between LHS quality indicator values and EGO quality indicator values is much higher in task AppAcid (parameter space of dimension 7) than in task Sonar (dimension 2).

In the future we plan to apply the multi criteria approach to other datasets as well, in order to reveal the challenge caused by high dimensions of the parameter space. Additionally, we want to explore the relationship between computation time and classification gain to enable the tuning of ML algorithms in a shorter timeframe.