Tuned Data Mining in R

**Patrick Koch, Wolfgang Konen,
Oliver Flasch, Martina Friese, Boris Naujoks, Martin Zaefferer,
Thomas Bartz-Beielstein**

Fakultät für Informatik und Ingenieurwissenschaften, Fachhochschule Köln
E-Mail: patrick.koch@fh-koeln.de

Abstract

In data mining (DM), the user has to deal with a variety of options to be set manually, e.g., which preprocessing is reasonable for the task, which models should be incorporated, what is an adequate parameter setting for models and preprocessing, or which variables should be selected. Such questions have to be answered anew for each task and illustrate the importance of a comprehensive framework, which can generate good models for DM tasks. We present the software package Tuned Data Mining in R (TDMR), which is a comprehensive framework for data mining offering a fully integrated concept for parameter tuning.

1 Introduction

Nowadays, DM has become more and more important in engineering and science. With the advanced computational power in today's workstations, more advanced techniques come into reach. The tuning of algorithmic parameters and preprocessing options can be valuable in order to build more accurate DM models. Whilst tuning was far too complex in the past, it can be supported by modern computer architectures and powerful optimization algorithms now. Despite this fact, tuning is still seldom applied in practice: users often rely on using rather simple methods as grid search (see e.g., Thisted [Thi88]) or hand-tuning for obtaining the required parameter settings. Unfortunately, these solutions usually perform worse compared to systematically optimized configurations. In this work, we propagate to optimize DM models using a comprehensive framework, which gives good and robust results on the one hand and is still easy to use on the other side.

This paper is structured as follows: in Sec. 2 we give a brief overview on comparable software frameworks for data mining followed by an introduction to parameter tuning algorithms for data mining tasks. We describe the features and the tuning concept of the TDMR framework in Sec. 3. In Sec. 4 we give an overview on tasks already solved using the TDMR framework. A short summary and an outlook is provided in Sec. 5.

2 Methods

2.1 Existing Frameworks

Other frameworks for simplifying and solving DM tasks are broadly available today. In this Sec., we give a short overview on some existing frameworks from industry and sciences. The Machine Learning in R (*mlr*) framework proposed by Bischl [Bis2009]

provides many machine learning methods and handles DM tasks including classification and regression. The toolbox *Gait-CAD* [Mik2006] is based on Matlab and includes tools for time series data and classification. Commercial tools include *RapidMiner* [Mie06] (formerly known as YALE), which provides a graphical user interface (GUI), where building blocks of data mining can be combined and drawn together. *ClearVu Analytics* [CVA2010] is similarly a graphical toolbox for modeling and optimization. *ClearVu Analytics* gives suggestions for outlier detection and allows very flexible graphical inspections of the data within the GUI, prior to model building and model optimization. A Mathematica [Wolf88] framework for data modeling was presented by Kotanchek *et al.* [Kota06]. The framework is based on symbolic regression using genetic programming [Koza92] as a modeling technique. Finally, *Rattle* [Will11] is another open-source GUI frontend for data mining using `R` and gaining increasing popularity.

While offering very powerful solutions, the drawback of all these frameworks is that they provide not enough support for tuning. They do not have a generic framework to couple data mining and feature preprocessing methods with advanced and exchangable tuning methods known from the optimization community (as there are, e.g., SPOT or CMA-ES, see Sec. 2.3).

2.2 Data Mining in the `R` Environment

`R` is an open-source software language and can be considered as a free implementation of John Chambers' `S` language. Today, `R` is probably the most often used language within the statistics community. Many software packages are provided through the project homepage CRAN (comprehensive `R` archive network, `http://cran.r-project.org/`). In addition to many statistical software packages, the CRAN platform also provides a lot of machine learning algorithms. Available software packages for machine learning in `R` reach from simple linear or quadratic models like linear or quadratic discriminant analysis to more complex algorithms as Random Forests (RF) or Support Vector Machines (SVM). Most of the learning algorithms support a simple `formula` interface, which makes it easy to use the same grammatical syntax for the algorithms. Recently, `R` has become very popular in data mining: a KDnuggets poll with 1100 voters [Piat11] sees `R` as second most used data mining tool for real projects (closely behind RapidMiner).

2.3 Tuning Algorithms

Parameter optimization is of large importance for data mining. Most parameter settings of learning algorithms are very sensitive, consider, for example, the kernel parameters of Support Vector Machines. Other parameters include preprocessing options, e.g., feature processing and selection. Here, additional parameters come into play which can greatly affect the model quality. In this Sec. we describe some algorithms, which can be used for parameter optimization. In general, a tuning algorithm can be any (numerical) optimizer, which supports box constraints. However, also optimization algorithms without supporting constraint handling can be used, e.g., by making use of penalty functions.

Sequential Parameter Optimization (SPO) [BLP05] is a framework for improving and understanding the behavior of search algorithms by experimentation. Both classical and modern methods from statistics are considered for optimizing algorithm parameters to

improve the performance of these algorithms. SPO sequentially performs a pre-defined number of algorithm runs and uses the information during exploration of the search space to build and refine one or multiple meta models of the true objective function. The use of such meta models (or surrogate models) allows SPO to reach very good results with only a small number of observations in the search space. An implementation of SPO, the Sequential Parameter Optimization Toolbox (SPOT), is available as open source software [Bar2010, Bart10e].

Originally, TDMR was designed to work with SPOT as the preferred tuner, but it also offers the possibility to use other tuners as well, e.g., the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [Han96], a latin hypercube design (LHD) or other direct-search (derivative-free) optimizers [BFGS, Powell] for comparison. At the moment, TDMR contains the tuning algorithms presented in Tab. 1. The list of tuners integrated in TDMR is not fixed, thus the framework can easily be extended by other tuning algorithms.

Table 1: Overview of tuning algorithms

Algorithm Descriptor	Description
spot	Sequential Parameter Optimization Toolbox [Bar2010]
lhd	Latin Hypercube Design (truncated SPOT, all budget spent for the initial step, parameters are equally distributed over the parameter space)
cmaes	Covariance Matrix Adaption Evolution Strategy [Han96], implementation by Trautmann *et al.* can be downloaded from `http://cran.r-project.org/web/packages/cmaes/index.html`
powell	Powell's Method (direct, local search, no constraint handling) [Pow1998]
bfgs	Broyden, Fletcher, Goldfarb and Shanno method (direct search, constraint handling [Liu89] available through `R` package *optimx*, download from `http://cran.r-project.org/web/packages/optimx/`)

TDMR allows to use $k > 1$ tuning algorithms for a task, yielding k final parameter sets at the end. Tuning with more than one optimization algorithm can be done in parallel using the `R` snowfall package (see Sec. 3.4). First experiments using TDMR and the tuning algorithms showed, that SPOT outperforms CMA-ES and is significantly better than simple LHD parameter designs or classical local search algorithms like BFGS [Kone11d].

3 TDMR Features

We now introduce the *Tuned Data Mining in* `R` framework which combines feature processing, machine learning and the possibility to tune both of these processing steps. With the TDMR framework tuned DM models can be built with minimal effort and little DM expertise from the user's point of view. The framework is written in `R`, because a lot of

models and optimization methods are available in this language. The TDMR package is available under the terms of the GNU public license from our webpage [TDMR11].

The workflow of TDMR can be divided into the following three phases with increasing (computational) complexity:

1. A DM model is built completely without tuning (using standard or user-defined parameter settings for the model).
2. Tuned Data Mining: In this phase, a tuning of relevant model parameters is conducted using a tuning method of choice (see Sec. 2.3).
3. Model Chains („The big loop“): Several TDM tasks are started (usually on the same DM task, but with different tuning configurations), optionally with several tuners. Their best solutions are compared with different modes of unbiased evaluations, e.g. on unseen test data or by starting a new, independent cross validation or resubsampling.

The three phases are explained in detail in the following Sections.

3.1 Phase 1: Simple Data Mining in `R`

In the first phase, a data mining model is built without optimization using standard or user-defined parameter settings for the model. The results obtained from this phase can be used as preliminary suggestions for the parameter ranges and methods to be used. Thus, within this phase, a preselection of models can be done. Later, this phase can also be used for model validation as a final step after TDMR phases 2 and 3.

3.1.1 Defining the Data Mining Task

In general, TDMR can handle two different kinds of data mining tasks, classification, and regression respectively. In the beginning, the user writes a task-specific function `mainTASK(opts=NULL)`, where the dataset is loaded and options for the TDMR framework are set. In the function, one of the task-independent functions `tdmRegressLoop` or `tdmClassifyLoop` is called, which starts the TDMR run. An example of a main function is given below. Here, a dataset is read from a comma separated value (CSV) file and the regression template `tdmRegressLoop` is called. All results from the run are written in a structure named `result` and returned to the user:

```
main_cpu <- function(opts=NULL) {
  # define dataset
  dir.data <- "./data/";
  filename = "cpu.csv";

  # fill opts structure with default values
  if (is.null(opts)) opts <- tdmOptsDefaultsSet();
```

```
  # graphics and log file initialization
  tdmGraAndLogInitialize(opts);

  cat1(opts,filename,": Read data ...\n")
  dset <- read.csv2(file=paste(dir.data, filename, sep=""));
  cat1(opts,filename,":", length(dset[,1]), "records read.\n");

  # set response variable
  response.variables <- "ERP";
  input.variables <- setdiff(names(dset), response.variables);

  # run regression template
  result <- tdmRegressLoop(dset, response.variables,
                           input.variables,opts);
  result <- tdmRegressSummary(dset,result,opts);

  # close graphics and log file
  tdmGraAndLogFinalize(opts);
  return(result);
}
```

3.1.2 The TDMR Objective Function

Every data mining task processed with TDMR is evaluated using a certain error measure or accuracy value, which is defined by a real-valued function. The objective function value is returned to the user at the end of the function call of `tdmRegressLoop` or `tdmClassifyLoop` as a feedback of the model quality. In other phases of TDMR as tuned data mining or model chain optimization the main task is called multiple times with different parameter settings for the given task(s). The objective function value can be set to different types using the TDMR variable `rgain.type`.

Table 2: Available objective functions for classification (C) and regression (R)

Objective function	Type	Description
rgain (default class.)	C	the relative gain in percent, i.e. the gain actually achieved divided by the maximal achievable gain on the given data set
meanCA	C	mean class accuracy: For each class, the accuracy on the data set is calculated and the mean over all classes is returned
minCA	C	same as „meanCA“, but with min instead of mean. For a two-class problem, this is equivalent to maximizing the min(Specifity,Sensitivity)
rmae (default regr.)	R	the relative mean absolute error RMAE, i.e. the mean <|y – y(pred)|> divided by the mean <|y|>
rmse	R	root mean squared error

In Tab. 2, the term „gain“ is associated with a user-specific gain matrix (square matrix with number of rows = number of classes), which allows to set a specific gain for each possible outcome „true vs. predicted“ (usually, all gains for misclassifications will be non-positive, but they can differ in magnitude). The default gain matrix is the identity matrix; in this case `rgain` reduces to the usual mean classification accuracy.

3.1.3 TDMR Options

All settings in TDMR are defined by a structure or list in `R` named `opts`. In the beginning of a TDMR run the list can be filled with default parameters. Changes to the `opts` list, e.g., substituting the base model or setting preprocessing options can be easily passed to the main task function `mainTASK(opts)` of a data mining task.

3.2 Phase 2: Tuned Data Mining

The second phase comprises a tuning of all relevant model parameters using a tuning algorithm selected by the user. Actually, the tuning algorithms presented in Tab. 1 are provided with TDMR for parameter optimization. As an additional feature of TDMR, the optimization algorithms can be run in parallel on multicore or cluster CPUs. For more details on parallelization see Sec. 3.4.

In TDMR, a parameter configuration usually is a 3-tuple consisting of the preprocessing options $\vec{p}_{pre}$, the model hyperparameters $\vec{p}_{model}$, and, optionally, the postprocessing parameters $\vec{p}_{post}$.

The preprocessing parameters include all parameters required for feature processing:

- feature selection: the number of input features to select using a selection method of choice (see Guyon and Elisseeff [Guy03] for a more thorough overview).
- feature construction: e.g., number of PCA features to select, number of monomials to build from the input features, or similar transformations on the input feature set.

The parameter vector $\vec{p}_{model}$ completely depends on the chosen model for the corresponding data mining task. Thus, the number of parameters can range from an empty set in case of a linear model to more elaborate settings of $\vec{p}_{model}$ in case of SVM or Random Forest models.

Finally, further options may be defined in $\vec{p}_{post}$ for the optional postprocessing of data mining tasks. This includes class weights, making the classifier cost-sensitive, forcing that also classes with few samples are classified correctly.

The flow chart of phase 2 is shown graphically in Fig. 1. In the beginning, a tuning algorithm of choice initializes a parameter configuration $\vec{p} = (\vec{p}_{pre}, \vec{p}_{model}, \vec{p}_{post})$. The configuration is then passed to the objective function, which handles preprocessing, model training, and evaluation on an independent validation set. The objective function value, e.g., the prediction accuracy, is returned to the tuning algorithm again. The tuning algorithm creates new parameter configurations from this information and tries to find the best parameter configuration for the data mining task.

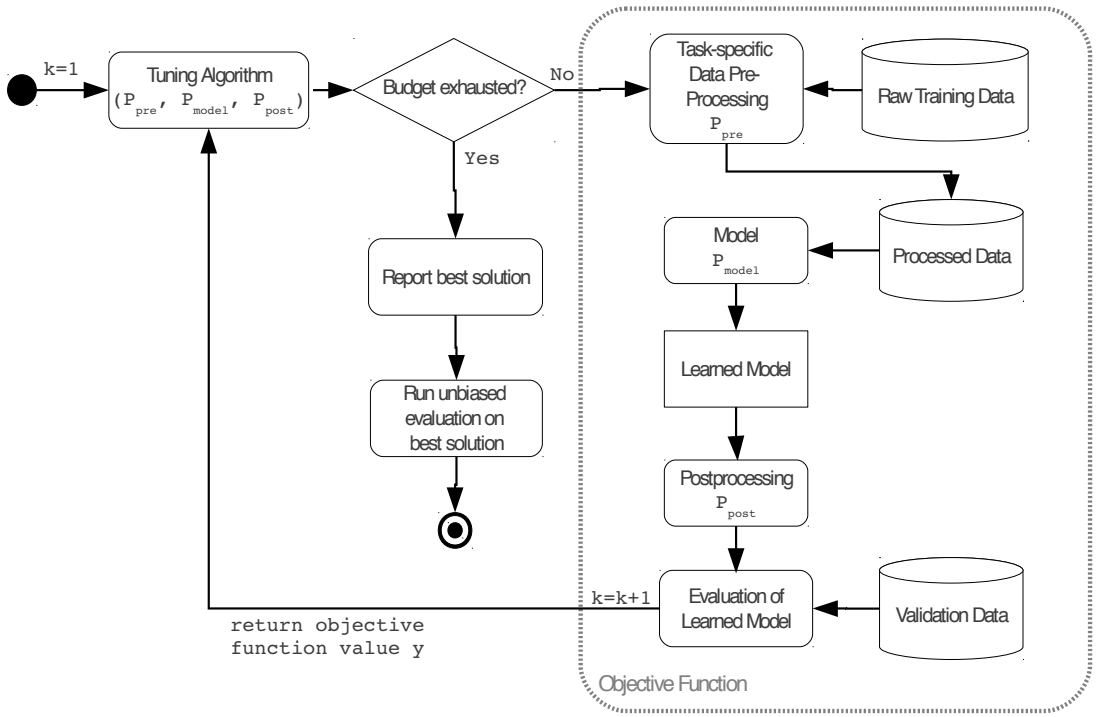

Figure 1: Tuned Data Mining flow chart.

In TDMR, tuning algorithms can be defined using the option:

```
opts$tuneMethod = c("Tuner1", "Tuner2",..., "TunerN"),
```

where `Tuner1` to `TunerN` are the descriptors of the tuning algorithms (see Tab. 1).

3.3 Phase 3: Model Chains or „The Big Loop“

The Big Loop is a script to start several phase-2 tasks (usually on the same DM task, but with different tuning configurations), optionally with several tuners. In this phase of TDMR for each task and tuner,

(a) the tuning process is started (if `spotStep=auto`) or a previous tuning result is read in from file (if `spotStep=rep`) and

(b) one or more unbiased evaluations are started. The unbiased evaluations are done for each element of `umode` by calling the function

```
unbiasedBestRun_*(...,umode,...)
```

[`*=C` for classification and `*=R` for regression]. The function `unbiasedBestRun_*` reads in the best solution of a tuning run from SPOT's `.bst` file, and performs a re-run (training + test) with these best parameters to see whether the result quality is reproducible on independently trained models and / or on independent test data.

The tuned models are subject to an unbiased evaluation using

- unseen test data (`umode=TST`),

- independent cross validation (`umode=CV`), or
- by starting a new, independent random subsampling (`umode=RSUB`).

The unbiased evaluations are a way to measure possible oversearching effects. In research of neural networks, the term *overfitting* has become popular, where models get to complex because of too long training periods, which lead to misfits of the models. For decision trees, similar effects can be observed, e.g., the growth of too large and specialized trees, which can't generalize enough any more. Here, pruning is one option to circumvent this effect. However, whilst overfitting is a term affecting the model of the data mining process, parameter tuning can lead to other effects, better known as *oversearching* (see Quinlan and Cameron-Jones [Qui95]). Oversearching occurs if a tuning process follows too closely the random fluctuations of a noisy objective function. Koch *et al.* [Koch10b] have shown a way to measure the oversearching effect for time series data. Another possibility to handle oversearching is to use multiobjective optimizers instead of single-objective optimization techniques minimizing the oversearching effect in an own objective.

3.4 Parallelizing Tuning Experiments

As an important feature for accelerating the tuning process TDMR comprises parallel computing with `snowfall` [Kna09], where DM tasks can be set up on parallel architectures like networks of workstations or grid computing nodes. TDMR experiments can profit from this parallelization, yielding a large speed-up in runtime.

Usually, each TDMR component is run in a sequential manner. Since some components are independent from each other, these steps can easily be accelerated by running them in parallel mode. When the TDMR parameter `parallelCPUs` is set to an integer value larger than 1, a cluster is initialized and the TDMR components will be assigned to the CPUs defined by the cluster.

The following list gives examples how TDMR can be accelerated by parallelized execution:

- In TDMR, experiments can be repeated multiple times with different seeds, e.g., to cope with non-deterministic models or random subsampling. If run on homogenous parallel architectures, a parallelization of experiments will lead to an almost linear speed-up.
- Tuned data mining experiments can be set up with a list of tuning algorithms. By setting multiple tuning algotithms in the `tuneMethod` parameter of TDMR, algorithms can be run in parallel:
  `opts$tuneMethod = c("spot", "cmaes", "bfgs",... ).`
  Please note that the result of the tuning algorithms will only be available when the slowest tuning algorithm has finished its run.

3.5 Extensibility

TDMR has been designed especially with options for easy extensibility in mind:

1. Tuning parameters: It is easily possible to extend the set of tuning parameters without the need to change core TDMR `R` -code. The new parameter is added to the list `opts`, to the mapping file `tdmMapDesign.csv`, and to SPOT's `.roi` file. Afterwards it is automatically included into the tuned parameter space.

2. Other tuners: The integration of further tuning algorithms beyond the set listed in Tab. 1 is easily possible. Implementation details are given in [Kone11a].

3. Other machine learning methods.

It is the aim of this extensibility options to proliferate the use of tuning in data mining tasks. Often, the tuning within a new data mining task is hindered by the fact that a certain parameter can not be included in the list of tunable parameters. TDMR should make such extensions fairly easy. Furthermore, the framework facilitates the comparison of different tuners. With TDMR, it becomes easier to run a list of tuning algorithms on a variety of tasks and also to benchmark a new tuning algorithm comprehensivly against existing ones.

4 Experiments

TDMR has recently been used for several data mining tasks. We used TDMR for optimizing both model and preprocessing parameters of a real-world regression problem [Koch10b, Koch10c]. We showed that parameter tuning of the preprocessing parameters in conjunction with model parameters gives comparable or even better results than special-purpose models for the problem at hand (predicting fill levels of stormwater overflow tanks). Konen *et al.* [Kone10a] use TDMR with SPOT as tuning algorithm and without any task-specific programming and reach a place in the top ten of the Data Mining Cup competition 2010 [Koge2010] with their models. (Without tuning, the same model without any task-specific programming and with default parameters would only reach place 47 of 67.) In [Kone11d] different tuning algorithms were compared to each other; here SPOT gave the best performance on the benchmarks from all tuning algorithms used in this study.

Still, a big issue in tuned data mining is the computation time required for the optimization. To cope with this problem, we analyzed settings a) with giving small and large budgets for the optimization and b) with and without Optimal Computing Budget Allocation (OCBA) [Che1997]. OCBA was used within SPOT to assign the available budgets optimally. With fixed repeated evaluations, too much time is spent in regions of the search space, where the parameter configurations appear to be bad. Thus, in Fig. 2 we vary both, the budget available for tuning and set the repeated evaluations one time fixed to a maximum value of 5 and one time dynamic (OCBA). The budget was varied between three different sizes $B \in \{100, 500, 1000\}$.

The plots show the results of ten repeated runs of TDMR on a real-world classification problem (prediction of acid concentrations in the fluids of a plant, AppAcid, see [Kone11d] for more details on the task). As classification method for this task, we chose a random forest, since it gave best results in prior experimentations [Kone11d]. In the plots, we also compare two meta models for the SPOT optimization, a kriging-based meta model (maximum likelihood estimated gaussian processes, MLEGP) and a random forest meta

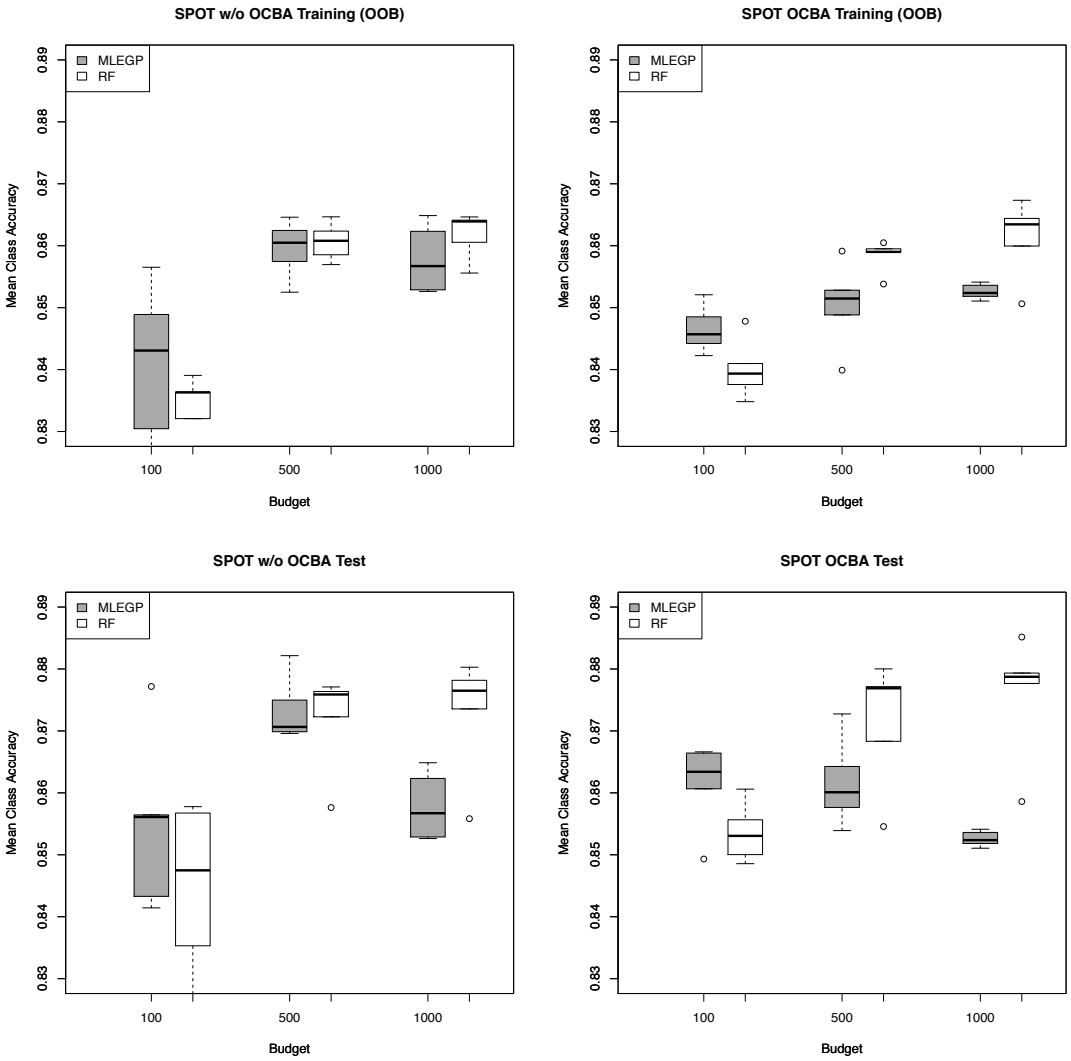

Figure 2: Variation of available budget on AppAcid task without (left) and with (right) OCBA.

model (RF). In preliminary studies, these two meta models always performed best for this task. The tuning parameters were the same as in [Kone11d], namely $CUTOFF$ and $CLASSWT$ to handle the class imbalances of the problem and also the number of trees, $ntree$, and number of splits tried in each step $mtry$. The initial design size of SPOT was set to 25 design points. As objective function, we used the mean class accuracy (MCA) of the five classes respecting each class fairly.

Summarizing the results, we always achieved a performance of more than 80% correctly classified patterns (MCA), even with the lowest of the three budgets. The boxplots in the top of Fig. 2 show the results of the best final parameter configurations obtained by SPOT evaluated repeatedly on a validation set (the set used during tuning the parameters). The boxplots in the bottom show the results of the best final parameter configurations obtained by SPOT evaluated on an independent test set. The difference between the left and the right boxplots is whether OCBA was used during tuning or not. It can be seen from the plots that all results with OCBA show a much smaller variance than without OCBA. This is a clear indicator that more stable results can be reached with OCBA. However, a

too small budget $B = 100$ leads to inferior results on the validaton set and of a higher variance as well. Interestingly, also very high budgets can result in negative effects as can be seen from the boxes of the MLEGP meta model evaluated on the independent test set using a large budget of 1000 evaluations. Here, even a slightly decreasing accuracy can be observed, which is remarkable, since the first assumption was that results should get better when giving more time for optimization. We think that this effect is caused by a certain oversearching or overtuning, which was also observed in previous studies (predicting fill levels of stormwater overflow tanks, see [Koch10c] for a more detailed description).

5 Summary and Outlook

In this paper, we presented the TDMR framework for solving data mining tasks. The framework is written in `R` and is available under the terms of the GNU public license [TDMR11]. It provides a comprehensive software tool to cycle through an entire data mining process. With the framework, it is possible to perform a systematic parameter tuning using state-of-the-art methods from computational intelligence (CI), machine learning, and optimization.

TDMR puts an emphasis on the extensibility of the tuning process (Sec. 3.5). It should be easy for the user to add new parameters to the tuning process and to change or extend the set of tuning algorithms. Nevertheless, we feel that SPOT, as a tuning algorithm, has shown considerable strengths for DM applications (especially when the computing budget is low), but more comprehensive experiments are needed to justify this claim.

Our experiments in Sec. 4 have shown that SPOT will, even with low budgets on average, reach astonishing good results, however, at the price of a large variance. Increasing the budget decreases the variance. OCBA, on the other hand, is a promising way to decrease the variance even for small budgets.

TDMR supports several ways of measuring possible oversearching effects (Sec. 3.3). Our results in Sec. 4 have shown that small changes in the tuning process (e.g. whether SPOT's surrogate model is MLEGP or RF) can have strong influence on whether oversearching is observed or not.

In the future, we want to run more experiments with TDMR, add more tuning algorithms and more machine learning algorithms (with suggested sets of tunable parameters) to TDMR in order to improve its performance. Furthermore, we plan to extend the framework with additional options for feature generation and feature selection. Finally, it is our aim to explore new ways to accelerate tuning runs without causing oversearching effects.

6 Acknowledgements

This work has been supported by the Bundesministerium für Bildung und Forschung (BMBF) under the grants FIWA and SOMA (AiF FKZ 17N2309 and 17N1009, Ingenieurnachwuchs) and by the Cologne University of Applied Sciences under the research focus grant COSA.

References

[Bar2006] Bartz-Beielstein, T., *Experimental Research in Evolutionary Computation—The New Experimentalism*, Springer, Berlin / Heidelberg, 2006.

[Bar2010] Bartz-Beielstein, T., *SPOT: A Toolbox for Interactive and Automatic Tuning in the R Environment*. In: Hoffmann, F. and Hüllermeier, E. (eds.), Proceedings 16. Workshop Computational Intelligence, p. 264-276, Universitätsverlag Karlsruhe, 2010.

[Bart10e] Bartz–Beielstein, T., SPOT: An R Package For Automatic and Interactive Tuning of Optimization Algorithms by Sequential Parameter Optimization, Research Center CIOP (Computational Intelligence, Optimization and Data Mining), CIOP Technical Report, 05/10, Comments: Related software can be downloaded from `http://cran.r-project.org/web/packages/SPOT/index.html`, Cologne University of Applied Sciences, Faculty of Computer Science and Engineering Science, 2010

[BLP05] Bartz–Beielstein, T. and Lasarczyk, C. and Preuß, M., Sequential Parameter Optimization, In: B. McKay *et al.* , Proceedings 2005 Congress on Evolutionary Computation (CEC'05), Edinburgh, Scotland, p. 773–780, IEEE Press, Piscataway NJ, 2005

[Bis2009] Bischl, B., *The mlr package: Machine Learning in R*. `http://mlr.r-forge.r-project.org`, Abruf 03.10.2011.

[Che1997] Chen, H.C. and Dai, L. and Chen, C.H. and Yücesan, E., *New development of optimal computing budget allocation for discrete event simulation*. In: Proceedings of the 29th conference on Winter simulation, p. 334–341, IEEE Computer Society, 1997.

[CVA2010] Bäck, T., Krause, P., *ClearVu Analytics*. `http://divis-gmbh.de/ClearVu`, Abruf: 03.10.2011.

[Guy03] Guyon, I. and Elisseeff, A., An introduction to variable and feature selection, In: The Journal of Machine Learning Research, vol. 3, p. 1157–1182, 2003.

[Han96] Hansen, N. and Ostermeier, A., Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In: Evolutionary Computation, 1996., Proceedings of IEEE International Conference on Evolutionary Computation, IEEE, p. 312–317, 1996

[Han06a] Hansen, N., The CMA evolution strategy: a comparing review. In Lozano, J., Larranaga, P., Inza, I., and Bengoetxea, E., (eds.), Towards a new evolutionary computation. Advances on estimation of distribution algorithms, pages 75–102. Springer, Heidelberg, 2006

[Kna09] Knaus, J. and Porzelius, C. and Binder, H. and Schwarzer, G., *Easier parallel computing in R with snowfall and sfCluster*. The R Journal, 1:5459, 2009.

[Koch10b] Koch, P., Konen, W., Flasch, O. Bartz-Beielstein, T. Bartz-Beielstein, T., Optimizing Support Vector Machines for Stormwater Prediction. In: Chiarandini, M.; Paquete, L. and Preuss, M. (Eds.), Proceedings of Workshop on Experimental Methods for the Assessment of Computational Systems joint to PPSN2010, 2010, p. 47–59

[Koch10c] Koch, P., Bartz-Beielstein, T., Konen, W., Optimization of Support Vector Regression Models for Stormwater Prediction. In: F. Hoffmann and E. Hüllermeier (Eds.), Proceedings 20th Workshop Computational Intelligence, Universitätsverlag Karlsruhe, 2010

[Koch11a] Koch, P., Bischl, B., Flasch, O., Bartz-Beielstein, T., and Konen, W., On the Tuning and Evolution of Support Vector Kernels. Technical Report, Research Center CIOP (Computational Intelligence, Optimization and Data Mining), 2011

[Koge2010] Kögel, S.: Data-Mining-Cup DMC, `http://www.data-mining-cup.de`, Abruf: 03.10.2011

[Kone10a] Konen, W., Koch, P., Flasch, O. and Bartz-Beielstein, T., Parameter-Tuned Data Mining: A General Framework. In: F. Hoffmann, and E. Hüllermeier (Eds.), Proceedings 20th Workshop Computational Intelligence, Universitätsverlag Karlsruhe, 2010

[Kone11a] Konen, W. The TDM Framework: Tuned Data Mining in R. Technical Report, Cologne University of Applied Sciences, 2011

[Kone11b] Konen, W.; Koch, P., Flasch, O., Bartz-Beielstein, T., Friese, M. and Naujoks, B. Tuned Data Mining: A Benchmark Study on Different Tuners. Technical Report, Cologne University of Applied Sciences, 2011

[Kone11d] Konen, W., Koch, P., Flasch, O., Bartz-Beielstein, T., Friese, M. and Naujoks, B., Tuned Data Mining: A Benchmark Study on Different Tuners. In: N. Krasnogor (Ed.), GECCO '11: Proceedings of the 13th annual Conference on Genetic and Evolutionary Computation, 2011

[Kota06] Smits, G. and Kordon, A. and Vladislavleva, K. and Jordaan, E. and Kotanchek, M., Variable selection in industrial datasets using pareto genetic programming, In: Genetic Programming Theory and Practice (III), Springer, New York, 2006, p. 79-92

[Koza92] Koza, J.R., Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, Cambridge MA, 1992

[Liu89] Liu, D.C. and Nocedal, J., On the limited memory BFGS method for large scale optimization, In: Mathematical Programming, 1989, vol. 45, p. 503–528

[Mie06] Mierswa, I. and Wurst, M. and Klinkenberg, R. and Scholz, M. and Euler, T.: YALE: Rapid Prototyping for Complex Data Mining Tasks, in Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-06), 2006.

[Mik2006] Mikut, R., Burmeister, O., Reischl, M., Loose, T., *Die MATLAB-Toolbox Gait-CAD*. In: Mikut, R., Reischl, M. (eds.), Proceedings 16. Workshop Computational Intelligence, p. 114-124, Universitätsverlag Karlsruhe, 2006

[Piat11] Piatetsky-Shapiro, G., Poll on DM tools, http://www.kdnuggets.com/2011/05/tools-used-analytics-data-mining.html

[Pow1998] Powell, M.J.D., Direct search algorithms for optimization calculations, In: Acta Numerica, vol. 7, no. 1, 1998, p. 287–336, Cambridge University Press

[Qui95] Quinlan, J.R. and Cameron-Jones, R., Oversearching and layered search in empirical learning, In: Breast Cancer, 1995, vol. 286, p. 2–7

[TDMR11] TDMR package, downloadable from http://gociop.de/research-projects/tuned-data-mining

[Thi88] Thisted, R. A., Elements of Statistical Computing. Chapman and Hall, 1988

[Will11] Williams, G.: Data Mining with R and Rattle: The Art of Excavating Data for Knowledge Discovery, In: Use R!, Springer, 2011

[Wolf88] Wolfram, S., Mathematica: a system for doing mathematics by computer, Addison-Wesley Longman Publishing Co., Inc., 1988