Automatic Error Detection in Gaussian Processes Regression Modeling for Production Scheduling

Bernd Scholz-Reiter, Jens Heger
BIBA - Bremer Institut für Produktion und Logistik GmbH
at the University of Bremen
Hochschulring 20, 28359 Bremen, Germany
e-mail: {bsr|heg}@biba.uni-bremen.de

***Abstract*— In the application field of production, scheduling with dispatching rules is facing the problem that no rule performs globally better than any other. Therefore, machine learning techniques can be used to calculate estimates of rule performances and select the best rule for each system state. A number of estimates are of poor quality and lead to a wrong selection of rules. Motivated by this problem, to further stabilize the selection approach a general approach, to automatically detect 'faulty' estimates from regression models is introduced and analyzed in this paper. Therefore, different models are learned and if their estimates differ strongly, it is likely that at least one model delivers poor estimates. Additionally, a difference-threshold for our example data is defined. As a machine learning technique, we use Gaussian process regression with different covariance functions (kernels). The results have shown that our automatic detection works in most cases and poorly tuned models can be detected.**

Keywords-Gaussian processes; dispatching rules; machine learning; scheduling; multiple classifier techniques.

I. INTRODUCTION

Alpaydin described machine learning in the following way: "The goal of machine learning is to program computers to use example data or experience to solve a given problem" [8]. In regression, this means that the given data is analyzed and used to calculate a regression function, which can be used get estimates for unknown data points. In Fig. 1, an example is depicted, where the regression function gives an estimate for point *c*.

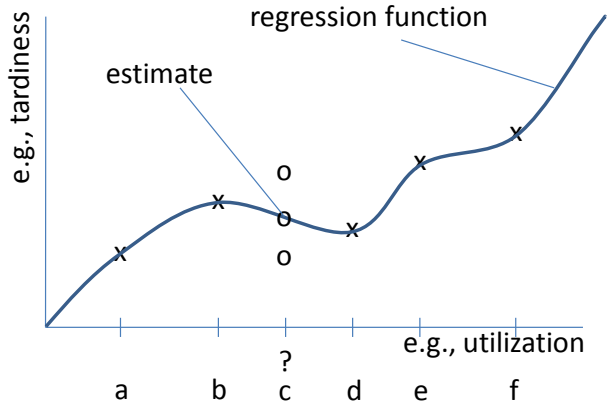

Figure 1: An example of regression function

Our general motivation is the optimization of a production system, e.g., reducing tardiness of jobs. Decentralized scheduling with dispatching rules is applied in many fields of logistics and production, which are characterized by high complexity and dynamics. Instead of calculating a global plan (a schedule for all jobs and machines), dispatching rules work in an autonomous, decentralized way. Dispatching rules, which are a special kind of priority rules, assign a job to a machine. Each time, the machine has finished a job and more jobs are waiting, the next job to be processed is selected by calculating a priority for each of the waiting jobs. This priority can be based on attributes of the job, the machines or the system. The job with the highest priority is chosen to be processed next. Dispatching rules have been developed and analyzed in the scientific literature for many years; see e.g., [1], [2] and [3]. The most well know rules are *Shortest Processing Time first* (SPT), *Earliest Due Date* (EDD) and *First in System First Out* (FSFO). Many dispatching rules perform well on different scenarios, but no rule has been found, which outperforms other rules across various objectives. For this reason, approaches to switch between rules depending on the current system conditions have been proposed. Most of these approaches use learning techniques (e.g., neural networks) to estimate the performance of each dispatching rule and select the best [4].

To calculate the performance of rules, we perform simulation runs of the production system with several dispatching rules and different system settings, e.g. different utilization levels. Since lots of possible variants exist and simulation runs are time consuming, we want to perform as few simulation runs as possible and use machine learning to estimate performances (e.g., tardiness) of not explicitly simulated settings. The results of simulation running of a production system are the learning data for models we learn with Gaussian Process Regression. Based on these models, the best dispatching rule for the current scenario is selected [5]. This procedure avoids costly and unnecessary simulation runs. It is practically not possible to run preliminary simulation runs for all parameter combinations.

In most cases, this approach works successfully, but in some cases the learning fails and results in improper

 ISBN: 978-1-61208-172-4

regression curves. This leads to a wrong selection of dispatching rules and thus to a poor performance of the production system. Our goal is to automatically detect, if the learned model might be 'faulty'. In this case, the learning data is supplemented using additional simulation runs.

In this paper, we concentrate on the problem of improper regression curves and their automatic detection. This is a general problem, which not only occurs in dispatching rule selection.

In this study, we use synthetic data instead of simulation runs. On the one hand these simulation runs are time consuming and on the other hand we need many different functions to test our error recognition. Therefore, we are using synthetic functions, which are similar to our simulation runs and concentrate on the error detection.

Gaussian processes (GP) is one promising machine learning technique [5]. --> These techniques have been introduced in 1996 and were promoted in the machine learning community by Williams et al. [6]. Analyses reveal their good prediction performance in comparison with other techniques [7]. As a further advantage, their formalism allows providing a measure of prediction quality with each predicted value in a natural way. Additionally, they are – their mathematical background aside – relatively easy to handle.

The obvious problem with machine learning is that learned models can only estimate values. Sometimes, their estimates differ strongly from the original value. These cases lead to wrong decisions. To provide more stability and to automatically detect wrong estimates, we suggest combining similar models. If one model strongly differs from the comparison data, further learning steps, such as adding more data, are necessary.

This paper is organized as follows: in Section 2, we introduce Gaussian processes and general problems with machine learning techniques. Section 3 comprises our approach and the settings of the performed experiments followed by the results in Section 4. The paper concludes with a short summary and provides directions towards future research.

II. PROBLEM DESCRIPTION AND STATE OF THE ART

Regression models are used to provide estimates of values, which are not exactly known, because it is too costly to calculate them or just not possible. In our application field, we experienced that a number of tuned models provided poor estimates, which is a general problem in regression. Therefore, in this paper, we determine how it is possible to automatically detect such bad models.

A. Gaussian Processes

1) Introduction

O'Hagan [9] represents an early reference from the statistics community for the use of a GP as a prior over functions, an idea which was only introduced to the machine learning community by Williams et al. [6].

We have a simulation model implicitly implementing a (noisy) mapping between a vector of state variable (in our case containing, e.g., utilization) and the objective function (mean tardiness) $y = f(x) + \varepsilon$. The learning consists of finding a good approximation f*(x) of f(x) to make predictions at new points x.

To tune such a model using GP requires some learning data as well as a so-called covariance function. This covariance function, sometimes called kernel, specifies the covariance between pairs of random variables and influences the possible form of the learned function f*.

Since we want to check and compare the tuned models, we use three different kernels, which are well suited for our application. These are the squared exponential (SE) covariance function (1), which is a common choice in Gaussian Process Regression. Additionally, we use 2 functions of the *Matérn* class (2), with parameter d=3 and d=5 (see [6], chapter 4), which are a good choice in many engineering applications.

$$k_y(x_p, x_q) = \sigma_f^2 \exp\left(-\frac{1}{2l^2}(x_p - x_q)^2\right) + \sigma_n^2 \delta_{pq} \tag{1}$$

$$\operatorname{cov}\left(f(x_p), f(x_q)\right) = k(x_p, x_q) = \sigma_f^2 f\left(\frac{\sqrt{d}}{l} r\right) \exp\left(-\sqrt{d}\, r\right) \tag{2}$$

with f(t)=1+t for d=3
f(t)=1+t+t²/3 for d=5.

The formulas further include the so-called hyperparameters. These parameters of a covariance function can be used to fine-tune the GP-model. The *squared exponential* covariance function used in our experiments has three hyperparameters. There is the length-scale *l*, the signal variance σ_f^2 and the noise variance σ_n^2. The *Matérn* functions have the signal variance σ_f^2 and factor *l* as well. Additionally, since hyperparameters can be interpreted as length-scale parameters in the case of the *squared exponential* covariance function, further optimization is possible. Rasmussen and Williams [10] describe the hyperparameters informally like this: "how far do you need to move (along a particular axis) in input space for the function values to become uncorrelated". Thus, the *squared exponential* covariance function implements an automatic relevance determination (ARD) [11], since the inverse of the length-scale determines how relevant an input is. A very large length-scale value means that the covariance will become almost independent of that input. ARD has been used successfully for removing irrelevant input by several authors, e.g., Williams et al. [6].

Gaussian processes provide a quality estimate of their predicted value, exemplarily denoted by the shaded area in Fig. 2. Fifteen noisy training points are given and since

 ISBN: 978-1-61208-172-4

there is noise, the standard deviation close to the training points is small, but not exactly zero. In between two points as well as at the beginning and the end, the quality of the estimates decreases.

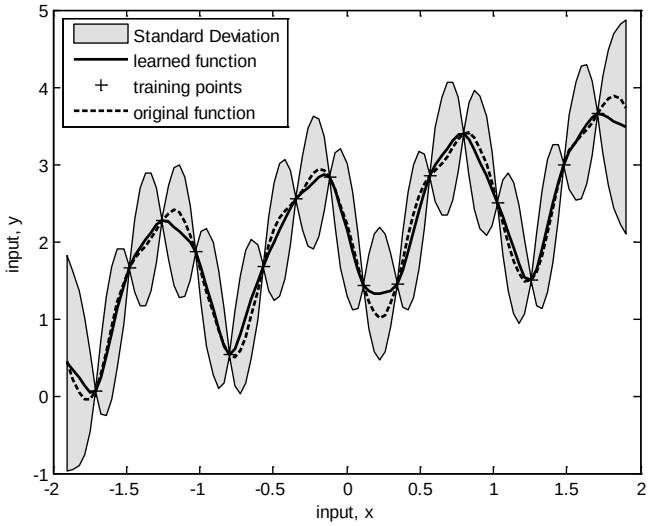

Figure 2. Example of a Gaussian process regression function with noisy training points observed. The mean prediction is shown as a black line and the shaded area denotes twice the standard deviation. The full underlying original function is also shown.

B. Application and Example

Learning with Gaussian processes is done by selecting a covariance function and setting its free hyperparameters. To learn or optimize the hyperparameters, the marginal likelihood should be maximized [10]. Further, the setting of the hyperparameters aims to a minimization of the generalization error, which denotes the average error on unseen test examples. This is done with cross-validation by splitting the training data in learning and test data. An optimization of the training error does not take place, this may lead to an over-fitting of the data (see below).

C. Machine learning – model quality

A typical task in data mining is learning a model that bases on available data. These models can be regression models or classifiers. The problem evaluating such a model is that it may have an adequate prediction capability, but might fail to predict future unseen data [13]. This problem is called overfitting, because the model fits well on the training data, but the general quality might be poor. To estimate the generalization performance in this context, a procedure called cross-validation is recommended. The idea of cross-validation originates in the 1930s and has been further developed by Mosteller and Wallace and others in the 1960s [12].

1) Cross-Validation

Cross-Validation is a statistical method for the evaluation of learning algorithms by dividing data into two parts. One is used to learn a model, the other used to validate it. The basic form of cross-validation is the *k*-fold cross-validation. In *k*-fold cross-validation, the data is first partitioned into *k* equally sized folds. Subsequently, *k* iterations of training and validation are performed. Within each iteration, a different fold of the data is held-out for validation, while the remaining *k*-1 folds are used for learning [13].

A special case of *k*-fold cross-validation is 'leave-one-out' cross-validation. In this case, *k* is set to the number of instances in the data. This means, that during each iteration, all data points are used for learning except one, which is used for testing. Leave-one-out cross-validation is used especially, when the available data are very rare.

2) Bootstrapping

Another method for assigning measures of accuracy to sample estimates is *bootstrapping*, which was introduced by Efron and Tibshirani [16]. Bootstrapping is a method that uses resampling to create sets of data derived from one original data set. The bootstrap process can be described in the following steps: *b* bootstrap samples are generated from the original data set. Each of these samples has *n* elements, which were generated by sampling with replacement *n* times. By calculating the value of the estimator of the replicates the bootstrap replicates can be obtained. The variance of the estimates can be determined by computing the variance of the estimates for the samples. The assumptions gained from bootstrapping are similar to those gained from cross-validation, i.e., stability of the algorithm on the dataset, which should closely approximate the real world [14]. More details and a comparison of cross-validation and bootstrapping was conducted by Kohavi [14].

3) Problem description

In Fig. 3, two different kernel functions are used to learn a model with 15 training points. Cross-validation is used to optimize the hyperparameters. The learned function 2 provides estimates close to the original function. Learned function 1 is not that close and has the form of a linear average function. Errors or problems like this occur regularly and deteriorate the estimates and the decisions based on them.

Cross-validation and bootstrapping are good accuracy estimation methods and are applicable for parameter setting or performance estimation. Still, it is hard to detect with these methods, if the learning model has obviously failed, like function 1 in Fig. 2 did. These cases are responsible for a high amount of the total error of the learning model. As a solution for this problem, this paper presents the approach of combining different models for automatic error recognition.

 ISBN: 978-1-61208-172-4

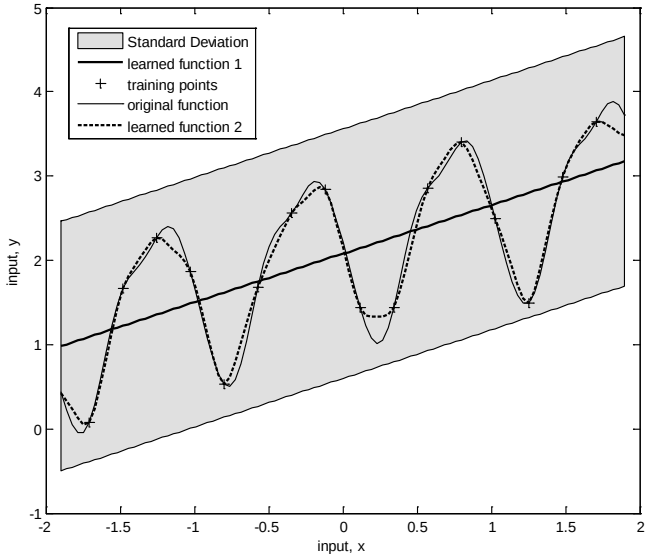

Figure 3: An example for two learned regression curves, where the learned function 1 one provided bad estimate and learned function 2 provides good estimates close to the original function

III. APPROACH AND EXPERIMENTS

The presented approach aims to stabilize the learning process by detecting 'faulty' regression curves automatically. Therefore, we tune different models with different kernels, e.g, covariance functions. If there are great differences between these models, it is obvious, that at least one model does not fit to the learning data. In these cases, more learning data can help to improve the models.

Generally, adding more learning data can also reduce errors. But this contradicts the objective of reducing costly simulation runs by using machine learning.

A. Experimental setup

For our experiments we use the framework provided by Rasmussen and Nickisch [10], [15]. Three covariance functions (*squared exponential* and *Matérn* 3 & 5) are implemented as learning kernels. The learning data is synthetically generated by a neural network based covariance function in combination with a periodic component (see framework documentation [15], covariance functions 'NNone' and 'Periodic'). Hyperparameters are set to l=1; sf =1 (NNone) and l=1/12, p=1, sf =1 (Periodic) and noise variance = 1. This way, we get smooth functions, which are similar to the curves resulting from simulation runs with dispatching rules scheduling. But since we need a lot of functions to get a general result, we use these synthetic functions instead of simulations runs.

B. Experiments

For the analysis, 1000 random functions are generated as described before, with a discretization level of 101 on the x-level. We have used 13 learning data points and the hyperparameters are optimized using cross-validation.

When using Gaussian processes regression, a covariance function needs to be selected. The most common choice is the *squared exponential*, but depending on the application other covariance functions or their compositions might fit better to the application data. In this paper two different approaches are analyzed: First, one covariance function is set as the default learning kernel and others are used to check the results for errors.

If no preferable covariance function is known in advance a more general approach is analyzed. In this case two or more covariance function are used to learn a model and each of their predictions are used to calculate a mean resulting prediction.

1) One main covariance function is selected

In the first set of experiments the *squared exponential* covariance function is set as the main learning model. To detect the cases, the considered model is of less quality, the *Matérn* functions with parameter d=5 (d=3 leads to very similar results) comes into operation. The learning error is determined by calculating the difference between the *squared exponential* and the original function. The differences between *Matérn* and *squared exponential* are also calculated. If these values correlate, a threshold level for the difference between the *squared exponential* and the *Matérn* functions indicates a higher error in the learning model.

2) Predictions based on multiple covariance functions

In the second set of experiments the *squared exponential* is not set as the standard model. Instead, both models are used equally. That means, the learning error in these experiments is the difference between the original function and the mean prediction of both models. The difference between both models is also calculated. If these data correlates, errors in the examined models can be detected, without knowing, which model works better to the application data in advance.

IV. RESULTS

Fig. 4 depicts the results of the differences between two learning models (*squared exponential* to *Matérn* 5) and the error to the original function (*squared exponential* to *original data*). Some correlating high errors for both are highlighted. Fig. 6 does the same for (mean (*squared exponential-Matérn 3)* to *original data*).

The first set of experiments show that the difference between the two models correlates to the error. There are many data pairs in Fig. 4 showing this for high values exemplarily. The higher the difference in the two models, the higher the error of the *squared exponential* learning model. Fig. 5 shows that there is a linear dependency between both values. Many data points are close to 0, which means, that most times the learning worked well and the difference between the models is small. This makes it easy to find different threshold levels to divide the functions into

 ISBN: 978-1-61208-172-4

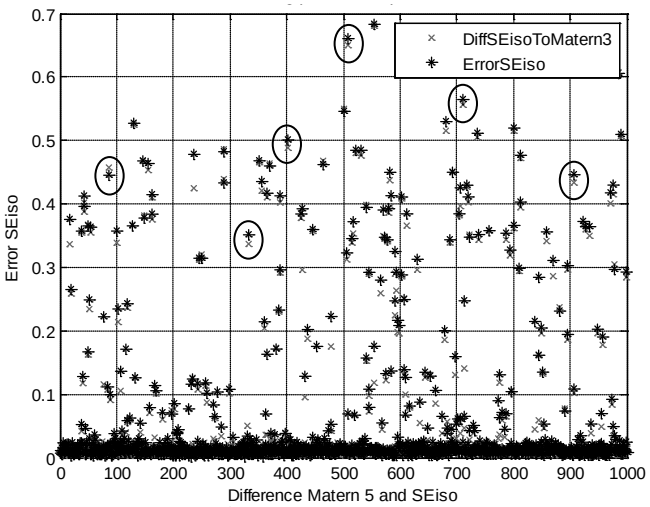

Figure 4: 1000 synthetic functions learned by the squared exponential and the Matérn 5 function. The error between the original function and the squared exponential function are depict as well as the difference between squared exponential and Matérn.

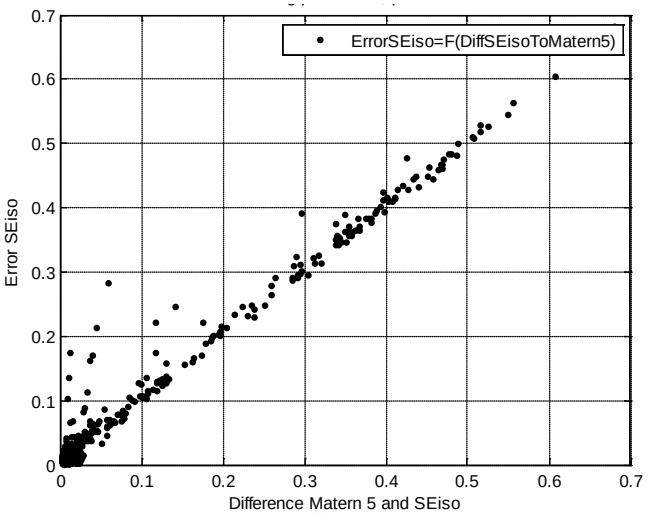

Figure 5: Correlation between differences in the two learned models (squared exponential and Matérn 5) and the error between the squared exponential function and the original data

TABLE I. THRESHOLD AND RESULTS DISTRIBUTION

threshold	number of functions above threshold in %	total error of functions above threshold in %
0.25	9.0	61.5
0.2	10.7	64.8
0.15	11.8	69.2
0.1	14.4	74.4
0.075	15.5	76.1
0.05	18.2	78.2
0.03	20.8	81.3
0.02	32.3	82.8

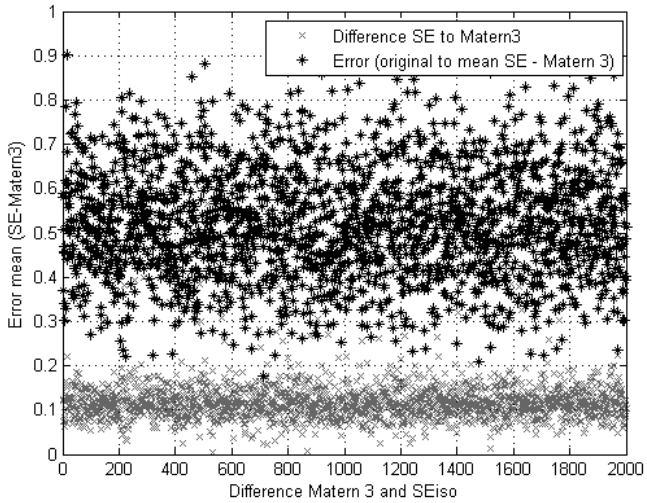

Figure 6: 2000 synthetic functions learned by the squared exponential and the Matérn 3 function. The error between the original function and the mean of squared exponential function and Matérn 3 are depict as well as the difference between squared exponential and Matérn 3.

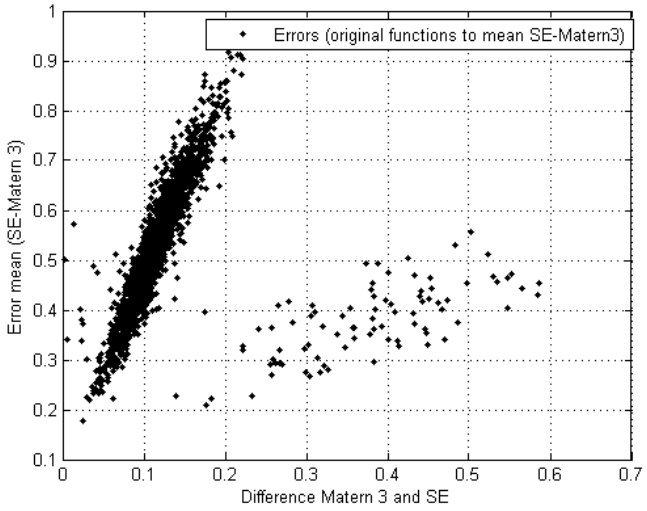

Figure 7: Correlation between differences in the two learned models (squared exponential and Matérn 3) and the error between the squared exponential function and the original data

TABLE II. THRESHOLD AND RESULTS DISTRIBUTION

threshold	number of functions above threshold	total error of functions above threshold in %
0.14	23.85	29.16
0.143	21.95	26.67
0.145	21.05	25.77
0.147	19.65	24.02
0.15	17.90	21.88
0.16	12.35	14.65

 ISBN: 978-1-61208-172-4

good and faulty learning models. Possible settings are depicted in table 1. A threshold level of 0.03 seems to be appropriate for this experimental setting. If the difference between *squared exponential* and *Matérn* 5 is higher than 0.03, about 20% of the functions are detected, which are responsible for about 80% of the total error. By adding more data to the learning process of these recognized 20% of functions up to 80% of the total error can be reduced.

The second set of experiments show similar results. There is also a linear dependency between the difference of the models and the total error of the learning model consisting of both covariance functions. That means, if the two learning models differ, the total error gets higher. Fig. 7 shows that in most cases at least one model differs from the original values, because there are only a few functions close to [0, 0]. This is different to the results depict in Fig. 5. The effect can be seen in table 2. The 19% of functions, which are over the threshold level of 0.147 are responsible for about 24% of the total error.

The results demonstrate that there is a correlation between the difference in the learning models and the total error. This is a promising result, however, the approach to work with the mean value of two learning models is only good for a small improvement. One possible reason for this can be the minimized number of learning data. At least one model does not provide good estimates in most cases, which reduces quality of the mean value estimates of both.

Even if the approach brings only a small improvement in our experiments here, the approach can be very useful in the real-life application, because there are regularly a few models, which are extremely wrong, e.g., some regression curves go up to infinity. If these cases can be found automatically, this stabilizes the general learning approach a lot.

V. CONCLUSION AND FURTHER STEPS

The presented experiments show that automatic error detection, i.e., faulty tuned models, in Gaussian processes with different kernels is a promising approach. The experiments have shown, that if estimates from different kernels differ strongly, the difference to the original function is high.

The benefit from the approach is, that if less regression models are 'faulty', the 'learning' is more stable and less unfavorable decisions are made. The approach to use different learning kernels, which all seem appropriate for this application, is promising. Since deviations in the models are a clear indication that at least one model provides poor estimates.

Next steps are to further improve the approach where the average estimates of two learning models are used. Some preliminary tests to find the most appropriate kernel and use this kernel mainly, can be a promising approach.

Nevertheless, the presented approach brings more stabilization to the learning process and further steps will be to implement it in the application of production scheduling in the future.

ACKNOWLEDGEMENTS

The authors are grateful to the generous support by the German Research Foundation (DFG), grant SCHO 540/17-1.

REFERENCES

[1] J. H. Blackstone, D. T. Phillips, and G. L. Hogg, "A state-of-the-art survey of dispatching rules for manufacturing job shop operations," *International Journal of Production Research*, vol. 20, no. 1, pp. 27–45, 1982.

[2] R. Haupt, "A survey of priority rule-based scheduling," *OR Spektrum*, vol. 11, no. 1, pp. 3–16, 1989.

[3] S. S. Panwalkar and W. Iskander, "A survey of scheduling rules," *Operations Research*, vol. 25, no. 1, pp. 45–61, 1977.

[4] W. Mouelhi-Chibani and H. Pierreval, "Training a neural network to select dispatching rules in real time," *Computers & Industrial Engineering*, vol. 58, no. 2, pp. 249 – 256, 2010, scheduling in Healthcare and Industrial Systems.

[5] B. Scholz-Reiter, J. Heger, and T. Hildebrandt, "Gaussian processes for dispatching rule selection in production scheduling," *Procceeding of the International Workshop on Data Mining Application in Government and Industry 2010 (DMAGI10) As Part of The 10th IEEE International Conference on Data Mining.*, 2010.

[6] C. K. I. Williams and C. E. Rasmussen, "Gaussian processes for regression," *Advances in Neural Information Processing Systems*, vol. 8, pp. 514–520, 1996.

[7] C. E. Rasmussen, "Evaluation of gaussian processes and other methods for non-linear regression," *PhD thesis, Department of Computer Science, University of Toronto*, 1996.

[8] E. Alpaydin, *Introduction to Machine Learning (Adaptive Computation and Machine Learning Series)*. The MIT Press, 2004, vol. 14, no. 1.

[9] A. O'Hagan, "Curve fitting and optimal design," *Journal of the Royal Statistical Society*, vol. 40, no. 1, pp. 1–42, 1978.

[10] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, December 2006.

[11] R. M. Neal, *Bayesian Learning for Neural Networks (Lecture Notes in Statistics)*, 1st ed. Springer, August 1996.

[12] F. Mosteller and D. L. Wallace, "Inference in an authorship problem," *Journal of the American Statistical Association*, vol. 58, no. 302, pp. 275–309, 1963. [Online]. Available: http://www.jstor.org/stable/2283270

[13] P. Refaeilzadeh, L. Tang, and H. Liu. (2008) Cross-validation, glossar, Arizona State University. last checked 07.07.2011. [Online]. Available: www.public.asu.edu/~ltang9/-papers/ency-cross-validation.pdf

[14] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *International joint conference on artificial Intelligence*, 1995, pp. 1137–1143.

[15] C. E. Rasmussen and H. Nickisch. (2011) Gpml matlab code version 3.1. last checked: 07.07.2011. [Online]. Available: www.gaussianprocesses.org

[16] Efron, B. and Tibshirani, R.: An introduction to the bootstrap, Chapman & Hall, 1993

 ISBN: 978-1-61208-172-4