Scalable Hyperparameter Optimization with Products of Gaussian Process Experts

Nicolas Schilling(✉), Martin Wistuba, and Lars Schmidt-Thieme

Information Systems and Machine Learning Lab,
Universitätsplatz 1, 31141 Hildesheim, Germany
{schilling,wistuba,schmidt-thieme}@ismll.uni-hildesheim.de

Abstract. In machine learning, hyperparameter optimization is a challenging but necessary task that is usually approached in a computationally expensive manner such as grid-search. Out of this reason, surrogate based black-box optimization techniques such as sequential model-based optimization have been proposed which allow for a faster hyperparameter optimization. Recent research proposes to also integrate hyperparameter performances on past data sets to allow for a faster and more efficient hyperparameter optimization.

In this paper, we use products of Gaussian process experts as surrogate models for hyperparameter optimization. Naturally, Gaussian processes are a decent choice as they offer good prediction accuracy as well as estimations about their uncertainty. Additionally, their hyperparameters can be tuned very effectively. However, in the light of large meta data sets, learning a single Gaussian process is not feasible as it involves inversion of a large kernel matrix. This directly limits their usefulness for hyperparameter optimization if large scale hyperparameter performances on past data sets are given.

By using products of Gaussian process experts the scalability issues can be circumvened, however, this usually comes with the price of having less predictive accuracy. In our experiments, we show empirically that products of experts nevertheless perform very well compared to a variety of published surrogate models. Thus, we propose a surrogate model that performs as well as the current state of the art, is scalable to large scale meta knowledge, does not include hyperparameters itself and finally is even very easy to parallelize. The software related to this paper is available at https://github.com/nicoschilling/ECML2016.

Keywords: Hyperparameter optimization · Sequential model-based optimization · Product of experts

1 Introduction

In recent years, machine learning and data mining has been gaining more and more attention by showing very good prediction performance in areas such as recommender systems, pattern, speech and visual object recognition and many more. The lift in prediction performance is usually due to the development of more complex models as we see for example in the area of deep learning. However, developing more complex models usually has drawbacks, which is the increasing time that is spent for learning the model plus the increasing dimensionality of the hyperparameter space of the associated model. By hyperparameters we denote parameters of a model that can not explicitly be learned from the data by a well-defined optimization criterion such as the minimization of a regularized loss functional. These hyperparameters can be continuous, the reader might consider a positive learning rate of a gradient descent optimization approach, or a regularization constant of a Tikhonov regularization term. However, by hyperparameters we also consider discrete choices, such as the dimensionality of a low-rank factorization or the number of nodes and layers in a deep feedforward neural network. Additionally, hyperparameters can also be categorical, for instance the choice of kernel function in a support vector machine, or even the choice of loss function to optimize within the optimization criterion. Finally, even model choice as well as preprocessing of the data can be understood as hyperparameters of a general learner. What all of these parameters have in common is that they cannot be optimized in a straightforward fashion, but usually their correct setting renders methods from producing weak predictions to state-of-the-art predictions. Due to this impact, practicioners that do not know the underlying techniques very well usually have a hard time optimizing hyperparameters and therefore rely on either choosing standard hyperparameters or on performing a grid-search, which tries many hyperparameters and in the end chooses the one that performs best. In this way, a lot of unnecessary computations are created.

Out of this reason, recent research proposes to use black-box optimization techniques such as sequential model-based optimization (SMBO) to allow for a more directed search in the hyperparameter space. Essentially, SMBO treats the hyperparameter configuration as input for a black box function and uses a surrogate model to learn on a few observed performances to then predict the performance of any arbitrary hyperparameter configuration. The predicted performance as well as the uncertainty of the surrogate model are then used within the context of an acquisition function to finally predict a hyperparameter configuration that likely performs better, while keeping a good balance between exploitation and exploration. On the one hand, exploitation is attained whenever the acquisition function chooses hyperparameter configurations that are very close to already observed well-performing configurations and therefore the surrogate model is quite certain about its estimation. On the other hand, exploration is met if the acquisition function chooses configurations that are very distant to all observed configurations, i.e. explores new areas of the hyperparameter space, where the surrogate model is quite uncertain about its prediction. Given that usually only a few initial observations are present and the amount of overall queries for hyperparameter configurations is limited, a decent tradeoff between both exploration and exploitation is desired.

More recent work is inspired by the area of meta learning, where the goal is to transfer knowledge for parameters of a given model from having learned this model already on other data sets [4]. Thus, these methods propose to also take into account the knowledge of hyperparameter performances on different (past) data sets, where hyperparameter opimization has already been done. This is quite intuitive, as every experienced practitioner, who has already learned a model many times on different data sets probably comes up with better hyperparameter configurations for the target data set to test initially. In many works, the surrogate model is then learned on the hyperparameter performances of past data sets and therefore has a better knowledge of well-performing hyperparameters to choose. In order for the surrogate model to not confuse performances of the same hyperparameter configuration on different data sets, the meta knowledge is usually augmented by additional meta features that describe characteristics of a data set.

Many surrogate models have been proposed, but one of the simplest surrogates is probably a Gaussian process (GP), as it is relatively simple to learn, delivers good predictions and furthermore, due to its probabilistic nature, allows for a direct estimation of uncertainties, which is a key ingredient for SMBO. Another advantage of using Gaussian processes compared to other surrogate models is that they are basically hyperparameter free, as all the parameters that we have to specify for the kernel can be learned by optimizing their marginal log likelihood. However, Gaussian processes have one huge drawback which lies in their scalability. In order to learn a Gaussian process, the kernel matrix computed over all observed instances has to be inverted which is an operation with cubic expense in the number of observations. Thus, if we seek to include meta knowledge of many past data sets into the training data of the Gaussian process, learning the Gaussian process might even take more time and memory than learning the model we seek to optimize the hyperparameters for, which then renders a Gaussian process infeasible, despite its advantages.

In this paper, we propose to use a product of Gaussian process experts as surrogate model, where basically an independent GP is learned for all the observation of one past data set and in the end all the predictions of the individual experts are assembled to predict hyperparameter performances of the target data set. Following this approach, our work has four main contributions:

- We learn a product of GP experts, which allows for the inclusion of a large amount of meta information,
- by using GPs as base surrogate model, we employ surrogates that are very easy and fast to learn, and do not require much memory
- additionally, by using GPs, we do not introduce additional surrogate-hyperparameters in opposition to many state of the art methods,
- finally, we show empirically that products of GP experts perform very competitively for hyperparameter optimization against a variety of published competitors, as well as make both the implementation and the meta data publicly available.

2 Related Work

As already mentioned, in the recent years there has been a growing interest in research regarding hyperparameter optimization. Random search has been proposed as an alternative to grid-search and works well in cases of low effective dimensionality, where a subspace of the hyperparameter space does not influence the results as much as the remaining hyperparameter dimensions [3].

In the context of SMBO, many different surrogate models have been proposed in a variety of papers. At first, an independent Gaussian process [17] was used. We denote it as independent as it does not learn across data [20]. Secondly, random forests have been proposed as surrogates and inherit the ability to work well with non numerical as well as hierarchical hyperparameters [13]. Regarding hyperparameter optimization using meta knowledge, a stacking of a GP on top of a ranking SVM was proposed [2], as well as a Gaussian process with a multi task kernel in two closely related works [21,26]. Furthermore, a mixture of a multilayer perceptron and a factorization machine has been employed as surrogate model [18], which automatically learns data set representations and therefore does not necessarily need meta features.

A different aspect of using meta knowledge is conducted through learning an initialization of well-performing hyperparameters. The first work in this context is [8] where the initial hyperparameters are chosen based on data sets that are closest with respect to the Euclidean distance evaluated on the meta features of the respective data sets. This intuition has been extended by [24] which uses a differentiable plug in estimator to compute initial hyperparameters. Finally, [25] employs a static sequence of hyperparameters that is learned using meta knowledge and does not need a surrogate model at all, however, it has the drawback that it needs meta information over different data sets evaluated on the same hyperparameter grids.

There is a plethora of other approaches that are either model specific [1] or use genetic algorithms [7,15], or do both in conjunction [9]. As these approaches are not embedded in the context of SMBO, we will leave them out of further discussions.

Since we are seeking to employ product of experts models in the framework of SMBO-based hyperparameter optimization, we also review the related work in this field as well as various techniques to speed up Gaussian process learning. Initially, product of experts models have been proposed by [11] alongside with a learning algorithm [12] to train the parameters of such a model. The *generalized* product of experts [5] introduces additional weighting factors within the product in order to reduce the overconfidence of the product of experts in unknown areas. Another model that also estimates a joint probability density given by a set of experts is the Bayesian committee machine [22], which includes the prior in its predictions. Finally, the work by [6] combines both the idea of the generalized POE with its weighting factors with the Bayesian committee machine. We do want to highlight that all of this work is not specifically tailored to Gaussian processes, however, [6] argues that using products of experts is an easy way to make Gaussian processes more scalable to larger training data sets.

Additionally, many efforts have been made by the means of sparse GPs, namely Gaussian processes learned on subsets of the original training data such as [19] which employs kd-trees for subsampling. There are many more works in this area such as [10,23] or [16], however, as we want to make use of the rich meta information of hyperparameter performance on other data sets using only a subset of the meta information seems counterintuitive. Due to this reason, we do not intend to use sparse GPs as surrogates for Bayesian hyperparameter optimization.

3 Background

In this section we first review hyperparameter optimization and sequential model-based optimization in general, secondly, we discuss Gaussian processes shortly and lastly we give a review of product of experts models which we ultimately seek to employ as surrogate models.

3.1 Problem Setting

Let $\mathcal{D}$ denote by the space of all data sets, following the notation by [3], we denote a learning algorithm for a fixed model class $\mathcal{M}$ by a mapping $\mathcal{A} : \Lambda \times \mathcal{D} \longrightarrow \mathcal{M}$. Thus, an algorithm $\mathcal{A}$ is essentially a mapping from a given hyperparameter configuration and training data to a model which is learned by minimizing a loss functional. In many cases, the hyperparameter space Λ is the cartesian product of lower dimensional spaces. Now we can define the problem of *hyperparameter optimization* as choosing the hyperparameter configuration $\lambda^\star$ which minimizes the loss of a learned model on given validation data:

$$\lambda^\star := \underset{\lambda \in \Lambda}{\arg\min}\ \mathcal{L}(\mathcal{A}(\lambda, D^{\text{train}}), D^{\text{val}}) =: \underset{\lambda \in \Lambda}{\arg\min}\ b(\lambda, D). \tag{1}$$

Please note that we use the short b as notation for the process of learning a model on training data with given hyperparameters and evaluating it on validation data. Clearly, b is the black box function that we seek to optimize using Bayesian optimization.

3.2 Sequential Model-Based Optimization

The SMBO framework is depicted in Algorithm 1. It starts by learning a surrogate model denoted by Ψ such that $\Psi \approx b$ on a set of given hyperparameter performances which are stored in the observation history $\mathcal{H}$. Secondly, the surrogate model will be used to predict the hyperparameter performance of unknown hyperparameters, these predictions as well as the uncertainties will be forwarded to the acquisition function a, which then picks a hyperparameter configuration to test. The most commonly used acquisition function is Expected Improvement (EI) and can be computed analytically if one assumes the probability of improvement to be Gaussian [14]. Having chosen a candidate configuration, b

Algorithm 1. Sequential model-based optimization across data sets

```
Input: Hyperparameter space $\Lambda$, observation history $\mathcal{H}$, target data set $D$, number of
    iterations $T$, acquisition function $a$, surrogate model $\Psi$, initial best hyperparameter
    configuration $\lambda^{\text{best}}$.
Output: Best hyperparameter configuration $\lambda^{\text{best}}$ for $D$
1: for $t = 1$ to $T$ do
2:     Fit $\Psi$ to $\mathcal{H}$
3:     $\lambda^{\text{new}} = \arg\max_{\lambda \in \Lambda} a\left(\Psi(\lambda, D), \mathcal{H}\right)$
4:     Evaluate $b\left(\lambda^{\text{new}}, D\right)$
5:     if $b(\lambda^{\text{new}}, D) < b(\lambda^{\text{best}}, D)$ then
6:         $\lambda^{\text{best}} = \lambda^{\text{new}}$
7:     $\mathcal{H} = \mathcal{H} \cup \left(\lambda^{\text{new}}, b\left(\lambda^{\text{new}}, D\right)\right)$
8: return $\lambda^{\text{best}}$
```

will be evaluated for the proposed hyperparameter configuration, the result will be fed into the observation history and the process is repeated for T many times until finally a best hyperparameter configuration λ^{best} is found. Additionally, the surrogate model's feature vector is usually augmented by meta features, which are descriptive features of a data set, to allow the surrogate model to distinguish between different data sets.

3.3 Gaussian Processes

We introduce Gaussian processes as we use them as base models in a product of experts. Given is a regression problem of the form

$$y(x) = f(x) + \epsilon, \tag{2}$$

where we assume i.i.d. noise $\epsilon \sim \mathcal{N}(0, \sigma^2)$. A Gaussian process assumes that for a given set of input variables $X = (x_1, ..., x_N)$ with associated labels $y = (y_1, ..., y_N)$ the labels are multivariate Gaussian distributed $y \sim \mathcal{N}(0, K)$, where K is a covariance matrix that is defined through a positive semidefinite kernel function $k(x, x')$. A very common choice for k is the squared exponential kernel

$$k(x, x') = \exp\left(\frac{-\|x - x'\|^2}{2\sigma_l^2}\right) + \sigma^2 \delta(x = x'), \tag{3}$$

where $\theta = (\sigma_l, \sigma)$ are denoted as the hyperparameters and the δ function returns 1 if its predicate is true and 0 otherwise. Given a set of known observations, the conditional distribution of a label $f_\star$ given its input $x_\star$ is Gaussian distributed with mean and covariance

$$\mu(f_\star) = k_\star^\top K^{-1} y \tag{4}$$

$$\sigma^2(f_\star) = k_{\star\star} - k_\star^\top K^{-1} k_\star, \tag{5}$$

where $k_\star = (k(x_1, x_\star), ..., k(x_n, x_\star))$ is the vector of kernel evaluations of the new input $x_\star$ to all observed inputs and $k_{\star\star} = k(x_\star, x_\star)$ is the prior covariance of $f_\star$.

As we can see, using a GP for predictions requires inverting the kernel matrix of size N, which is an operation of $\mathcal{O}(N^3)$ and thus becomes infeasible for data sets with many instances. Recalling that our primary goal was to include large scale meta knowledge of past hyperparameter performances, this boundary might be reached very soon, which would force us to throw valuable data away or rely on other surrogate models. However, solving the linear system of equations for inversion of K can be reduced to $\mathcal{O}(N^2)$ by using a Cholesky decomposition of the kernel matrix [17].

The kernel hyperparameters can be learned by maximizing their marginal log likelihood using standard optimization techniques such as gradient ascent. Again, for optimizing the kernel hyperparameters, we have to invert the kernel matrix as well as compute its determinant, which also both scale cubically in the dimension of K. As gradient ascent might use several iterations to converge to a useful θ, this inversion becomes even more the bottle neck with respect to both computational speed as well as memory usage.

3.4 (Generalized) Product of Experts (POE)

In order to scale Gaussian processes to a large training data set (i.e. observation history) we will use product of experts models [11], of which several variants have been proposed. Within a product of GP experts, a set of M invididual Gaussian processes are learned on M disjoint subsets of the training data, so let us decompose our training data as

$$X = (X^{(1)}, ..., X^{(M)}) \qquad y = (y^{(1)}, ..., y^{(M)}), \tag{6}$$

such that the individual subsets of instances and labels are disjoint. Then, following the independence assumption, the marginal joint likelihood factorizes into a product of single likelihoods

$$p(y \mid X, \theta) = \prod_{i=1}^{M} p_i \left(y^{(i)} \mid X^{(i)}, \theta^{(i)} \right). \tag{7}$$

Thus, in order to learn the individual experts, we only need to invert kernel matrices of the size of roughly N/M, thus learning the individual experts can be done in $\mathcal{O}(N^3/M^3)$ which is a reasonable reduction for a sufficiently large enough M. In this way, we also learn M many different sets of kernel hyperparameters.

As the M experts have been learned, we can compute the marginal likelihood by multiplying all individual likelihoods. The *generalized* product of experts [5] introduces additional weighting factors β_i such that:

$$p(y \mid X, \theta) = \prod_{i=1}^{M} {p_i}^{\beta_i} \left(y^{(i)} \mid X^{(i)}, \theta^{(i)} \right). \tag{8}$$

Naturally, if all $\beta_i = 1$, we arrive at the initial formulation of Eq. 7. Computing the product of the individual likelihoods yields a density that is proportional to a Gaussian with following mean and precision:

$$\mu^{\text{poe}}(f_\star) = (\sigma^{\text{poe}}(f_\star))^2 \sum_{i=1}^{M} \beta_i \sigma_i^{-2}(f_\star)\mu_i(f_\star) \tag{9}$$

$$(\sigma^{\text{poe}}(f_\star))^{-2} = \sum_{i=1}^{M} \beta_i \sigma_i^{-2}(f_\star) \tag{10}$$

Essentially, by replacing $\sigma_i^{-2}(f_\star) = \tau_i(f_\star)$ with the precision, we see that the mean predicted by the product of experts is a sum of means, weighted by the product of the individual β_i and the precision τ_i, which is then divided by the total sum of weighting factors. Usually, the β_i are set such that $\sum_i \beta_i = 1$, surprisingly, this already works quite well for $\beta_i \equiv 1/M$. This does not change the mean as the multiplication with the precision cancels out the effect, however, the precisions effectively get weighed down and decrease the overconfidence of the initial product of experts without any weights.

3.5 Product of Experts in SMBO

Having introduced the product of experts models, their implementation for hyperparameter optimization in the SMBO framework seems straightforward. However, a few questions still remain unanswered. At first, we split the meta knowledge into all the instances belonging to hyperparameter performances of one data set. In this way, each expert will be learned on the meta information of one distinct data set. If this would still be too large for a GP to learn, we could further subdivide them into smaller subsets.

Secondly, the question of how the information on the target data set will be incorporated into the surrogate model remains. We seek for two alternatives, in the first one we simply add the information of new points on the target data set to all the experts in the ensemble. Doing this, we effectively train all experts to be expert for two data sets, the initial one they have been trained on plus the target data set. In our implementation, we then follow the intution of all weights summing up to one, thus we set all $\beta_i = 1/M$.

As an independent Gaussian process that is learned without any meta knowledge already behaves reasonably well as a surrogate model, we also tried another alternative. We still feed the target data set information into all experts learned in the ensemble but additionally create a new GP that carries only the information of the target data set and is weighed much higher than the individual experts. Specifically, we use $\beta_i = 1/2M$ for the individual experts and $\beta_{M+1} = 1/2$ for the GP learned on only the target data set responses. In this way, we use the meta information as well as the strength of an independent Gaussian process.

Additionally, we seek to scale the hyperparameter performances observed in the meta data as well as the hyperparameter performances of the target data set. This is due to the fact that the range of b naturally depends on the data set. Consider for example a classification problem where b models the misclassification rate of a classifier for some test data. Naturally, for some data sets, very low misclassification rates might be achieved in contrast to other data sets that are simply harder to classify. A POE might then be biased towards choosing hyperparameter configurations that produce good results on simple data sets, which is something we want to prevent. In order to do so, we scale the labels of the meta data to become standard Gaussian distributed, for the target data set, we do this on-the-fly every time we see a new response of b as was also proposed by [26].

4 Experiments

To evaluate the proposed surrogate models for hyperparameter optimization, we conduct hyperparameter optimization within the SMBO framework including a variety of published baselines. The experiments are performed on two meta data sets that we have created ourselves.

4.1 Meta Data Set Creation

We have created two meta data sets for the task of classification using two distinct classifiers, namely being a support vector machine (SVM) and AdaBoost. These meta data sets consists of a complete grid search for both classifiers on 50 classification data sets that we have taken from the UCI repository[1]. If splits were already given, we merged them into one complete data set, shuffled the resulting data set and then took 80 % of the data for training and the remaining 20 % for testing. The AdaBoost meta data set was created by running AdaBoost[2] with hyperparameters $I \in \{2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000\}$ and $M \in \{2, 3, 4, 5, 7, 10, 15, 20, 30\}$. This yields 108 meta instances per data set and therefore the overall meta data set contains 5400 instances.

The second meta data set was created by running an SVM[3] on all of the data sets, with four hyperparameters. The first one resembles the choice of kernel and is categorical between a linear, a polynomial and an RBF kernel, thus introduces three binary hyperparameters. The second hyperparameter is the tradeoff parameter, usually denoted as C, the third and fourth hyperparameter are the degree d of the polynomial kernel and the width γ of the RBF kernel. If the kernel hyperparameters are not used, i.e. the polynomial degree for an RBF kernel, we set them to a constant value of zero. As for the AdaBoost meta data set, we computed the misclassification rates using grid-search, where C was chosen from the set $\{2^{-5}, \ldots, 2^{6}\}$, the polynomial degree d was chosen from $\{2, \ldots, 10\}$ and γ was chosen from $\{0.0001, 0.001, 0.01, 0.05, 0.1, 0.5, 1, 2, 5, 10, 20, 50, 100, 1000\}$. This results in 288 runs per data set, and therefore the overall meta data set contains up to $14,400$ instances.

Finally, we also added meta features to the meta data set, to allow the surrogate models to distinguish between the same hyperparameter configurations

[1] http://archive.ics.uci.edu/ml/index.html.
[2] http://www.multiboost.org.
[3] http://svmlight.joachims.org.

Table 1. List of all meta-features used.

Number of Classes	Log Inverse Data Set Dimensionality	Kurtosis Mean
Number of Instances	Class Cross Entropy	Kurtosis Standard Deviation
Log Number of Instances	Class Probability Min	Skewness Min
Number of Features	Class Probability Max	Skewness Max
Log Number of Features	Class Probability Mean	Skewness Mean
Data Set Dimensionality	Class Probability Standard Deviation	Skewness Standard Deviation
Log Data Set Dimensionality	Kurtosis Min	
Inverse Data Set Dimensionality	Kurtosis Max	

evaluated on different data sets. A list of all employed meta features can be seen in Table 1. For our experiments, all features in the meta data set, namely the computed meta features as well as the hyperparameter configurations have been scaled to values in $[0, 1]$.

4.2 Competing Surrogate Models

Random Search (RANDOM). This is a surrogate that simply picks a random point out of the grid.

Random Forests (RF). Sequential Model-based Algorithm Configuration [13] employs a random forest as surrogate model and computes uncertainties using the learned ensemble by estimating empirical means and standard deviations.

Independent Gaussian Process (IGP). An independent Gaussian process with SE-ARD kernel that is only learned on the observations on the target data set, this was proposed by [20].

Surrogate-based Collaborative Tuning (SCOT). This surrogate model is effectively a stacking of an SVM^{RANK} and a Gaussian process and was proposed by [2]. The ranking SVM learns how to rank hyperparameter configurations across data sets, uncertainties are estimated by stacking a GP on the ranked output.

Full Gaussian Process (FGP). A Gaussian process with SE-ARD kernel that is learned on the whole meta data set. This is basically the model we seek to approximate by learning a product of experts.

Gaussian Process with MKL (MKLGP). This surrogate was proposed by [26] and learns basically a full GP over the whole meta data set using a combination of an SE-ARD kernel and a kernel function that models the distances between data sets based on meta features.

Factorized Multilayer Perceptron (FMLP). The surrogate model that was proposed by [18], which learns a multilayer perceptron and factorizes the weights in the first layer in order to learn latent data set and hyperparameter representations. Uncertainties are estimated by learning an ensemble of FMLPs.

Product of Gaussian Process Experts (POGPE). This surrogate model learns a product of GP experts as described in Sect. 3.4. Each expert employs an SE-ARD kernel. Information of the target data set is distributed to all experts, which are all weighted equally.

Single Gaussian Process Expert (SGPE). This surrogate model also learns a product of GP experts as described in Sect. 3.4, however, also learns an independent GP for the target data set only and weighs the target GP as much as the whole set of experts.

4.3 Experimental Setup

Our experiments are performed in a leave-one-out fashion, meaning that we train the surrogate model on 49 data sets and use the meta knowledge to start SMBO on the remaining test data set. To cancel out random effects, we ran all experiments for a total of 100 times and averaged the results in the end. In total, each SMBO run was allowed to test $T = 70$ different hyperparameter configurations on the test data. As acquisition function we employed the popular expected improvement, which is by now the most widely used acquisition function in hyperparameter optimization using the SMBO framework.

As evaluation metric, we use the average rank, where, for each target data set, we rank all competing surrogate models based on the best misclassification rate they have found so far. Ties are being solved by granting the average rank, i.e. if one surrogate models find the misclassification rates 0.2, another two find 0.25 and a third one finds only 0.5, we would rank the surrogates with 1, 2.5, 2.5 and 4. As we run the experiments for 50 different target data sets, we report the average of all average ranks.

The implementations were largely done by ourselves, except for SMAC and SCOT, where we used MLTK[4] for the former and the implementation by Joachims[5] for the ranking SVM used in SCOT. All hyperparameters of the GP based models have been automatically tuned by maximizing their marginal likelihood, for FMLP we used the setting proposed by the authors. For SMAC, SCOT and MKLGP we used leave-one-out cross validation to tune the hyperparameters. For all GP-based models, we implemented the Cholesky decomposition to speed up the inversion of kernel matrices. In order to facilitate reproducibility of our experimental results, we make the program code as well as the employed meta data sets publicly available on Github[6].

[4] http://www.cs.cornell.edu/~yinlou/projects/mltk/.
[5] http://svmlight.joachims.org/.
[6] https://github.com/nicoschilling/ECML2016.

4.4 Performance in SMBO

The average rank among all competing methods can be seen in Fig. 1, where the left plot shows the average rank on AdaBoost versus the number of trials conducted, and the right one shows the results for the SVM meta data set. First of all, we see that for both meta data sets the random baseline shows the worst performance as is expected. Surprisingly, for both meta data sets, POGPE and SGPE find the best hyperparameter among the competitors in the first trial. During the SMBO procedure, both the full Gaussian process as well as MKLGP perform better on the AdaBoost data set, however, we observe that POGPE performs better on the SVM data set which is quite a surprise. POGPE, despite its good starting point, is being outperformed by FMLP on the SVM data set in the first 15 trials, which then degrades in performance and performs worse than both full GP approaches. Comparing both of these with each other, we see that they perform almost equally, however, MKLGP tends to have worse starting points than a simple full GP. For both meta data sets, we see the lift of including meta knowledge through comparison with the independent GP, which performs reasonably on AdaBoost but degrades on SVM. This observation leads us to the conclusion that optimizing the hyperparameters of AdaBoost seems an easier task than on SVM.

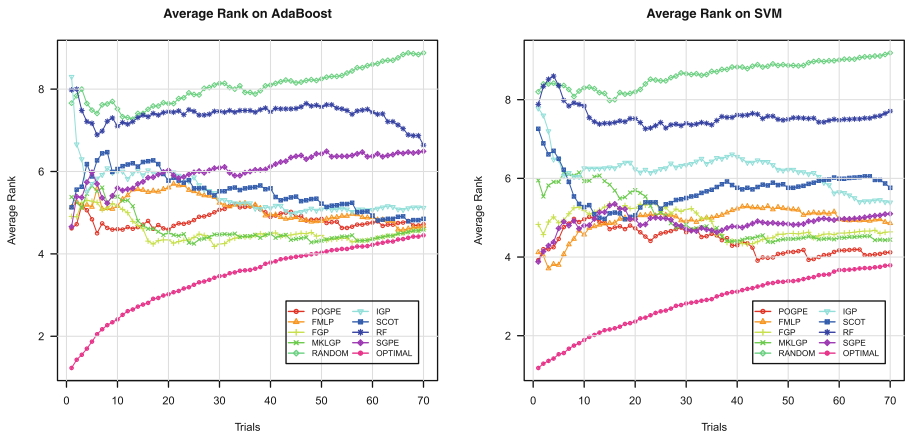

Fig. 1. Average Rank of all competing methods. The left plot shows results for AdaBoost, the right plot shows results for the SVM meta data set.

In contrast to POGPE, SGPE does not seem to perform that well, maybe the tradeoff between product of experts and single GP has to be adjusted for each trial, however, this would introduce another hyperparameter for the surrogate model, which we do not seek to do.

Overall, we conclude that POGPE, FMLP, FGP and MKLGP are among the best performing surrogate models, so for these models we also computed the

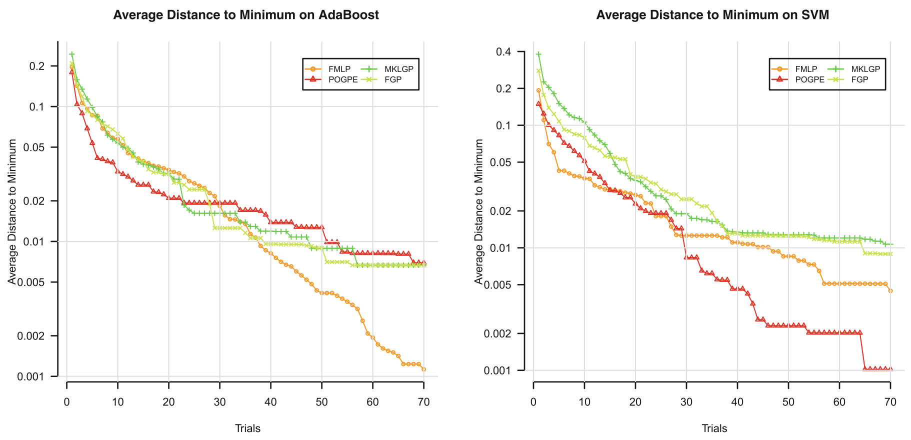

Fig. 2. Average Distance to Minimum of the best performing methods. The left plot shows results for AdaBoost, the right plot shows results for the SVM meta data set.

average distance to the minimum in terms of b. For each data set, we scale all values of b (in this case accuracies) to be in $[0, 1]$. Then, again for each data set, we compute the distance of the best hyperparameter response so far to the best on the overall grid. This value is then averaged to become the average distance to the minimum, which gives an idea of how a surrogate model makes use of the responses it gets on the target data set. The results can be seen in Fig. 2, where the left plot shows the results for AdaBoost and the right for SVM. Overall, we see the same behaviour as we have seen in average rank, however, FMLP seems to be a little bit better here. For AdaBoost, it achieves the lowest average distance, this is due to FMLP winning severely against its competitors on the data set *sonar-scale*, where in the average rank, this win does not count that much. In conclusion, we see that POGPE works really well on both evaluatuion metrics, especially when we consider its simplicity in the light of POGPE actually being an approximation of a full GP.

4.5 Runtime Experiments

In order to demonstrate the scalability of using POGPE, we have also conducted a runtime experiment. We have measured the runtime of the most competitive methods, namely being POGPE, FMLP, and both of the full Gaussian process approaches FGP and MKLGP. Experiments were conducted on a Xeon E5-2670v2 with 2.50 GHz clock speed and 64 GB of RAM, where we again performed a total of 70 trials of an SMBO run on the SVM meta data set. To account for measurement noise, we repeated all experiments 10 times.

The results can be seen in Fig. 3, where the left plot shows the cumulative runtime in seconds without the initial training time of the surrogate in opposition to the right plot which includes it. We plot both results as the training of the surrogate model can be performed in an offline fashion while waiting for new

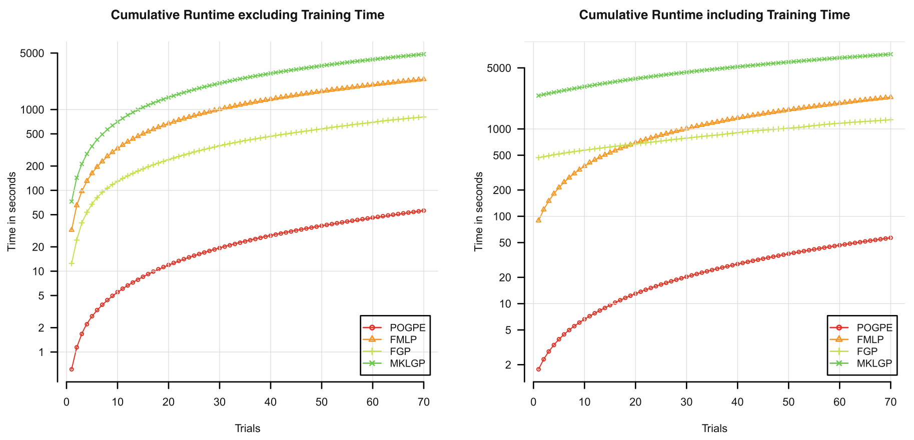

Fig. 3. Runtime comparison among the most competitive surrogate models. The left plot shows the cumulative runtime in seconds.

data. As we do not take into account the learning of the actual model, i.e. the evaluation of b, both plots can be understood as the total overhead time of running SMBO instead of using default hyperparameters. We can observe that POGPE consumes drastically less time than all its competitors, simply due to the fact that we have to invert much smaller kernel matrices. By excluding the potentially offline training time, a full GP is faster than FMLP, however, if the full GP needs to be trained first an FMLP is faster but gets overtaken with respect to computation time if only enough trials are performed. In both plots, MKLGP requires the most computation time. Considering that these differences will be bigger if we use more meta information, we conclude that POGPE is very fast while performing also very well in the SMBO procedure.

5 Conclusions

In this paper, we proposed to choose POE models as surrogate models for hyperparameter tuning, specifically we chose to employ Gaussian processes because of their fairly easy implementation as well as their predictive performance in the field of Bayesian optimizazion. We do acknowledge that POGPE is not the best model in all experiments, but is quite competitive which is a surprise due to its simplicity and its approximative nature. In the very first trial both POGPE and SGPE (as they start out the same) on average pick the best hyperparameter configuration compared to all competitor methods, which shows how efficient usage of the meta data can simply be made by learning a product of experts on each invidual data set and querying the committee. Moreover, the other competitive surrogate models such as FMLP and MKLGP introduce additional hyperparameters for the surrogate model that need to be optimized. For FMLP, tuning of the network architecture such as number of layers and number of nodes per layer as well as setting correct learning rates is demanded. MKLGP requires tuning of the number of neighboring data sets and the tradeoff term between both employed kernels. In comparison, a simple product of GP experts does not require any hyperparameter tuning, as the GP parameters can be learned by maximizing their marginal likelihood quite effectively.

As we have seen in the results, POGPE can also be trained much faster than the other competitive surrogate models. In the light of big data we will probably have access to also growing meta data sets that we can employ for hyperparameter optimization, which makes scalable use of the meta data a necessity. Moreover, POE models are easy to parallelize which allows easy usage in distributed scenarios. Out of all these reasons we see them as a very reasonable choice to pick as surrogate models for hyperparameter optimization including large scale meta data.

Acknowledgments. The authors gratefully acknowledge the co-funding of their work by the German Research Foundation (DFG) under grant SCHM 2583/6-1.

References

1. Adankon, M.M., Cheriet, M.: Model selection for the LS-SVM. Appl. Handwriting Recogn. Pattern Recognit. **42**(12), 3264–3270 (2009)
2. Bardenet, R., Brendel, M., Kegl, B., Sebag, M.: Collaborative hyperparameter tuning. In: Dasgupta, S., Mcallester, D. (eds.) Proceedings of the 30th International Conference on Machine Learning (ICML-13). vol. 28, pp. 199–207. JMLR Workshop and Conference Proceedings, May 2013
3. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. **13**, 281–305 (2012)
4. Brazdil, P., Carrier, C.G., Soares, C., Vilalta, R.: Metalearning: Applications to data mining. Springer, Heidelberg (2008)
5. Cao, Y., Fleet, D.J.: Generalized product of experts for automatic and principled fusion of gaussian process predictions. arXiv preprint (2014). arXiv:1410.7827
6. Deisenroth, M.P., Ng, J.W.: Distributed gaussian processes. Int. Conf. Mach. Learn. (ICML) **2**, 5 (2015)
7. Escalante, H.J., Montes, M., Sucar, L.E.: Particle swarm model selection. J. Mach. Learn. Res. **10**, 405–440 (2009)
8. Feurer, M., Springenberg, J.T., Hutter, F.: Initializing bayesian hyperparameter optimization via meta-learning. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (2015)
9. Guo, X.C., Yang, J.H., Wu, C.G., Wang, C.Y., Liang, Y.C.: A novel ls-svms hyperparameter selection based on particle swarm optimization. Neurocomput **71**(16–18), 3211–3215 (2008)
10. Hensman, J., Fusi, N., Lawrence, N.D.: Gaussian processes for big data. arXiv preprint (2013). arXiv:1309.6835
11. Hinton, G.E.: Products of experts. In: Ninth International Conference on (Conf. Publ. No. 470) Artificial Neural Networks, ICANN 99. vol. 1, pp. 1–6. IET (1999)
12. Hinton, G.E.: Training products of experts by minimizing contrastive divergence. Neural Comput. **14**(8), 1771–1800 (2002)

13. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Dhaenens, C., Jourdan, L., Marmion, M.-E. (eds.) LION 2015. LNCS, vol. 8994, pp. 507–523. Springer, Heidelberg (2011). doi:10.1007/978-3-642-25566-3_40
14. Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black-box functions. J. Global Optim. **13**(4), 455–492 (1998)
15. Koch, P., Bischl, B., Flasch, O., Bartz-Beielstein, T., Weihs, C., Konen, W.: Tuning and evolution of support vector kernels. Evol. Intell. **5**(3), 153–170 (2012)
16. Quinonero-Candela, J., Rasmussen, C.E.: A unifying view of sparse approximate gaussian process regression. J. Mach. Learn. Res. **6**, 1939–1959 (2005)
17. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning). The MIT Press (2005)
18. Schilling, N., Wistuba, M., Drumond, L., Schmidt-Thieme, L.: Hyperparameter optimization with factorized multilayer perceptrons. In: Machine Learning and Knowledge Discovery in Databases, pp. 87–103. Springer, Heidelberg (2015)
19. Shen, Y., Ng, A., Seeger, M.: Fast gaussian process regression using kd-trees. In: Proceedings of the 19th Annual Conference on Neural Information Processing Systems. No. EPFL-CONF-161316 (2006)
20. Snoek, J., Larochelle, H., Adams, R.P.: Practical bayesian optimization of machine learning algorithms. In: Pereira, F., Burges, C., Bottou, L., Weinberger, K. (eds.) Advances in Neural Information Processing Systems 25, pp. 2951–2959. Curran Associates, Inc. (2012)
21. Swersky, K., Snoek, J., Adams, R.P.: Multi-task bayesian optimization. In: Burges, C., Bottou, L., Welling, M., Ghahramani, Z., Weinberger, K. (eds.) Advances in Neural Information Processing Systems 26, pp. 2004–2012. Curran Associates, Inc. (2013)
22. Tresp, V.: A bayesian committee machine. Neural Comput. **12**(11), 2719–2741 (2000)
23. Williams, C., Seeger, M.: Using the nyström method to speed up kernel machines. In: Proceedings of the 14th Annual Conference on Neural Information Processing Systems, pp. 682–688. No. EPFL-CONF-161322 (2001)
24. Wistuba, M., Schilling, N., Schmidt-Thieme, L.: Learning hyperparameter optimization initializations. In: IEEE International Conference on Data Science and Advanced Analytics (DSAA), 36678 2015, pp. 1–10. IEEE (2015)
25. Wistuba, M., Schilling, N., Schmidt-Thieme, L.: Sequential model-free hyperparameter tuning. In: 2015 IEEE International Conference on Data Mining (ICDM), pp. 1033–1038. IEEE (2015)
26. Yogatama, D., Mann, G.: Efficient transfer learning method for automatic hyperparameter tuning. In: International Conference on Artificial Intelligence and Statistics (AISTATS 2014) (2014)