Hyperparameter Optimization Across Problem Tasks

Nicolas Schilling, Torben Windler and Lars Schmidt-Thieme

Abstract Hyperparameter Optimization is a task that is generally hard to accomplish as the correct setting of hyperparameters cannot be learned from the data directly. However, finding the right hyperparameters is necessary as the performance on test data can differ a lot under various hyperparameter settings. Many researchers rely on search techniques such as grid-search, having the downside that they require a lot of computation time, as prediction models are learned for a wide range of possible hyperparameter configurations which is only feasible in a parallel computing environment. Recently, search methods based on Bayesian optimization such as SMAC have been proposed and extended to include hyperparameter performance of the same model on another data set. These meta learning approaches show that the search for well-performing hyperparameters can be steered in a more intelligent manner. In this work, we aim to accomplish hyperparameter optimization across problem tasks where

Nicolas Schilling · Lars Schmidt-Thieme
University of Hildesheim, Information Systems and Machine Learning Lab (ISMLL)
Universitätsplatz 1, 31141 Hildesheim, Germany
✉ nicolas.schilling@xing.com
✉ schmidt-thieme@ismll.de

Torben Windler
University of Hildesheim
Universitätsplatz 1, 31141 Hildesheim, Germany
✉ windler@uni-hildesheim.de

Archives of Data Science, Series A
(Online First)
KIT Scientific Publishing
Vol. 4, No. 1, 2018

DOI 10.5445/KSP/1000085951/14
ISSN 2363-9881

we specifically target regression and classification problems. We show, that the incorporation of hyperparameter performance on a classification task is helpful when optimizing hyperparameters for a regression task and vice versa.

1 Introduction

Hyperparameter Optimization is a task that has to be successfully accomplished whenever competitive results of a machine learning model are needed. For scientific publications hyperparameters are optimized across all competing models in order to ensure a fair comparison between all approaches. For companies working on a machine learning project, hyperparameter optimization is usually conducted to maximize prediction performance for the task at hand. Hyperparameter optimization usually renders a model from delivering poor results to state-of-the art performance and is therefore inevitable.

Many researchers bank on using grid-search to optimize hyperparameters where a large set of hyperparameter configurations is simply being tested and the best performing configuration is used to predict for the test data. Being easily parallelizable, grid-search is simple to implement but requires a parallel computing environment such as a compute cluster to be scalable, especially when the amount of hyperparameters to tune is large, as the number of computations grows exponentially. On the downside, grid-search computations are independent of each other and knowledge about well performing hyperparameter configurations is not used. Ultimately, grid-search is a technique that is simple to implement, but scales very poorly and therefore interferes with the goal of making machine learning techniques available to a broad area of companies and organizations.

Recent research attempts to solve the problem of hyperparameter optimization by using approaches based on Bayesian optimization techniques. One instance of Bayesian optimization is surrogate model-based optimization (SMBO), where a surrogate model is learned on observed performances of a small set of hyperparameter configurations. Thereafter, the surrogate model can be queried for a large set of hyperparameter configurations allowing for a more principled search. After the evaluation is completed, the surrogate model is fitted to the newly observed performances to increase its prediction quality. The overall process is repeated until either a cost budget is exhausted, or a performance criterion on the hyperparameter configurations is met.

More recently, researchers have followed the idea of injecting meta knowledge into the learning process of the surrogate model. This is accomplished by learning the surrogate model on a large set of hyperparameter performances of the same model on different data sets, where hyperparameter optimization has been completed already. This approach has shown to work better than learning the surrogate model from scratch for every new data set.

In this paper, we add another dimension to the metalearning aspect of recent work, by learning hyperparameter performance not only across data sets, but also across problem tasks. Specifically, we chose regression, binary and n-ary classification, as we need a model that is capable of handling all tasks, we optimize the hyperparameters of feedforward neural networks. The only difference between the tasks is the activation function in the output layer, all other hyperparameters are shared across all tasks and, therefore, a knowledge transfer between tasks seems possible. On a large meta data set of observed performances on 70 data sets, we are able to show that the current state of the art surrogate models are capable of learning across tasks.

2 Related work

In the field of SMBO-based hyperparameter optimization, there are several publications who propose different surrogate models. The work by Bergstra and Bengio (2012) suggests to sample hyperparameters from random variables instead of a grid-search. The authors of Snoek et al (2012) propose to use Gaussian processes (GP) as surrogate models, as they naturally predict a distribution over target values. The work by Hutter et al (2011) suggests random forests as surrogate as they can naturally deal with hierarchical hyperparameters. The first publication to include hyperparameter performance on other data sets is Bardenet et al (2013) which learns a ranking SVM to select hyperparameters. A variant of a neural network that uses a factorization machine as signal function in the first layer of the network was proposed by Schilling et al (2015). The factorization machine allows the model to estimate latent features for binary indicator variables. Finally, Schilling et al (2016) proposes to use a product of GP experts, as GPs are in general a decent surrogate, but do not scale to large amounts of meta data.

In another direction, there are works that try to initialize the hyperparameter search in a principled manner such as Feurer et al (2015), Wistuba et al (2015), that answer the question of which hyperparameter configurations to test initially. Additionally, there are approaches that use genetic algorithms as for example Reif et al (2012) and Koch et al (2012) or take learning curves into account as van Rijn et al (2015). However, as these approaches do not propose surrogates for SMBO-based hyperparameter optimization, we will restrict our experimental section on comparing the former group of publications on our meta data.

3 SMBO-Based Hyperparameter Optimization

In this section, we will first introduce the problem of hyperparameter optimization in general and subsequently show how it is approached using sequential model-based optimization.

3.1 Hyperparameter Optimization

Let us by $\mathcal{D}$ denote the space of all data sets for both our targeted tasks of regression and classification. Additionally, denote by $\mathcal{M}$ the space of all feedforward neural networks, essentially this is the space of all DAGs associated with parameters for each of the graph's edges. The space of all possible hyperparameter configurations is denoted by Λ, which for example specifies the structure of the network to be learned. We call a learning algorithm $\mathcal{A}$ a mapping $\mathcal{A} : \mathcal{D} \times \Lambda \longrightarrow \mathcal{M}$, that takes as input a data set $D \in \mathcal{D}$ and a hyperparameter configuration $\lambda \in \Lambda$ to then estimate a model $M \in \mathcal{M}$. For many machine learning problems, $\mathcal{A}$ searches through the model space and returns a model by minimizing the regularized empirical loss over training partition D^{train} of the data set D given the model M

$$\mathcal{A}(D, \lambda) = \underset{M \in \mathcal{M}}{\arg\min}\ \mathcal{L}(M, D^{\text{train}}) + \mathcal{R}(M). \tag{1}$$

With these expressions, we can easily define the problem of *hyperparameter optimization* as finding the hyperparameter configuration $\lambda^\star$, where the associated model minimizes the loss on the validation partition

$$\lambda^\star \in \underset{\lambda \in \Lambda}{\arg\min}\ \mathcal{L}(\mathcal{A}(D, \lambda), D^{\text{valid}}) =: \underset{\lambda \in \Lambda}{\arg\min}\ b(\lambda, D), \tag{2}$$

where we use the short form of $b(\lambda, D)$ for the function we seek to optimize. While the loss function $\mathcal{L}$ usually has an analytical representation, the evaluation of the final model parameters returned by the learning algorithm $\mathcal{A}$ itself is a black box function, which makes finding $\lambda^\star$ using standard optimization techniques such as gradient descent impossible. Additionally, not every hyperparameter dimension of Λ is represented by continuous values, as for example the number of latent features in a factorization model or the number of layers and neurons in neural networks are discrete hyperparameters, also rendering continuous optimization techniques inapplicable. For this reason, black box optimization techniques such as SMBO have been employed to solve the hyperparameter optimization problem.

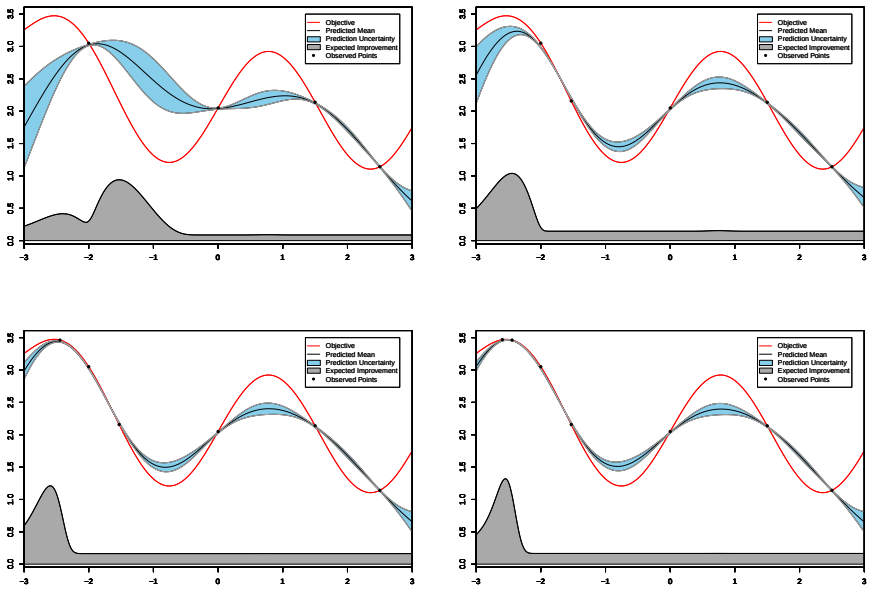

Figure 1: Four consecutive SMBO iterations using a Gaussian process as surrogate model and expected improvement as acquisition function. The x-axis describes a one-dimensional hyperparameter while the y-axis is the resulting validation performance, with fictional values. In the first trial, the expected improvement chooses a point with high uncertainty, after having observed this configuration the exploration is stopped and the true objective is maximized.

3.2 Sequential Model-Based Optimization

SMBO starts by evaluating a set of initial observations of b, precisely a few hyperparameter configurations are chosen, evaluated and then fed into an observation history

$$\mathcal{H}(D) = \{(\lambda_1, b(\lambda_1, D), ..., (\lambda_I, b(\lambda_I, D)\}. \tag{3}$$

This observation history is used as training data for the surrogate model $\Psi(\lambda)$. After learning Ψ, it is queried for a range of possible hyperparameter configurations where its prediction is taken into account and its uncertainty as well. Then, the hyperparameter configuration that maximizes the acquisition function is chosen, the configuration is evaluated by running $\mathcal{A}$ and then fed back into the observation history $\mathcal{H}$. At this point, a new iteration of SMBO is started, where Ψ is fitted to the updated observation history and queried again, until some well-performing configuration is found or the time budget is consumed. An example of four SMBO iterations using a Gaussian process surrogate on some toy data is given in Figure 1.

The role of the acquisition function a is to find a decent tradeoff between exploration of Λ and exploitation of the knowledge of Ψ. Always evaluating the hyperparameter configuration where Ψ is maximized would highly exploit it, however, this leads to low exploration of Λ, as the maximum is usually in the vicinity of already observed configurations. On the other hand, one could always choose the λ where the uncertainty is maximal, this leads to high exploration of Λ, but may end up finding only mediocre hyperparameter configurations. In many works the expected improvement (EI) is used as acquisition function (Jones et al, 1998), and is defined as the expected value of the improvement function. If we assume the probability for improvement to follow a Gaussian distribution centered around the predicted mean $\mu(\lambda)$ and having variance equal to the predicted variance $\sigma(\lambda)$ by the surrogate, then we can compute EI analytically using

$$EI(\lambda) = \begin{cases} \left(b(\lambda^{\text{best}}) - \mu(\lambda)\right) \Phi(Z) + \sigma(\lambda)\, \phi(Z) & \text{if } \sigma(\lambda) > 0 \\ 0 & \text{if } \sigma(\lambda) = 0 \end{cases}, \tag{4}$$

$$Z = \frac{b(\lambda^{\text{best}}) - \mu(\lambda)}{\sigma(\lambda)}, \tag{5}$$

where $\Phi(Z)$ is the cumulative distribution function and $\phi(Z)$ the probability density function of a standard Gaussian.

4 Creation of the Meta Data

As the goal is to optimize hyperparameters across the tasks of regression, binary and n-ary classification, we have to chose a model that is capable of handling all tasks well, in this section we introduce feedforward neural networks as they are able to solve all tasks. An artificial feedforward neural network is a machine learning model that has a structure of a directed acyclic graph, where we restrict the model space in two ways. At first, we do not allow connections to skip a layer in our formulation, secondly we assume that the network is always fully connected, meaning that each hidden neuron receives an input of each other hidden neuron in the previous layer of the network. With these restrictions, the state of the l-th hidden layer h_l in a neural network with L many layers can be written as

$$h_l = \sigma_l(W_l^\top h_{l-1} + b_l) \qquad \forall l \in 1, ..., L, \tag{6}$$

where $W_l \in \mathbb{R}^{n_{l-1} \times n_l}$ is the weight matrix connecting each neuron of layer $l-1$ with each neuron in the next layer, $b_l \in R^{n_l}$ is a bias vector and σ_l is an element-wise applied nonlinear activation function. The input layer is simply the input $x \in R^{n_0}$ that we want to make predictions for, i.e. $h_0 = x$ and the output of the L-th layer will be our prediction

$$\hat{y}(x) = h_L = \sigma_L(W_L^\top h_{L-1} + b_L), \tag{7}$$

where σ_L is chosen depending on the task. For regression, σ_L is simply the identity function, for binary classification it is a sigmoid function and for multiclass classification problems the softmax activation function is used.

4.1 Hyperparameter in Neural Networks

Neural networks can only be learned if a set of hyperparameters has been specified prior to training, we will now discuss which parameters we treat as hyperparameters and then discuss the creation of the meta data. The hyperparameters of feedforward neural networks can be grouped into three different classes:

1. **Structure-based hyperparameters**:
   The number of layers L as well as the number of neurons n_l for all layers are hyperparameters that determine the structure of the model to be learned. We make a slight simplificiation by assuming that n_l

amounts to the same value for all layers. Additionally, the activation function being used is a hyperparameter, where one can choose between a sigmoid function, tanh or activation functions based on rectified linear units (ReLU). We again slightly simplify this choice by assuming that σ_l is equal in all layers, except for the output layer, where the activation function is defined by the task to be solved.

2. **Optimization-based hyperparameters**:
   The choice of the optimization algorithm is a hyperparameter that crucially decides whether predictions are useful or not. Usually, one chooses between plain SGD (Bottou, 2010) or more sophisticated techniques such as AdaGrad, AdaDelta, RMSProp, ADAM, NADAM and the likes (Duchi et al, 2011; Zeiler, 2012; Tieleman and Hinton, 2012; Kingma and Ba, 2014; Dozat, 2016). Some of these optimizers adjust learning rates and momentums automatically, where only initial values need to be given. In addition to that, also the mini batch size is an optimization-based hyperparameter.

3. **Regularization-based hyperparameters**: In order to avoid overfitting, neural networks are usually regularized, this may be through L1 or L2 regularization, with associated regularization constant, by dropout regularization with the associated dropout percentage or by other techniques as early stopping (Prechelt, 1998).

The hyperparameter configurations that have been used in our meta data are summarized in Table 1. Overall, we end up with 2916 experiments per data set using these hyperparameter configurations. In order to obtain the ground truth hyperparameter performance, we have performed a grid search on 70 data sets that have been downloaded mainly from the UCI machine learning repository (provided on `https://archive.ics.uci.edu/ml/index.php` by the University of California, Irvine (USA)). Among these data sets we have 44 binary classification problems, 12 multiclass classification problems and 14 regression problems, in total the meta data consists of $204,120$ (70 datasets $\times$ 2,916 hyperparameters) experiments. To make these experiments comparable between each other, we have to bring RMSE values on the scale of accuracy values, which is the interval $[0,1]$. In order to do so, for each data set we scale the final RMSE values between 0 and 1 where we assign 1 for the lowest RMSE and 0 for the highest RMSE.

For RMSE, this is accomplished by computing:

$$b^{\text{scaled}}(\lambda) = 1 - \frac{b(\lambda) - \min_\lambda b(\lambda)}{\max_\lambda b(\lambda) - \min_\lambda b(\lambda)} \tag{8}$$

For the classification problems, we simply compute the value that we would subtract from 1 in the above equation. Doing this assigns 1 to the maximum accuracy and 0 to the minimal accuracy. Scaling the labels is crucial as it allows the surrogate model to make a fair comparison between data sets and not be biased towards the easier problems.

Table 1: Overview of the hyperparameter grid used to create the meta data set.

Structure	**Values**		
Activation Function	ReLU	leakyReLU	tanh
Number of Layers	5	10	20
Number of Neurons	10	20	50
Optimization	**Values**		
Optimizer	Adam	AdaGrad	AdaDelta
Number of Epochs	10	100	
Regularization	**Values**		
Dropout	0	0.2	0.4
L_p Regularization	L_1	L_2	
Regularization Constant	0.01	0.001	0.0001

5 Experiments

In this section we will conduct two experiments, first we use the meta data of all problem tasks to learn the surrogate and therefore compare the overall performances of different surrogate models. In the second experiment we will evaluate the benefit of including meta data from other tasks, this is accomplished

by comparing the surrogate's performances in two variations. The first variant allows the surrogate to include meta data of other tasks, where in the second variant they are only learned on the meta data of the target task.

We start by describing both evaluation metrics used by us. The first evaluation metric is the average rank among all optimizers. This can be obtained by comparing for every trial the average performance of each surrogate and then computing a ranking between the surrogate models, where ties are solved by granting the average rank. If we, for example, have four surrogates that in trial 20 have achieved an average performance of 0.8, 0.7, 0.7 and 0.4 the average rank would be 1, 2.5, 2.5 and 4.

The second evaluation metric is the average hyperparameter rank achieved by the individual optimizers. This is computed by simply ranking all hyperparameter performances for a given data set and then, for each trial computing the average hyperparameter rank that has been achieved.

5.1 Using all Meta Data

For the first experiment we proceed as follows: For each surrogate and for each target data set, we learn the surrogate model on the observed performances on the 69 remaining training data sets and then perform 300 SMBO trials, thus allowing the surrogate to only evaluate a good tenth of the original hyperparameter grid. Overall, we compare six different surrogate models: At first, we use a random surrogate model called RANDOM, that randomly picks one hyperparameter without any knowledge as was proposed in Bergstra and Bengio (2012). Secondly, we learn a Gaussian Process only on the observed target performances, without any usage of the meta data, this is called SPEARMINT (Snoek et al, 2012). A third surrogate model that also does not take the meta data into account is SMAC (Hutter et al, 2011), which uses a random forest as surrogate. We augmented SMAC by including the meta-data and call the resulting surrogate SMAC++, this is the first surrogate that makes use of meta knowledge. Finally, we employ two surrogate models that are also able to learn hyperparameter performance across tasks, the first one being FMLP (Schilling et al, 2015), which uses a combination of a neural network and a factorization machine as surrogate model, finally, we use a product of GP experts (Schilling et al, 2016) called POGPE, where an independent GP is learned on each data set in the meta data, and these GPs are then combined to make predictions on the target data.

The average rank is shown in Figure 2, where clearly, we see that the random surrogate RANDOM performs bad. Both surrogates that are agnostic to the meta data also do not show a good performance, SMAC performs better than SPEARMINT in the beginning, but gets overtaken around trial 50. A difference because of including meta knowledge can be observed when comparing SMAC++ with SMAC, the former consistently shows better results. FMLP produces decent results, we highlight that for the very first trial - where no information of the target data has been gathered - FMLP is the best surrogate model, but it quickly loses agains POGPE, which shows the best performance of all competitors.

The average hyperparameter rank can be seen in Figure 3 and shows a similar behaviour. Again, the surrogate models that do not include meta data are showing a much worse performance than the ones that do, please note that the y-axis is plotted in log-scale. While these surrogates start at an average hyperparameter rank of 2000, FMLP and POGPE start somewhere around 400 and reach an average hyperparameter rank of 20 already after having conducted trial 35.

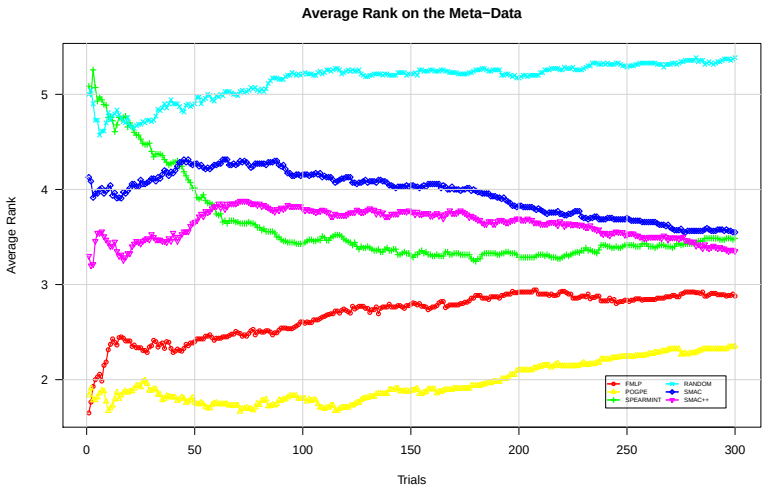

Figure 2: Average Rank of all competing surrogates using all Meta Data.

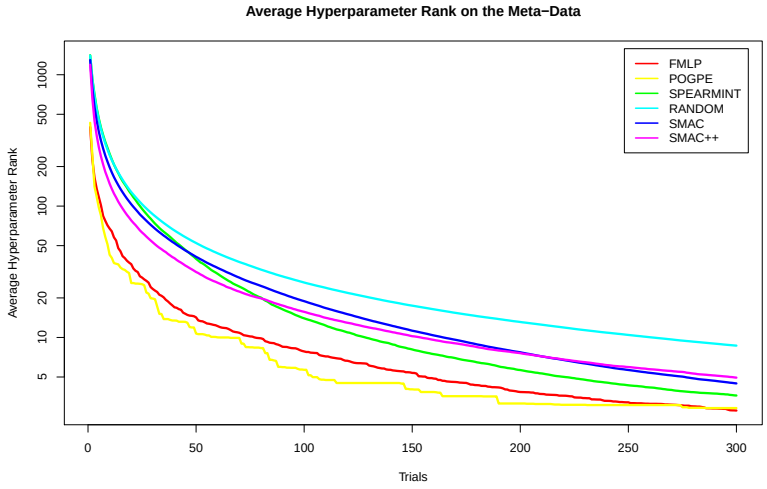

Figure 3: Average Hyperparameter Rank of all competing surrogates using all Meta Data.

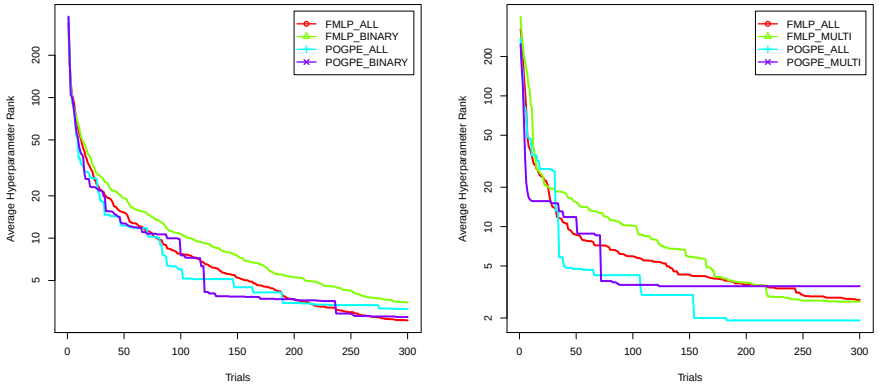

Figure 4: Average hyperparameter rank for binary and multiclass classification tasks.

5.2 Using Task-Specific Meta Data

Now we would like to assess the performance gain that is achieved by additional meta data. For each target task, we show the average hyperparameter rank where hyperparameters are only optimized for all data sets belonging to the target task in Figure 4 and 5. For both FMLP and POGPE, we compare two variants, one where we use all meta data to learn the surrogate and the second variant where we learn the surrogate only on the observed performances of data sets from the target task.

Figure 4 shows results for the targets tasks of binary and multiclass classification. We see that for both tasks, adding meta data of the other two tasks increases the performance, however, for POGPE the difference is quite small. For some of the trials the variant without the additional meta data even performs better. FMLP shows a more principled lift when including additional meta data.

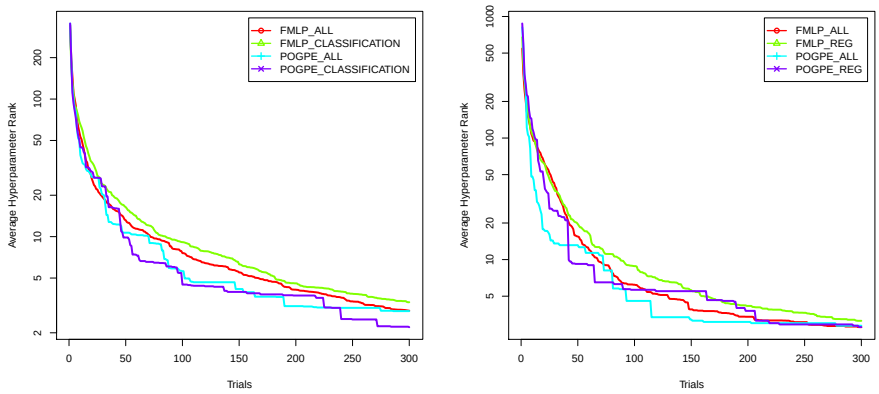

Figure 5: Average Hyperparameter Rank for the joint classification task and for regression.

As one may argue whether binary and multiclass classification are two different tasks, we have also merged both tasks to a joint classification task and evaluated if the additional meta knowledge from the regression problems is helpful. The results for the merged classification task can be seen in Figure 5 on the left plot. The difference between both variants is rather small, considering that the only difference is the 14 additional regression problems, this result is not surprising.

However, there still is a principled lift in the results of FMLP, showing that also little information can be used adequately. The results for regression as targeted problem are shown in the same figure on the right plot. Similar to the other experiments, we see a consistent lift for FMLP and POGPE, although the difference again is quite small for POGPE. Overall, we conclude that using additional meta data from other tasks for hyperparameter optimization improves the performance of the learned surrogate models.

6 Conclusion

In this paper, we have built a meta data set consisting of roughly 200,000 experiments solving the three tasks of binary and n-ary classification as well as regression for a range of data sets. We have chosen neural networks as prediction models, as they are able to solve a range of different tasks but still use almost the same hyperparameter space. Our results show that learning hyperparameter performance across problem tasks is possible and generally improves the performance. For future work, we seek to extend our meta data set firstly by adding more experiments, this can be accomplished by solving more data sets or by increasing the hyperparameters that are being optimized. Additionally, we seek to extend the meta data by adding additional tasks, one may for example add ranking problems or regression problems of certain types such as recommender systems.

References

Bardenet R, Brendel M, Kegl B, Sebag M (2013) Collaborative Hyperparameter Tuning. In: Proceedings of the 30th International Conference on Machine Learning (ICML-13), Proceedings of Machine Learning Research (PMLR), Dasgupta S, Mcallester D (eds), vol. 28, pp. 199–207.

Bergstra J, Bengio Y (2012) Random Search for Hyper-parameter Optimization. Journal of Machine Learning Research (JMLR) 13:281–305.

Bottou L (2010) Large-Scale Machine Learning with Stochastic Gradient Descent. In: Proceedings of COMPSTAT'2010, Physica-Verlag HD, pp. 177–186. DOI: 10.1007/978-3-7908-2604-3_16.

Dozat T (2016) Incorporating Nesterov Momentum into Adam. International Conference for Learning Representations (ICLR) 2016 Workshop. URL: https://openreview.net/forum?id=OM0jvwB8jIp57ZJjtNEZ.

Duchi J, Hazan E, Singer Y (2011) Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research 12:2121–2159.

Feurer M, Springenberg JT, Hutter F (2015) Initializing Bayesian Hyperparameter Optimization via Meta-Learning. Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, pp. 1128–1135.

Hutter F, Hoos HH, Leyton-Brown K (2011) Sequential Model-based Optimization for General Algorithm Configuration. In: Proceedings of the 5th International Conference on Learning and Intelligent Optimization (LION 2011), Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, vol. 6683, pp. 507–523. DOI: 10.1007/978-3-642-25566-3_40.

Jones DR, Schonlau M, Welch WJ (1998) Efficient Global Optimization of Expensive Black-Box Functions. Journal of Global Optimization 13(4):455–492, Kluwer Academic Publishers, Hingham (USA). DOI: 10.1023/A:1008306431147.

Kingma DP, Ba J (2014) ADAM: A Method for Stochastic Optimization. arXiv preprint (arXiv:1412.6980). URL: https://arxiv.org/abs/1412.6980.

Koch P, Bischl B, Flasch O, Bartz-Beielstein T, Weihs C, Konen W (2012) Tuning and Evolution of Support Vector Kernels. Evolutionary Intelligence 5(3):153–170, Springer. DOI: 10.1007/s12065-012-0073-8.

Prechelt L (1998) Early stopping – But When? In: Neural Networks: Tricks of the trade, Montavon G, Orr G, Müller K (eds), Lecture Notes in Computer Science, vol. 7700, Springer, pp. 55–69. DOI: 10.1007/978-3-642-35289-8_5.

Reif M, Shafait F, Dengel A (2012) Meta-Learning for Evolutionary Parameter Optimization of Classifiers. Machine Learning 87(3):357–380, Springer US. DOI: 10.1007/s10994-012-5286-7.

van Rijn JN, Abdulrahman SM, Brazdil P, Vanschoren J (2015) Fast Algorithm Selection Using Learning Curves. In: International Symposium on Intelligent Data Analysis (IDA 2015), Springer, Cham, Lecture Notes in Computer Science, vol. 9385, pp. 298–309. DOI: 10.1007/978-3-319-24465-5_26.

Schilling N, Wistuba M, Drumond L, Schmidt-Thieme L (2015) Hyperparameter Optimization with Factorized Multilayer Perceptrons. In: Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2015), Lecture Notes in Computer Science, vol. 9285, Springer, Cham, pp. 87–103. DOI: 10.1007/978-3-319-23525-7_6.

Schilling N, Wistuba M, Schmidt-Thieme L (2016) Scalable Hyperparameter Optimization with Products of Gaussian Process Experts. In: Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2016), Springer, Cham, Lecture Notes in Computer Science, vol. 9851, pp. 33–48. DOI: 10.1007/978-3-319-46128-1_3.

Snoek J, Larochelle H, Adams RP (2012) Practical Bayesian Optimization of Machine Learning Algorithms. In: Advances in Neural Information Processing Systems 25, Curran Associates, Inc., pp. 2951–2959.

Tieleman T, Hinton G (2012) Lecture 6.5-rmsprop: Divide the Gradient by a Running Average of its Recent Magnitude. Neural Networks for Machine Learning 4(2):26–31, COURSERA.

UCI Machine Learning Repository (2018) Datasets. URL: https://archive.ics.uci.edu/ml/datasets.php.

Wistuba M, Schilling N, Schmidt-Thieme L (2015) Learning Data Set Similarities for Hyperparameter Optimization Initializations. In: Proceedings of the 2015 International Conference on Meta-Learning and Algorithm Selection (MetaSel'15), CEUR-WS.org, Aachen (Germany), vol. 1455, pp. 15–26.

Zeiler MD (2012) ADADELTA: An Adaptive Learning Rate Method. arXiv preprint (arXiv:1212.5701). URL: https://arxiv.org/abs/1212.5701.

Appendix

Table 2: Dataset overview (1/3), mainly provided via the UCI machine learning repository (dataset names are searchable on UCI Machine Learning Repository (2018)), except for the energy dataset (being a private dataset and not available via the repository).

Name	#Classes	#Features	#Instances
Regression			
Airfoil-self-noise		6	1,503
bodyfat		15	252
cadata		8	20,640
CASP		9	45,730
cpusmall		12	8,192
energy (private dataset, not via UCI)		102	50,088
eunite2001		16	336
housing		13	506
mg		6	1,385
mpg		7	392
pyrim		27	74
slice-localization		386	53,500
space-ga		6	3,107
triazines		60	186
Binary Classification			
A1A	2	123	30,956
A9A	2	123	16,281
appendicitis	2	7	106
australian	2	14	690
banana	2	2	5,300
bands	2	19	539
bupa	2	6	345
census	2	41	142,521
chess	2	36	3,196
cod-rna	2	9	488,565
coil2000	2	85	9,822
covtype-binary	2	54	581,012
credit-a	2	16	690
credit-g	2	21	1,000

Table 3: Dataset overview (2/3), provided via the UCI machine learning repository (dataset names are searchable on UCI Machine Learning Repository (2018)).

Name	#Classes	#Features	#Instances
crx	2	15	653
diabetes	2	8	768
german-numer	2	24	1,000
heart	2	13	270
hepatitis	2	19	155
housevotes	2	16	435
ijcnn1	2	22	91,701
ionosphere	2	34	351
kr-vs-kp	2	6	28,056
liver-disorders	2	5	200
magic	2	10	19,020
mammographic	2	5	961
mushrooms	2	112	8,124
phoneme	2	5	5,404
pima	2	8	768
real-sim	2	20,959	72,309
ring	2	20	7,400
saheart	2	9	462
sonar	2	60	208
spambase	2	57	4,597
spectfheart	2	44	267
splice	2	60	2,175
svmguide1	2	4	4,000
svmguide3	2	21	1,243
tic-tac-toe	2	9	958
twonorm	2	20	7,400
W1A	2	300	47,272
W8A	2	300	49,749
wdbc	2	30	569
wisconsin	2	9	699

Table 4: Dataset overview (3/3), provided via the UCI machine learning repository (dataset names are searchable on UCI Machine Learning Repository (2018)).

Name	#Classes	#Features	#Instances
Multi-Class Classification			
covtype	7	54	581,012
glass	6	9	216
mnist	10	780	60,000
poker	10	10	1,000,000
satimage	6	36	4,435
SensIT Vehicle (combined)	3	100	78,823
sensorless	11	48	58,509
shuttle	7	9	43,500
svmguide2	3	20	391
svmguide4	6	10	312
usps	10	256	7,291
vowel	11	10	528