

Learning Multiple Defaults for Machine Learning Algorithms

Florian Pfisterer* and Jan N. van Rijn† and Philipp Probst* and Andreas Müller† and Bernd Bischl*

*Ludwig Maximilian University of Munich, Germany

†Columbia University, New York, U.S.A.

Abstract

The performance of modern machine learning methods highly depends on their *hyperparameter configurations*. One simple way of selecting a configuration is to use *default settings*, often proposed along with the publication and implementation of a new algorithm. Those default values are usually chosen in an ad-hoc manner to work *good enough* on a wide variety of datasets. To address this problem, different automatic hyperparameter configuration algorithms have been proposed, which select an optimal configuration per dataset. This principled approach usually improves performance, but adds additional algorithmic complexity and computational costs to the training procedure. As an alternative to this, we propose learning a set of complementary default values from a large database of prior empirical results. Selecting an appropriate configuration on a new dataset then requires only a simple, efficient and embarrassingly parallel search over this set. We demonstrate the effectiveness and efficiency of the approach we propose in comparison to random search and Bayesian Optimization.

Introduction

parameters at all can be detrimental, defaults provide a fallback for cases, where no additional knowledge is available. Wistuba, Schilling, and Schmidt-Thieme (2015b) proposed to extend the notion of pre-specified defaults to ordered sets of defaults, combining the prior knowledge encoded in default values with the flexibility of optimization procedures. This work directly builds upon this notion. Our ordered sets of defaults are diverse lists of parameter settings for a particular algorithm, ordered by their performance across datasets. This can be seen as an extension of the classical exhaustive grid-search: Instead of searching all possible combinations in the grid, we keep only those configurations that historically (on a collection of benchmark datasets) performed well. Given that we eliminate most candidates using prior data, we can then afford to start with a very fine grid, approximating the results of a continuous optimization procedure.

A different perspective on multiple defaults is as a special case of meta-learning: we build a model using a collection of benchmark datasets, that allows us to predict good candidate parameters for a new dataset. Only we do not use any properties of the new dataset, and always predict the same ordered set of candidates

Selection of hyperparameter setting used until now:

- ① software defaults
- ② manual configuration
- ③ tuning procedure

Selection of hyperparameter setting used until now:

- ① software defaults
- ② manual configuration
- ③ tuning procedure

Problem: using historic results we try to find fixed size set Θ_{def} defaults which will be complementary and **together** perform well on new datasets.

Multiple defaults have several benefits: ease of use, strong anytime performance, parallel.

Goals of paper:

- description of two methods to acquire a list of defaults,
- comparison of these methods,
- benchmark to evaluate methods of multiple defaults, random search and Bayesian optimization.

Notation

$\hat{f}_\theta(X)$ -prediction model controlled by multidimensional hyperparameter configuration θ , $\theta \in \Theta$

$L(y, \hat{f}_\theta(X))$ - loss function

$\mathcal{R}_{\mathcal{P}}(\theta) = \mathbb{E}(L(y, \hat{f}_\theta(X))|\mathcal{P})$ - expected risk

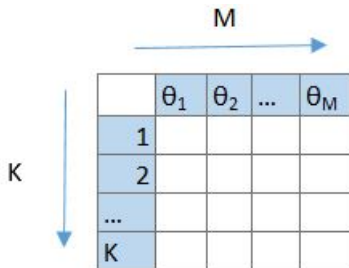
Notation

K - different datasets ($\mathcal{P}_1, \dots, \mathcal{P}_K$)

M - different hyperparameters configuration ($\Theta_M = \{\theta_1, \dots, \theta_M\}$)

$$R_k(\theta) = \mathbb{E}(L(y, \hat{f}_\theta(X)) | \mathcal{P}_k), \quad k = 1, \dots, K$$

$$R(\Theta_M) = (R_k(\theta_m)), \quad k = 1, \dots, K, m = 1, \dots, M$$



	θ_1	θ_2	...	θ_M
1				
2				
...				
K				

Risk of a set of configuration

$$G(\Theta_{def}) = h \left(\min_{j=1,\dots,n} R_1(\theta_j), \dots, \min_{j=1,\dots,n} R_K(\theta_j) \right)$$

$$\arg \min_{|\Theta_{def}|=n} G(\Theta_{def})$$

Function h - mean.

Problem: We assume that measure performance are commensurable across datasets. (standardization of results)

Methods

- exact discretized optimization
- greedy search
for $i = 1, \dots, n$:

$$\theta_{def,i} = \arg \min_{\theta \in \Theta} G(\{\theta\} \cup \Theta_{def,i-1})$$

$$\Theta_{def,i} = \{\theta_{def,1}, \dots, \theta_{def,i}\}$$

Cons: we could not find global optimum

Pros: greedy search results in a growing sequence of default subset

Methods

- exact discretized optimization
- greedy search
for $i = 1, \dots, n$:

$$\theta_{def,i} = \arg \min_{\theta \in \Theta} G(\{\theta\} \cup \Theta_{def,i-1})$$

$$\Theta_{def,i} = \{\theta_{def,1}, \dots, \theta_{def,i}\}$$

Cons: we could not find global optimum

Pros: greedy search results in a growing sequence of default subset

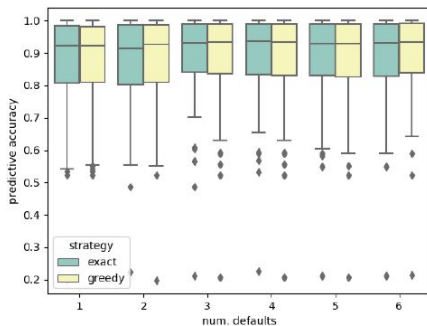
Estimation of $R_k(\theta)$:

- 1 crossvalidation
- 2 surrogate model

Exact Discretized vs. Greedy Defaults

Model SVM, 100 datasets.

Generation of $n = 1, \dots, 6$ set of defaults (gamma and complexity - 16 choices for both)



Sets of defaults from both methods perform approximately the same.
Greedy defaults has the benefit of being computationally much cheaper.

Benchmark setup

- mlr:
 - ▶ glmnet(2), rpart(4), xgboost(10) - AUC
 - ▶ 38 datasets
- scikit-learn:
 - ▶ Adaboost(5), SVM (6), random forest (6) - Accuracy
 - ▶ 100 datasets
- Leave-One-Dataset-Out crossvalidation

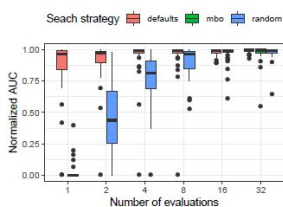
Greedy Defaults

Comparison method of multiple defaults (greedy search) with traditional approach. For each $k \in 1, \dots, K$:

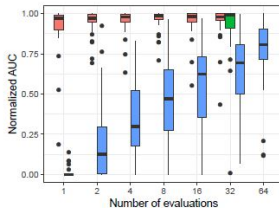
- multiple defaults:
 - ▶ for $n = \{1, 2, 4, 8, 16, 32\}$ find a set of n defaults Θ_{def} on $\{\{1, \dots, K\} \setminus k\}$
 - ▶ on dataset k - CV
- random search ($iter = 4, 8, 16, 32, 64$)
- Bayesian optimization ($iter = 32$)

Performance estimates across all evaluated methods are obtained from a fixed outer 10-fold cross-validation loop on each left out dataset.

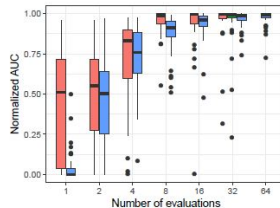
Results



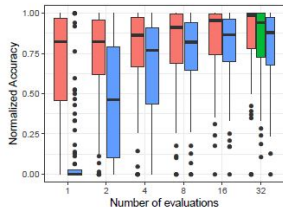
(a) Elastic net



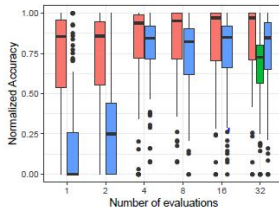
(b) Decision tree



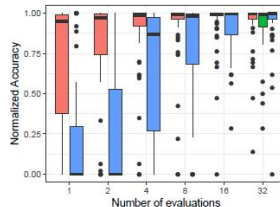
(c) Gradient boosting



(d) Adaboost



(e) Random forest



(f) SVM

Results

- multiple defaults and random search having more iterations improves performance
- Bayesian optimization is often among the highest ranked strategy
- using only a few defaults is competitive with BO and RS with higher budget

Conclusion

- sets of defaults is especially worthwhile when either computation time or expertise on hyperparameter optimization is lacking
- A potential drawback is that defaults are optimal with respect to a single metric (extension: multi-objective measure)