Initializing Bayesian Hyperparameter Optimization via Meta-Learning

Matthias Feurer
feurerm@cs.uni-freiburg.de

Jost Tobias Springenberg
springj@cs.uni-freiburg.de

Frank Hutter
fh@cs.uni-freiburg.de

Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

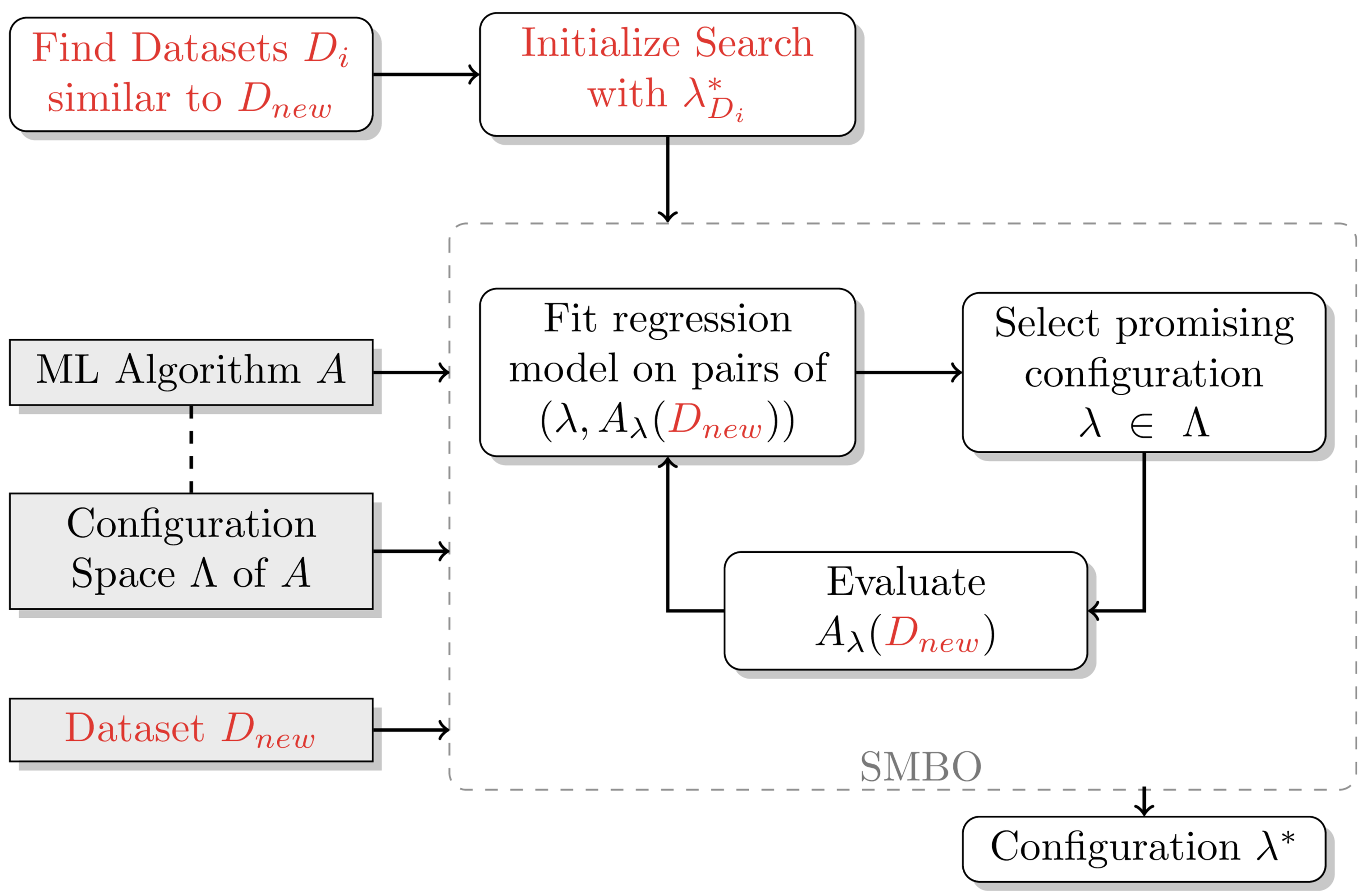
… in 30 seconds

- Hyperparameters of machine learning algorithms should be optimized by automated methods, not by humans
- **Bayesian Optimization** is a powerful hyperparameter optimization tool
- In contrast to human domain experts, Bayesian Optimization does not use knowledge from previous runs on different datasets
- We employ **meta-learning** to obtain promising configurations to **warmstart Bayesian Optimization**

MI-SMBO

- **M**eta-learning **I**nitialized **S**equential **M**odel-based **B**ayesian **O**ptimization
- Mimics human domain experts: uses configurations which are known to work well on similar datasets
- Similarity is defined by a distance between datasets based on metafeatures

SMBO with Meta-Learning



Standard Bayesian Optimization (**black**) together with meta-learning initialization (**red**).

Dataset Similarity

Similarity of datasets is defined by a distance function between dataset metafeatures. Some examples of metafeatures for the Iris dataset:

Metafeatures for the Iris dataset	
# training examples	150
# classes	3
# features	4
# numerical features	4
# categorical features	0
# categorical features	No

We compared two distance functions:

- L_1 norm:

$$d_p(D_{new}, D_j) = \sum_i |m_i^{new} - m_i^j|$$

- Spearman correlation coefficient between known model performances:

$$d_c(D_i, D_j) = 1 - \mathrm{Corr}\left(\begin{matrix}[f^{D_i}(\lambda_1), \ldots, f^{D_i}(\lambda_n)], \\ [f^{D_j}(\lambda_1), \ldots, f^{D_j}(\lambda_n)]\end{matrix}\right)$$

Caveats:

- This only works for a fixed set of hyperparameters
- Cannot be computed for a new dataset D_{new}
  Solution: compute $d_c(D_i, D_j)$ for all 1 ≤ I, j ≤ N and use regression to learn a mapping from $\langle m^i, m^j \rangle$ to $d_c(D_i, D_j)$. We used a random forest for this mapping.

Experiments

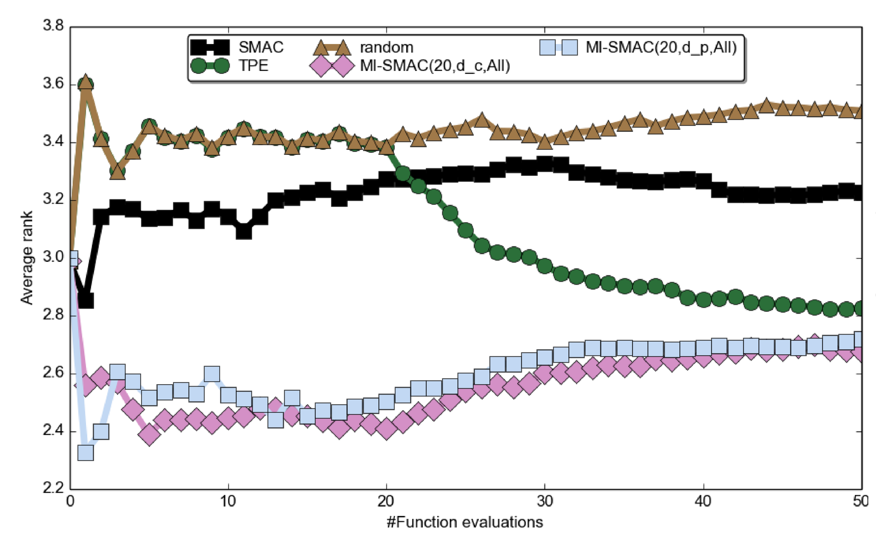
Setup

- Two experiments:
  1. Tuned the hyperparameters of an SVM (see paper)
  2. Combined algorithm selection and hyperparameter optimization (CASH) for scikit-learn: AutoSklearn

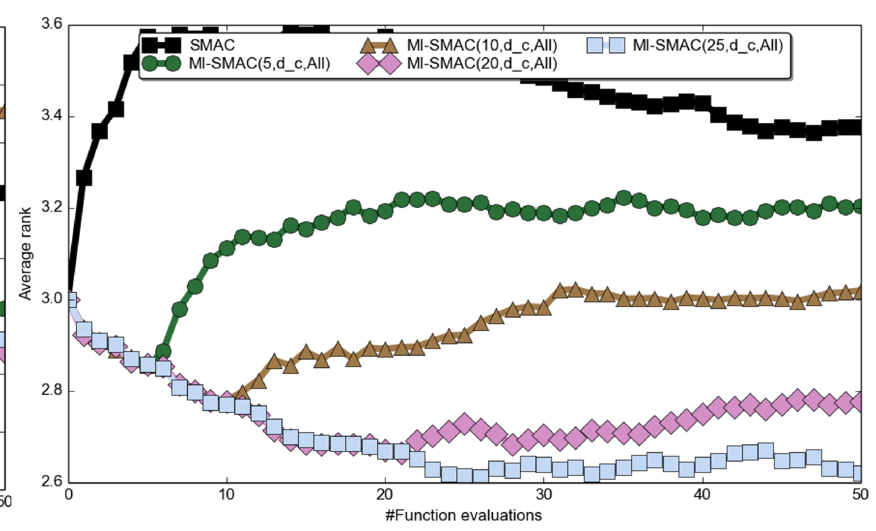
Component	Hyperparameter	#Values
Main	Classifier	3
Main	Preprocessing	2
SVM	$\log_2(C)$	21
SVM	$\log_2(\gamma)$	19
LinearSVM	$\log_2(C)$	21
LinearSVM	Penalty	1
RF	Max features	5
RF	Min splits	10
RF	Criterion	2
PCA	Variance to keep	2

- Validated our approach on 57 datasets from `OpenML.org`
  - Leave one dataset out: Ran MI-SMBO on one dataset and assumed knowledge of performance on all other 56
- Precomputed a dense grid of 1623 hyperparameter configurations
- Ran each optimization algorithm 10 times on each dataset
- Used 46 metafeatures from the literature
- Tried 40 different instantiations of MI-SMAC

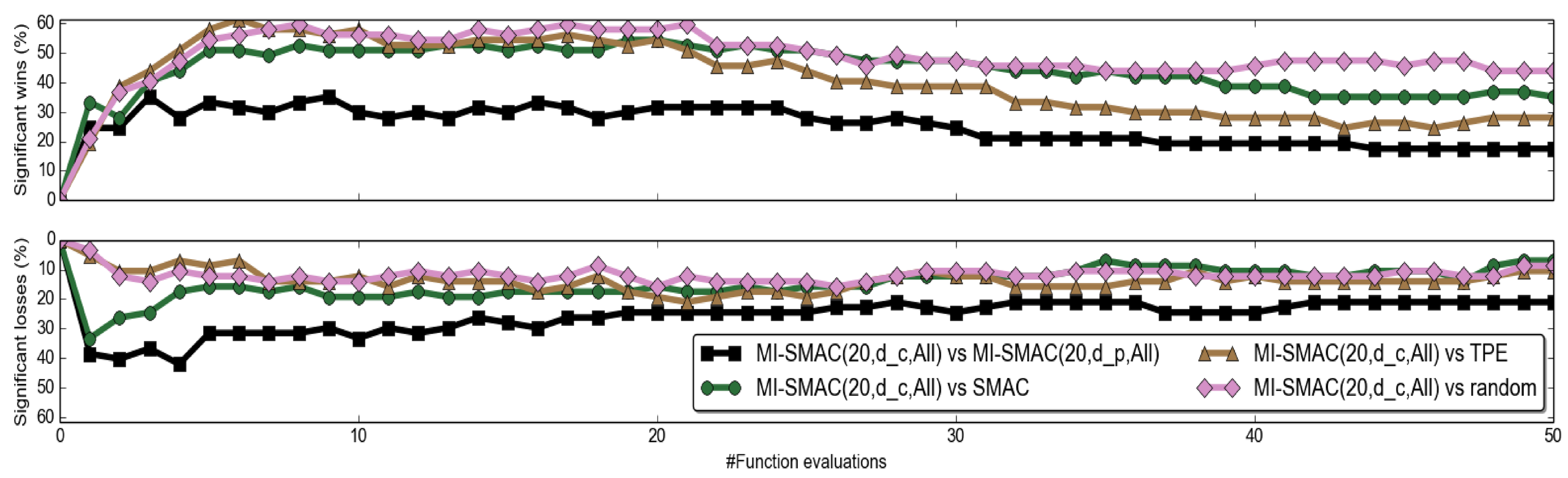
Results



Average rank of different optimization algorithms. Since we ran each algorithm ten times on each dataset, we drew a bootstrap sample of 1000 joint runs and computed the average across these runs. We then further averaged these ranks across all 57 datasets.



Average rank of MI-SMAC with different number of initial configurations.



Top: Percentage of datasets on which MI-SMAC performs statistically better than its competitors.
Bottom: As above, but percentage of losses.
This plot shows that MI-SMAC improves over vanilla SMAC on 36% of the datasets, while it is worse on only 8%. We also observe that metalearning leads to a great performance boost in the beginning of SMBO.