Auto-WEKA: Automatic model selection and hyperparameter optimization in WEKA

Lars Kotthoff and Chris Thornton and Holger H. Hoos and Frank Hutter and Kevin Leyton-Brown

University of British Columbia

Abstract. Many different machine learning algorithms exist; taking into account each algorithm's hyperparameters, there is a staggeringly large number of possible alternatives overall. We consider the problem of simultaneously selecting a learning algorithm and setting its hyperparameters. We show that this problem can be addressed by a fully automated approach, leveraging recent innovations in Bayesian optimization. Specifically, we consider feature selection techniques and all machine learning approaches implemented in WEKA's standard distribution, spanning 2 ensemble methods, 10 meta-methods, 28 base learners, and hyperparameter settings for each learner. On each of 21 popular datasets from the UCI repository, the KDD Cup 09, variants of the MNIST dataset and CIFAR-10, we show performance often much better than using standard selection and hyperparameter optimization methods. We hope that our approach will help non-expert users to more effectively identify machine learning algorithms and hyperparameter settings appropriate to their applications, and hence to achieve improved performance.

1 Introduction

Increasingly, users of machine learning tools are non-experts who require off-the-shelf solutions. The machine learning community has much aided such users by making available a wide variety of sophisticated learning algorithms and feature selection methods through open source packages, such as WEKA [15] and mlr [7]. Such packages ask a user to make two kinds of choices: selecting a learning algorithm and customizing it by setting hyperparameters (which also control feature selection, if applicable). It can be challenging to make the right choice when faced with these degrees of freedom, leaving many users to select algorithms based on reputation or intuitive appeal, and/or to leave hyperparameters set to default values. Of course, adopting this approach can yield performance far worse than that of the best method and hyperparameter settings.

This suggests a natural challenge for machine learning: given a dataset, automatically and simultaneously choosing a learning algorithm and setting its hyperparameters to optimize empirical performance. We dub this the *combined algorithm selection and hyperparameter optimization (CASH) problem*; we formally define it in Section 3. There has been considerable past work separately

addressing model selection, e.g. [1,6,8,9,11,23,24,31,], and hyperparameter optimization, e.g. [3,4,5,14,27,29,22,]. In contrast, despite its practical importance, we are surprised to find only limited variants of the CASH problem in the literature; furthermore, these consider a fixed and relatively small number of parameter configurations for each algorithm, see e.g. [21,].

A likely explanation is that it is very challenging to search the combined space of learning algorithms and their hyperparameters: the response function is noisy and the space is high dimensional, involves both categorical and continuous choices, and contains hierarchical dependencies (e.g., the hyperparameters of a learning algorithm are only meaningful if that algorithm is chosen; the algorithm choices in an ensemble method are only meaningful if that ensemble method is chosen; etc). Another related line of work is on meta-learning procedures that exploit characteristics of the dataset, such as the performance of so-called landmarking algorithms, to predict which algorithm or hyperparameter configuration will perform well [2,21,25,30]. While the CASH algorithms we study in this chapter start from scratch for each new dataset, these meta-learning procedures exploit information from previous datasets, which may not always be available.

In what follows, we demonstrate that CASH can be viewed as a single hierarchical hyperparameter optimization problem, in which even the choice of algorithm itself is considered a hyperparameter. We also show that—based on this problem formulation—recent Bayesian optimization methods can obtain high quality results in reasonable time and with minimal human effort. After discussing some preliminaries (Section 2), we define the CASH problem and discuss methods for tackling it (Section 3). We then define a concrete CASH problem encompassing a wide range of learners and feature selectors in the open source package WEKA (Section 4), and show that a search in the combined space of algorithms and hyperparameters yields better-performing models than standard algorithm selection and hyperparameter optimization methods (Section 5). More specifically, we show that the recent Bayesian optimization procedures TPE [4] and SMAC [16] often find combinations of algorithms and hyperparameters that outperform existing baseline methods, especially on large datasets.

2 Preliminaries

We consider learning a function $f : \mathcal{X} \mapsto \mathcal{Y}$, where $\mathcal{Y}$ is either finite (for classification), or continuous (for regression). A *learning algorithm* A maps a set $\{d_1, \ldots, d_n\}$ of training data points $d_i = (\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ to such a function, which is often expressed via a vector of *model parameters*. Most learning algorithms A further expose *hyperparameters* $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$, which change the way the learning algorithm $A_{\boldsymbol{\lambda}}$ itself works. For example, hyperparameters are used to describe a description-length penalty, the number of neurons in a hidden layer, the number of data points that a leaf in a decision tree must contain to be eligible for splitting, etc. These hyperparameters are typically optimized in an "outer

loop" that evaluates the performance of each hyperparameter configuration using cross-validation.

2.1 Model Selection

Given a set of learning algorithms $\mathcal{A}$ and a limited amount of training data $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, the goal of model selection is to determine the algorithm $A^* \in \mathcal{A}$ with optimal generalization performance. Generalization performance is estimated by splitting $\mathcal{D}$ into disjoint training and validation sets $\mathcal{D}^{(i)}_{\text{train}}$ and $\mathcal{D}^{(i)}_{\text{valid}}$, learning functions f_i by applying A^* to $\mathcal{D}^{(i)}_{\text{train}}$, and evaluating the predictive performance of these functions on $\mathcal{D}^{(i)}_{\text{valid}}$. This allows for the model selection problem to be written as:

$$A^* \in \operatorname*{argmin}_{A \in \mathcal{A}} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}(A, \mathcal{D}^{(i)}_{\text{train}}, \mathcal{D}^{(i)}_{\text{valid}}),$$

where $\mathcal{L}(A, \mathcal{D}^{(i)}_{\text{train}}, \mathcal{D}^{(i)}_{\text{valid}})$ is the loss achieved by A when trained on $\mathcal{D}^{(i)}_{\text{train}}$ and evaluated on $\mathcal{D}^{(i)}_{\text{valid}}$.

We use k-fold cross-validation [19], which splits the training data into k equal-sized partitions $\mathcal{D}^{(1)}_{\text{valid}}, \ldots, \mathcal{D}^{(k)}_{\text{valid}}$, and sets $\mathcal{D}^{(i)}_{\text{train}} = \mathcal{D} \setminus \mathcal{D}^{(i)}_{\text{valid}}$ for $i = 1, \ldots, k$.[1]

2.2 Hyperparameter Optimization

The problem of optimizing the hyperparameters $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ of a given learning algorithm A is conceptually similar to that of model selection. Some key differences are that hyperparameters are often continuous, that hyperparameter spaces are often high dimensional, and that we can exploit correlation structure between different hyperparameter settings $\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2 \in \boldsymbol{\Lambda}$. Given n hyperparameters $\lambda_1, \ldots, \lambda_n$ with domains $\Lambda_1, \ldots, \Lambda_n$, the hyperparameter space $\boldsymbol{\Lambda}$ is a subset of the crossproduct of these domains: $\boldsymbol{\Lambda} \subset \Lambda_1 \times \cdots \times \Lambda_n$. This subset is often strict, such as when certain settings of one hyperparameter render other hyperparameters inactive. For example, the parameters determining the specifics of the third layer of a deep belief network are not relevant if the network depth is set to one or two. Likewise, the parameters of a support vector machine's polynomial kernel are not relevant if we use a different kernel instead.

More formally, following [17], we say that a hyperparameter λ_i is *conditional* on another hyperparameter λ_j, if λ_i is only active if hyperparameter λ_j takes values from a given set $V_i(j) \subsetneq \Lambda_j$; in this case we call λ_j a *parent* of λ_i. Conditional hyperparameters can in turn be parents of other conditional hyperparameters, giving rise to a tree-structured space [4] or, in some cases, a directed

[1] There are other ways of estimating generalization performance; e.g., we also experimented with repeated random subsampling validation [19], and obtained similar results.

Algorithm 1 SMBO

1: initialise model $\mathcal{M}_L$; $\mathcal{H} \leftarrow \emptyset$
2: **while** time budget for optimization has not been exhausted **do**
3: $\quad \boldsymbol{\lambda} \leftarrow$ candidate configuration from $\mathcal{M}_L$
4: $\quad$ Compute $c = \mathcal{L}(A_{\boldsymbol{\lambda}}, \mathcal{D}^{(i)}_{\text{train}}, \mathcal{D}^{(i)}_{\text{valid}})$
5: $\quad \mathcal{H} \leftarrow \mathcal{H} \cup \{(\boldsymbol{\lambda}, c)\}$
6: $\quad$ Update $\mathcal{M}_L$ given $\mathcal{H}$
7: **end while**
8: **return** $\boldsymbol{\lambda}$ from $\mathcal{H}$ with minimal c

acyclic graph (DAG) [17]. Given such a structured space $\boldsymbol{\Lambda}$, the (hierarchical) hyperparameter optimization problem can be written as:

$$\boldsymbol{\lambda}^* \in \operatorname*{argmin}_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}(A_{\boldsymbol{\lambda}}, \mathcal{D}^{(i)}_{\text{train}}, \mathcal{D}^{(i)}_{\text{valid}}).$$

3 Combined Algorithm Selection and Hyperparameter Optimization (CASH)

Given a set of algorithms $\mathcal{A} = \{A^{(1)}, \ldots, A^{(k)}\}$ with associated hyperparameter spaces $\boldsymbol{\Lambda}^{(1)}, \ldots, \boldsymbol{\Lambda}^{(k)}$, we define the combined algorithm selection and hyperparameter optimization problem (CASH) as computing

$$A^*{}_{\boldsymbol{\lambda}^*} \in \operatorname*{argmin}_{A^{(j)} \in \mathcal{A}, \boldsymbol{\lambda} \in \boldsymbol{\Lambda}^{(j)}} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}(A^{(j)}_{\boldsymbol{\lambda}}, \mathcal{D}^{(i)}_{\text{train}}, \mathcal{D}^{(i)}_{\text{valid}}). \tag{1}$$

We note that this problem can be reformulated as a single combined hierarchical hyperparameter optimization problem with parameter space $\boldsymbol{\Lambda} = \boldsymbol{\Lambda}^{(1)} \cup \cdots \cup \boldsymbol{\Lambda}^{(k)} \cup \{\lambda_r\}$, where λ_r is a new root-level hyperparameter that selects between algorithms $A^{(1)}, \ldots, A^{(k)}$. The root-level parameters of each subspace $\boldsymbol{\Lambda}^{(i)}$ are made conditional on λ_r being instantiated to A_i.

In principle, problem 1 can be tackled in various ways. A promising approach is Bayesian Optimization [10], and in particular Sequential Model-Based Optimization (SMBO) [16], a versatile stochastic optimization framework that can work with both categorical and continuous hyperparameters, and that can exploit hierarchical structure stemming from conditional parameters. SMBO (outlined in Algorithm 1) first builds a model $\mathcal{M}_{\mathcal{L}}$ that captures the dependence of loss function $\mathcal{L}$ on hyperparameter settings $\boldsymbol{\lambda}$ (line 1 in Algorithm 1). It then iterates the following steps: use $\mathcal{M}_{\mathcal{L}}$ to determine a promising candidate configuration of hyperparameters $\boldsymbol{\lambda}$ to evaluate next (line 3); evaluate the loss c of $\boldsymbol{\lambda}$ (line 4); and update the model $\mathcal{M}_{\mathcal{L}}$ with the new data point $(\boldsymbol{\lambda}, c)$ thus obtained (lines 5–6).

In order to select its next hyperparameter configuration $\boldsymbol{\lambda}$ using model $\mathcal{M}_{\mathcal{L}}$, SMBO uses a so-called *acquisition function* $a_{\mathcal{M}_{\mathcal{L}}} : \boldsymbol{\Lambda} \mapsto \mathbb{R}$, which uses the predictive distribution of model $\mathcal{M}_{\mathcal{L}}$ at arbitrary hyperparameter configurations

$\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ to quantify (in closed form) how useful knowledge about $\boldsymbol{\lambda}$ would be. SMBO then simply maximizes this function over $\boldsymbol{\Lambda}$ to select the most useful configuration $\boldsymbol{\lambda}$ to evaluate next. Several well-studied acquisition functions exist [18,26,28]; all aim to automatically trade off exploitation (locally optimizing hyperparameters in regions known to perform well) versus exploration (trying hyperparameters in a relatively unexplored region of the space) in order to avoid premature convergence. In this work, we maximized *positive expected improvement (EI)* attainable over an existing given loss c_{min} [26]. Let $c(\boldsymbol{\lambda})$ denote the loss of hyperparameter configuration $\boldsymbol{\lambda}$. Then, the positive improvement function over c_{min} is defined as

$$I_{c_{min}}(\boldsymbol{\lambda}) := \max\{c_{min} - c(\boldsymbol{\lambda}), 0\}.$$

Of course, we do not know $c(\boldsymbol{\lambda})$. We can, however, compute its expectation with respect to the current model $\mathcal{M}_{\mathcal{L}}$:

$$\mathbb{E}_{\mathcal{M}_{\mathcal{L}}}[I_{c_{min}}(\boldsymbol{\lambda})] = \int_{-\infty}^{c_{min}} \max\{c_{min} - c, 0\} \cdot p_{\mathcal{M}_L}(c \mid \lambda)\, dc. \tag{2}$$

We briefly review the SMBO approach used in this chapter.

3.1 Sequential Model-based Algorithm Configuration (SMAC)

Sequential model-based algorithm configuration (SMAC) [16] supports a variety of models $p(c \mid \boldsymbol{\lambda})$ to capture the dependence of the loss function c on hyperparameters $\boldsymbol{\lambda}$, including approximate Gaussian processes and random forests. In this chapter we use random forest models, since they tend to perform well with discrete and high-dimensional input data. SMAC handles conditional parameters by instantiating inactive conditional parameters in $\boldsymbol{\lambda}$ to default values for model training and prediction. This allows the individual decision trees to include splits of the kind "is hyperparameter λ_i active?", allowing them to focus on active hyperparameters. While random forests are not usually treated as probabilistic models, SMAC obtains a predictive mean $\mu_{\boldsymbol{\lambda}}$ and variance ${\sigma_{\boldsymbol{\lambda}}}^2$ of $p(c \mid \boldsymbol{\lambda})$ as frequentist estimates over the predictions of its individual trees for $\boldsymbol{\lambda}$; it then models $p_{\mathcal{M}_{\mathcal{L}}}(c \mid \boldsymbol{\lambda})$ as a Gaussian $\mathcal{N}(\mu_{\boldsymbol{\lambda}}, {\sigma_{\boldsymbol{\lambda}}}^2)$.

SMAC uses the expected improvement criterion defined in Equation 2, instantiating c_{min} to the loss of the best hyperparameter configuration measured so far. Under SMAC's predictive distribution $p_{\mathcal{M}_{\mathcal{L}}}(c \mid \boldsymbol{\lambda}) = \mathcal{N}(\mu_{\boldsymbol{\lambda}}, {\sigma_{\boldsymbol{\lambda}}}^2)$, this expectation is the closed-form expression

$$\mathbb{E}_{\mathcal{M}_{\mathcal{L}}}[I_{c_{min}}(\boldsymbol{\lambda})] = \sigma_{\boldsymbol{\lambda}} \cdot [u \cdot \Phi(u) + \varphi(u)],$$

where $u = \frac{c_{\min} - \mu_{\boldsymbol{\lambda}}}{\sigma_{\boldsymbol{\lambda}}}$, and φ and Φ denote the probability density function and cumulative distribution function of a standard normal distribution, respectively [18].

SMAC is designed for robust optimization under noisy function evaluations, and as such implements special mechanisms to keep track of its best known configuration and assure high confidence in its estimate of that configuration's performance. This robustness against noisy function evaluations can be exploited in

combined algorithm selection and hyperparameter optimization, since the function to be optimized in Equation (1) is a mean over a set of loss terms (each corresponding to one pair of $\mathcal{D}^{(i)}_{\text{train}}$ and $\mathcal{D}^{(i)}_{\text{valid}}$ constructed from the training set). A key idea in SMAC is to make progressively better estimates of this mean by evaluating these terms one at a time, thus trading off accuracy and computational cost. In order for a new configuration to become a new incumbent, it must outperform the previous incumbent in every comparison made: considering only one fold, two folds, and so on up to the total number of folds previously used to evaluate the incumbent. Furthermore, every time the incumbent survives such a comparison, it is evaluated on a new fold, up to the total number available, meaning that the number of folds used to evaluate the incumbent grows over time. A poorly performing configuration can thus be discarded after considering just a single fold.

Finally, SMAC also implements a diversification mechanism to achieve robust performance even when its model is misled, and to explore new parts of the space: every second configuration is selected at random. Because of the evaluation procedure just described, this requires less overhead than one might imagine.

4 Auto-WEKA

To demonstrate the feasibility of an automatic approach to solving the CASH problem, we built Auto-WEKA, which solves this problem for the learners and feature selectors implemented in the WEKA machine learning package [15]. Note that while we have focused on classification algorithms in WEKA, there is no obstacle to extending our approach to other settings. Indeed, another successful system that uses the same underlying technology is auto-sklearn [12].

Table 1 shows all supported learning algorithms and feature selectors with the number of hyperparameters. algorithms. Meta-methods take a single base classifier and its parameters as an input, and the ensemble methods can take any number of base learners as input. We allowed the meta-methods to use any base learner with any hyperparameter settings, and allowed the ensemble methods to use up to five learners, again with any hyperparameter settings. Not all learners are applicable on all datasets (e.g., due to a classifier's inability to handle missing data). For a given dataset, our Auto-WEKA implementation automatically only considers the subset of applicable learners. Feature selection is run as a preprocessing phase before building any model.

The algorithms in Table 1 have a wide variety of hyperparameters, which take values from continuous intervals, from ranges of integers, and from other discrete sets. We associated either a uniform or log uniform prior with each numerical parameter, depending on its semantics. For example, we set a log uniform prior for the ridge regression penalty, and a uniform prior for the maximum depth for a tree in a random forest. Auto-WEKA works with continuous hyperparameter values directly up to the precision of the machine. We emphasize that this combined hyperparameter space is *much* larger than a simple union of the base learners' hyperparameter spaces, since the ensemble methods allow up to 5 *in-*

Base Learners			
BayesNet	2	NaiveBayes	2
DecisionStump*	0	NaiveBayesMultinomial	0
DecisionTable*	4	OneR	1
GaussianProcesses*	10	PART	4
IBk*	5	RandomForest	7
J48	9	RandomTree*	11
JRip	4	REPTree*	6
KStar*	3	SGD*	5
LinearRegression*	3	SimpleLinearRegression*	0
LMT	9	SimpleLogistic	5
Logistic	1	SMO	11
M5P	4	SMOreg*	13
M5Rules	4	VotedPerceptron	3
MultilayerPerceptron*	8	ZeroR*	0
Ensemble Methods			
Stacking	2	Vote	2
Meta-Methods			
LWL	5	Bagging	4
AdaBoostM1	6	RandomCommittee	2
AdditiveRegression	4		
AttributeSelectedClassifier	2	RandomSubSpace	3
Feature Selection Methods			
BestFirst	2	GreedyStepwise	4

Fig. 1. Learners and methods supported by Auto-WEKA, along with number of hyperparameters $|\Lambda|$. Every learner supports classification; starred learners also support regression.

dependent base learners. The meta- and ensemble methods as well as the feature selection contribute further to the total size of AutoWEKA's hyperparameter space.

Auto-WEKA uses the SMAC optimizer described above to solve the CASH problem and is available to the public through the WEKA package manager; the source code can be found at `https://github.com/automl/autoweka` and the official project website is at `http://www.cs.ubc.ca/labs/beta/Projects/autoweka`. For the experiments described in this chapter, we used Auto-WEKA version 0.5. The results the more recent versions achieve are similar; we did not replicate the full set of experiments because of the large computational cost.

5 Experimental evaluation

We evaluated Auto-WEKA on 21 prominent benchmark datasets (see Table 1): 15 sets from the UCI repository [13]; the 'convex', 'MNIST basic' and 'rotated MNIST with background images' tasks used in [5]; the appentency task from the KDD Cup '09; and two versions of the CIFAR-10 image classification task [20] (CIFAR-10-Small is a subset of CIFAR-10, where only the first 10 000 training

Table 1. Datasets used; *Num. Discr.*. and *Num. Cont.* refer to the number of discrete and continuous attributes of elements in the dataset, respectively.

Name	Num Discr.	Num Cont.	Num Classes	Num Training	Num Test
Dexter	20 000	0	2	420	180
GermanCredit	13	7	2	700	300
Dorothea	100 000	0	2	805	345
Yeast	0	8	10	1 038	446
Amazon	10 000	0	49	1 050	450
Secom	0	591	2	1 096	471
Semeion	256	0	10	1 115	478
Car	6	0	4	1 209	519
Madelon	500	0	2	1 820	780
KR-vs-KP	37	0	2	2 237	959
Abalone	1	7	28	2 923	1 254
Wine Quality	0	11	11	3 425	1 469
Waveform	0	40	3	3 500	1 500
Gisette	5 000	0	2	4 900	2 100
Convex	0	784	2	8 000	50 000
CIFAR-10-Small	3 072	0	10	10 000	10 000
MNIST Basic	0	784	10	12 000	50 000
Rot. MNIST + BI	0	784	10	12 000	50 000
Shuttle	9	0	7	43 500	14 500
KDD09-Appentency	190	40	2	35 000	15 000
CIFAR-10	3 072	0	10	50 000	10 000

data points are used rather than the full 50 000.) Note that in the experimental evaluation, we focus on classification. For datasets with a predefined training/test split, we used that split. Otherwise, we randomly split the dataset into 70% training and 30% test data. We withheld the test data from all optimization method; it was only used once in an offline analysis stage to evaluate the models found by the various optimization methods.

For each dataset, we ran Auto-WEKA with each hyperparameter optimization algorithm with a total time budget of 30 hours. For each method, we performed 25 runs of this process with different random seeds and then—in order to simulate parallelization on a typical workstation—used bootstrap sampling to repeatedly select four random runs and report the performance of the one with best cross-validation performance.

In early experiments, we observed a few cases in which Auto-WEKA's SMBO method picked hyperparameters that had excellent training performance, but turned out to generalize poorly. To enable Auto-WEKA to detect such overfitting, we partitioned its training set into two subsets: 70% for use inside the

SMBO method, and 30% of validation data that we only used after the SMBO method finished.

5.1 Baseline Methods

Auto-WEKA aims to aid non-expert users of machine learning techniques. A natural approach that such a user might take is to perform 10-fold cross validation on the training set for each technique with unmodified hyperparameters, and select the classifier with the smallest average misclassification error across folds. We will refer to this method applied to our set of WEKA learners as *Ex-Def*; it is the best choice that can be made for WEKA with default hyperparameters.

For each dataset, the second and third columns in Table 2 present the best and worst "oracle performance" of the default learners when prepared given all the training data and evaluated on the test set. We observe that the gap between the best and worst learner was huge, e.g. misclassification rates of 4.93% vs 99.24% on the Dorothea dataset. This suggests that some form of algorithm selection is essential for achieving good performance.

Table 2. Performance on both 10-fold cross-validation and test data. Ex-Def and Grid Search are deterministic. Random search had a time budget of 120 CPU hours. For Auto-WEKA, we performed 25 runs of 30 hours each. We report results as mean loss across 100 000 bootstrap samples simulating 4 parallel runs. We determined test loss (misclassification rate) by training the selected model/hyperparameters on the entire 70% training data and computing accuracy on the previously unused 30% test data. Bold face indicates the lowest error within a block of comparable methods that was statistically significant.

Dataset	**Oracle Perf. (%)**				**10-Fold C.V. Performance (%)**				**Test Performance (%)**			
	Ex-Def		Grid Search		Ex-Def	Grid Search	Rand. Search	Auto-WEKA	Ex-Def	Grid Search	Rand. Search	Auto-WEKA
	Best	Worst	Best	Worst								
Dexter	7.78	52.78	**3.89**	63.33	10.20	**5.07**	10.60	5.66	8.89	**5.00**	9.18	7.49
GermanCredit	26.00	38.00	**25.00**	68.00	22.45	20.20	20.15	**17.87**	27.33	**26.67**	29.03	28.24
Dorothea	4.93	99.24	**4.64**	99.24	6.03	6.73	8.11	**5.62**	6.96	5.80	**5.22**	6.21
Yeast	40.00	68.99	**36.85**	69.89	39.43	39.71	38.74	35.51	40.45	42.47	43.15	40.67
Amazon	28.44	99.33	**17.56**	99.33	43.94	**36.88**	59.85	47.34	28.44	**20.00**	41.11	33.99
Secom	7.87	14.26	**7.66**	92.13	6.25	6.12	5.24	**5.24**	8.09	8.09	8.03	**8.01**
Semeion	8.18	92.45	**5.24**	92.45	6.52	4.86	6.06	**4.78**	8.18	6.29	6.10	**5.08**
Car	0.77	29.15	**0.00**	46.14	2.71	0.83	**0.53**	0.61	0.77	0.97	**0.01**	0.40
Madelon	**17.05**	50.26	**17.05**	62.69	25.98	26.46	27.95	**20.70**	21.38	21.15	24.29	**21.12**
KR-vs-KP	0.31	48.96	**0.21**	51.04	0.89	0.64	0.63	**0.30**	0.31	1.15	0.58	**0.31**
Abalone	73.18	84.04	**72.15**	92.90	73.33	72.15	72.03	**71.71**	73.18	73.42	74.88	73.51
Wine Quality	36.35	60.99	**32.88**	99.39	38.94	35.23	35.36	**34.65**	37.51	34.06	34.41	33.95
Waveform	14.27	68.80	**13.47**	68.80	12.73	12.45	12.43	**11.92**	14.40	14.66	14.27	14.42
Gisette	2.52	50.91	**1.81**	51.23	3.62	2.59	4.84	**2.43**	2.81	2.40	4.62	**2.24**
Convex	25.96	50.00	**19.94**	71.49	28.68	**22.36**	33.31	25.93	25.96	23.45	31.20	**23.17**
CIFAR-10-Small	65.91	90.00	**52.16**	90.36	66.59	**53.64**	67.33	58.84	65.91	56.94	66.12	**56.87**
MNIST Basic	5.19	88.75	**2.58**	88.75	5.12	**2.51**	5.05	3.75	5.19	**2.64**	5.05	3.64
Rot. MNIST + BI	63.14	88.88	**55.34**	93.01	66.15	**56.01**	68.62	57.86	63.14	57.59	66.40	**57.04**
Shuttle	0.0138	20.8414	**0.0069**	89.8207	0.0328	0.0361	0.0345	**0.0224**	0.0138	0.0414	0.0157	**0.0130**
KDD09-Appentency	1.7400	6.9733	**1.6332**	54.2400	1.8776	1.8735	1.7510	**1.7038**	1.7405	1.7400	1.7400	**1.7358**
CIFAR-10	64.27	90.00	**55.27**	90.00	65.54	**54.04**	69.46	62.36	64.27	63.13	69.72	**61.15**

A stronger baseline we will use is an approach that in addition to selecting the learner, also sets its hyperparameters optimally from a predefined set. More precisely, this baseline performs an exhaustive search over a grid of hyperparameter settings for each of the base learners, discretizing numeric parameters into three points. We refer to this baseline as *grid search* and note that—as an optimization approach in the joint space of algorithms and hyperparameter settings—it is a simple CASH algorithm. However, it is quite expensive, requiring more than 10 000 CPU hours on each of Gisette, Convex, MNIST, Rot MNIST + BI, and both CIFAR variants, rendering it infeasible to use in most practical applications. (In contrast, we gave Auto-WEKA only 120 CPU hours.)

Table 2 (columns four and five) shows the best and worst "oracle performance" on the test set across the classifiers evaluated by grid search. Comparing these performances to the default performance obtained using Ex-Def, we note that in most cases, even WEKA's best default algorithm could be improved by selecting better hyperparameter settings, sometimes rather substantially: e.g., in the CIFAR-10 small task, grid search offered a 13% reduction in error over Ex-Def.

It has been demonstrated in previous work that, holding the overall time budget constant, grid search is outperformed by random search over the hyperparameter space [5]. Our final baseline, *random search*, implements such a method, picking algorithms and hyperparameters sampled at random, and computes their performance on the 10 cross-validation folds until it exhausts its time budget. For each dataset, we first used 750 CPU hours to compute the cross-validation performance of randomly sampled combinations of algorithms and hyperparameters. We then simulated runs of random search by sampling combinations without replacement from these results that consumed 120 CPU hours and returning the sampled combination with the best performance.

5.2 Results for Cross-Validation Performance

The middle portion of Table 2 reports our main results. First, we note that grid search over the hyperparameters of all base-classifiers yielded better results than Ex-Def in 17/21 cases, which underlines the importance of not only choosing the right algorithm but of also setting its hyperparameters well.

However, we note that we gave grid search a very large time budget (often in excess 10 000 CPU hours for each dataset, in total more than 10 CPU years), meaning that it would often be infeasible to use in practice.

In contrast, we gave each of the other methods only 4×30 CPU hours per dataset; nevertheless, they still yielded substantially better performance than grid search, outperforming it in 14/21 cases. Random search outperforms grid search in 9/21 cases, highlighting that even exhaustive grid search with a large time budget is not always the right thing to do. We note that sometimes Auto-WEKA's performance improvements over the baselines were substantial, with relative reductions of the cross-validation loss (in this case the misclassification rate) exceeding 10% in 6/21 cases.

5.3 Results for Test Performance

The results just shown demonstrate that Auto-WEKA is effective at optimizing its given objective function; however, this is not sufficient to allow us to conclude that it fits models that generalize well. As the number of hyperparameters of a machine learning algorithm grows, so does its potential for overfitting. The use of cross-validation substantially increases Auto-WEKA's robustness against overfitting, but since its hyperparameter space is much larger than that of standard classification algorithms, it is important to carefully study whether (and to what extent) overfitting poses a problem.

To evaluate generalization, we determined a combination of algorithm and hyperparameter settings A_λ by running Auto-WEKA as before (cross-validating on the training set), trained A_λ on the entire training set, and then evaluated the resulting model on the test set. The right portion of Table 2 reports the test performance obtained with all methods.

Broadly speaking, similar trends held as for cross-validation performance: Auto-WEKA outperforms the baselines, with grid search and random search performing better than Ex-Def. However, the performance differences were less pronounced: grid search only yields better results than Ex-Def in 15/21 cases, and random search in turn outperforms grid search in 7/21 cases. Auto-WEKA outperforms the baselines in 15/21 cases. Notably, on 12 of the 13 largest datasets, Auto-WEKA outperforms our baselines; we attribute this to the fact that the risk of overfitting decreases with dataset size. Sometimes, Auto-WEKA's performance improvements over the other methods were substantial, with relative reductions of the test misclassification rate exceeding 16% in 3/21 cases.

As mentioned earlier, Auto-WEKA only used 70% of its training set during the optimization of cross-validation performance, reserving the remaining 30% for assessing the risk of overfitting. At any point in time, Auto-WEKA's SMBO method keeps track of its *incumbent* (the hyperparameter configuration with the lowest cross-validation misclassification rate seen so far). After its SMBO procedure has finished, Auto-WEKA extracts a trajectory of these incumbents from it and computes their generalization performance on the withheld 30% validation data. It then computes the Spearman rank coefficient between the sequence of training performances (evaluated by the SMBO method through cross-validation) and this generalization performance.

6 Conclusion

In this work, we have shown that the daunting problem of combined algorithm selection and hyperparameter optimization (CASH) can be solved by a practical, fully automated tool. This is made possible by recent Bayesian optimization techniques that iteratively build models of the algorithm/hyperparameter landscape and leverage these models to identify new points in the space that deserve investigation.

We built a tool, Auto-WEKA, that draws on the full range of learning algorithms in WEKA and makes it easy for non-experts to build high-quality clas-

sifiers for given application scenarios. An extensive empirical comparison on 21 prominent datasets showed that Auto-WEKA often outperformed standard algorithm selection and hyperparameter optimization methods, especially on large datasets.

6.1 Community Adoption

Auto-WEKA was the first method to use Bayesian optimization to automatically instantiate a highly parametric machine learning framework at the push of a button. Since its initial release, it has been adopted by many users in industry and academia; the 2.0 line, which integrates with the WEKA package manager, has been downloaded more than 30,000 times, averaging more than 550 downloads a week. It is under active development, with new features added recently and in the pipeline.

References

1. Adankon, M., Cheriet, M.: Model selection for the LS-SVM. application to handwriting recognition. Pattern Recognition 42(12), 3264–3270 (2009)
2. Bardenet, R., Brendel, M., Kégl, B., Sebag, M.: Collaborative hyperparameter tuning. In: Proc. of ICML-13 (2013)
3. Bengio, Y.: Gradient-based optimization of hyperparameters. Neural Computation 12(8), 1889–1900 (2000)
4. Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for Hyper-Parameter Optimization. In: Proc. of NIPS-11 (2011)
5. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. JMLR 13, 281–305 (2012)
6. Biem, A.: A model selection criterion for classification: Application to HMM topology optimization. In: Proc. of ICDAR-03. pp. 104–108. IEEE (2003)
7. Bischl, B., Lang, M., Kotthoff, L., Schiffner, J., Richter, J., Studerus, E., Casalicchio, G., Jones, Z.M.: mlr: Machine Learning in R. Journal of Machine Learning Research 17(170), 1–5 (2016), `http://jmlr.org/papers/v17/15-066.html`
8. Bozdogan, H.: Model selection and Akaike's information criterion (AIC): The general theory and its analytical extensions. Psychometrika 52(3), 345–370 (1987)
9. Brazdil, P., Soares, C., Da Costa, J.: Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results. Machine Learning 50(3), 251–277 (2003)
10. Brochu, E., Cora, V.M., de Freitas, N.: A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. Tech. Rep. UBC TR-2009-23 and arXiv:1012.2599v1, Department of Computer Science, University of British Columbia (2009)
11. Chapelle, O., Vapnik, V., Bengio, Y.: Model selection for small sample regression. Machine Learning (2001)
12. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems 28, pp. 2962–2970. Curran Associates, Inc. (2015), `http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf`

13. Frank, A., Asuncion, A.: UCI machine learning repository (2010), http://archive.ics.uci.edu/ml, uRL: http://archive.ics.uci.edu/ml. University of California, Irvine, School of Information and Computer Sciences
14. Guo, X., Yang, J., Wu, C., Wang, C., Liang, Y.: A novel LS-SVMs hyper-parameter selection based on particle swarm optimization. Neurocomputing 71(16), 3211–3215 (2008)
15. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.: The WEKA data mining software: an update. ACM SIGKDD Explorations Newsletter 11(1), 10–18 (2009)
16. Hutter, F., Hoos, H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Proc. of LION-5. pp. 507–523 (2011)
17. Hutter, F., Hoos, H., Leyton-Brown, K., Stützle, T.: ParamILS: an automatic algorithm configuration framework. JAIR 36(1), 267–306 (2009)
18. Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black box functions. Journal of Global Optimization 13, 455–492 (1998)
19. Kohavi, R.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Proc. of IJCAI-95. pp. 1137–1145 (1995)
20. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Master's thesis, Department of Computer Science, University of Toronto (2009)
21. Leite, R., Brazdil, P., Vanschoren, J.: Selecting classification algorithms with active testing. In: Proc. of MLDM-12. pp. 117–131 (2012)
22. López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M.: The irace package, iterated race for automatic algorithm configuration. Tech. Rep. TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium (2011), http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf
23. Maron, O., Moore, A.: Hoeffding races: Accelerating model selection search for classification and function approximation. In: Proc. of NIPS-94. pp. 59–66 (1994)
24. McQuarrie, A., Tsai, C.: Regression and time series model selection. World Scientific (1998)
25. Pfahringer, B., Bensusan, H., Giraud-Carrier, C.: Meta-learning by landmarking various learning algorithms. In: Proc. of ICML-00. pp. 743–750 (2000)
26. Schonlau, M., Welch, W.J., Jones, D.R.: Global versus local search in constrained optimization of computer models. In: Flournoy, N., Rosenberger, W., Wong, W. (eds.) New Developments and Applications in Experimental Design, vol. 34, pp. 11–25. Institute of Mathematical Statistics, Hayward, California (1998)
27. Snoek, J., Larochelle, H., Adams, R.P.: Practical bayesian optimization of machine learning algorithms. In: Proc. of NIPS-12 (2012)
28. Srinivas, N., Krause, A., Kakade, S., Seeger, M.: Gaussian process optimization in the bandit setting: No regret and experimental design. In: Proc. of ICML-10. pp. 1015–1022 (2010)
29. Strijov, V., Weber, G.: Nonlinear regression model generation using hyperparameter optimization. Computers & Mathematics with Applications 60(4), 981–988 (2010)
30. Vilalta, R., Drissi, Y.: A perspective view and survey of meta-learning. Artif. Intell. Rev. 18(2), 77–95 (Oct 2002)
31. Zhao, P., Yu, B.: On model selection consistency of lasso. JMLR 7, 2541–2563 (Dec 2006)