Journal of Machine Learning Research 18 (2017) 1-5 Submitted 5/16; Revised 11/16; Published 3/17

Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA

Lars Kotthoff LARSKO@CS.UBC.CA
Chris Thornton CWTHORNT@CS.UBC.CA
Holger H. Hoos HOOS@CS.UBC.CA
Frank Hutter FH@CS.UNI-FREIBURG.DE
Kevin Leyton-Brown KEVINLB@CS.UBC.CA
Department of Computer Science
University of British Columbia
2366 Main Mall, Vancouver, B.C. V6T 1Z4 Canada

Editor: Geoff Holmes

Abstract

WEKA is a widely used, open-source machine learning platform. Due to its intuitive interface, it is particularly popular with novice users. However, such users often find it hard to identify the best approach for their particular dataset among the many available. We describe the new version of *Auto-WEKA*, a system designed to help such users by automatically searching through the joint space of WEKA's learning algorithms and their respective hyperparameter settings to maximize performance, using a state-of-the-art Bayesian optimization method. Our new package is tightly integrated with WEKA, making it just as accessible to end users as any other learning algorithm.

Keywords: Hyperparameter Optimization, Model Selection, Feature Selection

1. The Principles Behind Auto-WEKA

The WEKA machine learning software (Hall et al., 2009) puts state-of-the-art machine learning techniques at the disposal of even novice users. However, such users do not typically know how to choose among the dozens of machine learning procedures implemented in WEKA and each procedure's hyperparameter settings to achieve good performance.

Auto-WEKA[1] addresses this problem by treating all of WEKA as a single, highly parametric machine learning framework, and using Bayesian optimization to find a strong instantiation for a given dataset. Specifically, it considers the combined space of WEKA's learning algorithms $\mathcal{A} = \{A^{(1)}, \ldots, A^{(k)}\}$ and their associated hyperparameter spaces $\Lambda^{(1)}, \ldots, \Lambda^{(k)}$ and aims to identify the combination of algorithm $A^{(j)} \in \mathcal{A}$ and hyperparameters $\lambda \in \Lambda^{(j)}$ that minimizes cross-validation loss,

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

1. Thornton et al. (2013) first introduced Auto-WEKA and empirically demonstrated state-of-the-art performance. Here we describe an improved and more broadly accessible implementation of Auto-WEKA, focussing on usability and software design.

where $\mathcal{L}(A_\lambda, \mathcal{D}^{(i)}_{\text{train}}, \mathcal{D}^{(i)}_{\text{test}})$ denotes the loss achieved by algorithm A with hyperparameters λ when trained on $\mathcal{D}^{(i)}_{\text{train}}$ and evaluated on $\mathcal{D}^{(i)}_{\text{test}}$. We call this the *combined algorithm selection and hyperparameter optimization (CASH)* problem. CASH can be seen as a blackbox function optimization problem: determining $\text{argmin}_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} f(\boldsymbol{\theta})$, where each configuration $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ comprises the choice of algorithm $A^{(j)} \in \mathcal{A}$ and its hyperparameter settings $\lambda \in \Lambda^{(j)}$. In this formulation, the hyperparameters of algorithm $A^{(j)}$ are *conditional* on $A^{(j)}$ being selected. For a given $\boldsymbol{\theta}$ representing algorithm $A^{(j)} \in \mathcal{A}$ and hyperparameter settings $\lambda \in \Lambda^{(j)}$, $f(\boldsymbol{\theta})$ is then defined as the cross-validation loss $\frac{1}{k}\sum_{i=1}^{k} \mathcal{L}(A^{(j)}_\lambda, \mathcal{D}^{(i)}_{\text{train}}, \mathcal{D}^{(i)}_{\text{test}})$.[2]

Bayesian optimization (see, e.g., Brochu et al., 2010), also known as sequential model-based optimization, is an iterative method for solving such blackbox optimization problems. In its n-th iteration, it fits a probabilistic model based on the first $n-1$ function evaluations $\langle \boldsymbol{\theta}_i, f(\boldsymbol{\theta}_i)\rangle_{i=1}^{n-1}$, uses this model to select the next $\boldsymbol{\theta}_n$ to evaluate (trading off exploration of new parts of the space *vs* exploitation of regions known to be good) and evaluates $f(\boldsymbol{\theta}_n)$. While Bayesian optimization based on Gaussian process models is known to perform well for low-dimensional problems with numerical hyperparameters (see, e.g., Snoek et al., 2012), tree-based models have been shown to be more effective for high-dimensional, structured, and partly discrete problems (Eggensperger et al., 2013), such as the highly conditional space of WEKA's learning algorithms and their corresponding hyperparameters we face here.[3] Thornton et al. (2013) showed that tree-based Bayesian optimization methods yielded the best performance in Auto-WEKA, with the random-forest-based SMAC (Hutter et al., 2011) performing better than the tree-structured Parzen estimator, TPE (Bergstra et al., 2011). Auto-WEKA uses SMAC to determine the classifier with the best performance on the given data.

2. Auto-WEKA 2.0

Since the initial release of a usable research prototype in 2013, we have made substantial improvements to the Auto-WEKA package described by Thornton et al. (2013). At a prosaic level, we have fixed bugs, improved tests and documentation, and updated the software to work with the latest versions of WEKA and Java. We have also added four major features.

First, we now support regression algorithms, expanding Auto-WEKA beyond its previous focus on classification (starred entries in Fig. 1). Second, we now support the optimization of all performance metrics WEKA supports. Third, we now natively support parallel runs (on a single machine) to find good configurations faster and save the N best configurations of each run instead of just the single best. Fourth, Auto-WEKA 2.0 is now fully integrated with WEKA. This is important, because the crux of Auto-WEKA lies in its simplicity: providing a push-button interface that requires no knowledge about the available learning algorithms or their hyperparameters, asking the user to provide, in addition to the dataset to be processed, only a memory bound (1 GB by default) and the overall time

2. In fact, on top of machine learning algorithms and their respective hyperparameters, we also include attribute selection methods and their respective hyperparameters in the configurations $\boldsymbol{\theta}$, thereby jointly optimizing over their choice and the choice of algorithms.
3. Conditional dependencies can also be accommodated in the Gaussian process framework (Hutter and Osborne, 2013; Swersky et al., 2013), but currently, tree-based methods achieve better performance.

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

Figure 1: Learners and methods supported by Auto-WEKA 2.0, along with number of hyperparameters $|\Lambda|$. Every learner supports classification; starred learners also support regression.

budget available for the entire learning process.[4] The overall budget is set to 15 minutes by default to accommodate impatient users; longer runs allow the Bayesian optimizer to search the space more thoroughly; we recommend at least several hours for production runs.

The usability of the earlier research prototype was hampered by the fact that users had to download Auto-WEKA manually and run it separately from WEKA. In contrast, Auto-WEKA 2.0 is now available through WEKA's package manager. Users do not need to install software separately; everything is included in the package and installed automatically upon request. After installation, Auto-WEKA 2.0 can be used in two different ways:

1. *As a meta-classifier:* Auto-WEKA can be run like any other machine learning algorithm in WEKA: via the GUI, the command-line interface, or the public API. Figure 2 shows how to run it from the command line.
2. *Through the Auto-WEKA tab:* This provides a customized interface that hides some of the complexity. Figure 3 shows the output of an example run.

Source code for Auto-WEKA is hosted on GitHub (`https://github.com/automl/autoweka`) and is available under the GPL license (version 3). Releases are published to the WEKA package repository and available both through the WEKA package manager and from the Auto-WEKA project website (`http://automl.org/autoweka`). A manual describes how to use the WEKA package and gives a high-level overview for developers; we also provide lower-level Javadoc documentation. An issue tracker on GitHub, JUnit tests and the continuous integration system Travis facilitate bug tracking and correctness of the code. Since its release on March 1, 2016, Auto-WEKA 2.0 has been downloaded more than 15 000 times, with an average of about 400 downloads per week.

4. Internally, to avoid using all its budget for executing a single slow learner, Auto-WEKA limits individual runs of any learner to 1/12 of the overall budget; it further limits feature search to 1/60 of the budget.

```
java -cp autoweka.jar weka.classifiers.meta.AutoWEKAClassifier
    -timeLimit 5 -t iris.arff -no-cv
```

Figure 2: Command-line call for running Auto-WEKA with a time limit of 5 minutes on training dataset `iris.arff`. Auto-WEKA performs cross-validation internally, so we disable WEKA's cross-validation (`-no-cv`). Running with `-h` lists the available options.

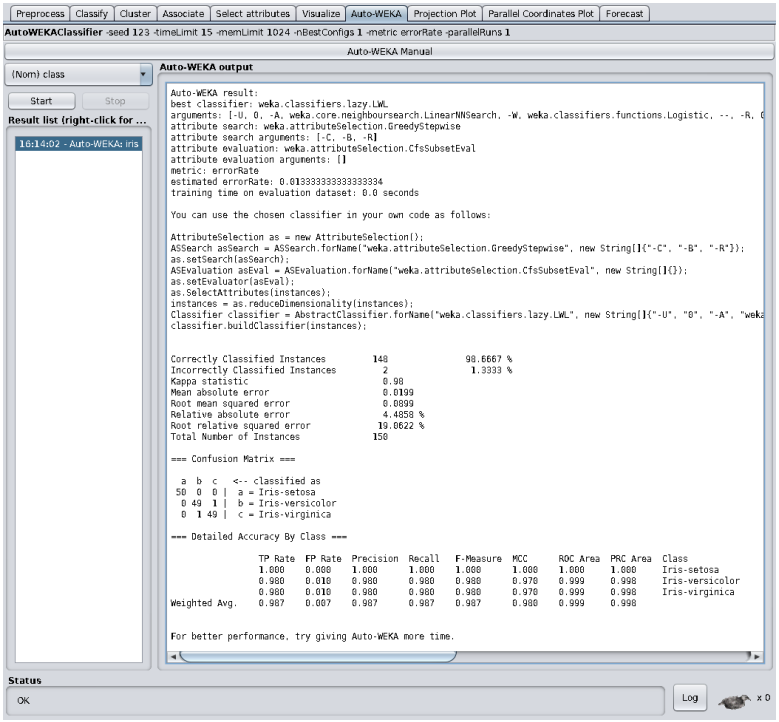

Figure 3: Example Auto-WEKA run on the iris dataset. The resulting best classifier along with its parameter settings is printed first, followed by its performance. While Auto-WEKA runs, it logs to the status bar how many configurations it has evaluated so far.

3. Related Implementations

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. This *automated machine learning (AutoML)* approach has recently also been applied to Python and scikit-learn (Pedregosa et al., 2011) in Auto-WEKA's sister package, *Auto-sklearn* (Feurer et al., 2015). Auto-sklearn uses the same Bayesian optimizer as Auto-WEKA, but comprises a smaller space of models and hyperparameters, since scikit-learn does not implement as many different machine learning techniques as WEKA; however, Auto-sklearn includes additional meta-learning techniques.

It is also possible to optimize hyperparameters using WEKA's own grid search and MultiSearch packages. However, these packages only permit tuning one learner and one filtering method at a time. Grid search handles only one hyperparameter. Furthermore, hyperparameter names and possible values have to be specified by the user.

References

J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems 24 (NIPS'11)*, pages 2546–2554, 2011.

E. Brochu, V. Cora, and N. de Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *Computing Research Repository (arXiv)*, abs/1012.2599, 2010.

K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. Hoos, and K. Leyton-Brown. Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In *NIPS Workshop on Bayesian Optimization (BayesOpt'13)*, 2013.

M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and Robust Automated Machine Learning. In *Advances in Neural Information Processing Systems 28 (NIPS'15)*, pages 2944–2952, 2015.

M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explor. Newsl.*, 11(1):10–18, Nov. 2009. ISSN 1931-0145.

F. Hutter and M. Osborne. A Kernel for Hierarchical Parameter Spaces. *Computing Research Repository (arXiv)*, abs/1310.5738, Oct. 2013.

F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential Model-Based Optimization for General Algorithm Configuration. In *Learning and Intelligent OptimizatioN Conference (LION 5)*, pages 507–523, 2011.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25 (NIPS'12)*, pages 2951–2959, 2012.

K. Swersky, D. Duvenaud, J. Snoek, F. Hutter, and M. Osborne. Raiders of the lost architecture: Kernels for Bayesian optimization in conditional parameter spaces. In *NIPS Workshop on Bayesian Optimization (BayesOpt'13)*, 2013.

C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *19th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'13)*, 2013.