as-asl: Algorithm Selection with Auto-sklearn

Brandon Malone BRANDON.MALONE@NECLAB.EU
NEC Laboratories Europe, Heidelberg, Germany

Kustaa Kangas JWKANGAS@MAPPI.HELSINKI.FI
Helsinki Institute for Information Technology HIIT
Department of Computer Science, Aalto University, Finland

Matti Järvisalo MATTI.JARVISALO@HELSINKI.FI
Helsinki Institute for Information Technology HIIT
Department of Computer Science, University of Helsinki, Finland

Mikko Koivisto MIKKO.KOIVISTO@HELSINKI.FI
Department of Computer Science, University of Helsinki, Finland

Petri Myllymäki PETRI.MYLLYMAKI@HELSINKI.FI
Helsinki Institute for Information Technology HIIT
Department of Computer Science, University of Helsinki, Finland

Editors: Marius Lindauer, Jan N. van Rijn and Lars Kotthoff

Abstract

In this paper, we describe our *algorithm selection with* Auto-sklearn (as-asl) software as it was entered in the 2017 Open Algorithm Selection Challenge. as-asl first selects informative sets of features and then uses those to predict distributions of algorithm runtimes. A classifier uses those predictions, as well as the informative features, to select an algorithm for each problem instance. Our source code is publicly available[1] with the permissive MIT license.

1. Introduction

The *algorithm selection* problem (Rice, 1976) entails selecting the best algorithm implementation (*solver*) to solve a given *instance* of a problem. In this work, we propose *algorithm selection with* Auto-sklearn (as-asl), a machine learning based software implementation to address this problem.[2]

In Section 2, we describe our approach to training the major components of as-asl. We then describe our computing infrastructure in Section 3 and conclude in Section 4.

2. Software Architecture

The Open Algorithm Selection Challenge (OASC) poses the algorithm selection problem as follows.[3]

1. https://github.com/bmmalone/as-asl
2. A preliminary version of this software was described in Malone et al. (2017).
3. An analogous formulation which focuses on solution quality is also posed.

INPUT: A training set of problem instances (represented as collections of feature values), the time required to observe sets of features, and the respective runtimes of a given set of solvers to find a solution.

TASK: Learn a *scheduler* function which constructs a schedule which minimizes the time required to find a solution on an unseen problem instance.

We adopt a three-step approach to learn a scheduler:

Step 1. Select informative feature sets.

Step 2. Train a model to select the best solver for a given instance.

Step 3. Select a presolver to run for a short amount of time before extracting features.

We describe each step in more detail.

2.1. Step 1: Feature set selection

First, we select sets of features to observe which lead to good algorithm selection performance. Intuitively, this step optimizes a tradeoff between the informativeness and cost of observation for the feature sets. AS-ASL selects informative features using a greedy, wrapper-based forward selection search (Guyon and Elisseeff, 2003) guided by the PAR10 metric; the quick AS-RF method described in Section 2.2 serves as the underlying model in the search.

2.2. Step 2: Solver selection

Second, we train a two-level stacking model (Wolpert, 1992) which selects the best solver for a given instance. The first level comprises one ensemble $\mathcal{E}_S$ for each solver S trained with AUTO-SKLEARN (Feurer et al., 2015) to predict its runtime (that is, these are regression models). For training, we use only those features selected during Step 1; in particular, the sets of observed features are fixed while learning the ensembles. The training objective is to minimize the root mean squared error of the predictions compared to the actual runtimes.

When presented with a new instance i, each member m_S of the ensemble predicts the runtime of the respective solver for that instance, $m_S(i)$; we propagate the uncertainty in these predictions by treating the true runtime i_S as a random variable whose mean and variance are the emprical mean and variance of the predictions $m_S(i)$ of all $m_S \in \mathcal{E}_S$.

A single ensemble also trained with AUTO-SKLEARN composes the second level; it is trained to predict the best solver from the selected features and runtime estimates from the regression models (that is, these are multi-class classifier models); specifically, the inputs for this model are the selected features from Step 1 and the mean and variance estimates from the solver runtime predictions. Each member of the ensemble is weighted according to its performance on the training data. The training objective is to minimize the multi-class micro F1 score (Lipton et al., 2014).

When presented with a new instance (and the predictions from the first level), the best solver is predicted based on a weighted voting scheme. We note that similar techniques as those for the ensembles for runtime prediction could be used to quantify uncertainty in the solver selection as well; however, at present AS-ASL does not incorporate this uncertainty.

Kotthoff (2012) proposed a similar stacking model architecture for algorithm selection. Our approach differs from it in several ways. First, we include the orginal features in the input to the classifier in the second level. Second, we explicitly account for the uncertainty in the runtime estimates from the first layer by representing them as distributions rather than point estimates.

as-rf For OASC, many of the scenarios included several dozen solvers; due to the computational expense of AUTO-SKLEARN (see Section 3), training complete ensembles during the feature set selection is not feasible. Thus, we also implemented a version of the stacking model approach using random forests with standard hyperparameters in place of the AUTO-SKLEARN ensembles; we refer to this method as AS-RF.

2.3. Step 3: Presolver selection

Finally, we select a presolver to run for a short amount of time in hopes of finding a solution quickly before even observing instance features. We use a grid search to optimize the choice of a single presolver and allocated time for presolving with respect to the PAR10 score.

In particular, we consider the selected features and solver selection model from Steps 1 and 2 fixed; each solver is then given a fixed time budget for presolving. We evaluate several time budgets, including no time for presolving, and select the presolver and budget which optimize the PAR10 score.

As described in Section 2.4, running the selected presolver is the *first* step in our schedule; however, since the efficacy of the presolver is dependent on the cost of observing the features and the quality of solver selections afterward, choosing the presolver and its budget is the *last* step in our learning algorithm.

2.4. Solver schedule selection

We use the above three steps during training to: (1) choose a presolver and its budget; (2) select sets of features to observe; and (3) create a model to select a solver based on an instance's features. AS-ASL creates a solver schedule for a new test instance using these three components as follows:

1. Run the presolver for the selected time budget. If the presolver solves the instance within the time budget, then AS-ASL does not perform any more work for that instance.
2. Observe the selected sets of features.
3. Run the selected solver for the remaining time budget.

3. Computing Infrastructure and Resources

We trained all models on Dell PowerEdge M610 computing nodes equipped with two 2.53-GHz Intel Xeon E5540 (quad-core) CPUs and 32 GB RAM. We limited total AUTO-SKLEARN training to 10 minutes per model, and we restricted training to at most 8 active threads at once.

The two primary factors affecting the empirical time complexity of our approach are: (1) the number of feature sets, which affects the size of the search space in Step 1; and (2)

the number of solvers, which determines the number of models required in Step 2. Step 3 can be performed rather quickly with a modest number of presolver time budgets. Typical runs for OASC required on the order of one to two hours.

The AS-ASL software is freely available[4] with the MIT license; installation is documented on GitHub.

4. Discussion

In this paper, we have described AS-ASL for solving the algorithm selection problem. It first identifies informative features and then trains a stacking model to select the best solver for each instance. Additionally, AS-ASL selects a presolver when beneficial. The implementation is freely available and a set of simple commands allow for easy execution.

Acknowledgements

This work is supported by Academy of Finland, grants #125637, #251170 (COIN Centre of Excellence in Computational Inference Research), #255675, #276412, and #284591; Finnish Funding Agency for Technology and Innovation (project D2I); and Research Funds of the University of Helsinki.

References

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, pages 2962–2970, 2015.

I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

L. Kotthoff. Hybrid regression-classification models for algorithm selection. In *Proceedings of the* 20^{th} *European Conference on Artificial Intelligence*, pages 480–485, 2012.

Z. C. Lipton, C. Elkan, and B. Naryanaswamy. Optimal thresholding of classifiers to maximize F1 measure. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 225–239, 2014.

B. Malone, K. Kangas, M. Järvisalo, M. Koivisto, and P. Myllymäki. Empirical hardness of finding optimal Bayesian network structures: Algorithm selection and runtime prediction. *Machine Learning*, To Appear, 2017.

J. R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

D. H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.

4. https://github.com/bmmalone/as-asl