A brief Review of the ChaLearn AutoML Challenge: 
Any-time Any-dataset Learning without Human Intervention

Isabelle Guyon U. Paris-Saclay, France, and ChaLearn, USA guyon@chalearn.org
Imad Chaabane U. Paris-Saclay, France imad-eddine.chaabane@u-psud.fr
Hugo Jair Escalante INAOE, Puebla, Mexico hugo.jair@gmail.com
Sergio Escalera U. Barcelona and Computer Vision Center, Spain sergio@maia.ub.es
Damir Jajetic IN2, Croatia damir.jajetic@in2.hr
James Robert Lloyd Qlearsite JAMES.ROBERT.LLOYD@GMAIL.COM
Núria Macià Universitat d’Andorra, Andorra macia.nuria@gmail.com
Bisakha Ray NYU School of Medicine, USA bisakha.ray@nyumc.org
Lukasz Romaszko University of Edinburgh, UK lukasz.romaszko@gmail.com
Michele Sebag U. Paris-Saclay, France michele.sebag@lri.fr
Alexander Statnikov NYU School of Medicine, USA statnikov@gmail.com
Sébastien Treguer Paris, France streguer@gmail.com
Evelyne Viegas Microsoft Research, USA evelynev@microsoft.com

Abstract

The ChaLearn AutoML Challenge team conducted a large scale evaluation of fully automatic, black-box learning machines for feature-based classification and regression problems. The test bed was composed of 30 data sets from a wide variety of application domains and ranged across different types of complexity. Over six rounds, participants succeeded in delivering AutoML software capable of being trained and tested without human intervention. Although improvements can still be made to close the gap between human-tweaked and AutoML models, this competition contributes to the development of fully automated environments by challenging practitioners to solve problems under specific constraints and sharing their approaches; the platform will remain available for post-challenge submissions at http://codalab.org/AutoML.

Keywords: AutoML Challenge, machine learning, model selection, meta-learning, representation learning, active learning

1. Introduction

Within ten years most expert-level predictive analytics/data science tasks will be automated according to a recent KD Nuggets poll (Piatetsky, 2015). In fact, domain-specific automated software has been in existence for a while (e.g., KXEN founded in 1998 and recently acquired by SAP) and many toolkits/APIs have lately flourished (e.g., Google Prediction API, Google CloudML, AzureML, BigML, Dataiku, DataRobot, KNIME, and RapidMiner) claiming easy-to-use or almost fully automated model construction. In the field of machine learning, practitioners use the Java-based toolkit Weka, the Matlab-based toolkit Spider (and its extension CLOP), the statistics platform R or the recent Python-based library scikit-learn.

© 2016 I. Guyon et al.
The AutoML challenge contributes to the transition towards a fully automated learning suite by testing machine learning code operated without any human intervention under strict execution time and memory usage constraints.

The purpose of this paper is to briefly describe the AutoML challenge and share the insights on how participants addressed such challenge.

2. The AutoML Challenge

The AutoML competition challenged participants’ code to be completely blind tested. The goal was to design a learning machine capable of performing all model selection and hyperparameter tuning without any human intervention over six rounds that included 30 data sets. (The data sets are now available at the website[^1].) This required machine learning and software engineering skills. The tasks were limited to (a) tabular data, i.e., examples were fixed-length feature vectors and (b) supervised learning problems, i.e., classification and regression tasks. Yet the participants had to face (a) data sets with a wide range of difficulties such as class imbalance, sparsity, missing values, and categorical variables, (b) training sets of diverse dimensionality in terms of number of instances and features, and (c) hardware constraints, i.e., train and test models within 20 minutes without running out of memory; memory was increased from 24 GB to 56 GB after phase AutoML3. The rounds alternated AutoML phases—in which submitted code was blind tested on the Codalab platform using unseen data sets and under limited time—and Tweakathon phases—in which the participants could improve their methods by tweaking them on those same data sets. During Tweakathon, the participants were free to use their own computational resources.

The challenge used the open-source CodaLab platform since it allowed to organize competitions with code submission and conduct fair evaluations. In order to encourage participants to try GPUs and deep learning, a GPU track sponsored by NVIDIA was included in Round 4. The reader is referred to (Guyon et al., 2015) for a detailed description of the challenge.

3. Results

Principled optimizers to search model space, which include Sequential Model-based Algorithm Configuration (SMAC) (Hutter et al., 2011) and hyperopt (Bergstra et al., 2013), combined with Weka (the Auto-WEKA software (Thornton et al., 2013)) or scikit-learn (the hyperopt-sklearn software (Komer et al., 2014)) were promising candidate solutions for the AutoML challenge. Although they have been great source of inspiration, this challenge turned out to be harder than it looked at first sight. Of hundreds of participants, only few passed the first rounds, which were of limited difficulty. The challenge experienced a critical turning point in Round 3 when sparse data were introduced; all but one participant (D. Jajetic) failed in the AutoML3 blind evaluation. Fortunately, the crowd recovered and the last rounds had enough successful teams to attribute all prizes.

Table 1 shows the results on the test set in the AutoML phases (blind testing) and the Final phases (one time testing on the test set revealed at the end of the Tweakathon phases). Ties were broken by giving preference to the first submission. Team names are abbreviated as follows.

[^1]: [http://automl.chalearn.org/data](http://automl.chalearn.org/data)
Table 1: **Results of the AutoML challenge winners.** \(< R >\) is the average rank over all five data sets of the round and was used to rank the participants. \(< S >\) is the average score over the five data sets of the round. UP is the percent increase in performance between the average performance of the winners in the AutoML phase and the Final phase of the same round.

<table>
<thead>
<tr>
<th>Rnd</th>
<th>Ended</th>
<th>AutoML</th>
<th>Final</th>
<th>UP (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Winners</td>
<td>&lt; R &gt;</td>
<td>&lt; S &gt;</td>
</tr>
<tr>
<td>0</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>02/15/15</td>
<td>1. aad</td>
<td>2.80</td>
<td>0.6401</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. jrl44</td>
<td>3.80</td>
<td>0.6226</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. tadej</td>
<td>4.20</td>
<td>0.6456</td>
</tr>
<tr>
<td>2</td>
<td>06/15/15</td>
<td>1. jrl44</td>
<td>1.80</td>
<td>0.4320</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. aad</td>
<td>3.40</td>
<td>0.3529</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. mat</td>
<td>4.40</td>
<td>0.3449</td>
</tr>
<tr>
<td>3</td>
<td>11/15/15</td>
<td>1. djaj</td>
<td>2.40</td>
<td>0.0901</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>4</td>
<td>02/20/16</td>
<td>1. aad</td>
<td>2.20</td>
<td>0.3881</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. djaj</td>
<td>2.20</td>
<td>0.3841</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. marc</td>
<td>2.60</td>
<td>0.3815</td>
</tr>
<tr>
<td>5</td>
<td>05/1/16</td>
<td>1. aad</td>
<td>1.60</td>
<td>0.5282</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. djaj</td>
<td>2.60</td>
<td>0.5379</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. post</td>
<td>4.60</td>
<td>0.4150</td>
</tr>
</tbody>
</table>

aad=aad_freiburg     djaj=djajetic     marc=marc.boule     tadej=tadejs
ahhh=ahhishek4      ideal=ideal.intel.analytics    mat=matthias.vonrohr
asml=asml.intel.com     jrl44=backstreet.bayes    post=postech.mlg.exbrain

AAD Freiburg is the overall winner, having totaled the largest number of wins in all phases; they won 3 out of 5 AutoML phases. The team, led by F. Hutter who codeveloped SMAC and Auto-WEKA, delivered a new tool called AutoSklearn (Feurer et al., 2015b). It is worth noting that the Codalab platform favored Python programming since the environment had Python installed and the sample code was written in Python. However, any Linux executable could be submitted. M. Boullé, for instance, submitted an executable of Khiops, the Orange Labs’ automatic tool for mining large databases.

It was also possible to enter the challenge without submitting code; participants could run their learning techniques on their own local environments and submit the results only. Following each AutoML phase, new data sets were released (labeled training set, unlabeled validation set, and test set), and participants could manually tune their models for over a month during the Tweakathon phases. Having a proprietary solution, the Intel team, led by E. Tuv, entered the no-code-releasing track. In this “free style” part of the competition, both the AAD Freiburg and the Intel teams were on equal par; they were always ranking in the top 3. Counting 1 point for the 3rd place, 2 for the 2nd, and 3 for the 1st, both scored 11 points. Interestingly, although both teams used ensemble methods, their approaches were radically different. The Intel team simply relied on their own C++ implementation of
CART-style tree-based methods for the feature selection and ensemble learning. Conversely, the AAD Freiburg team devised heterogeneous ensembles of predictors based on scikit-learn pipelines, using a combination of meta-learning and Bayesian hyper-parameter optimization. While we have no way of comparing computational efficiency, it is likely that Intel is much faster. However, the organizers are proud of the ADD Freiburg team for making their solution publicly available, encouraging this practice in future challenges.

4. Insights on the AutoML Solutions

4.1. Survey Analysis

Twenty-eight teams responded to a survey we conducted on methods used in the challenge. **Preprocessing.** Preprocessing consisted in normalization, feature extraction, and dimensionality reduction. About one half of the respondents performed classical preprocessing steps, including feature standardization, sample normalization, and replacement of missing values. This is consistent with the frequent use of ensembles of decision trees based on decision thresholds, which do not require complex preprocessing. Other preprocessing steps included grouping modalities for categorical variables (20%) and discretization (4%). Few participants also reported having used non-linear transforms such as log. Most participants did not perform any feature engineering, which can largely be explained by the fact that they did not know the application domain of the data sets. Those who reported using feature extraction either relied on the (embedded) feature learning of their algorithm (21%) or applied random functions (36%). More than 2/3 of the participants used dimensionality reduction, linear manifold transformations (e.g., PCA, ICA) being the most popular (43%). About 1/3 used feature selection alone. Other methods included non-linear dimensionality reduction (e.g., KPCA, MDS, LLE, Laplacian Eigenmaps) and clustering (e.g., K-means).

**Predictor.** The methods most frequently used involved (ensembles of) decision trees; 75% of the participants reported having used them, alone or in combination with other methods. The challenge setting lent itself well to such methods because each individual base learner trains rapidly and performance improves by increasing the number of learners, making such methods ideal any-time learning machines. Almost 1/2 of the participants used linear methods and about 1/3 used at least one of the following methods: Neural Nets, Nearest Neighbor, and Naive Bayes. The logistic loss was frequently used (75%). This may be due to the fact that producing probability-like scores is the most versatile when it comes to being able to be judged with a variety of loss functions. About 2/3 of the participants reported having used knowingly some form of regularization; two-norm regularization was slightly more popular than one-norm regularization.

**Model selection and ensembling.** About 2/3 of the respondents used one form of cross-validation for model selection; the rest used just the leaderboard. This may be due to the fact that the validation sets were not small for the most part. While K-fold cross-validation and leave-one-out remain the most popular, 20% of the respondents used the out-of-bag estimator of bagging methods and 10% used bi-level optimization methods. 4% reported transferring knowledge from phase to phase. However, such a strategy may be worth considering since both winners of phase AutoML5 used it. Only 18% of the respondents did not choose ensemble methods. For those who did, boosting and bagging were the most common—60% reported having used one of the two.
Implementation. Most respondents could not reliably evaluate how their methods scaled computationally. We are at least assured that they delivered results in less than 20 minutes on every data set, because this was the time limit for the execution. Most respondents claimed to have developed a simple method, easy to implement and parallelize (75% used multi-processor machines, 32% used algorithms run in parallel on different machines), but few claimed that their method was original or principled, and most relied on third-party libraries; scikit-learn, which was used in the starting kit, was frequently used. Luckily, this also resulted in code that was made available as open source—with only 10% exceptions. Python was used by 82% of the respondents. This is also explained by the fact that the starting kit was in Python. Although Codalab allows participants to submit any Linux executable, the organizers provided no support for this. Even then, 25% used at least one of the following languages: C/C++, Java, or R, sometimes in combination with Python. The fact that the Codalab backend ran on Linux may also explain that 86% of the respondents ran on Linux; others used Windows or MacOS. Memory consumption was generally high (more than half of the respondents used between 8 and 32 GB, and 18% used more than 32 GB). Indeed, when we introduced sparse data in Round 3, the sample code was memory demanding and we had to increase the memory on the server up to 56 GB. Unfortunately, this remained a problem until the end of the challenge—which we traced to an inefficient implementation of the data reader and of Random Forest for sparse matrices.

4.2. Best Methods

This section reviews the solutions of the two top ranking participants.

The proprietary solution of the Intel team is a fast implementation of tree-based methods in C/C++, which was developed to drive acceleration of yield learning in semiconductor process development. Using this software, the Intel team consistently has ranked high in ChaLearn challenges since 2003. The method is based on gradient boosting of trees built on a random subspace dynamically adjusted to reflect learned features relevance. A Huber loss function is used. No pre-processing was done, except for feature selection (Tuv et al., 2009). The classification method called Stochastic Gradient Tree and Feature Boosting selects a small sample of features at every step of the ensemble construction. The sampling distribution is modified at every iteration to promote more relevant features.

The open-source solution of the AAD Freiburg team uses a heterogeneous ensemble of learning machines (AutoSklearn (Feurer et al., 2015a,c)) combining the machine learning library scikit-learn (Pedregosa et al., 2011) with the state-of-the-art SMBO method SMAC to find suitable machine learning pipelines for a data set at hand. This is essentially a reimplementation of Auto-WEKA. To speed up the optimization process they employed a meta-learning technique (Feurer et al., 2015b) which starts SMAC from promising configurations of scikit-learn. Furthermore, they used the outputs of all models and combined these into an ensemble using ensemble selection. Their latest implementation uses the new version of SMAC (SMA) of Bayesian Optimization with Random Forests applied to a flexible configuration space describing scikit-learn. For the GPU version, they used the Java version of SMAC to tune AutoSklearn and deep neural networks implemented in Lasagne/Theano (Dieleman et al., 2015; Theano Development Team, 2016).
4.3. Other Notable Contributions

This section briefly reviews other solutions that ranked in the top 3 multiple times.

**Freeze Thaw Ensemble Construction** (Lloyd) of J. Lloyd (a.k.a. jrl44 and backstreet.bayes) is a modified version of the Freeze Thaw Bayesian optimization algorithm (Swerksky et al., 2014) for ensemble construction. The strategy is to keep training the most promising members of an ensemble, while freezing the least promising ones, which may be thawed later. Probabilistic models based on Gaussian processes and decision trees are used to predict which ensemble member should be trained further. Joining late in the challenge, L. Sun made an entry in AutoML5 that ranked third using a similar approach.

**AutoCompete** of Thakur and Krohn-Grimberghe (2015) is an automated machine learning framework for tackling Machine Learning competitions. Their solution performed well in late rounds of the AutoML challenge and won the GPU track. The pipeline includes (1) stratified data splitting, (2) building features, (3) feature selection, (4) performing model and hyper-parameter selection (Random Forests, Logistic Regression, Ridge Regression, Lasso, SVM, Naive Bayes, and Nearest Neighbors), and (5) ensembling solutions. Search space is specified with prior knowledge on similar data sets (a form of meta-learning). Thakur found that this strategy is faster and yields comparable results to hyperopt. The underlying implementation is based purely on Python and scikit-learn with some modules in Cython. Their GPU solution is an advanced version of the AutoCompete solution, which uses Neural Networks built with Keras (Chollet, 2015).

**Djajetic** (Jajetic, a) of D. Jajetic is based on heterogeneous ensembles of models obtained by searching through model-space and adjusting hyper-parameters without any communication between models. Jajetic believes that this makes search more effective in non-convex search spaces. This strategy lends itself well to efficient and simple parallelization. The search space and ensembling properties for each individual model is defined in a separate Python script. Each model is trained and explores its own parameter space and only communicates its training error and best prediction results to the outside. The ensembling module operates in a hierarchical manner. It uses only the N best hyper-parameter settings from each model, based on the training error, and only M best models from each model group. For the GPU track, Jajetic used a Neural Network (Jajetic, b) based on the Lasagne and Theano libraries.

5. Further work

We are currently analyzing the behavior of those solutions that run on all 30 data sets using systematic experiments. Our analysis will investigate relationships between algorithm and data complexity. We see opportunities for improvement because there are significant differences between Tweakathon and AutoML results, as shown in Table 1. The last column refers to the percent increase in test performance between the first trial in the AutoML phase (blind testing) and the second trial (end of Tweakathon, which is a one-month period of human model search). Except for Round 3, the improvements gained by human intervention are 15-35%—which can be due to external factors such as computational power. Hence, the learning schemas can still be optimized to close this gap. Other directions remain to be explored, including use of non vectorial data, non i.i.d. data, unsegmented data, and semi-supervised problems, for which devising fair benchmarks is challenging.
Acknowledgements

We gratefully acknowledge those that have participated as organizers or sponsors, see https://competitions.codalab.org/competitions/2321#learn_the_details-credits. Codalab runs on Azure and we are also grateful to Microsoft for supporting this challenge.

References


