JMLR: Workshop and Conference Proceedings 64:21-30, 2016 ICML 2016 AutoML Workshop

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.

1. Introduction

Within ten years most expert-level predictive analytics/data science tasks will be automated according to a recent KDNuggets poll (Piatetsky, 2015). In fact, domain-specific automated software has been in existence for a while 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 machine learning, practitioners use Java-based Weka, Matlab-based Spider/CLOP, stats-oriented R or Python-based scikit-learn. The purpose of this paper is to describe the results of the AutoML challenge, a benchmark for automated machine learning systems that can be operated without any human intervention, under strict execution time and memory usage constraints.

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 hyper-parameter tuning without any human intervention over six rounds that included 30 data sets[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 us organizing a competition with code submission thus conducting a fair evaluation with uniform hardware resources. 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., 2015a,b) 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.

aad=aad_freiburg	djaj=djajetic	marc=marc.boulle	tadej=tadejs
abhi=abhishek4	ideal=ideal.intel.analytics	mat=matthias.vonrohr	lisheng=lise_sun
asml=amsl.intel.com	jlr44 = backstreet.bayes	post = postech.mlg_exbrain	ref=reference

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).

1. 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.

	AutoML				**Final**				
Rnd	Ended	Winners	$< R >$	$< S >$	Ended	Winners	$< R >$	$< S >$	UP (%)
0	NA	NA	NA	NA	02/14/15	1. ideal 2. abhi 3. aad	1.40 3.60 4.00	0.8159 0.7764 0.7714	NA
1	02/15/15	1. aad 2. jrl44 3. tadej	2.80 3.80 4.20	0.6401 0.6226 0.6456	06/14/15	1. aad 2. ideal 3. amsl	2.20 3.20 4.60	0.7479 0.7324 0.7158	15
2	06/15/15	1. jrl44 2. aad 3. mat	1.80 3.40 4.40	0.4320 0.3529 0.3449	11/14/15	1. ideal 2. djaj 3. aad	2.00 2.20 3.20	0.5180 0.5142 0.4977	35
3	11/15/15	1. djaj 2. NA 3. NA	2.40 NA NA	0.0901 NA NA	02/19/16	1. aad 2. djaj 3. ideal	1.80 2.00 3.80	0.8071 0.7912 0.7547	481
4	02/20/16	1. aad 2. djaj 3. marc	2.20 2.20 2.60	0.3881 0.3841 0.3815	05/1/16	1. aad 2. ideal 3. abhi	1.60 3.60 5.40	0.5238 0.4998 0.4911	31
GPU	NA	NA	NA	NA	05/1/16	1. abhi 2. djaj 3. aad	5.60 6.20 6.20	0.4913 0.4900 0.4884	NA
5	05/1/16	1. aad 2. djaj 3. post	1.60 2.60 4.60	0.5282 0.5379 0.4150	NA	NA	NA	NA	NA

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.

Table 2: **Systematic study:** The team abbreviations are given in Table 1. The colors indicate the rounds.

Datasets	aad	abhi	djaj	ideal	jrl44	lisheng	marc	ref
ADULT	0.82	0.82	0.81	0.83	0.81	0.8	0.81	0.82
CADATA	0.8	0.79	0.78	0.81	0.09	0.79	0.64	0.76
DIGITS	0.95	0.94	0.83	0.96	0.73	0.95	0.86	0.87
DOROTHEA	0.66	0.87	0.82	0.89	0.82	0.84	0.79	0.7
NEWSGROUPS	0.48	0.46	0.64	0.59	0.33	0.05	0.38	0.56
CHRISTINE	0.49	0.46	0.48	0.55	0.48	0.46	0.45	0.42
JASMINE	0.63	0.61	0.62	0.65	0.62	0.61	0.56	0.56
MADELINE	0.82	0.59	0.64	0.81	0.57	0.58	0.18	0.53
PHILIPPINE	0.66	0.53	0.52	0.72	0.52	0.52	0.45	0.51
SYLVINE	0.9	0.87	0.89	0.93	0.89	0.87	0.83	0.89
ALBERT	0.38	0.32	0.36	0.37	0.32	0.34	0.35	0.32
DILBERT	0.94	0.79	0.75	0.98	0.21	0.24	0.46	0.79
FABERT	0.36	0.19	0.33	0.35	0.03	0.18	0.21	0.24
ROBERT	0.46	0.33	0.33	0.51	0.21	0.4	0.37	0.36
VOLKERT	0.33	0.26	0.28	0.37	0.11	0.15	0.14	0.25
ALEXIS	0.75	0.65	0.67	0.76	0.62	0.68	0.62	0.64
DIONIS	0.9	0.32	0.75	0.93	0.02	0.87	0.81	0.31
GRIGORIS	0.73	0.76	0.8	0.97	0.54	0.88	0.96	0.75
JANNIS	0.55	0.38	0.41	0.42	0.24	0.36	0.39	0.4
WALLIS	0.71	0.63	0.74	0.71	0.12	0.23	0.58	0.62
EVITA	0.59	0.59	0.58	0.61	0.59	0.59	0.52	0.41
FLORA	0.5	0.51	0.5	0.53	0.02	0.42	0.51	0.37
HELENA	0.22	0.23	0.15	0.25	0.06	0.2	0.19	0.08
TANIA	0.47	0.76	0.39	0.73	0.53	0.6	0.66	0.54
YOLANDA	0.32	0.37	0.29	0.39	0.02	0.24	0.19	0.26
ARTURO	0.75	0.8	0.78	0.77	0.3	0.72	0.7	0.77
CARLO	0.45	0.37	0.43	0.18	0.36	0.4	0.37	0.14
MARCO	0.55	0.71	0.69	0.54	0.66	0.54	0.68	0.25
PABLO	0.3	0.29	0.31	0.27	0.03	0.29	0.25	0.28
WALDO	0.59	0.56	0.57	0.61	0.56	0.56	0.46	0.56

4. Systematic study

We conducted a systematic study (Table 2, Figure 1). These results evaluating the "empirical difficulty" of the datasets have been thoroughly analyzed (Chaabane et al., 2016), but did not reveal obvious patterns relating empirical difficulty and classical metrics of complexity, such as the ratio number of features over number of training examples.

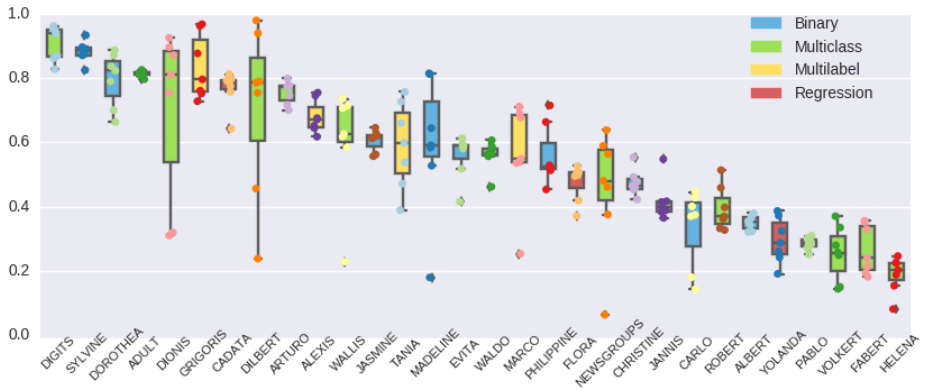

Figure 1: **Systematic study result distribution.** We represent the distribution of results of Table 2. The datasets are ordered with decreasing median test score.

5. Insights on the AutoML Solutions

5.1. 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 that 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.

5.2. 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 AAD Freiburg 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 (Hutter et al.) of Bayesian Optimization with Random Forests applied to a flexible configuration space describing scikit-learn. For the GPU version (Mendoza et al., 2016), 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).

5.3. 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 (Swersky 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 (Sun, 2016).
AutoCompete of Thakur and Krohn-Grimberghe (2015) is an automated machine learning framework for tackling Machine Learning competitions. This solution performed well in late rounds of the AutoML challenge and won the GPU track (Thakur, 2016). 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, 2016a) is based on heterogeneous ensembles of models obtained by searching through model-space and adjusting hyper-parameters (HP) 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 HP 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, 2016b) based on the Lasagne and Theano libraries.

6. Conclusion and further work

This first AutoML challenge is a stepping stone towards attaining fully automated machine learning, which will require the organization of more elaborate benchmarks (with less constraints on data representation and type of tasks). Practical solutions were obtained and open-sourced, such as the solution of the winners (Feurer et al., 2015b). On the tasks of this challenge, ensemble learning is the big winner, but it is unclear whether homogeneous ensembles of trees or heterogenous ensembles are preferable. There is still room for improvement, as revealed by the significant differences remaining between Tweakathon and AutoML results (Table 1). Except for Round 3, human intervention and additional compute power yielded a 15 to 35% improvement. Hence, the learning schemas can still be optimized to close this gap. An upcoming more detailed paper will study more in depth the relationships between algorithm and data complexity.

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.

Rnd	DATASET	Task	Metric	Time	C	Cbal	Sparse	Miss	Cat	Irr	Pte	Pva	Ptr	N	Ptr/N
0	1 ADULT	multilabel	F1	300	3	1	0.16	0.011	1	0.5	9768	4884	34190	24	1424.58
0	2 CADATA	regression	R2	200	0	NaN	0	0	0	0.5	10640	5000	5000	16	312.5
0	3 DIGITS	multiclass	BAC	300	10	1	0.42	0	0	0.5	35000	20000	15000	1568	9.57
0	4 DOROTHEA	binary	AUC	100	2	0.46	0.99	0	0	0.5	800	350	800	100000	0.01
0	5 NEWSGROUPS	multiclass	PAC	300	20	1	1	0	0	0	3755	1877	13142	61188	0.21
1	1 CHRISTINE	binary	BAC	1200	2	1	0.071	0	0	0.5	2084	834	5418	1636	3.31
1	2 JASMINE	binary	BAC	1200	2	1	0.78	0	0	0.5	1756	526	2984	144	20.72
1	3 MADELINE	binary	BAC	1200	2	1	1.2e-06	0	0	0.92	3240	1080	3140	259	12.12
1	4 PHILIPPINE	binary	BAC	1200	2	1	0.0012	0	0	0.5	4664	1166	5832	308	18.94
1	5 SYLVINE	binary	BAC	1200	2	1	0.01	0	0	0.5	10244	5124	5124	20	256.2
2	1 ALBERT	binary	F1	1200	2	1	0.049	0.14	1	0.5	51048	25526	425240	78	5451.79
2	2 DILBERT	multiclass	PAC	1200	5	1	0	0	0	0.16	9720	4860	10000	2000	5
2	3 FABERT	multiclass	PAC	1200	7	0.96	0.99	0	0	0.5	2354	1177	8237	800	10.3
2	4 ROBERT	multiclass	BAC	1200	10	1	0.01	0	0	0	5000	2000	10000	7200	1.39
2	5 VOLKERT	multiclass	PAC	1200	10	0.89	0.34	0	0	0	7000	3500	58310	180	323.94
3	1 ALEXIS	multilabel	AUC	1200	18	0.92	0.98	0	0	0	15569	7784	54491	5000	10.9
3	2 DIONIS	multiclass	BAC	1200	355	1	0.11	0	0	0	12000	6000	416188	60	6936.47
3	3 GRIGORIS	multilabel	AUC	1200	91	0.87	1	0	0	0	9920	6486	45400	301561	0.15
3	4 JANNIS	multiclass	BAC	1200	4	0.8	7.3e-05	0	0	0.5	9851	4926	83733	54	1550.61
3	5 WALLIS	multiclass	AUC	1200	11	0.91	1	0	0	0	8196	4098	10000	193731	0.05
4	1 EVITA	binary	AUC	1200	2	0.21	0.91	0	0	0.46	14000	8000	20000	3000	6.67
4	2 FLORA	regression	ABS	1200	0	NaN	0.99	0	0	0.25	2000	2000	15000	200000	0.08
4	3 HELENA	multiclass	BAC	1200	100	0.9	6e-05	0	0	0	18628	9314	65196	27	2414.67
4	4 TANIA	multilabel	PAC	1200	95	0.79	1	0	0	0	44635	22514	157599	47236	3.34
4	5 YOLANDA	regression	R2	1200	0	NaN	1e-07	0	0	0.1	30000	30000	400000	100	4000
5	1 ARTURO	multiclass	F1	1200	20	1	0.82	0	0	0.5	2733	1366	9565	400	23.91
5	2 CARLO	binary	PAC	1200	2	0.097	0.0027	0	0	0.5	10000	10000	50000	1070	46.73
5	3 MARCO	multilabel	AUC	1200	24	0.76	0.99	0	0	0	20482	20482	163860	15299	10.71
5	4 PABLO	regression	ABS	1200	0	NaN	0.11	0	0	0.5	23565	23565	188524	120	1571.03
5	5 WALDO	multiclass	BAC	1200	4	1	0.029	0	1	0.5	2430	2430	19439	270	72

Table 3: **Datasets of the AutoML challenge.** C=number of classes. Cbal=class balance. Sparse=sparsity. Miss=fraction of missing values. Cat=categorical variables. Irr=fraction of irrelevant variables. Pte, Pva, Ptr=number of examples of the test, validation, and training sets, respectively. N=number of features. Ptr/N=aspect ratio of the dataset.

References

J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the International Conference on Machine Learning*, volume 28, pages 115–123, 2013.

I. Chaabane, I. Guyon, and M. Sebag. Automl challenge result collection technical memorandum. https://www.authorea.com/users/94951/articles/114450/_show_article?access_token=I_fJAKswMQ5FpNT8WMQn_w, July 2016.

F. Chollet. Keras. https://github.com/fchollet/keras, 2015.

S. Dieleman, J. Schlter, C. Raffel, E. Olson, S. K. Snderby, D. Nouri, D. Maturana, M. Thoma, E. Battenberg, J. Kelly, J. De Fauw, M. Heilman, diogo149, B. McFee, H. Weideman, takacsg84, peterderivaz, Jon, instagibbs, Dr. Kashif Rasul, CongLiu, Britefury, and J. Degrave. Lasagne: First release. http://dx.doi.org/10.5281/zenodo.27878, August 2015.

M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Methods for improving bayesian optimization for automl. In *Proceedings of the International Conference on Machine Learning 2015, Workshop on Automatic Machine Learning*, 2015a.

M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In *Proceedings of the Neural Information Processing Systems*, pages 2962–2970. 2015b. URL https://github.com/automl/auto-sklearn.

M. Feurer, J.T. Springenberg, and F. Hutter. Initializing bayesian hyperparameter optimization via meta-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1128–1135, 2015c.

I. Guyon, K. Bennett, G. Cawley, H. J. Escalante, S. Escalera, T. Kam Ho, N. Macià, B. Ray, M. Saeed, A. Statnikov, and E. Viegas. AutoML challenge 2015: Design and first results. In *AutoML workshop*, 2015a. URL https://drive.google.com/file/d/0BzRGLkqgrI-qWkpzcGw4bFpBMUk/view.

I. Guyon, K. Bennett, G. Cawley, H. J. Escalante, S. Escalera, Tin Kam Ho, N. Macià, B. Ray, M. Saeed, A. Statnikov, and E. Viegas. Design of the 2015 ChaLearn AutoML challenge. In *Proceedings of the International Joint Conference on Neural Networks*, 2015b. URL http://www.causality.inf.ethz.ch/AutoML/automl_ijcnn15.pdf.

F. Hutter, H. Hoos, K. Murphy, and S. Ramage. Sequential Model-based Algorithm Configuration (SMAC). http://www.cs.ubc.ca/labs/beta/Projects/SMAC/.

F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the International Conference on Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.

D. Jajetic. Djajetic Implementation. https://github.com/djajetic/AutoML5, 2016a.

D. Jajetic. GPU_djajetic Implementation. https://github.com/djajetic/GPU_djajetic, 2016b.

B. Komer, J. Bergstra, and C. Eliasmith. Hyperopt-sklearn: Automatic hyperparameter configuration for scikit-learn. In *Proceedings of the International Conference on Machine Learning 2014, Workshop on Automatic Machine Learning*, 2014.

J. Lloyd. Freeze Thaw Ensemble Construction. https://github.com/jamesrobertlloyd/automl-phase-2.

H. Mendoza, A. Klein, M. Feurer, J. Tobias Springenberg, and Frank Hutter. Towards automatically-tuned neural networks. June 2016. URL https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbnxhdXRvbWwyMDE2fGd4OjMzYjQ4OWNhNTFhNzlhNGE.

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 É. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

G. Piatetsky. Data scientists automated and unemployed by 2025? http://www.kdnuggets.com/2015/05/data-scientists-automated-2025.html, 2015.

L. Sun. Automl challenge: System description of lisheng sun. In *ICML 2016 workshop on AutoML*, June 2016. URL http://dx.doi.org/10.5281/zenodo.27878.

K. Swersky, J. Snoek, and R. P. Adams. Freeze-thaw bayesian optimization. *Computing Research Repository*, abs/1406.3896, 2014.

A. Thakur. AutoML challenge: Rules for selecting neural network architectures for automl-gpu challenge. In *ICML 2016 workshop on AutoML*, June 2016. URL https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbnxhdXRvbWwyMDE2fGd4OjNmY2M0N2JhZGViZWY3ZDY.

A. Thakur and A. Krohn-Grimberghe. AutoCompete: A framework for machine learning competitions. In *Proceedings of the International Conference on Machine Learning 2015, Workshop on Automatic Machine Learning*, 2015. URL https://docs.google.com/a/chalearn.org/viewer?a=v&pid=sites&srcid=Y2hhbGVhcm4ub3JnfGF1dG9tbHxneDo3YThhNmNiNDA0M2Q2NjM5.

Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, 2016.

C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proc. ACM SIGKDD Conf. on Knowledge Discovery and Data Mining*, pages 847–855. ACM, 2013.

E. Tuv, A. Borisov, G. Runger, and K. Torkkola. Feature selection with ensembles, artificial variables, and redundancy elimination. *Journal of Machine Learning Research*, 10:1341–1366, January 2009.