Chapter 10

Analysis of the AutoML Challenge series 2015-2018

Isabelle Guyon and Lisheng Sun-Hosoya and Marc Boullé and Hugo Jair Escalante and Sergio Escalera and Zhengying Liu and Damir Jajetic and Bisakha Ray and Mehreen Saeed and Michèle Sebag and Alexander Statnikov and Wei-Wei Tu and Evelyne Viegas

Abstract

The ChaLearn AutoML Challenge[1] (NIPS 2015 - ICML 2016) consisted of six rounds of a machine learning competition of progressive difficulty, subject to limited computational resources. It was followed by a one-round AutoML challenge (PAKDD 2018). The AutoML setting differs from former model selection/hyper-parameter selection challenges, such as the one we previously organized for NIPS 2006: the participants aim to develop fully automated and computationally efficient systems, capable of being trained and tested without human intervention, with code submission. This chapter analyzes the results of these competitions and provides details about the datasets, which were not revealed to the participants. The solutions of the winners are systematically benchmarked over all datasets of all rounds and compared with canonical machine learning algorithms available in scikit-learn. All materials discussed in this chapter (data and code) have been made publicly available at `http://automl.chalearn.org/`.

This chapter is in part based on material that has appeared previously [36, 33, 32, 34].

[1]The authors are in alphabetical order of last name, except the first author who did most of the writing and the second author who produced most of the numerical analyses and plots.

10.1 Introduction

Until about ten years ago, machine learning (ML) was a discipline little known to the public. For ML scientists, it was a "seller's market": they were producing hosts of algorithms in search for applications and were constantly looking for new interesting datasets. Large internet corporations accumulating massive amounts of data such as Google, Facebook, Microsoft and Amazon have popularized the use of ML and data science competitions have engaged a new generation of young scientists in this wake. Nowadays, government and corporations keep identifying new applications of ML and with the increased availability of open data, we have switched to a "buyer's market": everyone seems to be in need of a learning machine. Unfortunately however, learning machines are not yet fully automatic: it is still difficult to figure out which software applies to which problem, how to horseshoe-fit data into a software and how to select (hyper-)parameters properly. The ambition of the ChaLearn AutoML challenge series is to channel the energy of the ML community to reduce step by step the need for human intervention in applying ML to a wide variety of practical problems.

Full automation is an unbounded problem since there can always be novel settings, which have never been encountered before. Our first challenges AutoML1 were limited to:

- **Supervised learning** problems (classification and regression).
- **Feature vector** representations.
- **Homogeneous datasets** (same distribution in the training, validation, and test set).
- **Medium size datasets** of less than 200 MBytes.
- **Limited computer resources** with execution times of less than 20 minutes per dataset on an 8 core $x86\_64$ machine with 56 GB RAM.

We excluded unsupervised learning, active learning, transfer learning, and causal discovery problems, which are all very dear to us and have been addressed in past ChaLearn challenges [31], but which require each a different evaluation setting, thus making result comparisons very difficult. We did not exclude the treatment of video, images, text, and more generally time series and the selected datasets actually contain several instances of such modalities. However, they were first preprocessed in a feature representation, thus de-emphasizing feature learning. Still, learning from data pre-processed in feature-based representations already covers a lot of grounds and a fully automated method resolving this restricted problem would already be a major advance in the field.

Within this constrained setting, we included a variety of difficulties:

- **Different data distributions:** the intrinsic/geometrical complexity of the dataset.
- **Different tasks:** regression, binary classification, multi-class classification, multi-label classification.

- **Different scoring metrics:** AUC, BAC, MSE, F_1, etc. (see Section 10.4.2).
- **Class balance:** Balanced or unbalanced class proportions.
- **Sparsity:** Full matrices or sparse matrices.
- **Missing values:** Presence or absence of missing values.
- **Categorical variables:** Presence or absence of categorical variables.
- **Irrelevant variables:** Presence or absence of additional irrelevant variables (distractors).
- **Number P_{tr} of training examples:** Small or large number of training examples.
- **Number N of variables/features:** Small or large number of variables.
- **Ratio P_{tr}/N of the training data matrix:** $P_{tr} \gg N, P_{tr} = N$ or $P_{tr} \ll N$.

In this setting, the participants had to face many modeling/hyper-parameter choices. Some other, equally important, aspects of automating machine learning were not addressed in this challenge and are left for future research. Those include data "ingestion" and formatting, pre-processing and feature/representation learning, detection and handling of skewed/biased data, inhomogeneous, drifting, multi-modal, or multi-view data (hinging on transfer learning), matching algorithms to problems (which may include supervised, unsupervised, or reinforcement learning, or other settings), acquisition of new data (active learning, query learning, reinforcement learning, causal experimentation), management of large volumes of data including the creation of appropriately-sized and stratified training, validation, and test sets, selection of algorithms that satisfy arbitrary resource constraints at training and run time, the ability to generate and reuse workflows, and generating meaningful reports.

This challenge series started with the NIPS 2006 "model selection game"[2] [37], where the participants were provided with a machine learning toolbox based on the Matlab toolkit CLOP [1] built on top of the "Spider" package [69]. The toolkit provided a flexible way of building models by combining preprocessing, feature selection, classification and post-processing modules, also enabling the building of ensembles of classifiers. The goal of the game was to build the best hyper-model: the focus was on model selection, not on the development of new algorithms. All problems were feature-based binary classification problems. Five datasets were provided. The participants had to submit the schema of their model. The model selection game confirmed the effectiveness of cross-validation (the winner invented a new variant called cross-indexing) and emphasized the need to focus more on search effectiveness with the deployment of novel search techniques such as particle swarm optimization.

New in the 2015/2016 AutoML challenge, we introduced the notion of "task": each dataset was supplied with a particular scoring metric to be optimized and a time budget. We initially intended to vary widely the time budget from

[2] `http://clopinet.com/isabelle/Projects/NIPS2006/`

dataset to dataset in an arbitrary way. We ended up fixing it to 20 minutes for practical reasons (except for Round 0 where the time budget ranged from 100 to 300 seconds). However, because the datasets varied in size, this put pressure on the participants to manage their allotted time. Other elements of novelty included the freedom of submitting any Linux executable. This was made possible by using automatic execution on the open-source platform Codalab[3]. To help the participants we provided a starting kit in Python based on the scikit-learn library [55][4]. This induced many of them to write a wrapper around scikit-learn. This has been the strategy of the winning entry "auto-sklearn" [26, 25, 27, 28][5]. Following the AutoML challenge, we organized a "beat auto-sklearn" game on a single dataset (madeline), in which the participants could provide hyper-parameters "by hand" to try to beat auto-sklearn. But nobody could beat auto-sklearn! Not even their designers. The participants could submit a json file which describes a sklearn model and hyper-parameter settings, via a GUI interface. This interface allows researchers who want to compare their search methods with auto-sklearn to use the exact same set of hyper-models.

A large number of satellite events including bootcamps, summer schools, and workshops have been organized in 2015/2016 around the AutoML challenge.[6] The AutoML challenge was part of the official selection of the competition program of IJCNN 2015 and 2016 and the results were discussed at the AutoML and CiML workshops at ICML and NIPS in 2015 and 2016. Several publications accompanied these events: in [33] we describe the details of the design of the AutoML challenge[7]. In [32] and [34] we review milestone and final results presented at the ICML 2015 and 2016 AutoML workshops. The 2015/2016 AutoML challenge had 6 rounds introducing 5 datasets each. We also organized a follow-up event for the PAKDD conference 2018[8] in only 2 phases, with 5 datasets in the development phase and 5 datasets in the final "blind test" round.

Going beyond the former published analyses, this chapter presents systematic studies of the winning solutions on all the datasets of the challenge and conducts comparisons with commonly used learning machines implemented in scikit-learn. It provides unpublished details about the datasets and reflective analyses.

[3] `http://competitions.codalab.org`
[4] `http://scikit-learn.org/`
[5] `https://automl.github.io/auto-sklearn/stable/`
[6] See `http://automl.chalearn.org`.
[7] `http://codalab.org/AutoML`
[8] `https://www.4paradigm.com/competition/pakdd2018`

10.2 Problem Formalization and Overview

10.2.1 Scope of the Problem

This challenge series focuses on supervised learning in ML and, in particular, solving classification and regression problems, without any further human intervention, within given constraints. To this end, we released a large number of datasets pre-formatted in given feature representations (i.e., each example consists of a fixed number of numerical coefficients; more in Section 10.3).

The distinction between input and output variables is not always made in ML applications. For instance, in recommender systems, the problem is often stated as making predictions of missing values for every variable rather than predicting the values of a particular variable [58]. In unsupervised learning [30], the purpose is to explain data in a simple and compact way, eventually involving inferred latent variables (e.g., class membership produced by a clustering algorithm).

We consider only the strict supervised learning setting where data present themselves as identically and independently distributed input-output pairs. The models used are limited to fixed-length vectorial representations, excluding problems of time series prediction. Text, speech, and video processing tasks included in the challenge have been preprocessed into suitable fixed-length vectorial representations.

The difficulty of the proposed tasks lies in the data complexity (class imbalance, sparsity, missing values, categorical variables). The testbed is composed of data from a wide variety of domains. Although there exist ML toolkits that can tackle all of these problems, it still requires considerable human effort to find, for a given dataset, task, evaluation metric, the methods and hyper-parameter settings that maximize performance subject to a computational constraint. The participant challenge is to create the *perfect black box* that removes human interaction, alleviating the shortage of data scientists in the coming decade.

10.2.2 Full Model Selection

We refer to participant solutions as *hyper-models* to indicate that they are built from simpler components. For instance, for classification problems, participants might consider a hyper-model that combines several classification techniques such as nearest neighbors, linear models, kernel methods, neural networks, and random forests. More complex hyper-models may also include preprocessing, feature construction, and feature selection modules.

Generally, a predictive model of the form $y = f(\mathbf{x}; \boldsymbol{\alpha})$ has:

• a set of parameters $\boldsymbol{\alpha} = [\alpha_0, \alpha_1, \alpha_2, ..., \alpha_n]$;

• a learning algorithm (referred to as trainer), which serves to optimize the parameters using training data;

• a trained model (referred to as predictor) of the form $y = f(\mathbf{x})$ produced by the trainer;

• a clear objective function $J(f)$, which can be used to assess the model's performance on test data.

Consider now the model hypothesis space defined by a vector $\boldsymbol{\theta} = [\theta_1, \theta_2, ..., \theta_n]$ of hyper-parameters. The hyper-parameter vector may include not only parameters corresponding to switching between alternative models, but also modeling choices such as preprocessing parameters, type of kernel in a kernel method, number of units and layers in a neural network, or training algorithm regularization parameters [59]. Some authors refer to this problem as *full model selection* [24, 62], others as the CASH problem (Combined Algorithm Selection and Hyperparameter optimization) [65]. We will denote hyper-models as

$$y = f(\mathbf{x}; \boldsymbol{\theta}) = f(\mathbf{x}; \boldsymbol{\alpha}(\boldsymbol{\theta}), \boldsymbol{\theta}), \tag{10.1}$$

where the model parameter vector $\boldsymbol{\alpha}$ is an implicit function of the hyper-parameter vector $\boldsymbol{\theta}$ obtained by using a trainer for a fixed value of $\boldsymbol{\theta}$, and training data composed of input-output pairs $\{\mathbf{x}_i, y_i\}$. The participants have to devise algorithms capable of training the hyper-parameters $\boldsymbol{\theta}$. This may require intelligent sampling of the hyper-parameter space and splitting the available training data into subsets for both training and evaluating the predictive power of solutions—one or multiple times.

As an optimization problem, model selection is a bi-level optimization program [18, 19, 7]; there is a lower objective J_1 to train the parameters $\boldsymbol{\alpha}$ of the model, and an upper objective J_2 to train the hyper-parameters $\boldsymbol{\theta}$, both optimized simultaneously (see Figure 10.1). As a statistics problem, model selection is a problem of multiple testing in which error bars on performance prediction ϵ degrade with the number of models/hyper-parameters tried or, more generally, the complexity of the hyper-model $C_2(\boldsymbol{\theta})$. A key aspect of AutoML is to avoid overfitting the upper-level objective J_2 by regularizing it, much in the same way as lower level objectives J_1 are regularized.

The problem setting also lends itself to using ensemble methods, which let several "simple" models vote to make the final decision [15, 29, 16]. In this case, the parameters $\boldsymbol{\theta}$ may be interpreted as voting weights. For simplicity we lump all parameters in a single vector, but more elaborate structures, such as trees or graphs can be used to define the hyper-parameter space [66].

10.2.3 Optimization of Hyper-Parameters

Everyone who has worked with data has had to face some common modeling choices: scaling, normalization, missing value imputation, variable coding (for categorical variables), variable discretization, degree of nonlinearity and model architecture, among others. ML has managed to reduce the number of hyper-parameters and produce *black-boxes* to perform tasks such as classification and regression [40, 21]. Still, any real-world problem requires at least some preparation of the data before it can be fitted into an "automatic" method, hence requiring some modeling choices. There has been much progress on end-to-end automatic ML for more complex tasks such as text, image, video, and speech processing with deep-learning methods [6]. However, even these methods have many modeling choices and hyper-parameters.

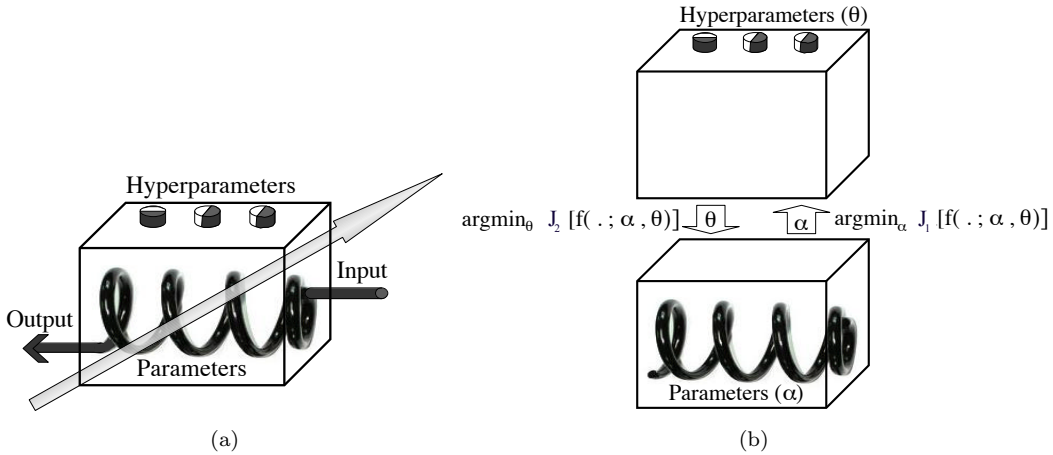
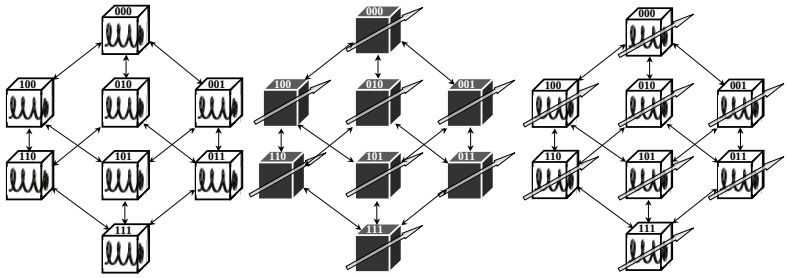


Figure 10.1: **Bi-level optimization.** (a) Representation of a learning machine with parameters and hyper-parameters to be adjusted. (b) De-coupling of parameter and hyper-parameter adjustment in two levels. The upper level objective J_2 optimizes the hyper-parameters $\boldsymbol{\theta}$; the lower objective J_1 optimizes the parameters $\boldsymbol{\alpha}$.

While producing models for a diverse range of applications has been a focus of the ML community, little effort has been devoted to the optimization of hyper-parameters. Common practices that include *trial and error* and grid search may lead to overfitting models for small datasets or underfitting models for large datasets. By overfitting we mean producing models that perform well on training data but perform poorly on unseen data, i.e., models that do not generalize. By underfitting we mean selecting too simple a model, which does not capture the complexity of the data, and hence performs poorly both on training and test data. Despite well-optimized off-the-shelf algorithms for optimizing parameters, end-users are still responsible for organizing their numerical experiments to identify the best of a number of models under consideration. Due to lack of time and resources, they often perform model/hyper-parameter selection with ad hoc techniques. [42, 47] examine fundamental, common mistakes such as poor construction of training/test splits, inappropriate model complexity, hyper-parameter selection using test sets, misuse of computational resources, and misleading test metrics, which may invalidate an entire study. Participants must avoid these flaws and devise systems that can be blind-tested.

An additional twist of our problem setting is that code is tested with limited computational resources. That is, for each task an arbitrary limit on execution time is fixed and a maximum amount of memory is provided. This places a constraint on the participant to produce a solution in a given time, and hence to optimize the model search from a computational point of view. In summary, participants have to jointly address the problem of over-fitting/under-fitting and



(a) Filter (b) Wrapper (c) Embedded

Figure 10.2: **Approaches to two-level inference.** (a) **Filter methods** select the hyper-parameters without adjusting the learner parameters. (No arrows indicates no parameter training.) (b) **Wrapper methods** select the hyper-parameters using trained learners, treating them as black-boxes. (c) **Embedded methods** use knowledge of the learner structure and/or parameters to guide the hyper-parameter search.

the problem of efficient search for an optimal solution, as stated in [43]. In practice, the computational contraints have turned out to be far more challenging to challenge participants than the problem of overfitting. Thus the main contributions have been to devise novel efficient search techniques with cutting-edge optimization methods.

10.2.4 Strategies of Model Search

Most practitioners use heuristics such as grid search or uniform sampling to sample $\boldsymbol{\theta}$ space, and use k-fold cross-validation as the upper-level objective J_2 [20]. In this framework, the optimization of $\boldsymbol{\theta}$ is not performed sequentially [8]. All the parameters are sampled along a regular scheme, usually in linear or log scale. This leads to a number of possibilities that exponentially increases with the dimension of $\boldsymbol{\theta}$. k-fold cross-validation consists of splitting the dataset into k folds; $(k-1)$ folds are used for training and the remaining fold is used for testing; eventually, the average of the test scores obtained on the k folds is reported. Note that some ML toolkits currently support cross-validation. There is a lack of principled guidelines to determine the number of grid points and the value of k (with the exception of [20]), and there is no guidance for regularizing J_2, yet this simple method is a good baseline approach.

Efforts have been made to optimize continuous hyper-parameters with bilevel optimization methods, using either the k-fold cross-validation estimator [7, 50] or the leave-one-out estimator as the upper-level objective J_2. The leave-one-out estimator may be efficiently computed, in closed form, as a by-product of training only one predictor on all the training examples (e.g., virtual-leave-one-out [38]). The method was improved by adding a regularization of J_2 [17]. Gradient descent has been used to accelerate the search, by making a local quadratic

approximation of J_2 [44]. In some cases, the full $J_2(\boldsymbol{\theta})$ can be computed from a few key examples [39, 54]. Other approaches minimize an *approximation* or an *upper bound* of the leave-one-out error, instead of its exact form [53, 68]. Nevertheless, these methods are still limited to specific models and continuous hyper-parameters.

An early attempt at full model selection was the *pattern search* method that uses k-fold cross-validation for J_2. It explores the hyper-parameter space by steps of the same magnitude, and when no change in any parameter further decreases J_2, the step size is halved and the process repeated until the steps are deemed sufficiently small [49]. [24] addressed the full model selection problem using Particle Swarm Optimization, which optimizes a problem by having a population of candidate solutions (particles), and moving these particles around the hyper-parameter space using the particle's position and velocity. k-fold cross-validation is also used for J_2. This approach retrieved the winning model in $\sim 76\%$ of the cases. Overfitting was controlled heuristically with early stopping and the proportion of training and validation data was not optimized. Although progress has been made in experimental design to reduce the risk of overfitting [42, 47], in particular by splitting data in a principled way [61], to our knowledge, no one has addressed the problem of optimally splitting data.

While regularizing the second level of inference is a recent addition to the frequentist ML community, it has been an intrinsic part of Bayesian modeling via the notion of hyper-prior. Some methods of multi-level optimization combine importance sampling and Monte-Carlo Markov Chains [2]. The field of Bayesian hyper-parameter optimization has rapidly developed and yielded promising results, in particular by using Gaussian processes to model generalization performance [60, 63]. But Tree-structured Parzen Estimator (TPE) approaches modeling $P(\mathbf{x}|y)$ and $P(y)$ rather than modeling $P(y|\mathbf{x})$ directly [10, 9] have been found to outperform GP-based Bayesian optimization for structured optimization problems with many hyperparameters including discrete ones [23]. The central idea of these methods is to fit $J_2(\boldsymbol{\theta})$ to a smooth function in an attempt to reduce variance and to estimate the variance in regions of the hyper-parameter space that are under-sampled to guide the search towards regions of high variance. These methods are inspirational and some of the ideas can be adopted in the frequentist setting. For instance, the random-forest-based SMAC algorithm [41], which has helped speed up both local search and tree search algorithms by orders of magnitude on certain instance distributions, has also been found to be very effective for the hyper-parameter optimization of machine learning algorithms, scaling better to high dimensions and discrete input dimensions than other algorithms [23]. We also notice that Bayesian optimization methods are often combined with other techniques such as meta-learning and ensemble methods [25] in order to gain advantage in some challenge settings with a time limit [32]. Some of these methods consider jointly the two-level optimization and take time cost as a critical guidance for hyper-parameter search [64, 45].

Besides Bayesian optimization, several other families of approaches exist in the literature and have gained much attention with the recent rise of deep learn-

ing. Ideas borrowed from *reinforcement learning* have recently been used to construct optimal neural network architectures [70, 4]. These approaches formulate the hyper-parameter optimization problem in a reinforcement learning flavor, with for example states being the actual hyper-parameter setting (e.g., network architecture), actions being adding or deleting a module (e.g., a CNN layer or a pooling layer), and reward being the validation accuracy. They can then apply off-the-shelf reinforcement learning algorithms (e.g., RENFORCE, Q-learning, Monte-Carlo Tree Search) to solve the problem. Other architecture search methods use *evolutionary* algorithms [57, 3]. These approaches consider a set (population) of hyper-parameter settings (individuals), modify (mutate and reproduce) and eliminate unpromising settings according to their cross-validation score (fitness). After several generations, the global quality of the population increases. One important common point of reinforcement learning and evolutionary algorithms is that they both deal with the exploration-exploitation trade-off. Despite the impressive results, these approaches require a huge amount of computational resources and some (especially evolutionary algorithms) are hard to scale. [56] recently proposed weight sharing among child models to speed up the process considerably [70] while achieving comparable results.

Note that splitting the problem of parameter fitting into two levels can be extended to more levels, at the expense of extra complexity—i.e., need for a hierarchy of data splits to perform multiple or nested cross-validation [22], insufficient data to train and validate at the different levels, and increase of the computational load.

Table 10.1 shows a typical example of multi-level parameter optimization in a frequentist setting. We assume that we are using an ML toolbox with two learning machines: Kridge (kernel ridge regression) and Neural (a neural network a.k.a. "deep learning" model). At the top level we use a test procedure to assess the performance of the final model (this is not an inference level). The top-level inference algorithm Validation({GridCV(Kridge, MSE), GridCV(Neural, MSE)}, MSE) is decomposed into its elements recursively. Validation uses the data split $D = [D_{Tr}, D_{Va}]$ to compare the learning machines Kridge and Neural (trained using D_{Tr} on the validation set D_{Va}, using the mean-square error (MSE) evaluation function. The algorithm GridCV, a grid search with 10-fold cross-validation (CV) MSE evaluation function, then optimizes the hyper-parameters $\boldsymbol{\theta}$. Internally, both Kridge and Neural use virtual leave-one-out (LOO) cross-validation to adjust γ and a classical L_2 regularized risk functional to adjust $\boldsymbol{\alpha}$.

Borrowing from the conventional classification of feature selection methods [46, 11, 38], model search strategies can be categorized into filters, wrappers, and embedded methods (see Figure 10.2). **Filters** are methods for narrowing down the model space, without training the learner. Such methods include preprocessing, feature construction, kernel design, architecture design, choice of prior or regularizers, choice of noise model, and filter methods for feature selection. Although some filters use training data, many incorporate human prior knowledge of the task or knowledge compiled from previous tasks. Recently, [5]

Table 10.1

Typical example of multi-level inference algorithm. The top-level algorithm Validation({GridCV(Kridge, MSE), GridCV(Neural, MSE)}, MSE) is decomposed into its elements recursively. Calling the method "train" on it using data D_{TrVa} results in a function f, then tested with test(f, MSE, D_{Te}). The notation $[.]_{CV}$ indicates that results are averages over multiple data splits (cross-validation). NA means "not applicable". A model family $\mathcal{F}$ of parameters $\boldsymbol{\alpha}$ and hyper-parameters $\boldsymbol{\theta}$ is represented as f$(\boldsymbol{\theta}, \boldsymbol{\alpha})$. We derogate to the usual convention of putting hyper-parameters last, the hyper-parameters are listed in decreasing order of inference level. $\mathcal{F}$, thought of as a bottom level algorithm, does not perform any training: $train$(f$(\boldsymbol{\theta}, \boldsymbol{\alpha})$) just returns the function $f(\mathbf{x}; \boldsymbol{\theta}, \boldsymbol{\alpha})$.

Level	**Algorithm**	**Parameters**		**Optimization performed**	**Data split**
		Fixed	**Varying**		
NA	f	All	All	Performance assessment (no inference).	D_{Te}
4	Validation	None	All	Final algorithm selection using validation data.	$D = [D_{Tr}, D_{Va}]$
3	GridCV	model index i	$\boldsymbol{\theta}, \gamma, \boldsymbol{\alpha}$	10-fold CV on regularly sampled values of $\boldsymbol{\theta}$.	$D_{Tr} = [D_{tr}, D_{va}]_{CV}$
2	Kridge$(\boldsymbol{\theta})$ Neural$(\boldsymbol{\theta})$	$i, \boldsymbol{\theta}$	$\gamma, \boldsymbol{\alpha}$	Virtual LOO CV to select regularization parameter γ	$D_{tr} = [D_{tr}^{\setminus\{d\}}, d]_{CV}$
1	Kridge$(\boldsymbol{\theta}, \gamma)$ Neural$(\boldsymbol{\theta}, \gamma)$	$i, \boldsymbol{\theta}, \gamma$	$\boldsymbol{\alpha}$	Matrix inversion of gradient descent to compute $\boldsymbol{\alpha}$.	D_{tr}
0	Kridge$(\boldsymbol{\theta}, \gamma, \boldsymbol{\alpha})$ Neural$(\boldsymbol{\theta}, \gamma, \boldsymbol{\alpha})$	All	None	NA	NA

proposed to apply collaborative filtering methods to model search. **Wrapper methods** consider learners as a black-box capable of learning from examples and making predictions once trained. They operate with a search algorithm in the hyper-parameter space (grid search or stochastic search) and an evaluation function assessing the trained learner's performance (cross-validation error or Bayesian evidence). **Embedded methods** are similar to wrappers, but they exploit the knowledge of the machine learning algorithm to make the search more efficient. For instance, some embedded methods compute the leave-one-out solution in a closed form, without leaving anything out, i.e., by performing a single model training on all the training data (e.g., [38]). Other embedded methods jointly optimize parameters and hyper-parameters [44, 51, 50].

In summary, many authors focus only on the efficiency of search, ignoring the problem of overfitting the second level objective J_2, which is often chosen to be k-fold cross-validation with an arbitrary value for k. Bayesian methods introduce techniques of overfitting avoidance via the notion of hyper-priors, but at the expense of making assumptions on how the data were generated and without providing guarantees of performance. In all the prior approaches to full model selection we know of, there is no attempt to treat the problem as the optimization of a regularized functional J_2 with respect to both (1) modeling choices and (2) data split. Much remains to be done to jointly address statistical and computational issues. The AutoML challenge series offers benchmarks to compare and contrast methods addressing these problems, free of the inventor/evaluator bias.

10.3 Data

We gathered a first pool of 70 datasets during the summer 2014 with the help of numerous collaborators and ended up selecting 30 datasets for the 2015/2016 challenge (see Table 10.2 and the appendix), chosen to illustrate a wide variety of domains of applications: biology and medicine, ecology, energy and sustainability management, image, text, audio, speech, video and other sensor data processing, internet social media management and advertising, market analysis and financial prediction. We preprocessed data to obtain feature representations (i.e., each example consists of a fixed number of numerical coefficients). Text, speech, and video processing tasks were included in the challenge, but not in their native variable-length representations.

For the 2018 challenge, three datasets from the first pool (but unused in the first challenge) were selected and 7 new datasets collected by the new organizers and sponsors were added (see Table 10.3 and the appendix).

Some datasets were obtained from public sources, but they were reformatted into new representations to conceal their identity, except for the final round of the 2015/2016 challenge and the final phase of the 2018 challenge, which included completely new data.

In the 2015/2016 challenge, data difficulty progressively increased from round to round. Round 0 introduced five (public) datasets from previous challenges

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 10.2: **Datasets of the 2015/2016 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.

Phase	DATASET	Cbal	Sparse	Miss	Cat	Irr	Pte	Pva	Ptr	N	Ptr/N
1	1 ADA	1	0.67	0	0	0	41471	415	4147	48	86.39
1	2 ARCENE	0.22	0.54	0	0	0	700	100	100	10000	0.01
1	3 GINA	1	0.03	0.31	0	0	31532	315	3153	970	3.25
1	4 GUILLERMO	0.33	0.53	0	0	0	5000	5000	20000	4296	4.65
1	5 RL	0.10	0	0.11	1	0	24803	0	31406	22	1427.5
2	1 PM	0.01	0	0.11	1	0	20000	0	29964	89	224.71
2	2 RH	0.04	0.41	0	1	0	28544	0	31498	76	414.44
2	3 RI	0.02	0.09	0.26	1	0	26744	0	30562	113	270.46
2	4 RICCARDO	0.67	0.51	0	0	0	5000	5000	20000	4296	4.65
2	5 RM	0.001	0	0.11	1	0	26961	0	28278	89	317.73

Table 10.3: **Datasets of the 2018 AutoML challenge.** All tasks are binary classification problems. The metric is the AUC for all tasks. The time budget is also the same for all datasets (1200s). Phase 1 was the development phase and phase 2 the final "blind test" phase.

illustrating the various difficulties encountered in subsequent rounds:

Novice. Binary classification problems only. No missing data; no categorical features; moderate number of features ($< 2,000$); balanced classes. Challenge lies in dealing with sparse and full matrices, presence of irrelevant variables, and various Ptr/N.

Intermediate. Binary and multi-class classification problems. Challenge lies in dealing with unbalanced classes, number of classes, missing values, categorical variables, and up to 7,000 features.

Advanced. Binary, multi-class, and multi-label classification problems. Challenge lies in dealing with up to 300,000 features.

Expert. Classification and regression problems. Challenge lies in dealing with the entire range of data complexity.

Master. Classification and regression problems of all difficulties. Challenge lies in learning from completely new datasets.

The datasets of the 2018 challenge were all binary classification problems. Validation partitions were not used because of the design of this challenge, even when they were available for some tasks. The three reused datasets had similar difficulty as those of rounds 1 and 2 of the 2015/2016 challenge. However, the 7 new data sets introduced difficulties that were not present in the former challenge. Most notably an extreme class imbalance, presence of categorical features and a temporal dependency among instances that could be exploited by participants to develop their methods[9]. The datasets from both challenges are downloadable from `http://automl.chalearn.org/data`.

[9]In RL, PM, RH, RI and RM datasets instances were chronologically sorted, this information was made available to participants and could be used for developing their methods.

10.4 Challenge Protocol

In this section, we describe design choices we made to ensure the thoroughness and fairness of the evaluation. As previously indicated, we focus on supervised learning tasks (classification and regression problems), without any human intervention, within given time and computer resource constraints (Section 10.4.1), and given a particular metric (Section 10.4.2), which varies from dataset to dataset. During the challenges, the identity and description of the datasets is concealed (except in the very first round or phase where sample data is distributed) to avoid the use of domain knowledge and to push participants to design fully automated ML solutions. In the 2015/2016 AutoML challenge, the datasets were introduced in a series of rounds (Section 10.4.3), alternating periods of code development (Tweakathon phases) and blind tests of code without human intervention (AutoML phases). Either results or code could be submitted during development phases, but code had to be submitted to be part of the AutoML "blind test" ranking. In the 2018 edition of the AutoML challenge, the protocol was simplified. We had only one round in two phases: a development phase in which 5 datasets were released for practice purposes, and a final "blind test" phase with 5 new datasets that were never used before.

10.4.1 Time Budget and Computational Resources

The Codalab platform provides computational resources shared by all participants. We used up to 10 compute workers processing in parallel the queue of submissions made by participants. Each compute worker was equipped with 8 cores $x86\_64$. Memory was increased from 24 GB to 56 GB after round 3 of the 2015/2016 AutoML challenge. For the 2018 AutoML challenge computing resources were reduced, as we wanted to motivate the development of more efficient yet effective AutoML solutions. We used 6 compute workers processing in parallel the queue of submissions. Each compute worker was equipped with 2 cores $x86\_64$ and 8 GB of memory.

To ensure fairness, when a code submission was evaluated, a compute worker was dedicated to processing that submission only, and its execution time was limited to a given time budget (which may vary from dataset to dataset). The time budget was provided to the participants with each dataset in its *info* file. It was generally set to 1200 seconds (20 minutes) per dataset, for practical reasons, except in the first phase of the first round. However, the participants did not know this ahead of time and therefore their code had to be capable to manage a given time budget. The participants who submitted results instead of code were not constrained by the time budget since their code was run on their own platform. This was potentially advantageous for entries counting towards the Final phases (immediately following a Tweakathon). Participants wishing to also enter the AutoML (blind testing) phases, which required submitting code, could submit both results and code (simultaneously). When results were submitted, they were used as entries in the on-going phase. They did not need to be produced by the submitted code; i.e., if a participant did not want to share

personal code, he/she could submit the sample code provided by the organizers together with his/her results. The code was automatically forwarded to the AutoML phases for "blind testing". In AutoML phases, result submission was not possible.

The participants were encouraged to save and submit intermediate results so we could draw learning curves. This was not exploited during the challenge. But we study learning curves in this chapter to evaluate the capabilities of algorithms to quickly attain good performances.

10.4.2 Scoring Metrics

The scores are computed by comparing submitted predictions to reference target values. For each sample $i, i = 1 : P$ (where P is the size of the validation set or of the test set), the target value is a continuous numeric coefficient y_i for regression problems, a binary indicator in $\{0, 1\}$ for two-class problems, or a vector of binary indicators $[y_{il}]$ in $\{0, 1\}$ for multi-class or multi-label classification problems (one per class l). The participants had to submit prediction values matching as closely as possible the target values, in the form of a continuous numeric coefficient q_i for regression problems and a vector of numeric coefficients $[q_{il}]$ in the range $[0, 1]$ for multi-class or multi-label classification problems (one per class l).

The provided starting kit contains an implementation in Python of all scoring metrics used to evaluate the entries. Each dataset has its own scoring criterion specified in its *info* file. All scores are normalized such that the expected value of the score for a random prediction, based on class prior probabilities, is 0 and the optimal score is 1. Multi-label problems are treated as multiple binary classification problems and are evaluated using the average of the scores of each binary classification subproblem.

We first define the notation $\langle\cdot\rangle$ for the average over all samples P indexed by i. That is,

$$\langle y_i \rangle = (1/P) \sum_{i=1}^{P} (y_i). \tag{10.2}$$

The score metrics are defined as follows:

R^2. The coefficient of determination is used for regression problems only. The metric is based on the mean squared error (MSE) and the variance (VAR), and computed as

$$R^2 = 1 - MSE/VAR, \tag{10.3}$$

where $MSE = \langle (y_i - q_i)^2 \rangle$ and $VAR = \langle (y_i - m)^2 \rangle$, with $m = \langle y_i \rangle$.

ABS. This coefficient is similar to R^2 but based on the mean absolute error (MAE) and the mean absolute deviation (MAD), and computed as

$$ABS = 1 - MAE/MAD, \tag{10.4}$$

where $MAE = \langle \text{abs}(y_i - q_i) \rangle$ and $MAD = \langle \text{abs}(y_i - m) \rangle$.

BAC. Balanced accuracy is the average of class-wise accuracy for classification

problems—and the average of *sensitivity* (true positive rate) and *specificity* (true negative rate) for binary classification:

$$BAC = \begin{cases} \frac{1}{2}[\frac{TP}{P} + \frac{TN}{N}], & \text{for binary} \\ \frac{1}{C}\sum_{i=1}^{C}\frac{TP_i}{N_i}, & \text{for multi-class} \end{cases} \tag{10.5}$$

where P (N) is the number of positive (negative) examples, TP (TN) is the number of well classified positive (negative) examples, C is the number of classes, TP_i is the number of well classified examples of class i and N_i the number of examples of class i.

For binary classification problems, the class-wise accuracy is the fraction of correct class predictions when q_i is thresholded at 0.5, for each class. For multi-label problems, the class-wise accuracy is averaged over all classes. For multi-class problems, the predictions are binarized by selecting the class with maximum prediction value $\arg\max_l q_{il}$ before computing the class-wise accuracy.

We normalize the metric as follows:

$$|BAC| = (BAC - R)/(1 - R), \tag{10.6}$$

where R is the expected value of BAC for random predictions (i.e., $R = 0.5$ for binary classification and $R = (1/C)$ for C-class problems).

AUC. The area under the ROC curve is used for ranking and binary classification problems. The ROC curve is the curve of *sensitivity* vs. *1-specificity* at various prediction thresholds. The AUC and BAC values are the same for binary predictions. The AUC is calculated for each class separately before averaging over all classes. We normalize the metric as

$$|AUC| = 2AUC - 1. \tag{10.7}$$

F1 score. The harmonic mean of precision and recall is computed as

$$F1 = 2 * (precision * recall)/(precision + recall), \tag{10.8}$$

$$precision = true\ positive/(true\ positive + false\ positive) \tag{10.9}$$

$$recall = true\ positive/(true\ positive + false\ negative) \tag{10.10}$$

Prediction thresholding and class averaging is handled similarly as in BAC. We normalize the metric as follows:

$$|F1| = (F1 - R)/(1 - R), \tag{10.11}$$

where R is the expected value of F1 for random predictions (see BAC).

PAC. Probabilistic accuracy is based on the cross-entropy (or log loss) and computed as

$$PAC = \exp(-CE), \tag{10.12}$$

$$CE = \begin{cases} average \sum_l \log(q_{il}), & \text{for multi-class} \\ -\langle y_i \log(q_i), & \\ +(1-y_i)\log(1-q_i)\rangle, & \text{for binary and multi-label} \end{cases} \tag{10.13}$$

Class averaging is performed after taking the exponential in the multi-label case. We normalize the metric as follows:

$$|PAC| = (PAC - R)/(1 - R), \tag{10.14}$$

where R is the score obtained using $q_i = \langle y_i \rangle$ or $q_{il} = \langle y_{il} \rangle$ (i.e., using as predictions the fraction of positive class examples, as an estimate of the prior probability).

Note that the normalization of R^2, ABS, and PAC uses the average target value $q_i = \langle y_i \rangle$ or $q_{il} = \langle y_{il} \rangle$. In contrast, the normalization of BAC, AUC, and F1 uses a random prediction of one of the classes with uniform probability.

Only R^2 and ABS are meaningful for regression; we compute the other metrics for completeness by replacing the target values with binary values after thresholding them in the mid-range.

Table 10.4: **Phases of round n in the 2015/2016 challenge.** For each dataset, one labeled training set is provided and two unlabeled sets (validation set and test set) are provided for testing.

Phase in round [n]	**Goal**	**Duration**	**Submissions**	**Data**	**Leaderboard scores**	**Prizes**
* AutoML[n]	Blind test of code	Short	NONE (code migrated)	New datasets, not downloadable	Test set results	Yes
Tweakathon[n]	Manual tweaking	Months	Code and/ or results	Datasets downloadable	Validation set results	No
* Final[n]	Results of Tweakathon revealed	Short	NONE (results migrated)	NA	Test set results	Yes

10.4.3 Rounds and Phases in the 2015/2016 Challenge

The 2015/2016 challenge was run in multiple phases grouped in six rounds. Round 0 (Preparation) was a practice round using publicly available datasets. It was followed by five rounds of progressive difficulty (Novice, Intermediate, Advanced, Expert, and Master). Except for rounds 0 and 5, all rounds included three phases that alternated AutoML and Tweakathons contests. These phases are described in Table 10.4.

Submissions were made in Tweakathon phases only. The results of the latest submission were shown on the leaderboard and such submission automatically

migrated to the following phase. In this way, the code of participants who abandoned before the end of the challenge had a chance to be tested in subsequent rounds and phases. New participants could enter at any time. Prizes were awarded in phases marked with a * during which there was no submission. To participate in phase AutoML[n], code had to be submitted in Tweakathon[n-1].

In order to encourage participants to try GPUs and deep learning, a GPU track sponsored by NVIDIA was included in Round 4.

To participate in the Final[n], code or results had to be submitted in Tweakathon[n]. If both code and (well-formatted) results were submitted, the results were used for scoring rather than rerunning the code in Tweakathon[n] and Final[n]. The code was executed when results were unavailable or not well formatted. Thus, there was no disadvantage in submitting both results and code. If a participant submitted both results and code, different methods could be used to enter the Tweakathon/Final phases and the AutoML phases. Submissions were made only during Tweakathons, with a maximum of five submissions per day. Immediate feedback was provided on the leaderboard on validation data. The participants were ranked on the basis of test performance during the Final and AutoML phases.

We provided baseline software using the ML library scikit-learn [55]. It uses ensemble methods, which improve over time by adding more base learners. Other than the number of base learners, the default hyper-parameter settings were used. The participants were not obliged to use the Python language nor the main Python script we gave as an example. However, most participants found it convenient to use the main python script, which managed the sparse format, the any-time learning settings and the scoring metrics. Many limited themselves to search for the best model in the scikit-learn library. This shows the importance of providing a good starting kit, but also the danger of biasing results towards particular solutions.

10.4.4 Phases in the 2018 Challenge

The 2015/2016 AutoML challenge was very long and few teams participated in all rounds. Further, even though there was no obligation to participate in previous rounds to enter new rounds, new potential participants felt they would be at a disadvantage. Hence, we believe it is preferable to organize recurrent yearly events, each with their own workshop and publication opportunity. This provides a good balance between competition and collaboration.

In 2018, we organized a single round of AutoML competition in two phases. In this simplified protocol, the participants could practice on five datasets during the first (development) phase, by either submitting code or results. Their performances were revealed immediately, as they became available, on the leaderboard.

The last submission of the development phase was automatically forwarded to the second phase: the AutoML "blind test" phase. In this second phase, which was the only one counting towards the prizes, the participants' code was automatically evaluated on five new datasets on the Codalab platform. The

datasets were not revealed to the participants. Hence, submissions that did not include code capable of being trained and tested automatically were not ranked in the final phase and could not compete towards the prizes.

We provided the same starting kit as in the AutoML 2015/2016 challenge, but the participants also had access to the code of the winners of the previous challenge.

Table 10.5: **Results of the 2015/2016 challenge winners.** $< R >$ is the average rank over all five data sets of the round and it 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. The GPU track was run in round 4. 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.

	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

10.5 Results

This section provides a brief description of the results obtained during both challenges, explains the methods used by the participants and their elements of novelty, and provides the analysis of post-challenge experiments conducted to answer specific questions on the effectiveness of model search techniques.

Table 10.6: **Results of the 2018 challenge winners.** Each phase was run on 5 different datasets. We show the winners of the AutoML (blind test) phase and for comparison their performances in the Feedback phase. The full tables can be found at `https://competitions.codalab.org/competitions/17767`.

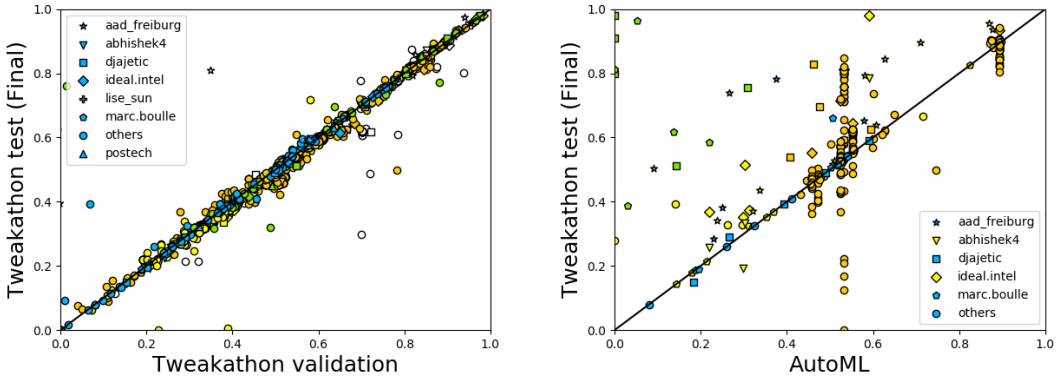
2. AutoML phase				**1. Feedback phase**			
Ended	Winners	$< R >$	$< S >$	Ended	Performance	$< R >$	$< S >$
	1. aad_freiburg	2.80	0.4341		aad_freiburg	9.0	0.7422
	2. narnars0	3.80	0.4180		narnars0	4.40	0.7324
03/31/18	3. wlWangl	5.40	0.3857	03/12/18	wlWangl	4.40	0.8029
	3. thanhdng	5.40	0.3874		thanhdng	14.0	0.6845
	3. Malik	5.40	0.3863		Malik	13.8	0.7116

10.5.1 Scores Obtained in the 2015/2016 Challenge

The 2015/2016 challenge lasted 18 months (December 8, 2014 to May 1, 2016). By the end of the challenge, practical solutions were obtained and open-sourced, such as the solution of the winners [25].

Table 10.5 presents 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 participant who submitted first. The table only reports the results of the top-ranking participants. We also show in Figure 10.3 a comparison of the leaderboard performances of all participants. We plot in Figure 10.3(a) the Tweakathon performances on the final test set vs. those on the validation set, which reveals no significant overfitting to the validation set, except for a few outliers. In Figure 10.3(b) we report the performance in AutoML result (blind testing) vs. Tweakathon final test results (manual adjustments possible). We see that many entries were made in phase 1 (binary classification) and then participation declined as the tasks became harder. Some participants put a lot of effort in Tweakathons and far exceeded their AutoML performances (e.g. Djajetic and AAD Freiburg).

There is still room for improvement, as revealed by the significant differences remaining between Tweakathon and AutoML (blind testing) results (Table 10.5 and Figure 10.3-b). In Round 3, all but one participant failed to turn in working solutions during blind testing, because of the introduction of sparse datasets. Fortunately, the participants recovered, and, by the end of the challenge, several submissions were capable of returning solutions on all the datasets of the challenge. But learning schemas can still be optimized because, even discarding Round 3, there is a 15 to 35% performance gap between AutoML phases (blind testing with computational constraints) and Tweakathon phases (human intervention and additional compute power). The GPU track offered (in round 4 only) a platform for trying Deep Learning methods. This allowed the participants to demonstrate that, given additional compute power, deep learning



(a) Leaderboard overfitting? (b) Gap AutoML/Tweakathon?

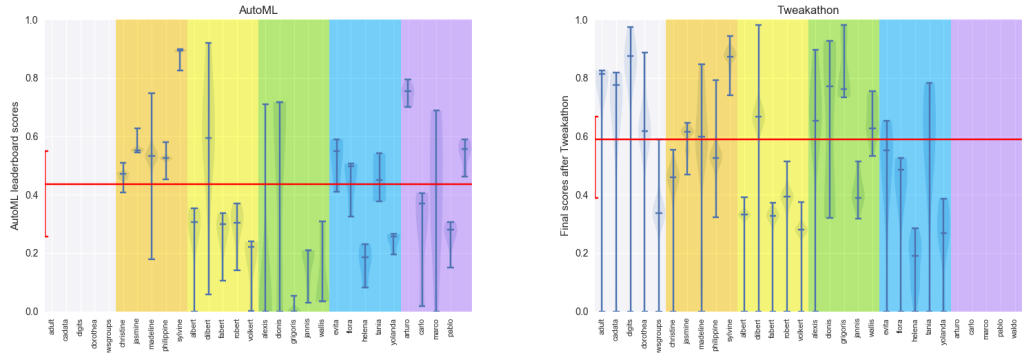
Figure 10.3: **Performances of all participants in the 2015/2016 challenge.** We show the last entry of all participants in all phases of the 2015/2016 challenge on all datasets from the competition leaderboards. The symbols are color coded by round, as in Table 10.5. (a) **Overfitting in Tweakathons?** We plot the performance on the final test set vs. the performance on the validation set. The validation performances were visible to the participants on the leaderboard while they were tuning their models. The final test set performances were only revealed at the end of the Tweakathon. Except for a few outliers, most participants did not overfit the leaderboard. (b) **Gap between AutoML and Tweakathons?** We plot the Tweakathons vs. AutoML performance to visualize improvements obtained by manual tweaking and additional computational resources available in Tweakathons. Points above the diagonal indicate such improvements.

methods were competitive with the best solutions of the CPU track. However, no Deep Learning method was competitive with the limited compute power and time budget offered in the CPU track.

10.5.2 Scores Obtained in the 2018 Challenge

The 2018 challenge lasted 4 months (November 30, 2017 to March 31, 2018). As in the previous challenge, top-ranked solutions were obtained and open sourced. Table 10.6 shows the results of both phases of the 2018 challenge. As a reminder, this challenge had a feedback phase and a blind test phase, the performances of the winners in each phase are reported.

Performance in this challenge was slightly lower than that observed in the previous edition. This was due to the difficulty of the tasks (see below) and the fact that data sets in the feedback phase included three deceiving datasets (associated to tasks from previous challenges, but not necessarily similar to the



(a) AutoML (test set) (b) Tweakathon (test set)

Figure 10.4: **Distribution of performance on the datasets of the 2015/2016 challenge (violin plots).** We show for each dataset the performances of participants at the end of AutoML and Tweakathon phases, as revealed on the leaderboard. The median and quartiles are represented by horizontal notches. The distribution profile (as fitted with a kernel method) and its mirror image are represented vertically by the gray shaded area. We show in red the median performance over all datasets and the corresponding quartiles. (a) **AutoML (blind testing).** The first 5 datasets were provided for development purpose only and were not used for blind testing in an AutoML phase. In round 3, the code of many participants failed because of computational limits. (b) **Tweakathon (manual tweaking).** The last five datasets were only used for final blind testing and the data were never revealed for a Tweakathon. Round 3 was not particularly difficult with additional compute power and memory.

data sets used in the blind test phase) out of five. We decided to proceed this way to emulate a realistic AutoML setting. Although harder, several teams succeeded at returning submissions performing better than chance.

The winner of the challenge was the same team that won the 2015/2016 AutoML challenge: AAD Freiburg [28]. The 2018 challenge helped to incrementally improve the solution devised by this team in the previous challenge. Interestingly, the second-placed team in the challenge proposed a solution that is similar in spirit to that of the winning team. For this challenge, there was a triple tie in the third place, prizes were split among the tied teams. Among the winners, two teams used the starting kit. Most of the other teams used either the starting kit or the solution open sourced by the AAD Freiburg team in the 2015/2016 challenge.

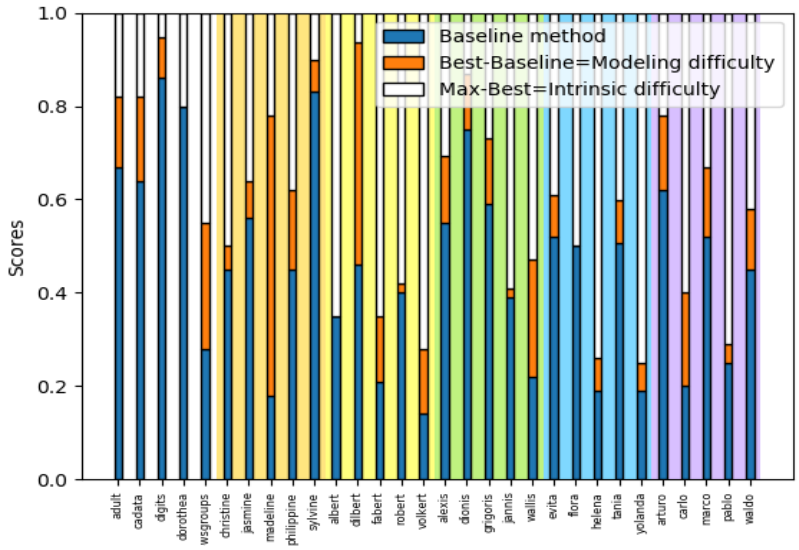
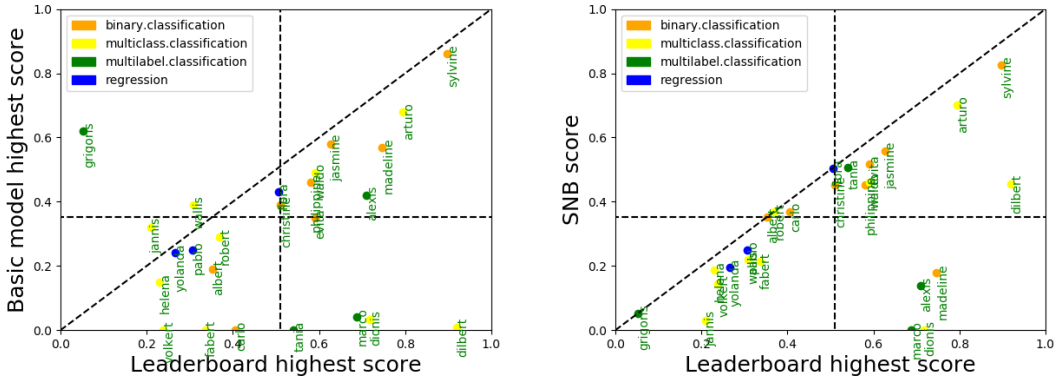


Figure 10.5: **Difficulty of tasks in the 2015/2016 challenge.** We consider two indicators of task difficulty (dataset, metric, and time budget are factored into the task): intrinsic difficulty (estimated by the performance of the winners) and modeling difficulty (difference between the performance of the winner and a baseline method, here Selective Naive Bayes (SNB)). The best tasks should have a relatively low intrinsic difficulty and a high modeling difficulty to separate participants well.

10.5.3 Difficulty of Datasets/Tasks

In this section, we assess dataset difficulty, or rather *task difficulty* since the participants had to solve prediction problems for given datasets, performance metrics, and computational time constraints. The tasks of the challenge presented a variety of difficulties, but those were not equally represented (Tables 10.2 and 10.3):

- **Categorical variables and missing data:** Few datasets had categorical variables in the 2015/2016 challenge (ADULT, ALBERT, and WALDO), and not very many variables were categorical in those datasets. Likewise, very few datasets had missing values (ADULT and ALBERT) and those included only a few missing values. So neither categorical variables nor missing data presented a real difficulty in this challenge, though ALBERT turned out to be one of the most difficult datasets because it was also one of the largest ones. This situation changed drastically for the 2018 challenge where five out of the ten datasets included categorical variables (RL, PM, RI, RH and RM) and missing values (GINA, PM, RL, RI and



(a) Leaderboard overfitting? (b) Gap AutoML/Tweakathon?

Figure 10.6: **Modeling Difficulty vs. intrinsic difficulty**. For the AutoML phases of the 2015/2016 challenge, we plot an indicator of modeling difficulty vs. and indicator of intrinsic difficulty of datasets (leaderboard highest score). (a) Modeling difficulty is estimated by the score of the best untuned model (over KNN, NaiveBayes, RandomForest and SGD(LINEAR)). (b) Modeling difficulty is estimated by the score of the Selective Naive Bayes (SNB) model. In all cases, higher scores are better and negative / NaN scores are replaced by zero. The horizontal and vertical separation lines represent the medians. The lower right quadrant represents the datasets with low intrinsic difficulty and high modeling difficulty: those are the best datasets for benchmarking purposes.

RM). These were among the main aspects that caused the low performance of most methods in the blind test phase.

- **Large number of classes.** Only one dataset had a large number of classes (DIONIS with 355 classes). This dataset turned out to be difficult for participants, particularly because it is also large and has unbalanced classes. However, datasets with large number of classes are not well represented in this challenge. HELENA, which has the second largest number of classes (100 classes), did not stand out as a particularly difficult dataset. However, in general, multi-class problems were found to be more difficult than binary classification problems.
- **Regression.** We had only four regression problems: CADATA, FLORA, YOLANDA, PABLO.
- **Sparse data.** A significant number of datasets had sparse data (DOROTHEA, FABERT, ALEXIS, WALLIS, GRIGORIS, EVITA, FLORA, TANIA, ARTURO, MARCO). Several of them turned out to be difficult, particularly ALEXIS, WALLIS, and GRIGORIS, which are large datasets in sparse format, which cause memory problems when they were introduced in round

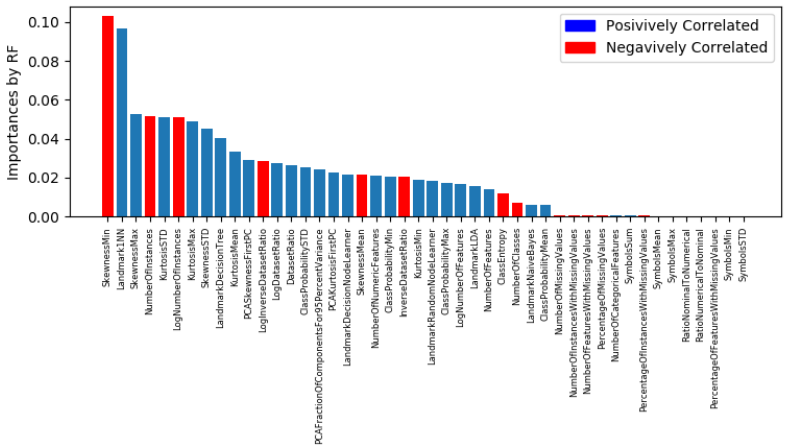


Figure 10.7: **Meta-features most predictive of dataset intrinsic difficulty (2015/2016 challenge data).** Meta-feature GINI importances are computed by a random forest regressor, trained to predict the highest participant leaderboard score using meta-features of datasets. Description of these meta-features can be found in Table 1 of the supplementary material of [25]. Blue and red colors respectively correspond to positive and negative correlations (Pearson correlations between meta features and score medians).

3 of the 2015/2016 challenge. We later increased the amount of memory on the servers and similar datasets introduced in later phases caused less difficulty.

- **Large datasets.** We expected the ratio of the number N of features over the number P_{tr} of training examples to be a particular difficulty (because of the risk of overfitting), but modern machine learning algorithm are robust against overfitting. The main difficulty was rather the PRODUCT $N * P_{tr}$. Most participants attempted to load the entire dataset in memory and convert sparse matrices into full matrices. This took very long and then caused loss in performances or program failures. Large datasets with $N * P_{tr} > 20.10^6$ include ALBERT, **ALEXIS**, DIONIS, **GRIGORIS, WALLIS, EVITA, FLORA, TANIA, MARCO**, GINA, GUILLERMO, PM, RH, RI, RICCARDO, RM. Those overlap significantly with the datasets with sparse data (in bold). For the 2018 challenge, all data sets in the final phase exceeded this threshold, and this was the reason of why the code from several teams failed to finish within the time budget. Only ALBERT and DIONIS were "truly" large (few features, but over 400,000 training examples).

- **Presence of probes:** Some datasets had a certain proportion of distractor features or irrelevant variables (probes). Those were obtained by randomly permuting the values of real features. Two-third of the datasets contained probes ADULT, CADATA, DIGITS, DOROTHEA, CHRISTINE, JASMINE, MADELINE, PHILIPPINE, SYLVINE, ALBERT, DILBERT, FABERT, JANNIS, EVITA, FLORA, YOLANDA, ARTURO, CARLO, PABLO, WALDO. This allowed us in part to make datasets that were in the public domain less recognizable.

- **Type of metric:** we used 6 metrics, as defined in section 10.4.2. The distribution of tasks in which they were used was not uniform: BAC (11), AUC (6), F1 (3), and PAC (6) for classification, and R2 (2) and ABS (2) for regression. This is because not all metrics lend themselves naturally to all types of applications.

- **Time budget:** Although in round 0 we experimented with giving different time budgets for the various datasets, we ended up assigning 1200 seconds (20 min) to all datasets in all other rounds. Because the datasets varied in size, this put more constraints on large datasets.

- **Class imbalance:** This was not a difficulty found in the 2015/2016 datasets. However, extreme class imbalance was the main difficulty for the 2018 edition. Imbalance ratios lower or equal to 1 to 10 were present in RL, PM, RH, RI, and **RM** datasets, in the latter data set class imbalance was as extreme as 1 to 1000. This was the reason why the performance of teams was low.

Figure 10.4 gives a first view of dataset/task difficulty for the 2015/2016 challenge. It captures, in a schematic way, the distribution of the participants' performance in all rounds on test data, in both AutoML and Tweakathon phases. One can see that the median performance over all datasets improves between AutoML and Tweakathon, as can be expected. Correspondingly, the average spread in performance (quartile) decreases. Let us take a closer look at the AutoML phases: The "accident" of round 3 in which many methods failed in blind testing is visible (introduction of sparse matrices and larger datasets)[10]. Round 2 (multi-class classification) appears to have also introduced a significantly higher degree of difficulty than round 1 (binary classification). In round 4, two regression problems were introduced (FLORA and YOLANDA), but it does not seem that regression was found significantly harder than multiclass classification. In round 5 no novelty was introduced. We can observe that, after round 3, the dataset median scores are scattered around the overall median. Looking at the corresponding scores in the Tweakathon phases, one can remark that, once the participants recovered from their surprise, round 3 was not particularly difficult for them. Rounds 2 and 4 were comparatively more difficult.

[10]Examples of sparse datasets were provided in round 0, but they were of smaller size.

For the datasets used in the 2018 challenge, the tasks' difficulty was clearly associated with extreme class imbalance, inclusion of categorical variables and high dimensionality in terms of $N \times P_{tr}$. However, for the 2015/2016 challenge data sets we found that it was generally difficult to guess what makes a task easy or hard, except for dataset size, which pushed participants to the frontier of the hardware capabilities and forced them to improve the computational efficiency of their methods. Binary classification problems (and multi-label problems) are intrinsically "easier" than multiclass problems, for which "guessing" has a lower probability of success. This partially explains the higher median performance in rounds 1 and 3, which are dominated by binary and multi-label classification problems. There is not a large enough number of datasets illustrating each type of other difficulties to draw other conclusions.

We ventured however to try to find summary statistics capturing overall taks difficulty. If one assumes that data are generated from an *i.i.d.*[11] process of the type:

$$y = F(\mathbf{x}, noise)$$

where y is the target value, $\mathbf{x}$ is the input feature vector, F is a function, and *noise* is some random noise drawn from an unknown distribution, then the difficuty of the learning problem can be separated in two aspects:

1. **Intrinsic difficulty**, linked to the amount of noise or the signal to noise ratio. Given an infinite amount of data and an unbiased learning machine $\hat{F}$ capable of identifying F, the prediction performances cannot exceed a given maximum value, corresponding to $\hat{F} = F$.

2. **Modeling difficulty**, linked to the bias and variance of estimators $\hat{F}$, in connection with the limited amount of training data and limited computational resources, and the possibly large number or parameters and hyper-parameters to estimate.

Evaluating the intrinsic difficulty is impossible unless we know F. Our best approximation of F is the winners' solution. **We use therefore the winners' performance as an estimator of the best achievable performance.** This estimator may have both bias and variance: it is possibly biased because the winners may be under-fitting training data; it may have variance because of the limited amount of test data. Under-fitting is difficult to test. Its symptoms may be that the variance or the entropy of the predictions is less than those of the target values.

Evaluating the modeling difficulty is also impossible unless we know F and the model class. In the absence of knowledge on the model class, data scientists often use generic predictive models, agnostic with respect to the data generating process. Such models range from very basic models that are highly biased towards "simplicity" and smoothness of predictions (e.g., regularized linear models) to highly versatile unbiased models that can learn any function

[11] Independently and Identically Distributed samples.

given enough data (e.g., ensembles of decision trees). To indirectly assess modeling difficulty, we resorted to use the difference in performance between *the method of the challenge winner* and that of (a) the best of four "untuned" basic models (taken from classical techniques provided in the scikit-learn library [55] with default hyper-parameters) or (b) Selective Naive Bayes (SNB) [12, 13], a highly regularized model (biased towards simplicity), providing a very robust and simple baseline.

Figures 10.5 and 10.6 give representations of our estimates of intrinsic and modeling difficulties for the 2015/2016 challenge datasets. It can be seen that the datasets of round 0 were among the easiest (except perhaps NEWSGROUP). Those were relatively small (and well-known) datasets. Surprisingly, the datasets of round 3 were also rather easy, despite the fact that most participants failed on them when they were introduced (largely because of memory limitations: scikit-learn algorithms were not optimized for sparse datasets and it was not possible to fit in memory the data matrix converted to a dense matrix). Two datasets have a small intrinsic difficulty but a large modeling difficulty: MADELINE and DILBERT. MADELINE is an artificial dataset that is very non-linear (clusters or 2 classes positioned on the vertices of a hyper-cube in a 5 dimensional space) and therefore very difficult for Naïve Bayes. DILBERT is an image recognition dataset with images of objects rotated in all sorts of positions, also very difficult for Naïve Bayes. The datasets of the last 2 phases seem to have a large intrinsic difficulty compared to the modeling difficulty. But this can be deceiving because the datasets are new to the machine learning community and the performances of the winners may still be far from the best attainable performance.

We attempted to predict the intrinsic difficulty (as measured by the winners' performance) from the set of meta features used by AAD Freiburg for meta-learning [25], which are part of OpenML [67], using a Random Forest classifier and ranked the meta features in order of importance (most selected by RF). The list of meta features is provided in the appendix. The three meta-features that predict dataset difficulty best (Figure 10.7) are:

- LandmarkDecisionTree: performance of a decision tree classifier.
- Landmark1NN: performance of a nearest neighbor classifier.
- SkewnessMin: min over skewness of all features. Skewness measures the symmetry of a distribution. A positive skewness value means that there is more weight in the left tail of the distribution.

10.5.4 Hyper-Parameter Optimization

Many participants used the scikit-learn (sklearn) package, including the winning group AAD Freiburg, which produced the auto-sklearn software. We used the auto-sklearn API to conduct post-challenge systematic studies of the effectiveness of hyper-parameter optimization. We compared the performances obtained with default hyper-parameter settings in scikit-learn and with hyper-

parameters optimized with auto-sklearn[12], both within the time budgets as imposed during the challenge, for four "representative" basic methods: k-nearest neighbors (KNN), naive Bayes (NB), Random Forest (RF), and a linear model trained with stochastic gradient descent (SGD-linear[13]). The results are shown in Figure 10.8. We see that hyper-parameter optimization usually improves performance, but not always. The advantage of hyper-parameter tuning comes mostly from its flexibility of switching the optimization metric to the one imposed by the task and from finding hyper-parameters that work well given the current dataset and metric. However, in some cases it was not possible to perform hyper-parameter optimization within the time budget due to the data set size (score ≤ 0). Thus, there remains future work on how to perform thorough hyper-parameter tuning given rigid time constraints and huge datasets.

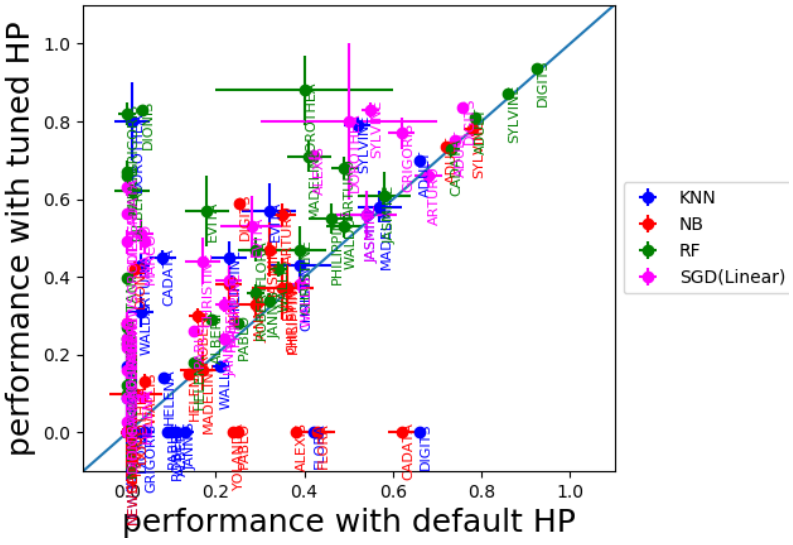


Figure 10.8: **Hyper-parameter tuning (2015/2016 challenge data).** We compare the performances obtained with default hyper-parameters and those with hyper-parameters optimized with auto-sklearn, within the same time budgets as given during the challenge. The performances of predictors which failed to return results in the alloted time are replaced by zero. Note that returning a prediction of chance level also resulted in a score of zero.

We also compared the performances obtained with different scoring metrics (Figure 10.10). Basic methods do not give a choice of metrics to be optimized, but auto-sklearn post-fitted the metrics of the challenge tasks. Consequently, when "common metrics" (BAC and R^2) are used, the method of the challenge

[12] we use sklearn 0.16.1 and auto-sklearn 0.4.0 to mimic the challenge environment
[13] we set the loss of SGD to be 'log' in scikit-learn for these experiments

winners, which is not optimized for BAC/R^2, does not usually outperfom basic methods. Conversely, when the metrics of the challenge are used, there is often a clear gap between the basic methods and the winners, but not always (RF-auto usually shows a comparable performance, sometimes even outperforms the winners).

Figure 10.9: **Comparison of metrics (2015/2016 challenge).** (a) We used the normalized balanced accuracy for all classification problems and the R^2 metric for regression problems. (b) We used the metrics of the challenge. By comparing the two figures, we can see that the winner remains top-ranking in most cases, regardless of the metric. There is no basic method that dominates all others. Though RF-auto (Random Forest with optimised HP) is very strong, it is often outperformed by other methods and sometimes by RF-def (Random Forest with default HP). Generally, under tight computational constraints, optimizing HP does not always pay, considering the number of hollow circles that come on top. For KNN though, time permitting, optimizing HP generally helps a lot. Interestingly, KNN can win, even over the challenge winners, on some datasets. Plain linear model SGD-def sometimes wins when common metrics are used, but the winners perform better with the metrics of the challenge. Overall, the technique of the winners proved to be effective.

(a) Challenge metrics (b) Common metrics

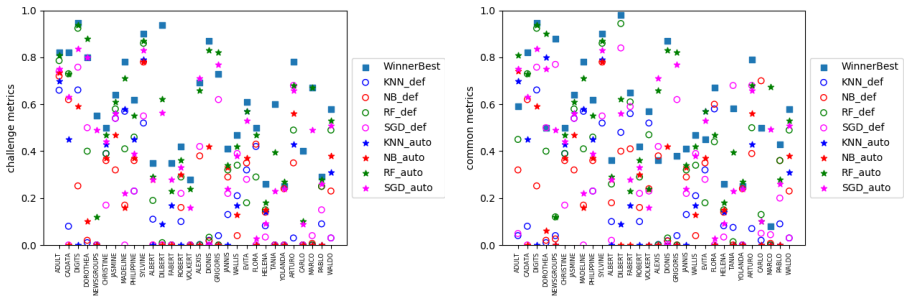
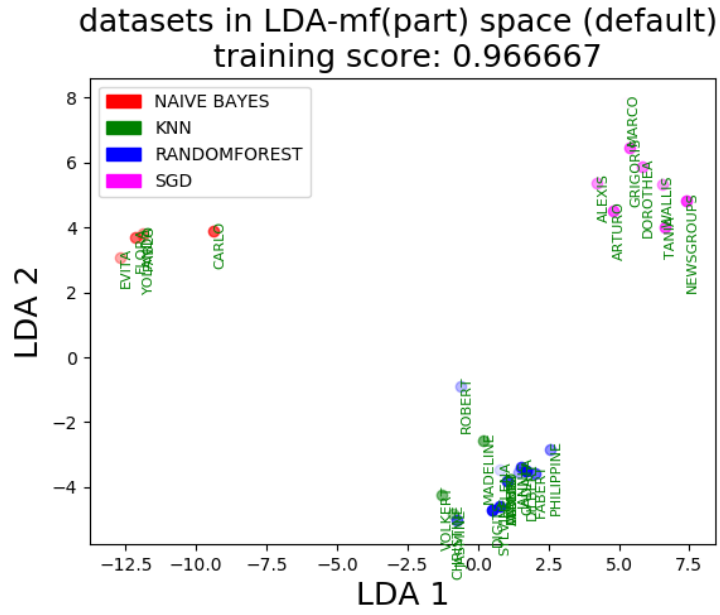
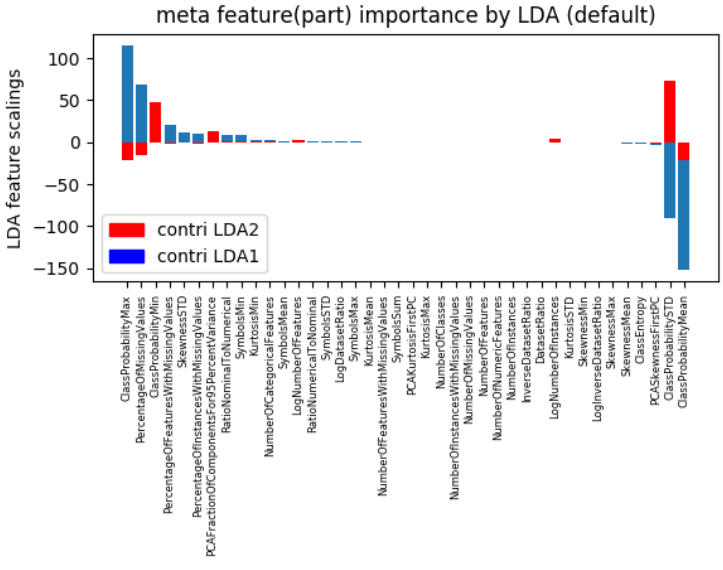


Figure 10.10: **Comparison of metrics (2015/2016 challenge).** (a) We used the metrics of the challenge. (b) We used the normalized balanced accuracy for all classification problems and the R^2 metric for regression problems. By comparing the two figures, we can see that the winner remains top-ranking in most cases, regardless of the metric. There is no basic method that dominates all others. Although RF-auto (Random Forest with optimized HP) is very strong, it is sometimes outperformed by other methods. Plain linear model SGD-def sometimes wins when common metrics are used, but the winners perform better with the metrics of the challenge. Overall, the technique of the winners proved to be effective.



(a)



(b)

Figure 10.11: **Linear Discriminant Analysis.** (a) **Dataset scatter plot in principal axes**. We have trained a LDA using X=meta features, except landmarks; y=which model won of four basic models (NB, SGD-linear, KNN, RF). The performance of the basic models is measured using the common metrics. The models were trained with default hyper-parameters. In the space of the two first LDA components, each point represents one dataset. The colors denote the winning basic models. The opacity reflects the scores of the corresponding winning model (more opaque is better). (b) **Meta feature importances** computed as scaling factors of each LDA component.

10.5.5 Meta-Learning

One question is whether meta-learning [14] is possible, that is learning to predict whether a given classifier will perform well on future datasets (without actually training it), based on its past performances on other datasets. We investigated whether it is possible to predict which basic method will perform best based on the meta-learning features of auto-sklearn (see the appendix). We removed the "Landmark" features from the set of meta features because those are performances of basic predictors (albeit rather poor ones with many missing values), which would lead to a form of "data leakage".

We used four basic predictors:

- NB: Naive Bayes
- SGD-linear: Linear model (trained with stochastic gradient descent)
- KNN: K-nearest neighbors
- RF: Random Forest

We used the implementation of the scikit-learn library with default hyper-parameter settings. In Figure 10.11, we show the two first Linear Discriminant Analysis (LDA) components, when training an LDA classifier on the meta-features to predict which basic classifier will perform best. The methods separate into three distinct clusters, one of them grouping the non-linear methods that are poorly separated (KNN and RF) and the two others being NB and linear-SGD.

The features that are most predictive all have to do with "ClassProbability" and "PercentageOfMissingValues", indicating that the class imbalance and/or large number of classes (in a multi-class problem) and the percentage of missing values might be important, but there is a high chance of overfitting as indicated by an unstable ranking of the best features under resampling of the training data.

10.5.6 Methods Used in the Challenges

A brief **description of methods** used in both challenges is provided in the appendix, together with the results of a survey on methods that we conducted after the challenges. In light of the overview of Section 10.2 and the results presented in the previous section, we may wonder whether a dominant methodology for solving the AutoML problem has emerged and whether particular technical solutions were widely adopted. In this section we call "model space" the set of all models under consideration. We call "basic models" (also called elsewhere "simple models", "individual models", "base learners") the member of a library of models from which our hyper-models of model ensembles are built.

Ensembling: dealing with over-fitting and any-time learning. Ensembling is the big AutoML challenge series winner since it is used by over 80% of the participants and by all the top-ranking ones. While a few years

ago the hottest issue in model selection and hyper-parameter optimization was over-fitting, in present days the problem seems to have been largely avoided by using ensembling techniques. In the 2015/2016 challenge, we varied the ratio of number of training examples over number of variables (Ptr/N) by several orders of magnitude. Five datasets had a ratio Ptr/N lower than one (dorothea, newsgroup, grigoris, wallis, and flora), which is a case lending itself particularly to over-fitting. Although Ptr/N is the most predictive variable of the median performance of the participants, there is no indication that the datasets with $Ptr/N < 1$ were particularly difficult for the participants (Figure 10.5). Ensembles of predictors have the additional benefit of addressing in a simple way the "any-time learning" problem by growing progressively a bigger ensemble of predictors, improving performance over time. All trained predictors are usually incorporated in the ensemble. For instance, if cross-validation is used, the predictors of all folds are directly incorporated in the ensemble, which saves the computational time of retraining a single model on the best HP selected and may yield more robust solutions (though slightly more biased due to the smaller sample size). The approaches differ in the way they weigh the contributions of the various predictors. Some methods use the same weight for all predictors (this is the case of bagging methods such as Random Forest and of Bayesian methods that sample predictors according to their posterior probability in model space). Some methods assess the weights of the predictors as part of learning (this is the case of boosting methods, for instance). One simple and effective method to create ensembles of heterogeneous models was proposed by [16]. It was used successfully in several past challenges, e.g., [52] and is the method implemented by the $aad\_freibug$ team, one of the strongest participants in both challenges [25]. The method consists in cycling several times over all trained model and incorporating in the ensemble at each cycle the model which most improves the performance of the ensemble. Models vote with weight 1, but they can be incorporated multiple times, which de facto results in weighting them. This method permits to recompute very fast the weights of the models if cross-validated predictions are saved. Moreover, the method allows optimizing the ensemble for any metric by post-fitting the predictions of the ensemble to the desired metric (an aspect which was important in this challenge).

Model evaluation: cross-validation or simple validation. Evaluating the predictive accuracy of models is a critical and necessary building block of any model selection of ensembling method. Model selection criteria computed from the predictive accuracy of basic models evaluated from training data, by training a single time on all the training data (possibly at the expense of minor additional calculations), such as performance bounds, were not used at all, as was already the case in previous challenges we organized [35]. Cross-validation was widely used, particularly K-fold cross-validation. However, basic models were often "cheaply" evaluated on just one fold to allow quickly discarding non-promising areas of model space. This is a technique used more and more frequently to help speed up search. Another speed-up strategy is to train on a subset of the training examples and monitor the learning curve. The "freeze-thaw" strategy [64] halts training of models that do not look promising on the

basis of the learning curve, but may restart training them at a later point. This was used, e.g., by [48] in the 2015/2016 challenge.

Model space: Homogeneous vs. heterogeneous. An unsettled question is whether one should search a large or small model space. The challenge did not allow us to give a definite answer to this question. Most participants opted for searching a relatively large model space, including a wide variety of models found in the scikit-learn library. Yet, one of the strongest entrants (the Intel team) submitted results simply obtained with a boosted decision tree (i.e., consisting of a homogeneous set of weak learners/basic models). Clearly, it suffices to use just one machine learning approach that is a universal approximator to be able to learn anything, given enough training data. So why include several? It is a question of rate of convergence: how fast we climb the learning curve. Including stronger basic models is one way to climb the learning curve faster. Our post-challenge experiments (Figure 10.10) reveal that the scikit-learn version of Random Forest (an ensemble of homogeneous basic models – decision trees) does not usually perform as well as the winners' version, hinting that there is a lot of know-how in the Intel solution, which is also based on ensembles of decision tree, that is not captured by a basic ensemble of decision trees such as RF. We hope that more principled research will be conducted on this topic in the future.

Search strategies: Filter, wrapper, and embedded methods. With the availability of powerful machine learning toolkits like scikit-learn (on which the starting kit was based), the temptation is great to implement **all-wrapper methods** to solve the CASH (or "full model selection") problem. Indeed, most participants went that route. Although a number of ways of optimizing hyper-parameters with **embedded methods** for several basic classifiers have been published [35], they each require changing the implementation of the basic methods, which is time-consuming and error-prone compared to using already debugged and well-optimized library version of the methods. Hence practitioners are reluctant to invest development time in the implementation of embedded methods. A notable exception is the software of *marc.boulle*, which offers a self-contained hyper-parameter free solution based on Naive Bayes, which includes re-coding of variables (grouping or discretization) and variable selection. See the appendix.

Multi-level optimization: Another interesting issue is whether multiple levels of hyper-parameters should be considered for reasons of computational effectiveness or overfitting avoidance. In the Bayesian setting, for instance, it is quite feasible to consider a hierarchy of parameters/hyper-parameters and several levels of priors/hyper-priors. However, it seems that for practical computational reasons, in the AutoML challenges, the participants use a shallow organization of hyper-parameter space and avoid nested cross-validation loops.

Time management: Exploration vs. exploitation tradeoff: With a tight time budget, efficient search strategies must be put into place to monitor the exploration/exploitation tradeoff. To compare strategies, we show in the appendix learning curves for two top ranking participants who adopted very different methods: *Abhishek* and *aad_freiburg*. The former uses heuristic meth-

ods based on prior human experience while the latter initializes search with models predicted to be best suited by meta-learning, then performs Bayesian optimization of hyper-parameters. *Abhishek* seems to often start with a better solution but explores less effectively. In contrast, *aad_freiburg* starts lower but often ends up with a better solution. Some elements of randomness in the search are useful to arrive at better solutions.

Preprocessing and feature selection: The datasets had intrinsic difficulties that could be in part addressed by preprocessing or special modifications of algorithms: sparsity, missing values, categorical variables, and irrelevant variables. Yet it appears that among the top-ranking participants, preprocessing has not been a focus of attention. They relied on the simple heuristics provided in the starting kit: replacing missing values by the median and adding a missingness indicator variable, one-hot-encoding of categorical variables. Simple normalizations were used. The irrelevant variables were ignored by 2/3 of the participants and no use of feature selection was made by top-ranking participants. The methods used that involve ensembling seem to be intrinsically robust against irrelevant variables. More details from the fact sheets are found in the appendix.

Unsupervised learning: Despite the recent regain of interest in unsupervised learning spurred by the Deep Learning community, in the AutoML challenge series, unsupervised learning is not widely used, except for the use of classical space dimensionality reduction techniques such as ICA and PCA. See the appendix for more details.

Transfer learning and meta learning: To our knowledge, only *aad_freiburg* relied on meta-learning to initialize their hyper-parameter search. To that end, they used datasets of OpenML[14]. The number of datasets released and the diversity of tasks did not allow the participants to perform effective transfer learning or meta learning.

Deep learning: The type of computations resources available in AutoML phases ruled out the use of Deep Learning, except in the GPU track. However, even in that track, the Deep Learning methods did not come out ahead. One exception is *aad_freiburg*, who used Deep Learning in Tweakathon rounds three and four and found it to be helpful for the datasets Alexis, Tania and Yolanda.

Task and metric optimization: There were four types of tasks (regression, binary classification, multi-class classification, and multi-label classification) and six scoring metrics (R2, ABS, BAC, AUC, F1, and PAC). Moreover, class balance and number of classes varied a lot for classification problems. Moderate effort has been put into designing methods optimizing specific metrics. Rather, generic methods were used and the outputs post-fitted to the target metrics by cross-validation or through the ensembling method.

Engineering: One of the big lessons of the AutoML challenge series is that most methods fail to return results in all cases, not a "good" result, but "any" reasonable result. Reasons for failure include "out of time" and "out of memory" or various other failures (e.g., numerical instabilities). We are still very far from

[14]https://www.openml.org/

having "basic models" that run on all datasets. One of the strengths of auto-sklearn is to ignore those models that fail and generally find at least one that returns a result.

Parallelism: The computers made available had several cores, so in principle, the participants could make use of parallelism. One common strategy was just to rely on numerical libraries that internally use such parallelism automatically. The *aad_freiburg* team used the different cores to launch in parallel model search for different datasets (since each round included 5 datasets). These different uses of computational resources are visible in the learning curves (see the appendix).

10.6 Discussion

We briefly summarize the main questions we asked ourselves and the main findings:

1. **Was the provided time budget sufficient to complete the tasks of the challenge?** We drew learning curves as a function of time for the winning solution of $aad\_freiburg$ (auto-sklearn, see the appendix). This revealed that for most datasets, performances still improved well beyond the time limit imposed by the organizers. Although for about half the datasets the improvement is modest (no more that 20% of the score obtained at the end of the imposed time limit), for some datasets the improvement was very large (more than 2x the original score). The improvements are usually gradual, but sudden performance improvements occur. For instance, for Wallis, the score doubled suddenly at 3x the time limit imposed in the challenge. As also noted by the authors of the auto-sklearn package [25], it has a slow start but in the long run gets performances close to the best method.

2. **Are there tasks that were significantly more difficult than others for the participants?** Yes, there was a very wide range of difficulties for the tasks as revealed by the dispersion of the participants in terms of average (median) and variability (third quartile) of their scores. Madeline, a synthetic dataset featuring a very non-linear task, was very difficult for many participants. Other difficulties that caused failures to return a solution included large memory requirements (particularly for methods that attempted to convert sparse matrices to full matrices), and short time budgets for datasets with large number of training examples and/or features or with many classes or labels.

3. **Are there meta-features of datasets and methods providing useful insight to recommend certain methods for certain types of datasets?** The *aad_freiburg* team used a subset of 53 meta-features (a superset of the simple statistics provided with the challenge datasets) to measure similarity between datasets. This allowed them to conduct hyper-parameter search more effectively by initializing the search with settings

identical to those selected for similar datasets previously processed (a form of meta-learning). Our own analysis revealed that it is very difficult to predict the predictors' performances from the meta-features, but it is possible to predict relatively accurately which "basic method" will perform best. With LDA, we could visualize how datasets recoup in two dimensions and show a clean separation between datasets "preferring" Naive Bayes, linear SGD, or KNN, or RF. This deserves further investigation.

4. **Does hyper-parameter optimization really improve performance over using default values?** The comparison we conducted reveals that optimizing hyper-parameters rather than choosing default values for a set of four basic predictive models (K-nearest neighbors, Random Forests, linear SGD, and Naive Bayes) is generally beneficial. In the majority of cases (but not always), hyper-parameter optimization (hyper-opt) results in better performances than default values. Hyper-opt sometimes fails because of time or memory limitations, which gives room for improvement.

5. **How do winner's solutions compare with basic scikit-learn models?** They compare favorably. For example, the results of basic models whose parameters have been optimized do not yield generally as good results as running auto-sklearn. However, a basic model with default HP sometimes outperforms this same model tuned by auto-sklearn.

10.7 Conclusion

We have analyzed the results of several rounds of AutoML challenges.

Our design of the first AutoML challenge (2015/2016) was satisfactory in many respects. In particular, we attracted a large number of participants (over 600), attained results that are statistically significant, and advanced the state of the art to automate machine learning. Publicly available libraries have emerged as a result of this endeavor, including auto-sklearn.

In particular, we designed a benchmark with a large number of diverse datasets, with large enough test sets to separate top-ranking participants. It is difficult to anticipate the size of the test sets needed, because the error bars depend on the performances attained by the participants, so we are pleased that we made reasonable guesses. Our simple rule-of-thumb "N=50/E" where N is the number of test samples and E the error rate of the smallest class seems to be widely applicable. We made sure that the datasets were neither too easy nor too hard. This is important to be able to separate participants. To quantify this, we introduced the notion of "intrinsic difficulty" and "modeling difficulty". Intrinsic difficulty can be quantified by the performance of the best method (as a surrogate for the best attainable performance, i.e., the Bayes rate for classification problems). Modeling difficulty can be quantified by the spread in performance between methods. Our best problems have relatively low "intrinsic difficulty" and high "modeling difficulty". However, the diversity of the 30 datasets of our first 2015/2016 challenge is both a feature and a curse: it

allows us to test the robustness of software across a variety of situations, but it makes meta-learning very difficult, if not impossible. Consequently, external meta-learning data must be used if meta-learning is to be explored. This was the strategy adopted by the AAD Freiburg team, which used the OpenML data for meta training. Likewise, we attached different metrics to each dataset. This contributed to making the tasks more realistic and more difficult, but also made meta-learning harder. In the second 2018 challenge, we diminished the variety of datasets and used a single metric.

With respect to task design, we learned that the devil is in the details. The challenge participants solve exactly the task proposed to the point that their solution may not be adaptable to seemingly similar scenarios. In the case of the AutoML challenge, we pondered whether the metric of the challenge should be the area under the learning curve or one point on the learning curve (the performance obtained after a fixed maximum computational time elapsed). We ended up favoring the second solution for practical reasons. Examining after the challenge the learning curves of some participants, it is quite clear that the two problems are radically different, particularly with respect to strategies mitigating "exploration" and "exploitation". This prompted us to think about the differences between "fixed time" learning (the participants know in advance the time limit and are judged only on the solution delivered at the end of that time) and "any time learning" (the participants can be stopped at any time and asked to return a solution). Both scenarios are useful: the first one is practical when models must be delivered continuously at a rapid pace, e.g. for marketing applications; the second one is practical in environments when computational resources are unreliable and interruption may be expected (e.g. people working remotely via an unreliable connection). This will influence the design of future challenges.

The two versions of AutoML challenge we have run differ in the difficulty of transfer learning. In the 2015/2016 challenge, round 0 introduced a sample of all types of data and difficulties (types of targets, sparse data or not, missing data or not, categorical variables of not, more examples than features or not). Then each round ramped up difficulty. The datasets of round 0 were relatively easy. Then at each round, the code of the participants was blind-tested on data that were one notch harder than in the previous round. Hence transfer was quite hard. In the 2018 challenge, we had 2 phases, each with 5 datasets of similar difficulty and the datasets of the first phase were each matched with one corresponding dataset on a similar task. As a result, transfer was made simpler.

Concerning the starting kit and baseline methods, we provided code that ended up being the basis of the solution of the majority of participants (with notable exceptions from industry such as Intel and Orange who used their own "in house" packages). Thus, we can question whether the software provided biased the approaches taken. Indeed, all participants used some form of ensemble learning, similarly to the strategy used in the starting kit. However, it can be argued that this is a "natural" strategy for this problem. But, in general, the question of providing enough starting material to the participants without biasing the challenge in a particular direction remains a delicate issue.

From the point of view of challenge protocol design, we learned that it is difficult to keep teams focused for an extended period of time and go through many challenge phases. We attained a large number of participants (over 600) over the whole course of the AutoML challenge, which lasted over a year (2015/2016) and was punctuated by several events (such as hackathons). However, it may be preferable to organize yearly events punctuated by workshops. This is a natural way of balancing competition and cooperation since workshops are a place of exchange. Participants are naturally rewarded by the recognition they gain via the system of scientific publications. As a confirmation of this conjecture, the second instance of the AutoML challenge (2017/2018) lasting only 4 months attracted nearly 300 participants.

One important novelty of our challenge design was code submission. Having the code of the participants executed on the same platform under rigorously similar conditions is a great step towards fairness and reproducibility, as well as ensuring the viability of solution from the computational point of view. We required the winners to release their code under an open source licence to win their prizes. This was good enough an incentive to obtain several software publications as the "product" of the challenges we organized. In our second challenge (AutoML 2018), we used Docker. Distributing Docker images makes it possible for anyone downloading the code of the participants to easily reproduce the results without stumbling over installation problems due to inconsistencies in computer environments and libraries. Still the hardware may be different and we find that, in post-challenge evaluations, changing computers may yield significant differences in results. Hopefully, with the proliferation of affordable cloud computing, this will become less of an issue.

The AutoML challenge series is only beginning. Several new avenues are under study. For instance, we are preparing the NIPS 2018 Life Long Machine Learning challenge in which participants will be exposed to data whose distribution slowly drifts over time. We are also looking at a challenge of automatic machine learning where we will focus on transfer from similar domains.

Acknowledgments

Microsoft supported the organization of this challenge and donated the prizes and cloud computing time on Azure. This project received additional support from the Laboratoire d'Informatique Fondamentale (LIF, UMR CNRS 7279) of the University of Aix Marseille, France, via the LabeX Archimede program, the Laboratoire de Recheche en Informatique of Paris Sud University, and INRIA-Saclay as part of the TIMCO project, as well as the support from the Paris-Saclay Center for Data Science (CDS). Additional computer resources were provided generously by J. Buhmann, ETH Zürich. This work has been partially supported by the Spanish project TIN2016-74946-P (MINECO/FEDER, UE) and CERCA Programme / Generalitat de Catalunya. The datasets released were selected among 72 datasets that were donated (or formatted using data publicly available) by the co-authors and by: Y. Aphinyanaphongs, O. Chapelle, Z. Iftikhar Malhi, V. Lemaire, C.-J. Lin, M. Madani, G. Stolovitzky,

H.-J. Thiesen, and I. Tsamardinos. Many people provided feedback to early designs of the protocol and/or tested the challenge platform, including: K. Bennett, C. Capponi, G. Cawley, R. Caruana, G. Dror, T. K. Ho, B. Kégl, H. Larochelle, V. Lemaire, C.-J. Lin, V. Ponce López, N. Macia, S. Mercer, F. Popescu, D. Silver, S. Treguer, and I. Tsamardinos. The software developers who contributed to the implementation of the Codalab platform and the sample code include E. Camichael, I. Chaabane, I. Judson, C. Poulain, P. Liang, A. Pesah, L. Romaszko, X. Baro Solé, E. Watson, F. Zhingri, M. Zyskowski. Some initial analyses of the challenge results were performed by I. Chaabane, J. Lloyd, N. Macia, and A. Thakur were incorporated in this paper. Katharina Eggensperger, Syed Mohsin Ali and Matthias Feurer helped with the organization of the Beat AutoSKLearn challenge. Matthias Feurer also contributed to the simulations of running auto-sklearn on 2015-2016 challenge datasets.

Bibliography

[1] Alamdari, A.R.S.A., Guyon, I.: Quick start guide for CLOP. Tech. rep., Graz University of Technology and Clopinet (May 2006)

[2] Andrieu, C., Freitas, N.D., Doucet, A.: Sequential MCMC for Bayesian model selection. In: IEEE Signal Processing Workshop on Higher-Order Statistics. pp. 130–134 (1999)

[3] Assunção, F., Lourenço, N., Machado, P., Ribeiro, B.: Denser: Deep evolutionary network structured representation. arXiv preprint arXiv:1801.01563 (2018)

[4] Baker, B., Gupta, O., Naik, N., Raskar, R.: Designing neural network architectures using reinforcement learning. arXiv preprint arXiv:1611.02167 (2016)

[5] Bardenet, R., Brendel, M., Kégl, B., Sebag, M.: Collaborative hyperparameter tuning. In: 30th International Conference on Machine Learning. vol. 28, pp. 199–207. JMLR Workshop and Conference Proceedings (May 2013)

[6] Bengio, Y., Courville, A., Vincent, P.: Representation learning: A review and new perspectives. IEEE Transactions on Pattern Analysis and Machine Intelligence 35(8), 1798–1828 (2013)

[7] Bennett, K.P., Kunapuli, G., Jing Hu, J.S.P.: Bilevel optimization and machine learning. In: Computational Intelligence: Research Frontiers, Lecture Notes in Computer Science, vol. 5050, pp. 25–47. Springer (2008)

[8] Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of Machine Learning Research 13(Feb), 281–305 (2012)

[9] Bergstra, J., Yamins, D., Cox, D.D.: Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In: 30th International Conference on Machine Learning. vol. 28, pp. 115–123 (2013)

[10] Bergstra, J.S., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Advances in Neural Information Processing Systems. pp. 2546–2554 (2011)

[11] Blum, A.L., Langley, P.: Selection of relevant features and examples in machine learning. Artificial Intelligence 97(1-2), 273–324 (December 1997)

[12] Boullé, M.: Compression-based averaging of selective naive bayes classifiers. Journal of Machine Learning Research 8, 1659–1685 (2007), http://dl.acm.org/citation.cfm?id=1314554

[13] Boullé, M.: A parameter-free classification method for large scale learning. Journal of Machine Learning Research 10, 1367–1385 (2009), http://doi.acm.org/10.1145/1577069.1755829

[14] Brazdil, P., Carrier, C.G., Soares, C., Vilalta, R.: Metalearning: Applications to data mining. Springer Science & Business Media (2008)

[15] Breiman, L.: Random forests. Machine Learning 45(1), 5–32 (2001)

[16] Caruana, R., Niculescu-Mizil, A., Crew, G., Ksikes, A.: Ensemble selection from libraries of models. In: 21st International Conference on Machine Learning. pp. 18–. ACM (2004)

[17] Cawley, G.C., Talbot, N.L.C.: Preventing over-fitting during model selection via Bayesian regularisation of the hyper-parameters. Journal of Machine Learning Research 8, 841–861 (April 2007)

[18] Colson, B., Marcotte, P., Savard, G.: An overview of bilevel programming. Annals of Operations Research 153, 235–256 (2007)

[19] Dempe, S.: Foundations of bilevel programming. Kluwer Academic Publishers (2002)

[20] Dietterich, T.G.: Approximate statistical test for comparing supervised classification learning algorithms. Neural Computation 10(7), 1895–1923 (1998)

[21] Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. Wiley, 2nd edn. (2001)

[22] Efron, B.: Estimating the error rate of a prediction rule: Improvement on cross-validation. Journal of the American Statistical Association 78(382), 316–331 (1983)

[23] Eggensperger, K., Feurer, M., Hutter, F., Bergstra, J., Snoek, J., Hoos, H., Leyton-Brown, K.: Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In: NIPS workshop on Bayesian Optimization in Theory and Practice (2013)

[24] Escalante, H.J., Montes, M., Sucar, L.E.: Particle swarm model selection. Journal of Machine Learning Research 10, 405–440 (2009)

[25] Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Proceedings of the Neural Information Processing Systems, pp. 2962–2970 (2015), `https://github.com/automl/auto-sklearn`

[26] Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Methods for improving bayesian optimization for automl. In: Proceedings of the International Conference on Machine Learning 2015, Workshop on Automatic Machine Learning (2015)

[27] Feurer, M., Springenberg, J., Hutter, F.: Initializing bayesian hyperparameter optimization via meta-learning. In: Proceedings of the AAAI Conference on Artificial Intelligence. pp. 1128–1135 (2015)

[28] Feurer, M., Eggensperger, K., Falkner, S., Lindauer, M., Hutter, F.: Practical automated machine learning for the automl challenge 2018. In: International Workshop on Automatic Machine Learning at ICML (2018), `https://sites.google.com/site/automl2018icml/`

[29] Friedman, J.H.: Greedy function approximation: A gradient boosting machine. The Annals of Statistics 29(5), 1189–1232 (2001)

[30] Ghahramani, Z.: Unsupervised learning. In: Advanced Lectures on Machine Learning. Lecture Notes in Computer Science, vol. 3176, pp. 72–112. Springer Berlin Heidelberg (2004)

[31] Guyon, I.: Challenges in Machine Learning book series. Microtome (2011-2016), `http://www.mtome.com/Publications/CiML/ciml.html`

[32] Guyon, I., Bennett, K., Cawley, G., Escalante, H.J., Escalera, S., Ho, T.K., Macià, N., Ray, B., Saeed, M., Statnikov, A., Viegas, E.: AutoML challenge 2015: Design and first results. In: Proc. of AutoML 2015@ICML (2015), `https://drive.google.com/file/d/0BzRGLkqgrI-qWkpzcGw4bFpBMUk/view`

[33] Guyon, I., Bennett, K., Cawley, G., Escalante, H.J., Escalera, S., Ho, T.K., Macià, N., Ray, B., Saeed, M., Statnikov, A., Viegas, E.: Design of the 2015 ChaLearn AutoML challenge. In: International Joint Conference on Neural Networks (2015), `http://www.causality.inf.ethz.ch/AutoML/automl_ijcnn15.pdf`

[34] Guyon, I., Chaabane, I., Escalante, H.J., Escalera, S., Jajetic, D., Lloyd, J.R., Macía, N., Ray, B., Romaszko, L., Sebag, M., Statnikov, A., Treguer, S., Viegas, E.: A brief review of the ChaLearn AutoML challenge. In: Proc. of AutoML 2016@ICML (2016), https://docs.google.com/a/chalearn.org/viewer?a=v&pid=sites&srcid=Y2hhbGVhcm4ub3JnfGF1dG9tbHxneDoyYThjZjhhNzRjMzI3MTg4

[35] Guyon, I., Alamdari, A.R.S.A., Dror, G., Buhmann, J.: Performance prediction challenge. In: the International Joint Conference on Neural Networks. pp. 1649–1656 (2006)

[36] Guyon, I., Bennett, K., Cawley, G., Escalante, H.J., Escalera, S., Ho, T.K., Ray, B., Saeed, M., Statnikov, A., Viegas, E.: Automl challenge 2015: Design and first results (2015)

[37] Guyon, I., Cawley, G., Dror, G.: Hands-On Pattern Recognition: Challenges in Machine Learning, Volume 1. Microtome Publishing, USA (2011)

[38] Guyon, I., Gunn, S., Nikravesh, M., Zadeh, L. (eds.): Feature extraction, foundations and applications. Studies in Fuzziness and Soft Computing, Physica-Verlag, Springer (2006)

[39] Hastie, T., Rosset, S., Tibshirani, R., Zhu, J.: The entire regularization path for the support vector machine. Journal of Machine Learning Research 5, 1391–1415 (2004)

[40] Hastie, T., Tibshirani, R., Friedman, J.: The elements of statistical learning: Data mining, inference, and prediction. Springer, 2nd edn. (2001)

[41] Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Proceedings of the conference on Learning and Intelligent OptimizatioN (LION 5) (2011)

[42] Ioannidis, J.P.A.: Why most published research findings are false. PLoS Medicine 2(8), e124 (August 2005)

[43] Jordan, M.I.: On statistics, computation and scalability. Bernoulli 19(4), 1378–1390 (September 2013)

[44] Keerthi, S.S., Sindhwani, V., Chapelle, O.: An efficient method for gradient-based adaptation of hyperparameters in SVM models. In: Advances in Neural Information Processing Systems (2007)

[45] Klein, A., Falkner, S., Bartels, S., Hennig, P., Hutter, F.: Fast bayesian hyperparameter optimization on large datasets. In: Electronic Journal of Statistics. vol. 11 (2017)

[46] Kohavi, R., John, G.H.: Wrappers for feature selection. Artificial Intelligence 97(1-2), 273–324 (December 1997)

[47] Langford, J.: Clever methods of overfitting (2005), blog post at http://hunch.net/?p=22

[48] Lloyd, J.: Freeze Thaw Ensemble Construction. `https://github.com/jamesrobertlloyd/automl-phase-2` (2016)

[49] Momma, M., Bennett, K.P.: A pattern search method for model selection of support vector regression. In: In Proceedings of the SIAM International Conference on Data Mining. SIAM (2002)

[50] Moore, G., Bergeron, C., Bennett, K.P.: Model selection for primal SVM. Machine Learning 85(1-2), 175–208 (October 2011)

[51] Moore, G.M., Bergeron, C., Bennett, K.P.: Nonsmooth bilevel programming for hyperparameter selection. In: IEEE International Conference on Data Mining Workshops. pp. 374–381 (2009)

[52] Niculescu-Mizil, A., Perlich, C., Swirszcz, G., Sindhwani, V., Liu, Y., Melville, P., Wang, D., Xiao, J., Hu, J., Singh, M., et al.: Winning the kdd cup orange challenge with ensemble selection. In: Proceedings of the 2009 International Conference on KDD-Cup 2009-Volume 7. pp. 23–34. JMLR. org (2009)

[53] Opper, M., Winther, O.: Gaussian processes and SVM: Mean field results and leave-one-out, pp. 43–65. MIT (10 2000), massachusetts Institute of Technology Press (MIT Press) Available on Google Books

[54] Park, M.Y., Hastie, T.: L1-regularization path algorithm for generalized linear models. Journal of the Royal Statistical Society: Series B (Statistical Methodology) 69(4), 659–677 (2007)

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

[56] Pham, H., Guan, M.Y., Zoph, B., Le, Q.V., Dean, J.: Efficient neural architecture search via parameter sharing. arXiv preprint arXiv:1802.03268 (2018)

[57] Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y.L., Le, Q., Kurakin, A.: Large-scale evolution of image classifiers. arXiv preprint arXiv:1703.01041 (2017)

[58] Ricci, F., Rokach, L., Shapira, B., Kantor, P.B. (eds.): Recommender Systems Handbook. Springer (2011)

[59] Schölkopf, B., Smola, A.J.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press (2001)

[60] Snoek, J., Larochelle, H., Adams, R.P.: Practical Bayesian optimization of machine learning algorithms. In: Advances in Neural Information Processing Systems 25, pp. 2951–2959 (2012)

[61] Statnikov, A., Wang, L., Aliferis, C.F.: A comprehensive comparison of random forests and support vector machines for microarray-based cancer classification. BMC Bioinformatics 9(1) (2008)

[62] Sun, Q., Pfahringer, B., Mayo, M.: Full model selection in the space of data mining operators. In: Genetic and Evolutionary Computation Conference. pp. 1503–1504 (2012)

[63] Swersky, K., Snoek, J., Adams, R.P.: Multi-task Bayesian optimization. In: Advances in Neural Information Processing Systems 26. pp. 2004–2012 (2013)

[64] Swersky, K., Snoek, J., Adams, R.P.: Freeze-thaw bayesian optimization. arXiv preprint arXiv:1406.3896 (2014)

[65] Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-weka: Automated selection and hyper-parameter optimization of classification algorithms. CoRR abs/1208.3719 (2012)

[66] Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In: 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 847–855. ACM (2013)

[67] Vanschoren, J., Van Rijn, J.N., Bischl, B., Torgo, L.: Openml: networked science in machine learning. ACM SIGKDD Explorations Newsletter 15(2), 49–60 (2014)

[68] Vapnik, V., Chapelle, O.: Bounds on error expectation for support vector machines. Neural computation 12(9), 2013–2036 (2000)

[69] Weston, J., Elisseeff, A., BakIr, G., Sinz, F.: Spider (2007), `http://mloss.org/software/view/29/`

[70] Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578 (2016)