Design of the 2015 ChaLearn AutoML Challenge

Isabelle Guyon, ChaLearn, USA, guyon@chalearn.org
Kristin Bennett, RPI, USA, bennek@rpi.edu
Gavin Cawley, UEA, UK, G.Cawley@uea.ac.uk
Hugo Jair Escalante, INAOE, Mexico, hugo.jair@gmail.com
Sergio Escalera, CVC, UAB & UB, Spain, sergio@maia.ub.es
Tin Kam Ho, IBM, USA, tho@us.ibm.com
Núria Macià, Andorra, macia.nuria@gmail.com
Bisakha Ray, NYU, USA, bisakha.ray@nyumc.org
Mehreen Saeed, FAST, Pakistan, mehreen.mehreen@gmail.com
Alexander Statnikov, NYU, USA, statnikov@gmail.com
Evelyne Viegas, Microsoft, USA, evelynev@microsoft.com

***Abstract*—ChaLearn is organizing the Automatic Machine Learning (AutoML) contest for IJCNN 2015, which challenges participants to solve classification and regression problems *without any human intervention*. Participants' code is automatically run on the contest servers to train and test learning machines. However, there is no obligation to submit code; half of the prizes can be won by submitting prediction results only. Datasets of progressively increasing difficulty are introduced throughout the six rounds of the challenge. (Participants can enter the competition in any round.) The rounds alternate phases in which learners are tested on datasets participants have not seen (AutoML), and phases in which participants have limited time to tweak their algorithms on those datasets to improve performance (Tweakathon). This challenge will push the state of the art in fully automatic machine learning on a wide range of real-world problems. The platform will remain available beyond the termination of the challenge.**

I. Introduction

An ever growing number of disciplines rely on machine learning (ML) techniques. However, the use of these techniques still requires human interaction; practitioners have to select appropriate features, workflows, algorithms, and hyper-parameters. To accelerate the applicability of ML, there is an increasing demand for easy-to-use, off-the-shelf methods that do not require ML expertise. We coin the term *AutoML* to denote the resulting research area which targets this progressive automation of machine learning.

AutoML refers to all aspects of automating the ML process beyond model selection, hyper-parameter optimization, and model search. The design of ML techniques and their user interface should include automatic (1) data loading and formatting (from raw data to miscellaneous formats), (2) detection and handling of skewed data and missing values, (3) selection of learning representation and feature extraction, (4) problem/algorithm match, (5) acquisition of new data (active learning), (6) creation of appropriately sized and stratified training, validation, and test sets, (7) selection of algorithms that satisfy resources constraints at training and run time, (8) ability to generate and reuse workflows, (9) meta-learning and learning transfer, and (10) explicative reports. Such automation is crucial for both robots and lifelong autonomous ML.

The purpose of this paper is to describe the design of the AutoML challenge[1]—part of the official IJCNN 2015 competition program. What is new in comparison to the previous competitions we have organized for IJCNN is code submission. Code submitted by participants is automatically executed on the open-source platform Codalab[2]. This ensures (1) no human intervention and (2) fair competition since all learning machines (also referred to as learners) are trained and tested on datasets unknown to participants using the same resources. However, there is no obligation to submit code; half of the prizes can be won by submitting prediction results alone. There are six rounds in which datasets of progressively increasing difficulty are introduced (see Fig. 1). The rounds alternate AutoML phases in which submitted code is tested, with a limited computational budget, on the Codalab platform with new datasets, and Tweakathon phases in which participants improve their methods by tweaking them on those same datasets. During Tweakathon phases participants are free to use their own computational resources.

This challenge advances the theoretical underpinnings of model selection by treating it as a joint problem of optimization and statistics. It provides a novel framework for numerical experimentation that reduces user intervention and allows practitioners to evaluate approaches on a large set of problems. The remainder of this paper details the protocol of the 2015 AutoML challenge and presents preliminary results.

II. Motivation and Scope

A. Motivation

There is a critical unmet need for data scientists who have the ability to transform raw data into knowledge. The data modeling process involves (1) formalizing a question into a modeling approach, (2) selecting appropriate data, (3) designing a model, (4) fitting the model to data (training), (5) making predictions on new data (testing), and (6) interpreting the results. Admittedly, completely automating the task of data scientists and replacing them with machines is a distant goal. One can envision that ultimately the data modeling process will be largely automated and simplified to a point that desktop data transformation tools will become as pervasive as text processing and spreadsheets. On the way to this grand goal, much can be done to accelerate the knowledge discovery process while improving its reliability.

Many ML packages currently provide highly optimized implementations of leading algorithms *for fixed hyper-parameters*

[1] http://codalab.org/AutoML

[2] codalab.org

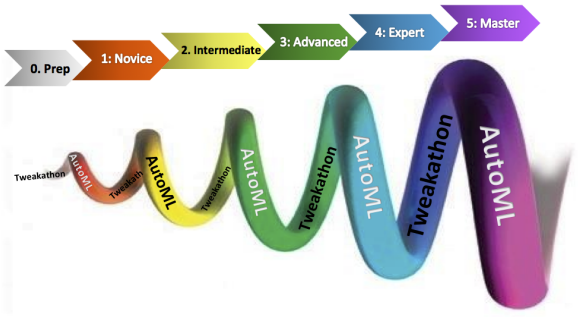


Fig. 1: **AutoML challenge.** The challenge includes six rounds numbered from 0 to 5. Except for rounds 0 (preparation) and 5 (termination), all rounds alternate AutoML and Tweakathon phases.

(*e.g.,* commercial platforms such as SAS, SPSS, AzureML, and the Google prediction API; freeware packages such as Weka, R, Lush, Scikit-learn, and Matlab libraries Spider and CLOP). In this context, the task of practitioners has shifted from implementing algorithms to learning how to use these packages—*i.e.,* how to create elaborate models from components and select the best hyper-parameters for every component. For example, to build a regression model one may choose among a set of preprocessing methods including variable scaling, apply a feature selection filter involving the choice of a threshold, and then train a support vector machine with the choice of kernel, loss, and regularization hyper-parameters.

Of all the steps in the data modeling process, model fitting can and should be completely automated. As it stands, this is not the case because model selection is the practical stumbling block that remains largely the user's responsibility. Even savvy users waste time reimplementing model selection techniques and non-savvy users make a variety of common mistakes resulting in suboptimal model choices. From our point of view, model selection is an integral part of model fitting and should be treated as an optimization and statistics problem—and not by applying haphazard heuristics. The goal of this competition is to design an automatic system to conduct rational selection of an optimal model (or ensemble of models) through (1) the search of associated hyper-parameters and (2) the assessment of performance—by making the best use of data, time, and computer resources available. Recent efforts based on old concepts [1]–[7] showcase that it is possible to develop software that can be used out of the box by ML novices [8]–[10]. Commercial enterprise platforms (such as SAP, Skytree, and RapidMiner) and open-source projects (such as H2O and E-Lico) also intend to reduce human intervention.

B. Scope

This challenge 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 are releasing 30 datasets[3] preformatted in given feature representations (*i.e.,* each example consists of a fixed number of numerical coefficients).

The distinction between input and output variables is not made in all 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 [11]. Another purpose may be to explain data in a simple, compact way and, eventually, infer latent variables (*e.g.,* class membership produced by a clustering algorithm). These tasks fall into the category of unsupervised learning [12]. We do not consider such cases and strictly limit the challenge to supervised learning where data present themselves as input-output pairs. Furthermore, the data pairs are identically and independently distributed. The models used are limited to fixed-length vectorial representations. Hence, we do not incorporate problems of time series prediction. Text, speech, and video processing tasks included in the challenge are not presented in their native data representations; datasets have been preprocessed in suitable fixed-length vectorial representations.

The difficulty of the challenge lies on 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 these problems, it still requires considerable human effort to find, for a given dataset, task, evaluation metric, and available computational time, the methods and hyper-parameter settings that maximize performance. The participant's challenge is to create the *perfect black box* that removes human interaction.

III. Brief Overview of Model Search

A. Full Model Selection

In what follows, we refer to participants' 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:

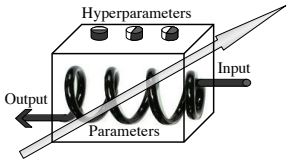
- a set of parameters $\boldsymbol{\alpha} = [\alpha_0, \alpha_1, \alpha_2, ..., \alpha_n]$;
- a learning algorithm (referred to as trainer), which serves to optimize the parameters using training data;
- a trained model (referred to as predictor) of the form $y = f(\mathbf{x})$ produced by the trainer;
- 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 variables corresponding to switching between alternative modules, 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 [13]. Some authors refer to this problem as *full model selection* [5], [14]. We will then denote the hyper-models as

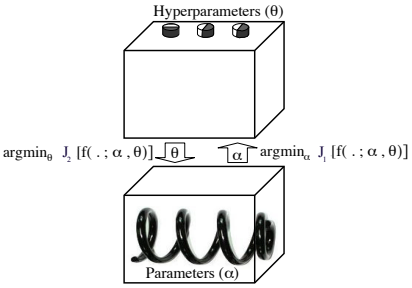
$$y = f(\mathbf{x}; \boldsymbol{\theta}) = f(\mathbf{x}; \boldsymbol{\alpha}(\boldsymbol{\theta}), \boldsymbol{\theta}), \quad (1)$$

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 $\{\mathbf{x}_i, y_i\}$. Participants have to devise algorithms

[3]The data will remain confidential until the challenge is over.



(a) **Learning machine**



(b) **Bi-level optimization**

Fig. 2: **Bi-level optimization.** (a) Representation of a learning machine with parameters and hyper-parameters to be adjusted. (b) Decoupling 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 $\boldsymbol{\alpha}$.

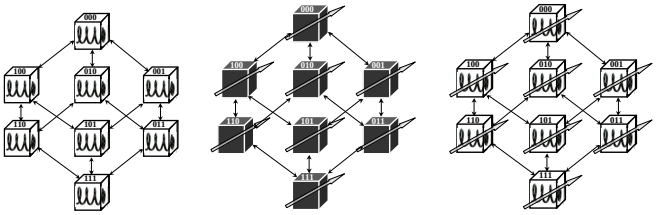
capable of training the hyper-parameters $\boldsymbol{\theta}$. This may require intelligent sampling of the hyper-parameter space and splitting the available training data into subsets for both training and evaluating the predictive power of solutions—one or multiple times.

As an optimization problem, model selection is a bi-level optimization program [15]–[17]; 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 Fig. 2). 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 in this challenge 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 challenge also lends itself to using ensemble methods, which let several "simple" models vote to make the final decision [4], [18], [19]. 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 [10].

B. Optimitzation of Hyper-parameters

Everyone who has modeled 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 [20], [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 surely has been much



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

Fig. 3: **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** uses knowledge of the learner structure and/or parameters to guide the hyper-parameter search.

progress on end-to-end automatic ML for more complex tasks such as text, image, video, and speech processing with deep-learning methods [22]. However, even these methods have many modeling choices and hyper-parameters.

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 data for small datasets or underfitting data 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. [23], [24] 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 the challenge is that code is tested with limited computational resources. That is, for each task an arbitrary limit on execution time is fixed. This places a constraint on the participant to produce a solution in a given time, hence to optimize the model search from a computational point of view. In summary, participants have to jointly address the problem of overfitting/underfitting and the problem of efficient search for an optimal solution, as stated in [25].

C. Strategies of Model Search

Most practitioners use heuristics such as grid search to sample $\boldsymbol{\theta}$ space, and use k-fold cross-validation as the upper-level objective J_2. In this framework, the optimization of $\boldsymbol{\theta}$ is not performed iteratively. All the parameters are sampled at regular intervals (usually using a linear or log scale). This leads to a combinatorial number of possibilities that exponentially increases with the dimension of $\boldsymbol{\theta}$. k-fold cross-validation consists of splitting the dataset into k parts; $(k-1)$ parts are

used for training and the remaining part is used for testing. The average of the test scores obtained on the k combinations is taken. There is a lack of principled guidelines to determine both the number of grid points and the value of k, and there is no guidance for regularizing J_2, yet this simple method is a good baseline approach. That is why some toolkits currently provide cross-validation modules.

Efforts have been made to optimize continuous hyper-parameters with bilevel optimization methods, using either the k-fold cross-validation estimator [17], [26] 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 [27]). The method was improved by adding a regularization of J_2 [28]. Gradient descent has been used to accelerate the search, by making a local quadratic approximation of J_2 [29]. Other approaches produce all values of $J_2(\boldsymbol{\theta})$ given only a few key examples [30], [31]. 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 [32]. [5] improves pattern search 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. Unfortunately, this approach retrieved the winning model in only $\sim 50\%$ 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 [23], [24], in particular by splitting data in a principled way [33], 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. Recent methods of multi-level Bayesian 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 [8], [9], [34], [35]. The central idea 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 could be adopted in the frequentist setting.

Note that splitting the problem of parameter fitting into two levels can be extended to multiple levels, at the expense of extra complexity—*i.e.,* need for a hierarchy of data splits to perform multiple or nested cross-validation [36], insufficient data to train and validate at the different levels, and increase of the computational load.

Borrowing from the conventional classification of feature selection methods [27], [37], [38], model search strategies can be categorized into filters, wrappers, and embedded methods (see Fig. 3). **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, [39] 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 learning machine 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.,* [27]). Other embedded methods jointly optimize parameters and hyper-parameters [26], [29], [40].

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. We expect this challenge offers an unbiased platform to compare and contrast methods addressing these problems.

IV. Challenge Design

A. Tasks

This challenge is concerned with regression and binary, multi-class, and multi-label classification problems from data formatted in fixed-length feature-vector representations. Each task is associated with a dataset coming from a real application. The domains of application are drawn from 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. All datasets present themselves in the form of data matrices with examples in rows and features in columns. The goal is to predict a target value. For instance, in a medical dataset, examples may represent patient records and features may represent results of laboratory analyses; the goal may be to predict the diagnosis of the patient (positive or negative). The identity and description of the datasets is concealed (except in round 0) to avoid the use of domain knowledge and to push participants to design fully automated ML solutions. In addition, the tasks are constrained by a time budget and a scoring metric.

B. Time Budget

The Codalab platform provides computational resources shared by all participants. To ensure fairness, when a code submission is evaluated, its execution time is limited to a given time budget, which varies from dataset to dataset. The time budget is provided with each dataset in its *info* file. The organizers reserve the right to adjust the time budget by supplying participants with updated *info* files. Participants who submit results—instead of code—are not constrained by the time budget since their code is run on their own platform. This may be advantageous for entries counting towards the Final phases (immediately following a Tweakathon). Participants wishing to also enter the AutoML phases, which require submitting code, can submit both results and code (simultaneously). The results do not need to be produced by the submitted code; if a participant does not want to share personal code, he/she can submit the sample code provided by the organizers together with his/her results.

C. Scoring Metrics

The score is 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, or a vector of binary indicators $[y_{il}]$ in $\{0, 1\}$, for multi-class or multi-label classification problems (one per class l). Participants must submit prediction values matching as closely as possible the target value, in the form of a continuous numeric coefficient q_i for regression problem 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 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. The score metrics are described 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

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

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

$$ABS = 1 - MAE/MAD, \tag{3}$$

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

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. 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.

$$|BAC| = (BAC - R)/(1 - R), \tag{4}$$

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

$$|AUC| = 2AUC - 1. \tag{5}$$

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

$$F1 = 2 * (precision * recall)/(precision + recall), \tag{6}$$

$$precision = true\ positive/(true\ positive + false\ positive) \tag{7}$$

$$recall = true\ positive/(true\ positive + false\ negative) \tag{8}$$

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

$$|F1| = (F1 - R)/(1 - R), \tag{9}$$

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

$$PAC = \exp(-CE), \tag{10}$$

$$CE = \begin{cases} \langle \sum_l \log(q_{il}) \rangle, & \text{for multi-class} \\ -\langle y_i \log(q_i) & \\ +(1 - y_i) \log(1 - q_i) \rangle, & \text{for binary and multi-label} \end{cases} \tag{11}$$

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

$$|PAC| = (PAC - R)/(1 - R), \tag{12}$$

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 R2, 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.

In all formulas the notation $\langle \cdot \rangle$ designates the average over all samples P indexed by i. That is,

$$\langle y_i \rangle = (1/P) \sum_i (y_i). \tag{13}$$

Only R2 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.

D. Leaderboard Score Calculation

Participants (or their submitted code) provide prediction results for the withheld target values for all datasets in each round. The scoring program supplied by the organizers is run on the server to compute the scores. For each dataset, participants are ranked in decreasing order of performance—based on the scoring metric associated with the given task. The overall score is computed by averaging the rank obtained on every dataset and shown on the leaderboard in column $\langle rank \rangle$.

The results of the last submission are used to compute the leaderboard results. Therefore, participants must resubmit older entries if they want them to be counted as final. In phases marked with a [+], participants with the three smallest $\langle rank \rangle$ are eligible for prizes (subject to compliance with the Terms and Conditions).

E. Learning Curves

Participants have to regularly test their systems while training to produce intermediate prediction results. This allows the organizers to make learning curves (performance as a function of training time) used to adjust the time budget in subsequent rounds—and eventually give more computational time.

F. Estimation of Performance on Future Data

In the Performance Prediction Challenge [41] participants had to both design models that generalized well on future data and predict their performance. This is a fundamental aspect of the AutoML field. However, the challenge does not address this issue to limit the complexity of design and evaluation.

Along with the learning curves as a function of time spent searching for the best hyper-model, we could draw learning curves as a function of the number of training examples. Such curves are useful in evaluating whether having more training examples significantly improves performance. This may be part of the post-challenge analysis.

G. Rounds and Phases

The challenge is run in multiple phases grouped in six rounds. Round 0 (Preparation) is a practice round with publicly available datasets which is followed by five rounds of progressive difficulty (Novice, Intermediate, Advanced, Expert, and Master). Except for rounds 0 and 5, all rounds include three phases that alternate AutoML and Tweakathons contests. These phases are described in Table I.

Submissions are made in Tweakathon phases only. The results of the latest submission are shown on the leaderboard and such submission automatically migrates to the next phase. Submitting code makes it possible to participate in subsequent phases without new submissions. However, participating in previous rounds is not a prerequisite for entering new rounds. Prizes are awarded in phases marked with a [+] during which there is no submission.

H. Code vs. Result Submission

To participate in phase AutoML[n], code must be submitted in Tweakathon[n-1]. To participate in the Final[n], code or results must be submitted in Tweakathon[n]. If both code and (well-formatted) results are submitted, the results are used for scoring rather than rerunning the code in Tweakathon[n] and Final[n]. The code is executed when results are unavailable or not well formatted. Thus, there is no disadvantage in submitting both results and code. When submitting results and code, different methods can be used to enter the Tweakathon/Final phases and the AutoML phases. Submissions are made only during Tweakathon with a maximum of five submissions per day. Immediate feedback is provided on the leaderboard on validation data. Participants rank on the basis of test performance during the Final and AutoML phases.

V. Data

Every round consists of five datasets spanning a range of difficulties. Data difficulty progressively increases from round to round. Datasets may be recycled, but are reformatted into new representations—except for the final round, which includes completely new data.

The datasets used in the challenge present the following range of difficulty, which requires demanding hyper-parameter choices:

Data distributions. Different intrinsic/geometrical complexity.

Tasks. Regression, binary classification, multi-class classification, and multi-label classification.

Scoring metrics. See Section IV-C.

Class imbalance. Balanced vs. unbalanced class proportions.

Sparsity. Full vs. sparse matrices.

Missing values. Presence vs. absence of missing values.

Categorical variables. Presence vs. absence of categorical features.

Irrelevant variables. Presence vs. absence of additional irrelevant data.

Number of training examples (P_{tr}). Small vs. large number of training examples.

Number of features (N). Small vs. large number of features.

Aspect ratio of the training data matrix (P_{tr}/N). $P_{tr} \gg N, P_{tr} = N$, or $P_{tr} \ll N$.

Round 0 uses five datasets—listed below—from previous challenges to define tasks illustrating a few of the aforementioned difficulties. Table II summarizes the statistical description of the datasets.

adult. Dense data; categorical features; multi-label classification. This is a rehash of the Adult 1994 census dataset from the UCI repository [42] donated by Kohavi and Becker.

cadata. Regression data. LibSVM dataset originally from Statlib house prices by Pace and Barry (1997).

digits. Semi-dense data; multi-class classification. Based on the MNIST dataset from LeCun, Cortes, and Burges (1998) of handwritten digits, extracted from a larger benchmark collected by the US National Institute of Standards and Technologies.

dorothea. Sparse binary data; binary classification. Data prepared for the NIPS 2003 feature selection challenge from one

TABLE I: **Phases of round n.** 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	1 month	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

TABLE III: Preparation phase results on validation data. Rank is shown in parentheses for each dataset. The winner is determined by the average rank.

Participant	Rank	DS 1	DS 2	DS 3	DS 4	DS 5
ideal.intel	1.2	0.82 (2)	0.81 (1)	0.96 (1)	0.90 (1)	0.60 (1)
abhishek	3.2	0.82 (4)	0.79 (4)	0.94 (3)	0.85 (3)	0.45 (2)
aad.freiburg	3.4	0.82 (3)	0.80 (2)	0.94 (4)	0.80 (5)	0.42 (3)
reference	7.0	0.81 (8)	0.78 (5)	0.81 (8)	0.70 (8)	0.35 (6)

of the Knowledge Discovery in Data Mining Cup 2001 tasks. DuPont Pharmaceuticals graciously provided the original data.

newsgroups. Sparse data; multi-class classification. Twenty datasets from Lang and Mitchell (1997).

All datasets have been pre-formatted in a fixed-length feature-based representation. In the following rounds described below, the number of variables and samples vary between thousands and millions.

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 5,000 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.

VI. Baseline Software and Preliminary Results

We provided baseline software using the ML library Scikit-learn [43]. 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 are used.

As of February 1, over 180 people registered and downloaded data, 20 teams are actively participating, and the top ranking participants in round 0 already significantly outperform the reference method on several datasets (see Table III).

VII. Discussion

Based on results of past challenges, we designed a competition where the strategy is to (1) reduce the search space with filter methods, (2) reduce the number of hyper-parameters using versions of the algorithms that optimize them with embedded methods, and (3) use an ensemble method to grow an ever improving ensemble until the computational budget is exhausted.

A few participants asked for more computational resources. In round 0 we provided one hour of computing per submission on an 8-core machine; we will progressively ramp up this time up to 10 hours throughout the challenge. Future editions might run on Hadoop to allow participants to process larger datasets and face the Big Data era.

In the post-challenge analysis, we plan to systematically test the winning entries on these datasets semi-synthetically modified to cover a wider range of problem complexity (*e.g.,* varying the proportion of training examples, missing data, and distractor variables). We intend to relate the results to a set of data complexity measures proposed in [44].

Acknowledgments

Microsoft supported the organization of this challenge and donated the prizes. 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. Computing resources were provided generously by J. Buhmann, ETH Zürich. This work also used computing resources at the High Performance Computing Facility of the Center for Health Informatics and Bioinformatics at the NYU Langone Medical Center. The datasets released were selected among 72 datasets that were donated (or formatted using data publicly available) 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: M. Boullé, C. Capponi, R. Caruana, G. Dror, C. Germain, B. Kégl, H. Larochelle, V. Lemaire, C.-J. Lin, V. Ponce López, S. Mercer, F. Popescu, M. Sebag, D. Silver, and I. Tsamardinos. The software developers who contributed to the implementation of the Codalab platform and the sample code include E. Camichael, I. Judson, C. Poulain, P. Liang, A. Pesah, L. Romaszko, X. Baro Solé, E. Watson, and M. Zyskowski. The paper was proofread by Nicola Talbot.

References

[1] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *15th International Conference on Artificial Intelligence*, 1995, pp. 1137–1143.

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

[3] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, March 2003.

[4] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes, "Ensemble selection from libraries of models," in *21st International Conference on Machine Learning*. ACM, 2004, pp. 18–.

[5] H. J. Escalante, M. Montes, and L. E. Sucar, "Particle swarm model selection," *Journal of Machine Learning Research*, vol. 10, pp. 405–440, 2009.

TABLE II: **Data for phase 0.** Legend: Name = dataset name; Task = corresponding task; Metric = scoring function; Cnum = number of classes; Cbal = entropy of class distribution; Sparse = sparsity; Missing = fraction of missing values; Catvar = 1 if has categorical variables; Irrvar = fraction of irrelevant variables; Pte = number of test examples; Pva = number of validation examples; Ptr = number of training examples; N = number of features; Ptr/N = aspect ratio of the data.

Name	Task	Metric	Time	Cnum	Cbal	Sparse	Missing	Catvar	Irrvar	Pte	Pva	Ptr	N	Ptr/N
adult	multi-label	F1	300	3	1	0.16	0.011	1	0.5	9,768	4,884	34,190	24	1424.58
cadata	regression	R2	200	0	NaN	0	0	0	0.5	10,640	5,000	5,000	16	312.5
digits	multi-class	BAC	300	10	1	0.42	0	0	0.5	35,000	20,000	15,000	1,568	9.57
dorothea	binary	AUC	100	2	0.46	0.99	0	0	0.5	800	350	800	100,000	0.01
newsgroups	multi-class	PAC	300	20	1	1	0	0	0	3,755	1,877	13,142	61,188	0.21

[6] I. Guyon, A. Saffari, G. Dror, and G. Cawley, "Model selection: Beyond the Bayesian/frequentist divide," *Journal of Machine Learning Research*, vol. 11, pp. 61–87, January 2010.

[7] T. Schaul, S. Zhang, and Y. LeCun, "No more pesky learning rates," in *30th International Conference on Machine Learning*, 2013.

[8] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems 25*, 2012, pp. 2951–2959.

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

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

[11] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, Eds., *Recommender Systems Handbook*. Springer, 2011.

[12] Z. Ghahramani, "Unsupervised learning," in *Advanced Lectures on Machine Learning*, ser. Lecture Notes in Computer Science, vol. 3176. Springer Berlin Heidelberg, 2004, pp. 72–112.

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

[14] Q. Sun, B. Pfahringer, and M. Mayo, "Full model selection in the space of data mining operators," in *Genetic and Evolutionary Computation Conference*, 2012, pp. 1503–1504.

[15] B. Colson, P. Marcotte, and G. Savard, "An overview of bilevel programming," *Annals of Operations Research*, vol. 153, pp. 235–256, 2007.

[16] S. Dempe, *Foundations of bilevel programming*. Kluwer Academic Publishers, 2002.

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

[18] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[19] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.

[20] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: Data mining, inference, and prediction*, 2nd ed. Springer, 2001.

[21] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed. Wiley, 2001.

[22] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[23] J. P. A. Ioannidis, "Why most published research findings are false," *PLoS Medicine*, vol. 2, no. 8, p. e124, August 2005.

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

[25] M. I. Jordan, "On statistics, computation and scalability," *Bernoulli*, vol. 19, no. 4, pp. 1378–1390, September 2013.

[26] G. Moore, C. Bergeron, and K. P. Bennett, "Model selection for primal SVM," *Machine Learning*, vol. 85, no. 1-2, October 2011.

[27] I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh, Eds., *Feature extraction, foundations and applications*, ser. Studies in Fuzziness and Soft Computing. Physica-Verlag, Springer, 2006.

[28] G. C. Cawley and N. L. C. Talbot, "Preventing over-fitting during model selection via Bayesian regularisation of the hyper-parameters," *Journal of Machine Learning Research*, vol. 8, pp. 841–861, April 2007.

[29] S. S. Keerthi, V. Sindhwani, and O. Chapelle, "An efficient method for gradient-based adaptation of hyperparameters in SVM models," in *Advances in Neural Information Processing Systems*, 2007.

[30] T. Hastie, S. Rosset, R. Tibshirani, and J. Zhu, "The entire regularization path for the support vector machine," *Journal of Machine Learning Research*, vol. 5, pp. 1391–1415, 2004.

[31] M. Y. Park and T. Hastie, "L1-regularization path algorithm for generalized linear models," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 69, no. 4, pp. 659–677, 2007.

[32] M. Momma and K. P. Bennett, "A pattern search method for model selection of support vector regression," in *In Proceedings of the SIAM International Conference on Data Mining*. SIAM, 2002.

[33] A. Statnikov, L. Wang, and C. F. Aliferis, "A comprehensive comparison of random forests and support vector machines for microarray-based cancer classification," *BMC Bioinformatics*, vol. 9, no. 1, 2008.

[34] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems*, 2011, pp. 2546–2554.

[35] K. Swersky, J. Snoek, and R. P. Adams, "Multi-task Bayesian optimization," in *Advances in Neural Information Processing Systems 26*, 2013, pp. 2004–2012.

[36] B. Efron, "Estimating the error rate of a prediction rule: Improvement on cross-validation," *Journal of the American Statistical Association*, vol. 78, no. 382, pp. 316–331, 1983.

[37] R. Kohavi and G. H. John, "Wrappers for feature selection," *Artificial Intelligence*, vol. 97, no. 1-2, pp. 273–324, December 1997.

[38] A. L. Blum and P. Langley, "Selection of relevant features and examples in machine learning," *Artificial Intelligence*, vol. 97, no. 1-2, pp. 273–324, December 1997.

[39] R. Bardenet, M. Brendel, B. Kégl, and M. Sebag, "Collaborative hyperparameter tuning," in *30th International Conference on Machine Learning*, vol. 28. JMLR Workshop and Conference Proceedings, May 2013, pp. 199–207.

[40] G. M. Moore, C. Bergeron, and K. P. Bennett, "Nonsmooth bilevel programming for hyperparameter selection," in *IEEE International Conference on Data Mining Workshops*, 2009, pp. 374–381.

[41] *Performance Prediction Challenge*, 2006.

[42] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

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

[44] T. K. Ho and M. Basu, "Complexity measures of supervised classification problems," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 3, pp. 289–300, March 2002.