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 for IJCNN 2015 an Automatic Machine Learning challenge (AutoML) to solve classification and regression problems from given feature representations, *without any human intervention*. This is a challenge with code submission: the code submitted can be executed automatically on the challenge servers to train and test learning machines on new datasets. However, there is no obligation to submit code. Half of the prizes can be won by just submitting prediction results. There are six rounds (Prep, Novice, Intermediate, Advanced, Expert, and Master) in which datasets of progressive difficulty are introduced (5 per round). There is no requirement to participate in previous rounds to enter a new round. The rounds alternate AutoML phases in which submitted code is "blind tested" on datasets the participants have never seen before, and Tweakathon phases giving time ($\simeq$ 1 month) to the participants to improve their methods by tweaking their code on those datasets. This challenge will push the state-of-the-art in fully automatic machine learning on a wide range of problems taken from real world applications. The platform will remain available beyond the termination of the challenge: http://codalab.org/AutoML.**

I. Introduction

Machine learning has achieved considerable success and an ever growing number of disciplines rely on it. However, these successes crucially rely on human machine learning experts, who select appropriate features, workflows, machine learning paradigms, algorithms, and hyper-parameters. To accelerate the applicability of machine learning in an ever growing number of domains, there is an increasing demand for off-the-shelf methods that can be used easily in the field, without expert machine learning knowledge. We call the resulting research area that targets progressive automation of machine learning: AutoML.

AutoML refers to all aspects of automating the machine learning process, beyond model selection, hyper-parameter optimization, and model search, including:

- Robot autonomous learning.
- Representation learning and automatic feature extraction/construction.
- Automatic generation and reuse of workflows.
- Meta learning and transfer learning.
- Automatic problem "ingestion" (from raw data and miscellaneous formats).
- Automatically detecting and handling skewed data, missing values, leakage detection and prevention.
- Matching problems to methods/algorithms.
- Automatic acquisition of new data (active learning, experimental design).
- Automatic report writing.
- User interfaces for AutoML.
- Automatic creation of appropriately sized and stratified train, validation, and test sets.
- Automatic selection of algorithms to satisfy time / space / power constraints at traintime or at runtime.
- Life long autonomous machine learning.

This year, we are organizing an AutoML competition[1], which covers specific aspects of the problem. The results of the AutoML competition, which is part of the official IJCNN 2015 competition program, will be discussed in an IJCNN 2015 workshop. The scope of the workshop is broader than that of the challenge; we welcome contributions on all aspects of the problem to stimulate creativity and prepare for new editions of AutoML competitions.

One novel design element compared to previous competitions we organized for IJCNN is "code submission". Code submitted by the participants is executed automatically on Codalab, an open-source platform hosting the competition[2]. This allows us to train and test learning machines with datasets unknown to the participants in identical conditions for all submissions, and to ensure that there is strictly no human intervention. However, there is **no obligation to submit code**. Half of the prizes can be won by just submitting prediction results. There are six rounds (Prep, Novice, Intermediate, Advanced, Expert, and Master) in which datasets of progressive difficulty are introduced, five per round (Figure 1). There is **no prerequisite to participate in previous rounds** to enter a new round. The rounds alternate AutoML phases in which submitted code is "blind tested" in limited time on the Codalab platform with new datasets, and Tweakathon phases giving time to the participants to improve your methods by tweaking them on those same datasets. During Tweakathon, the participants are free to use their own computational resources.

This challenge will accomplish multiple goals: (1) Advancing the theoretical underpinnings of model selection by treating it as a joint problem of optimization and statistics. (2) Designing and implementing a novel software framework for numerical experiment planning, eliminating user interventions. (3) Evaluating the approaches on a wide variety of problems

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

and comparing performance with "human" model selection (Tweakathon phases). (4) Disseminating results by making the code of the top ranking participants open source, publishing papers, organizing a workshop and editing proceedings. This paper summarizes the protocol of the 2015 AutoML challenge and presents preliminary results, which will be updated at revision time (if the paper is accepted).

II. Motivation and scope

A. Motivations

The "big data" era is full of promises and frustrations. A few industries have already harvested some of the big data potential seemingly with ease. Recently, Google's Research Director Peter Norvig claimed, "We don't have better algorithms. We just have more data." to explain the success of several Google services relying on machine learning. In apparent contradiction with this perceived simplicity of the problem, we read in the press that there is presently a shortage of data scientists to analyze big data [1]. Clearly, there is a critical unmet need for data scientists who have the ability to transform raw data into useful information. The data modeling chain 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 (5) interpreting the results. Admittedly, completely automating the task of data scientists and replacing them by machines is a distant goal. One can envision that ultimately the data modeling chain will be largely automated and simplified to a point that desktop data transformation tools will become as pervasive as text processing and spreadsheets; Like driving a car, the user will focus on the route to the destination without worrying about how the engine works. On the way to this grand goal, much can be done to accelerate the knowledge discovery process while improving its reliability.

Available today to data scientists are many machine learning or data mining packages providing highly optimized implementations of leading algorithms *for fixed hyper-parameters*, including commercial platforms like SAS [2], SPSS [3], AzureML, and the Google prediction API, and freeware packages like Weka [4], R [5], Lush [6] and several Matlab libraries like the Spider [7] and CLOP [8]. The newcomer Scikit-learn [9] implemeted in Python has gained rapid popularity and it is the basis of our competition starting kit. The MLOSS open source repository [10] indexes such valuable resources. In this context, the task of practitioners has shifted from that of identifying and implementing algorithms to that of learning how to use such packages, creating elaborate models from components, and selecting the best hyper-parameters for every component. For example, to create a regression model one may choose among various preprocessing methods including variable scaling, apply one or the other feature selection filter involving the choice of a threshold, and then train a kernel support vector regression model with a choice of kernel, loss, and regularization hyper-parameters.

The angle we take is that, of all the steps in the data modeling chain, **model fitting can and should be completely automated**. As it stands, this is not the case because model selection is the practical stumbling block in the model creation process that remains largely the user's responsibility. Savvy users waste time re-implementing model selection techniques and non-savvy users make a variety of common mistakes resulting in sub-optimal model choices. From our point of view, model selection is part of model fitting and should be treated with rigor as an **optimization and statistics problem**, not by applying haphazard heuristics. The charter of this competition is to design an automatic system to conduct rational selection of an optimal model or ensemble of models through a search of associated hyper-parameters and the assessment of their performance on future data, by making the best possible use of data, time, and computer resources available. Recent efforts building on top of old concepts [11]–[17] showcase what is possible to create software that can be used out-of-the-box by ML novices [18]–[20]. Commercial enterprise platforms (such as SAP, Skytree, and RapidMiner) and everal open-source efforts (such as H2O and E-Lico) are also progressing is the direction of accelerating model production by reducing human intervention.

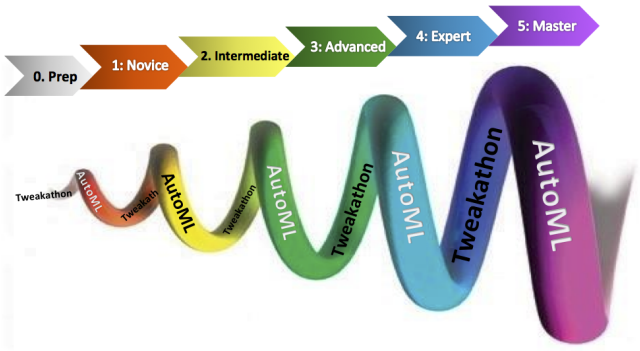


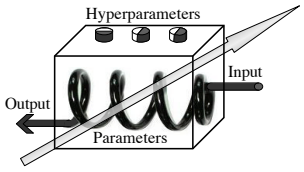
Fig. 1: **AutoML challenge organization.** The challenge include 6 rounds numbered 0 to 6. Except for rounds 0 (preparation) and 5 (termination), all rounds alternate AutoML and Tweakathon phases.

B. Scope

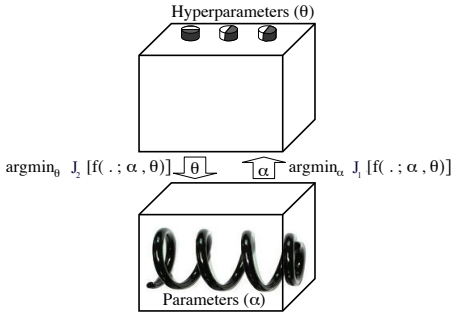
This is a "supervised learning" challenge in machine learning. We are releasing 30 datasets[3], pre-formatted in given feature representations (this means that each example consists of a fixed number of numerical coefficients). The challenge is to solve classification and regression problems, without any further human intervention, within the constraint of a certain time budget.

The division between input and output variable is not made in all machine learning applications. For example, 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 [21]. Another objective of modeling may be to explain data in a simple and compact way, eventually introducing new "latent" variables (for example, the class membership produced by a clustering algorithm). A wide range of "unsupervised learning" algorithms fall into that category [22]. However, in this challenge we do not consider such cases and strictly limit ourselves to the **supervised learning** setting in which data present themselves as input-output pairs $\{\mathbf{x}, y\}$. Furthermore, the data pairs are indentically and independently distributed (i.i.d.). In addition,

[3]For obvious reasons, the identity and nature of the data will remain confidential until the challenge is over



(a) **A learning machine**



(b) **Bi-level optimization**

Fig. 2: **Bi-level optimization.** (a) A pictorial representation of a learning machines 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 $\boldsymbol{\alpha}$.

the models we consider in this challenge are limited to **fixed-length vectorial representations**. Hence, we do not consider problems of time series prediction. Text processing, speech processing, and video processing tasks included in the challenge are not presented in their native data representations: the data have been preprocesssed in suitable fixed-length vectorial representations by the organizers.

The difficulty of the challenge lies in the diversity of data types and distributions (including balanced or unbalanced classes, sparse or dense feature representations, with or without missing values or categorical variables, various metrics of evaluation, various proportions of number of features and number of examples). The problems are drawn from a wide variety of domains, which include medical diagnosis from laboratory analyses, speech recognition, credit rating, prediction or drug toxicity or efficacy, classification of text, prediction of customer satisfaction, object recognition, protein structure prediction, action recognition in video data, etc. While there exist machine learning toolkits including methods that can solve all these problems, it is still considerable human effort to find, for a given combination of dataset, task, metric of evaluation, and available computational time, the combination of methods and hyper-parameter setting that is best suited. The participant's challenge is to create the "perfect black box" eliminating the human in the loop.

III. Brief overview of existing approaches

A. What is AutoML?

For a given class of model, best practices for model selection have emerged over the years, grounded in a wide variety of theories (regularization, Bayesian priors, Minimum Description Length (MDL), Structural Risk Minimization (SRM), bias/variance tradeoff, etc. [23]–[27]. Interestingly, all those theories converge towards the same principle stated already in the 14^{th} century by William of Ockham: "Pluralitas non est ponenda sine neccesitate", which prescribes limiting model complexity to the minimum necessary to explain the data, or *shave off unnecessary parameters* (Ockham's razor). This principle has been widely applied to parameter learning within a particular class of functions. However, the optimization of "hyper-parameters" with respect to model architecture, choices of preprocessing, feature selection, or choice of learning algorithms, is largely performed ignoring such theories and principles, and relying on optimizing simply an empirical statistic such as the cross-validation (CV) error using simple algorithms like grid search that are not practical for many hyper-parameters. An effective model selection strategy involves an effective plan to estimate generalization error by data sampling, to search or optimize the hyper-parameter space within the constraint of a certain time budget. Poor planning lacking consideration for overfitting, multiple-testing, data sampling and hyper-parameter optimization methods can lead to models with poor generalization.

In what follows, we refer to the solutions of challenge participants as "hyper-models" to indicate that they are elaborated from simpler components, which may include "models" already available in machine learning tookits. For example, for classification problems, the participants might want to consider a hyper-model made of alternative classification techniques such as nearest neighbors, linear models, kernel methods, neural networks, and random forests. More complex hyper-models may also include chains of alternative preprocessing, feature construction, feature selection, and classification 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, which we refer to as a "trainer", which serves to optimize the parameters using some training data (input-output pairs);
- a trained model called "predictor" of the form $y = f(\mathbf{x})$, produced by the trainer;
- a clear objective function, $J(f)$, which we can use to assess the model performance on new data not used for training (test data).

Consider now model hypothesis space parameterized by a vector $\boldsymbol{\theta} = [\theta_1, \theta_2, ..., \theta_n]$ of hyper-parameters. The hyper-parameter vector may include not only indicator variables corresponding to switching between alternative modules, but also modeling choices such as preprocessing parameters, type of kernel in a kernel method, the number of units and layers in a neural network, or training algorithm regularization parameters [28]. Some authors refer to this problem as "full model selection" [29], [30]. We will denote hyper-models as:

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

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 some training data D_{tr} including n_{tr} examples of input/output pairs $\{\mathbf{x}_k, y_k\}$. The challenge participants will need to devise algorithms capable of training the hyper-parameters $\boldsymbol{\theta}$. This may require intelligently sampling hyper-parameter space and splitting available training data into subsets to train and to evaluate the predictive power of candidate solutions, one or multiple times (cross-validation).

As an optimization problem, model selection is a "bi-level optimization program" [31]–[36]: 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 (Figure 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 capacity or complexity of the hyper-model $C_2(\boldsymbol{\theta})$. Statistical learning theory has taught us how to avoid overfitting with methods such as Structural Risk Minimization (SRM) [27]. One central question of this challenge will be to devise means to avoid over-fitting 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 [14], [37], [38]. In this case, the parameters $\boldsymbol{\theta}$ may be interpreted as voting weights. Although, for simplicity, we lump all parameters in a single vector, several authors have proposed more elaborate structures of hyper-parameter space involving trees or graphs [20].

B. Why is AutoML so hard?

If the model selection and hyper-parameter tuning problems have been around and understood for so long, why are they not solved already? Everyone who has modeled data has had to face some modeling choices: scaling, normalization, missing value imputation, variable coding (for categorical variables), variable discretization, degree of nonlinearity, model architecture, etc. Machine learning has considerably progressed in the direction of eliminating as many hyper-parameters as possible and producing the "perfect black-box" to perform tasks such as classification and regression [26], [39]. Still, any real world problem requires at least preparing data before it can be fitted into an "automatic" method, hence requiring some modeling choices. There surely has been much progress also made into the direction of having end-to-end automatic machine learning of more complex tasks such as text, image, video, and speech processing with deep-learning methods [40]. However, even these methods have still many modeling choices and hyper-parameters.

Although all formatted in a similar way (in fixed length feature representations), the datasets of the challenge present a range 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 IV-C).
- **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.
- **Aspect ratio P_{tr}/N of the training data matrix:** $P_{tr} \gg N, P_{tr} = N$ or $P_{tr} \ll N$.

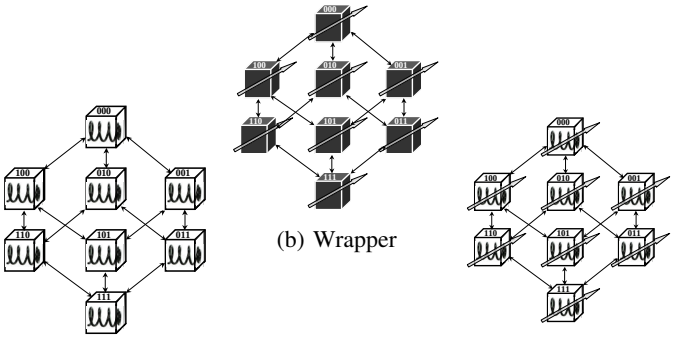
Hence this will necessarily require many modeling/hyper-parameter choices.

While producing models for a very rich and diverse range of applications has been a focus of data science research, relatively very little effort has been devoted to the optimization of hyper-parameters. Common practices including "trial and error" and "grid search" may lead to overfitting data for small datasets or underfitting data for large datasets, if conducted without care. By "overfitting" we mean producing models that perform very well on data used for training but perform poorly on new data (test data), *i.e.,* a model that does not "generalize". By "underfitting" we mean selecting too simple a model, which does not capture the complexity of the data, hence perform poorly both on training and test data. While there exist off-the-shelf well-optimized learning algorithms for optimizing parameters, end-users are still responsible for organising their numerical experiments to identify the best of a number of models under consideration and, due to lack of time and resources, they often perform model/hyper-parameter selection with ad hoc techniques. Many published papers include fundamental mistakes, which may invalidate an entire study [41], [42], including poor construction of training/testing splits, inappropriate model complexity, cheating by hyper-parameter selection using the test set, poor use of computational resources, and inappropriate test metrics. The challenge participants must avoid these flaws and devise systems, which can be blind tested on new data.

An additional twist of the challenge is that, in AutoML phases in which code is blind tested on the platform on new data, the computational resources are limited. For each task and arbitrary limit on execution time is fixed. This places on the participant the constraint to produce a solution in a given limited time, hence to optimize the model search from a computational point of view. In summary, the participants will have to address jointly the problem of overfitting/under-fitting, **a statistics problem** and the problem of efficient search for an optimal solution, **a computational problem**, as recently stated [43].

C. Strategies of model search

Most practitioners use simple 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, optimization of $\boldsymbol{\theta}$ is not performed iteratively. In grid search, all the parameters are sampled at regular intervals (usually using a linear or a log scale), leading to a combinatorial number of possibilities increasing exponentially with the dimension of $\boldsymbol{\theta}$. K-fold cross-validation consists in splitting the training data in K parts, training on $(K-1)$ parts and testing on the remaining part. An average is taken over all the choices of the left out part. There is a lack of principled guidelines to determine the number of grid points and value of K and no "prescription" for regularizing J_2. Still, this



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

Fig. 3: **Principal approaches to two-level inference.** (a) **Filter methods** select the hyper-parameters (with or without using training data), without first adjusting the learning machine parameters (the absence of an arrow means no training of the parameters). (b) **Wrapper methods** select the hyper-parameters using trained learning machines, treating them as black-boxes. (c) **Embedded methods** proceed like wrappers, but use knowledge of the learning machine structure, parameters and/or learning algorithm to guide the hyper-parameter search.

simple method is a good baseline approach, which is hard to beat [44]. Some toolkits provide already programmed cross-validation modules (see for instance the well documented Scikit-learn Python library for cross-validation http://scikit-learn.org/stable/modules/cross_validation.html).

Several authors have proposed optimizing continuous hyper-parameters with recent bilevel optimization methods, using as the upper-level objective J_2 either the K-fold cross-validation estimator [34]–[36], [45] or the leave-one-out estimator. For instance, in [46] the authors used the Nelder-Mead simplex. The leave-one-out estimator computed efficiently in a closed form as a by-product of training only one predictor on all the training examples (virtual-leave-one-out, see *e.g.,* [47]). The authors perfected this method by adding a regularization of J_2 in their subsequent work [48]. To accelerate search compared to the simplex method, some authors have been using gradient descent, making a local quadratic approximation of J_2 [49]. Other authors have devised methods to produce all values of $J_2(\boldsymbol{\theta})$ given only a few key samples [50], [51]. All these methods are of great practical importance, but they are limited to specific models and continuous hyper-parameters; they do not address the problem of full model selection.

The problem of full model selection recently started attracting the attention of the research community. One early proposed algorithm is an optimization method called "pattern search" using K-fold cross-validation for J_2. It explores hyper-parameter space by steps of the same magnitude, and when no change in any one parameter further decreases J_2, the step size is halved and the process repeated until the steps are deemed sufficiently small [52]. In [29], the authors improve on pattern search using Particle Swarm Optimization (PSO), which optimizes a problem by having a population of candidate solutions, here dubbed particles, and moving these particles around in hyper-parameter space using the particle's position and velocity. K-fold cross-validation is also used for J_2. Using challenges and known benchmarks they apply their method to search for the winning model in a space of hyper-models containing it. They succeed in getting good performance on every task, but retrieve the winning model (or a model performing similarly) in only about 50% of the cases. Overfitting is controlled heuristically with early stopping and the proportion of training and validation data is not optimized. Although several authors have increased awareness of good practice of numerical experimental design to reduce the risk of overfitting [41], [42], in particular by splitting data in a principled way [53], to our knowledge, no one has addressed the problem of optimally splitting data.

While regularizing the second level of inference is a relatively recent addition in the frequentist machine learning community, it has always been a natural and intrinsic part of Bayesian modeling via the notion of hyper-prior. Recent efficient methods of multi-level Bayesian optimization combining importance sampling and Monte-Carlo Markov Chains (MCMC) have been proposed [12]. The field of Bayesian hyper-parameter optimization has rapidly developed and yielded very promising results, particular by making use of Gaussian processes to model generalization performance [18], [19], [54]–[57]. 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 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 maybe be translated to the frequentist setting.

Although splitting the problem of parameter fitting in two levels is common practice, the idea can be extended to splitting the problem in multiple levels, at the expense of extra complexity. This may require creating a hierarchy of data splits to perform multiple or nested cross-validations [58]. This may yield to insufficient data to train or validate at the various levels. Additionally, this increases to computational load.

We can put a unified framework around the various approaches. Borrowing from the conventional classification of feature selection methods [59]–[61], model search strategies can be categorized into filters, wrappers, and embedded methods (Figure 3). **Filters** are methods for narrowing down the model space, without training the learning machine. Such methods include preprocessing, feature construction, kernel design, architecture design, choice of prior or regularizers, choice of a 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 (a form of meta learning or transfer learning). Recently, it has been proposed to apply collaborative filtering methods to model search [62]. **Wrapper methods** consider the learning machine as a **black-box** capable of learning from examples and making predictions once trained. They operate with a search algorithm in hyper-parameter space (for example grid search or stochastic search) and an evaluation function assessing the trained learning machine performances (for example the cross-validation error or the 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.,* [47]). Other embedded methods jointly optimize parameters and hyper-parameters [34]–[36], [45], [49].

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 over-fitting avoidance via the notion of hyper-priors, but at the expense of making assumptions on how the data were generated (which underly all Bayesian approaches) 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 both to (1) modeling choices and (2) data split. Much remains to be done to address joinly statistical and computational issues. We hope that this challenge will offer an unbiased platform to compare and contrast methods addressing these problems.

IV. Challenge design

A. Tasks

This challenge is concerned with regression and classification problems (binary, multi-class, or multi-label) from data already formatted in fixed-length feature-vector representations. Each task is associated with a dataset coming from a real application. The domains of application are very diverse and 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 samples in rows and features (or variables) in columns. For instance, in a medical application, the samples may represent patient records and the features may represent results of laboratory analyses. The goal is to predict a target value, for instance the diagnosis "diseased" or "healthy" in the case of a medical diagnosis problem. The identity of the datasets and the features is concealed (except in round 0) to avoid the use of domain knowledge and push the participants to design fully automated machine learning solutions. In addition, the tasks are constrained by a "Time Budget" and a "Scoring Metric" Task, scoring metric and time budget are provided with the data, in a special "info" file.

B. Time Budget

The Codalab platform provides computational resources shared by all participants. To ensure the fairness of the evaluation, 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 the participants with new "info" files. The participants who submit results (instead of code) are NOT constrained by the time budget, since they can run their code on their own platform. This may be advantageous for entries counting towards the Final phases (immediately following a Tweakathon). The participants wishing to also enter the AutoML phases, which require submitting code, can submit BOTH results and code (simultaneously). There is no disadvantage in submitting both results and code. The results do not need to have been produced by the code submitted. For instance, if a participant does not want to submit personal code, he may submit the sample code provided by the organizers together with his submitted results.

C. Scoring Metrics

The scoring program computes a score by comparing submitted predictions with reference "target values". For each sample i, i=1:P (P being the size of the vaildation set or of the test set), the target value is a continuous numeric coefficient y_i, for regression problem or a vector of binary indicators $[y_{ik}]$ in $\{0, 1\}$, for multi-class or multi-label classification problems (one per class k). The participants must turn in prediction values matching as closely as possible the target value, in the form of a continuous numeric coefficient q_i for regression problem or a vector of numeric coefficients $[q_{ik}]$ in the range $[0, 1]$ for multi-class or multi-label classification problems (one per class k).

The Starting Kit contains the Python implementation of all scoring metrics used to evaluate the entries. Each dataset has its own metric (scoring criterion), specified in its "info" file. All scores are re-normalized such that the expected value of the score for a "trivial guess" 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 by the average of the scores of each binary classification sub-problem. The scores are taken from the following list:

- **R2:** R-square or "coefficient of determination" used for regression problems: R2 = 1-MSE/VAR, where MSE=$< (y_i - q_i)^2 >$ is the mean-square-error and VAR= $< (y_i - m)^2 >$ is the variance, with m=$< y_i >$.
- **ABS:** A coefficient similar to the R2 but based on mean absolute error (MAE) and mean absolute deviation (MAD): ABS = 1-MAE/MAD, with MAE=$< abs(y_i - q_i) >$ and MAD=$< abs(y_i - m) >$.
- **BAC:** Balanced accuracy, which is the average of class-wise accuracy for classification problems (or the average of sensitivity (true positive rate) and specificity (true negative rate) for the special case of 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. The class-wise accuracy is averaged over all classes for multi-label problems. For multi-class classification problems, the predictions are binarized by selecting the class with maximum prediction value $argmax_k q_{ik}$ before computing the class-wise accuracy. We normalize the BAC with the formula BAC := (BAC-R)/(1-R), 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 classification problems).
- **AUC:** Area under the ROC curve, used for ranking and for binary classification problems. The ROC curve is the curve of sensitivity vs. 1-specificity, when a threshold is varied on the predictions. The AUC is identical to the BAC for binary predictions. The AUC is calculated for each class separately before averaging over all classes. We normalize it with the formula: AUC := 2AUC-1, making it de-facto identical to the so-called Gini index.
- **F1 score:** The harmonic mean of precision and recall. Precision=positive predictive value=true positive/all called positive. Recall=sensitivity=true positive

rate=true positive/all real positive. Prediction thresholding and class averaging is handled similarly as in the case of the BAC. We also normalize F1 with F1 := (F1-R)/(1-R), where R is the expected value of F1 for random predictions (i.e. R=0.5 for binary classification and R=(1/C) for C-class classification problems).

- **PAC:** Probabilistic accuracy PAC = exp(- CE) based on the cross-entropy or log loss, CE = - $< \sum_k \log(q_{ik}) >$ for multi-class classification and CE = - $< y_i \log(q_i) + (1-y_i) \log(1-q_i) >$ for binary classification and multi-label problems. Class averaging is performed after taking the exponential in the multi-label case. We normalize with PAC := (PAC-R)/(1-R), where R is the score obtained using $q_i =< y_i >$ or $q_{ik} =< y_{ik} >$ (*i.e.,* using as predictions the fraction of positive class examples, as an estimate of the prior probability). We note that for R2, ABS, and PAC the normalization uses a "trivial guess" corresponding to the average target value $q_i =< y_i >$ or $q_{ik} =< y_{ik} >$. In contrast, for BAC, AUC, and F1 the "trivial guess" is a random prediction of one of the classes with uniform probability.

In all formulas the brackets $< . >$ designates the average over all P samples indexed by i: $< y_i >= (1/P) \sum_i (y_i)$. Only R2 and ABS make sense for regression; we compute the other scores for completeness by replacing the target values by binary values after thresholding them in the mid-range.

D. Leaderboard score calculation

Each round includes five datasets from different application domains, spanning various levels of difficulty. The participants (or their submitted programs) provide prediction results for the withheld target values (called "solution"), for all 5 datasets. Independently of any intervention of the participants, the original version of the scoring program supplied by the organizers is run on the server to compute the scores. For each dataset, the participants are ranked in decreasing order of performance for the prescribed scoring metric associated with the given task. The overall score is computed by averaging the ranks over all 5 datasets and shown in the column $< rank >$ on the leaderboard.

The results of the LAST submission made are used to compute the leaderboard results (so you must re-submit an older entry that you prefer if you want it to count as your final entry). This is what is meant by Leaderboard modifying disallowed. In phases marked with a [+], the participants with the three smallest $< rank >$ are eligible for prizes, if they meet the Terms and Conditions.

E. Learning curves

We ask the participants to test their systems regularly while training to produce intermediate prediction results, which will allow us to make learning curves (performance as a function of traintime). Using such **learning curves**, we will adjust the "time budget" in subsequent rounds (eventually giving more computational time to the participants). But only the last point (corresponding to the file with the largest order number) is used for leaderboard calculations.

F. Estimation of performance on future data

In a challenge we organized in 2006 (The Performance Prediction Challenge) [63], we asked the participants not only to design good models that generalize well on future data, but to predict their preformance. We believe this is an essential part of the AutoML problem. However, in this challenge, to limit the complexity of design and evaluation, we did not introduce this additonal task.

Besides the learning curves as a function of time spend searching for the best "hyper-model", we could draw learning curves as a function of the number of training examples. Such curves are useful to evaluate whether having more training examples would significantly improve performance. Evaluating such learning curves on the basis of training data only and providing extrapolations are also important AutoML tasks, which are not part of this challenge, but could be added to future evaluations.

G. Phases and rounds

The challenge is run in multiple phases grouped in rounds, alternating AutoML contests and Tweakathons. There are 6 six rounds: Round 0 (Preparation round), followed by 5 rounds of progressive difficulty (Novice, Intermediate, Advanced, Expert, and Master). Except for round 0 (preparation) and round 5 (termination), all rounds include 3 phases, alternating Tweakathons and AutoML contests:

The results of the last submission made are shown on the leaderboard. Submissions are made in Tweakathon phases only. The last submission of one phase migrates automatically to the next one. If code is submitted, this makes it possible to participate to subsequent phases without making new submissions. Prizes are attributed for phases marked with a [+] during which there is NO submission. The total prize pool is $30,000 (see Rewards and Terms and Conditions for details).

H. Code vs. result submission

To participate in the AutoML[n] phase, 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, in Tweakathon[n] the results are used for scoring rather than re-running the code in Tweakathon[n] and Final[n]. The code is executed when results are unavailable or not well formatted. Hence there is no disadvantage to submitting both results and code. There is no obligation to submit the code, which has produced the results provided. Using mixed submissions of results and code, different methods can be used to enter the Tweakathon/Final phases and to enter the AutoML phases. Submissions are made only during Tweakathon, with a maximum of 5 submissions per day. Immediate feed-back is provided on the leaderboard on validation data. The participants are ranked on the basis of test data performance during the Final and AutoML phases.

V. DATA

In every round, there are 5 datasets in each round spanning a range of difficulties. We will progressively introduce difficulties from round to round (each round cumulating all the difficulties of the previous ones plus new ones): Some datasets

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 I: **Phases of round n.** For each dataset, a labeled training set is provided for training and two unlabeled sets (validation set and test set) are provided for testing.

may be recycled from previous challenges, but reformatted into new representations, except for the final MASTER round, which includes only completely new data. The 5 rounds following the preparation round (round 0) are:

1) NOVICE: Binary classification problems only; no missing data; no categorical variables; moderate number of features (< 2000); balanced classes; BUT sparse and full matrices; presence of irrelevant variables; various Ptr/N.
2) INTERMEDIATE: Multi-class and binary classification problems + additional difficulties including: unbalanced classes; small and large number of classes (several hundred); some missing values; some categorical variables; up to 5000 features.
3) ADVANCED: All types of classification problems, including multi-label + additional difficulties including: up to 300,000 features.
4) EXPERT: Classification and regression problems, covering the entire range of data complexity.
5) MASTER: Classification and regression problems, all difficulties, completely new datasets.

The datasets cover a wide range of application areas: pharmacology, medicine, marketing, ecology, text, image, video and speech processing. The number of variables and samples vary between thousands and millions. The datasets span a range difficulties (sparsity, missing data, noise, categorical variables, etc.). All datasets will be pre-formatted in a fixed-length feature-based representation.

In round 0, we used data from previous challenges to carve out tasks illustrating various difficulties you might encounter in this new challenge:

1) **adult** = Dense data; categorical variables; multilabel classification: A rehash of the Adult 1994 census dataset (from UCI repository), donated by Ronny Kohavi and Barry Becker.
2) **cadata** = Regression data: LibSVM dataset originally from Statlib house prices, Pace and Barry (1997).
3) **digits** = Semi-dense data; multiclass classification: A dataset carved out of 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 (NIST).
4) **dorothea** = Sparse binary data; binary classification: Dataset prepared for the NIPS 2003 feature selection challenge from one of the KDD (Knowledge Discovery in Data Mining) Cup 2001 tasks. DuPont Pharmaceuticals graciously provided this data set for the KDD Cup 2001 competition.
5) **newsgroups** = Sparse data; multiclass classification: The 20 NewsGroups data set (Ken Lang and Tom Mitchell (1997)). One of the most used data sets for text categorization available in the UCI repository.

TABLE II: Preparation phase (phase 0) results on validation data. We show the rank in parentheses for each dataset. The winner will be determined by average rank.

User	Rank	Set 1	Set 2	Set 3	Set 4	Set 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)

The statistics of the data are summarized in table 4.

From round 1, we will no longer disclose the identity of the datasets nor the meaning of the features.

VI. Baseline software and preliminary results

We provided baseline software written in Python, using the machine learning library Scikit-learn [9]. The software 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 outperform significantly the baseline method (reference) on several datasets, see Table II.

VII. Discussion

Several participants asked us "what would you do"? Drawing on results of past challenges, the following strategy seems reasonable: (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"; (3) use an ensemble method to grow an ever improving ensemble until the time is out, using *e.g.,* the method proposed in [14].

Some participants asked for increasing the computer resources. In phase 0 we provided 1 hour of computing on an 8-core machine per submission. We will progressively ramp up the computer time up to 10 hours per submission throughout the challenge. Future editions of the challenge might run on Hadoop to allow the participants to process much larger datasets and enter the big data era.

We made an effort to provide a wide variety of datasets covering many application domains and illustrating different types of tasks, metrics of success, and difficulties. In post-challenge analyses, we plan to complement this effort by systematically testing of the winning entries on variants of the datasets semi-synthetically generated to cover a range of problem complexity (*e.g.,* varying the proportion of training

Num	Name	Task	Metric	Time	Cnum	Cbal	Sparse	Missing	Catvar	Irrvar	Pte	Pva	Ptr	N	Ptr/N
1	adult	multilabel	f1_metric	300	3	1	0.16	0.011	1	0.5	9768	4884	34190	24	1424.58
2	cadata	regression	r2_metric	200	0	NaN	0	0	0	0.5	10640	5000	5000	16	312.5
3	digits	multiclass	bac_metric	300	10	1	0.42	0	0	0.5	35000	20000	15000	1568	9.57
4	dorothea	binary	auc_metric	100	2	0.46	0.99	0	0	0.5	800	350	800	100000	0.01
5	newsgroups	multiclass	pac_metric	300	20	1	1	0	0	0	3755	1877	13142	61188	0.21

Fig. 4: **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 (large means more zeros); Missing = fraction of missing values; Catvar = one if has categorical variables; Irrvar = fraction of irrelevant variables; Pte = number of test patterns; Pva = number of validation patterns; Ptr = number of training patterns; N = number of features; Ptr/N = aspect ratio of the data.

examples, missing data, distractor variables, the metrics of success, and the traintime). We intend to relate the results with some metrics of data complexity recently proposed [64], [65].

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 Joachim Buhmann, ETH Zürich. We selected the 30 datasets used in the challenge among 72 datasets that were donated or formatted using data publicly available by: Yindalon Aphinyanaphongs, Olivier Chapelle, Hugo Jair Escalante, Sergio Escalera, Isabelle Guyon, Zainab Iftikhar Malhi, Vincent Lemaire, Chih Jen Lin, Meysam Madani, Bisakha Ray, Mehreen Saeed, Alexander Statnikov, Gustavo Stolovitzky, Hans-Jürgen Thiesen, and Ioannis Tsamardinos. Many people provided feed-back to early designs of the protocol and/or tested the challenge platform, including: Kristin Bennett, Marc Boullé, Cécile Capponi, Richard Caruana, Gavin Cawley, Gideon Dror, Hugo Jair Escalante, Sergio Escalera, Tin Kam Ho, Hugo Larochelle, Vincent Lemaire, Chih Jen Lin, Víctor Ponce López, Nuria Macia, Simon Mercer, Florin Popescu, Mehreen Saeed, Danny Silver, Alexander Statnikov, and Ioannis Tsamardinos. The software developers who contributed to the implementation of the Codalab platform and the sample code include: Eric Camichael, Isabelle Guyon, Ivan Judson, Christophe Poulain, Percy Liang, Arthur Pesah, Xavier Baro Solé, Erick Watson, Michael Zyskowski.

REFERENCES

[1] T. H. Davenport and D. Patil, "Data scientist: The sexiest job of the 21st century," *Harvard Business Review*, October 2012.

[2] S. Slaughter and L. Delwiche, *The Little SAS Book for Enterprise Guide 4.2*. Sas Inst, 2010. [Online]. Available: http://books.google.com/books?id=mdtHcilcfHIC

[3] A. Field, *Discovering Statistics Using SPSS*, ser. Introducing Statistical Methods Series. SAGE Publications, 2009. [Online]. Available: http://books.google.com/books?id=5253SAL5nDgC

[4] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, Nov. 2009. [Online]. Available: http://doi.acm.org/10.1145/1656274.1656278

[5] M. J. Crawley, *Statistics: An Introduction using R*. Wiley, 2005, iSBN 0-470-02297-3. [Online]. Available: http://www.bio.ic.ac.uk/research/crawley/statistics/

[6] Y. LeCun and L. Bottou, "Lush reference manual," NYU, Tech. Rep., 2002, code available at http://lush.sourceforge.net. [Online]. Available: http://lush.sourceforge.net

[7] J. Weston, A. Elisseeff, G. BakIr, and F. Sinz, "Spider," 2007, http://mloss.org/software/view/29/.

[8] A. Saffari and I. Guyon, "Quick start guide for CLOP," Graz University of Technology and Clopinet, Tech. Rep., May 2006. [Online]. Available: http://clopinet.com/CLOP/

[9] 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 E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[10] S. Sonnenburg, M. Braun, and C. S. Ong, 2007-2012. [Online]. Available: http://mloss.org

[11] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *IJCAI*, 1995, pp. 1137–1145.

[12] C. Andrieu, N. De Freitas, and A. Doucet, "Sequential MCMC for bayesian model selection," in *Higher-Order Statistics, 1999. Proceedings of the IEEE Signal Processing Workshop on*. IEEE, 1999, pp. 130–134.

[13] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *JMLR*, vol. 3, pp. 1157–1182, March 2003. [Online]. Available: http://jmlr.csail.mit.edu/papers/volume3/guyon03a/guyon03a.pdf

[14] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes, "Ensemble selection from libraries of models," in *Proceedings of the Twenty-first International Conference on Machine Learning*, ser. ICML '04. New York, NY, USA: ACM, 2004, pp. 18–. [Online]. Available: http://doi.acm.org/10.1145/1015330.1015432

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

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

[17] T. Schaul, S. Zhang, and Y. LeCun, "No more pesky learning rates," *CoRR*, vol. abs/1206.1106, 2012. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1206.html#abs-1206-1106

[18] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems 25*, 12/2012 2012.

[19] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search," *CoRR*, vol. abs/1209.5111, 2012.

[20] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-weka: Combined selection and hyperparameter optimization of classification algorithms," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '13. New York, NY, USA: ACM, 2013, pp. 847–855. [Online]. Available: http://doi.acm.org/10.1145/2487575.2487629

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

[22] Z. Ghahramani, "Unsupervised learning," in *Advanced Lectures on Machine Learning*. Springer-Verlag, 2004, pp. 72–112.

[23] J. Rissanen, "Modeling by shortest data description," *Automatica*, vol. 14, pp. 465–471, 1978.

[24] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural Comput.*, vol. 4, no. 1, pp. 1–58, 1992.

[25] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer series in statistics. New York: Springer, 2001.

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

[27] V. Vapnik, *Statistical learning theory*. Wiley, 1998.

[28] B. Scholkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, 2001.

[29] H. J. Escalante, M. Montes, and L. E. Sucar, "Particle swarm model selection," *J. Mach. Learn. Res.*, vol. 10, pp. 405–440, 2009.

[30] Q. Sun, B. Pfahringer, and M. Mayo, "Full model selection in the space of data mining operators," in *GECCO (Companion)*, 2012, pp. 1503–1504.

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

[32] S. Dempe, *Foundations of Bilevel Programming*. Dordrecht, The Netherlands: Kluwer Academic Publishers, 2002.

[33] G. Kunapuli, K. P. Bennett, and J.-S. Pang, "Bilevel model selection for support vector machines," in *Data Mining and Mathematical Programming*, P. Pardalos and P. Hansen, Eds. AMS, 2008, vol. 45, pp. 129–158.

[34] K. P. Bennett, G. Kunapuli, J. Hu, and J.-S. Pang, "Bilevel optimization and machine learning," in *Computational Intelligence: Research Frontiers: IEEE WCCI 2008, Hong Kong, China, June 1-6, 2008 : Plenary/invited Lectures*, ser. Lecture Notes in Computer Science, J. Zurada and et al, Eds. Springer, 2008, vol. 5050, pp. 25–47.

[35] K. P. Bennett, J. Hu, X. Ji, G. Kunapuli, and J.-S. Pang, "Model selection via bilevel optimization," *International Joint Conference on Neural Networks*, pp. 1922–1929, 2006.

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

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

[38] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, pp. 1189–1232, 2000.

[39] T. Hastie, R. Tibshirani, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. New York: Springer-Verlag, 2001.

[40] Y. Bengio, A. C. Courville, and P. Vincent, "Unsupervised feature learning and deep learning: A review and new perspectives," *CoRR*, vol. abs/1206.5538, 2012.

[41] J. P. A. Ioannidis, "Why most published research findings are false," *PLoS Med*, vol. 2, no. 8, p. e124, 08 2005.

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

[43] M. I. Jordan, "On statistics, computation and scalability," *Bernoulli*, vol. 19, no. 4, pp. 1378–1390, Sep. 2013. [Online]. Available: http://dx.doi.org/10.3150/12-BEJSP17

[44] G. Cawley, "Grid search based model selection," Tech. Rep., preprint.

[45] G. Moore, C. Bergeron, and K. P. Bennett, "Model selection for primal SVM," in *Machine Learning*, To appear.

[46] G. Cawley, "Leave-one-out cross-validation based model selection criteria for weighted ls-svms," in *IJCNN*, 2006, pp. 1661–1668.

[47] I. Guyon and G. Dreyfus, *Feature Extraction, Foundations and Applications*, ser. Series Studies in Fuzziness and Soft Computing. Physica-Verlag, Springer, 2006, ch. 2: Assessment methods.

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

[49] S. Keerthi, V. Sindhwani, and O. Chapelle, "An efficient method for gradient-based adaptation of hyperparameters in svm models," *NIPS*, 2006.

[50] 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, no. 2, p. 1391, 2005.

[51] M. 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.

[52] M. Momma and K. P. Bennett, "A pattern search method for model selection of support vector regression," in *SDM*, 2002.

[53] A. Statnikov, L. Wang, and C. Aliferis, "A comprehensive comparison of random forests and support vector machines for microarray-based cancer classification," *BMC bioinformatics*, vol. 9, no. 1, p. 319, 2008.

[54] 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.

[55] J. Snoek, H. Larochelle, and R. Adams, "Practical bayesian optimization of machine learning algorithms," *NIPS*, 2012.

[56] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures." in *ICML (1)*, ser. JMLR Proceedings, vol. 28. JMLR.org, 2013, pp. 115–123. [Online]. Available: http://dblp.uni-trier.de/db/conf/icml/icml2013.html#BergstraYC13

[57] K. Swersky, J. Snoek, and R. P. Adams, "Multi-task bayesian optimization," in *Advances in Neural Information Processing Systems 26*, 12/2013 2013.

[58] 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. [Online]. Available: http://dx.doi.org/10.2307/2288636

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

[60] A. Blum and P. Langley, "Selection of relevant features and examples in machine learning," *Artificial Intelligence*, vol. 97, no. 1-2, pp. 245–271, December 1997.

[61] I. Guyon, S. Gunn, M. Nikravesh, and L. Z. Editors, *Feature Extraction, Foundations and Applications*, ser. Studies in Fuzziness and Soft Computing. With data, results and sample code for the NIPS 2003 feature selection challenge. Physica-Verlag, Springer, 2006. [Online]. Available: http://clopinet.com/fextract-book/

[62] R. Bardenet, M. Brendel, B. Kégl, and M. Sebag, "Collaborative hyperparameter tuning," in *30th International Conference on Machine Learning (ICML 2013)*, S. Dasgupta and D. McAllester, Eds., vol. 28. Atlanta, United States: Acm Press, Jun. 2013, pp. 199–207. [Online]. Available: http://hal.in2p3.fr/in2p3-00907381

[63] *Performance Prediction Challenge*, 2006. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1716305

[64] T. K. Ho, M. Basu, and S. Member, "Complexity measures of supervised classification problems," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, pp. 289–300, 2002.

[65] N. Maci, T. K. Ho, A. Orriols-Puig, and E. Bernad-Mansilla, "The landscape contest at icpr 2010." in *ICPR Contests*, ser. Lecture Notes in Computer Science, D. nay, Z. ataltepe, and S. Aksoy, Eds., vol. 6388. Springer, 2010, pp. 29–45. [Online]. Available: http://dblp.uni-trier.de/db/conf/icpr/contests2010.html#MaciaHOB10