Fast and Informative Model Selection Using Learning Curve Cross-Validation

Felix Mohr and Jan N. van Rijn

***Abstract*—Common cross-validation (CV) methods like k-fold cross-validation or Monte Carlo cross-validation estimate the predictive performance of a learner by repeatedly training it on a large portion of the given data and testing it on the remaining data. These techniques have two major drawbacks. First, they can be unnecessarily slow on large datasets. Second, beyond an estimation of the final performance, they give almost no insights into the learning process of the validated algorithm. In this article, we present a new approach for validation based on learning curves (LCCV). Instead of creating train-test splits with a large portion of training data, LCCV iteratively increases the number of instances used for training. In the context of model selection, it discards models that are unlikely to become competitive. In a series of experiments on 75 datasets, we could show that in over 90% of the cases using LCCV leads to the same performance as using 5/10-fold CV while substantially reducing the runtime (median runtime reductions of over 50%); the performance using LCCV never deviated from CV by more than 2.5%. We also compare it to a racing-based method and successive halving, a multi-armed bandit method. Additionally, it provides important insights, which for example allows assessing the benefits of acquiring more data.**

***Index Terms*—Decision making, learning curves, model selection, supervised machine learning.**

I. Introduction

The model selection task is to pick, from a set or sequence of machine learning models, the one that has the best predictive performance. An important sub-task of model selection is to estimate the performance of each candidate model, which is done by so-called *validation methods*. For example, tools for automated machine learning (AutoML) make excessive use of validation techniques to assess the performance of machine learning pipelines [9], [28], [31], [38]. To this end, model selection approaches can often be *configured* with a validation function, which maps models to performance quality estimates. Typical validation functions are k-fold cross-validation (CV) and Monte-Carlo Cross-Validation (MCCV).

Manuscript received 27 November 2021; revised 14 December 2022; accepted 13 February 2023. Date of publication 8 March 2023; date of current version 30 June 2023. Recommended for acceptance by B. Hammer. *(Corresponding author: Felix Mohr.)*

Felix Mohr is with the Faculty of Engineering, Universidad de La Sabana, Chía 250001, Colombia (e-mail: felix.mohr@unisabana.edu.co).

Jan N. van Rijn is with the Leiden Institute of Advanced Computer Science (LIACS), Leiden University, 2333 CA Leiden, The Netherlands (e-mail: j.n.van.rijn@liacs.leidenuniv.nl).

This article has supplementary downloadable material available at https://doi.org/10.1109/TPAMI.2023.3251957, provided by the authors.

Digital Object Identifier 10.1109/TPAMI.2023.3251957

One problem of these methods is that they cannot easily early discard candidate models that do not seem to be competitive. In many cases, an inspection of the *learning curve*, i.e., the curve describing the performance of a candidate at different training set sizes on which it is being trained, shows at an early stage that a candidate, even if being trained on the target size, will not be competitive with a current baseline [25]. This target size often lies between 70% and 90% of the available data for CV and MCCV, but sub-optimal candidates could often be detected and dropped already at much smaller sizes, sometimes below 5% of the available data.

Additionally, vanilla validation procedures provide little additional information to the data owner. Data owners usually need to ask themselves the question of which of the conventional options would improve the quality of their current model: (i) gather more data points, (ii) gather more attributes, or (iii) apply a wider range of models. Common validation techniques only provide information for (iii).

In this article, we address these shortcomings through *learning-curve-based cross-validation* (LCCV). Instead of evaluating a candidate just for one training set size, it is evaluated, in increasing order, at different so-called *anchors* of training fold sizes, e.g., for 64, 128, 256, 512,... instances. At each anchor, several evaluations are conducted until a stability criterion is met. Other authors also utilized learning curves [20], [25], [39] or sub-sampling [16], [21], [33], [35], [36] to speed-up model selection by discarding unpromising candidates early. The main contribution of LCCV is that it *combines* empirical learning curves with an *convexity assumption* on learning curves to early prune candidates only if they are unlikely to be an optimal solution. The discarding decision in LCCV is model-free, which means that it does not rely on an imposed learning curve model like a power law.

Our work is based on observation-based learning curves rather than *iteration-based* curves, which are obtained from learner performance observations across iterations like in neural networks [2], [7], [18], [25]. First, even though incremental versions have been proposed for many learners, these are often not realized in widely used implementations like WEKA [14] or scikit-learn [32] and hence not available for data scientists using those libraries. Iteration-based learning curves are hence largely limited to neural networks in practice. Second, empirical evidence supports the convexity assumption of observation-based learning curves (see Appendix F), (available online), whereas for iteration-based learning curves there is even contrary evidence, e.g., double descent [41].

In summary, LCCV has two advantages over classical validation methods in the context of AutoML [9], [24], [31]:

1) If the learning curve of the model chosen by common CV is convex, LCCV will also choose it and, despite its potential drawback that it builds multiple models at different anchors from scratch, usually in substantially less time.
2) Instead of a single-point observation, LCCV gives insights about the learning *behaviour* of a candidate. This is very useful to answer the question of whether more data would help to obtain better results; information that is not naturally provided by any of the above validation techniques. Note that this is not the same as retroactively generating the learning curve based on the best performing learner after a cross-validation procedure. There is no guarantee that this learner (or even only any of the k best learners) would also be the best if more data were available; this is precisely something that can be revealed with LCCV since it computes the learning curves of *all* learners that it considers.

We sustain these claims empirically by comparing LCCV in the context of an algorithm configuration setting against three other methods, i.e., vanilla cross-validation, racing, and successive halving, over 75 datasets, a strict superset of the AutoML benchmark [13]. Our experiments show that LCCV is on average more than 50% faster than vanilla cross-validation, while usually choosing an equally performing algorithm configuration. Compared to the other baselines we see an interesting Pareto front, where the racing-based implementation yields the fewest speed-ups, but often the most superior performance result, successive halving yields most speed-ups, but due to its aggressive pruning it also misses the optimal configuration in several cases, whereas LCCV performs on both axes very well.

In addition, the experiments show that the learning curves obtained using LCCV can be directly used to assert whether or not more data could improve the overall result of the model selection process.

II. Problem Statement

Our focus is on supervised machine learning. For an *instance space* $\mathcal{X}$ and a *label space* $\mathcal{Y}$, the hypothesis space $H = \{h \mid h : \mathcal{X} \to \mathcal{Y}\}$ is the set of all mappings between the instance space and the label space. The performance of a hypothesis is its *out-of-sample error*

$$\mathcal{R}(h) = \int_{\mathcal{X},\mathcal{Y}} loss(y, h(x))\, d\mathbb{P}(x, y). \tag{1}$$

Here, $loss(y, h(x)) \in \mathbb{R}$ is the penalty for predicting $h(x)$ for an instance $x \in \mathcal{X}$ when the true label of that instance is $y \in \mathcal{Y}$. Finally, $\mathbb{P}$ is a joint probability measure on $\mathcal{X} \times \mathcal{Y}$ from which the available dataset d has been generated.

The problem of *model selection* is to select, from some set or sequence of *learners*, the one that creates the best hypotheses on average for some given data. A learner is a function $a : D \to H$, where $D = \{d \mid d \subset \mathcal{X} \times \mathcal{Y}\}$ is the space of all *datasets*, each of which is a finite relation between the two spaces. Given a dataset d, a hypothesis $h = a(d')$ is obtained by training a given learner on (parts) d' of the given data. The performance of such a hypothesis in (1) is a theoretical quantity that is estimated in practice by disclosing some instances from the learning process. Since the portion of data disclosed this way introduces a random factor, it is common to create several *splits* of the data d into training and validation folds d_{train}, d_{valid}, and to use d_{valid} to estimate the performance of the hypothesis $a(d_{train})$ for each split. Thereby, the goal of model selection is not to optimize over individual hypotheses but over *learners* whose performance is measured by the *average performance* they obtain when being trained on a specific portion (often 80% or 90%) of the available data. This logic is realized by a *validation* function $\nu : D \times A \to \mathbb{R}$. Given a dataset d and a set $A = (a_1, a_2, \ldots)$ of learners, a *model selector* must select $\arg\min_{a\in A} \nu(d, a)$.

We are interested in *sequential* model selection, in which A is a *stream* of candidates (usually generated by an optimizer). To make a choice, the model selector will keep track of the score r of the best candidate seen so far. To this end, it will, for each incoming candidate $a \in A$, (i) determine whether $\nu(d, a)$ will be better than r, and (ii) update r with $\nu(d, a)$ if the first case applies. Such a model selector can use an *informed* and *relaxed* validation function

$$\tilde{\nu} : D \times A \times \mathbb{R} \to \mathbb{R} \cup \{\perp\}, \tag{2}$$

which takes as an additional argument the reference score r, and $\tilde{\nu}(d, a, r)$ may return $\perp$ if $\nu(d, a)$ is worse than r. Some generators of A might want to use exact results of ν to *steer* their behaviour (e.g., Bayesian optimization, evolutionary algorithms, or tree search). However, recent works suggest that discarding exact results of sub-optimal candidates has a negligible effect on the search process [7], and other (tree-based) approaches are, by concept, robust with respect to this issue [28].

Using an informed and relaxed validation function in sequential model selection enables substantial runtime improvements. As such $\tilde{\nu}(d, a, r)$ does not need to compute $\nu(d, a)$ as long as it is worse than r. This economizes on compute resources, by not computing exact ν-scores of sub-optimal performing learners.

III. Related Work

Model selection is at the heart of many data science approaches. When provided with a new dataset, a data scientist is confronted with the question of which model to apply to this. This problem is typically abstracted as the *combined algorithm selection and hyperparameter optimization problem* [38]. To properly deploy model selection methods, there are three important components:

1) A *configuration space*, which specifies the set of algorithms to be considered and, perhaps, their hyperparameter domains
2) A *search procedure*, which determines the order in which these algorithms are considered
3) An *evaluation mechanism*, which assesses the quality of a certain algorithm

Most research addresses the question of how to efficiently search the configuration space, leading to a wide range of methods such as random search [4], [27], Bayesian optimization [5], [15], [37], evolutionary optimization [23], meta-learning [6], [34] and planning-based methods [28]. Our work aims to make the evaluation mechanism faster, while at the same time not compromising the performance of the algorithm selection procedure. As such, it can be applied orthogonal to the many advances made on the components of the configuration space and the search procedure.

The typical methods used as evaluation mechanisms are using classical methods such as a holdout set, cross-validation, leave-one-out cross-validation, and bootstrapping. This can be sped up by applying racing methods, i.e., to stop the evaluation of a given model once a statistical test lets the possibility of improving over the best seen so far appear unlikely. Some notable methods that make use of this are ROAR [15] and iRace [22]. The authors of [8] focus on setting the hyperparameters of AutoML methods, among others selecting the evaluation mechanism. Additionally, model selection methods are accelerated by considering only subsets of the data. By sampling various subsets of the dataset of increasing size, one can construct a learning curve [25], [30], [41]. While these methods at their core have remained largely unchanged, there are two directions of research building upon this basis: (i) model-free learning curves, and (ii) learning curve prediction models.

Model-Free Learning Curves: One basic procedure is simply training and evaluating a model based upon a small fraction of the data [33]. The authors of [35] propose progressive sampling methods using batches of increasing sample sizes (which we also leverage in our work) and propose mechanisms for detecting whether a given algorithm has already converged. However, the proposed convergence detection does not take into account randomness from selecting a given set of sub-samples, making the method fast but at risk of terminating early.

The authors of [42] introduce FLAML, which uses multiple sample sizes, by adding a sample size hyperparameter to the configuration space. The reasoning behind this design choice is to allow low-complexity learners to be tested on small amounts of data, and high-complexity learners to be tested on larger amounts of data. FLAML starts with evaluating configurations on low sample sizes and gradually extends this to larger sample sizes.

Successive halving addresses the model selection problem as a multi-armed bandit problem, that can be solved by progressive sampling [16]. All models are evaluated on a small number of instances, and the best models get a progressively increasing number of instances. While this method yields good performance, it does not take into account the development of learning curves, e.g., some learners might be slow starters, and will only perform well when supplied with a large number of instances. For example, the extra tree classifier [12] is the best algorithm on the PHISHINGWEBSITES dataset when training with all (10 000) instances but the *worst* when using 1 000 or fewer instances; it will be discarded by successive halving (based on all scikit-learn algorithms with default hyperparameters). Hyperband aims to address this by building a loop around successive halving, allowing learners to start at various budgets [21]. While the aforementioned methods all work well in practice, these are all greedy in the fact that they might disregard a certain algorithm too fast, leading to fast model selection but sometimes sub-optimal performances.

Another important aspect is that the knowledge of the whole portfolio plays a key role in successive halving and Hyperband. Other than our approach, which is a *validation* algorithm and does not have knowledge about the portfolio to be evaluated (and makes no global decisions on budgets), successive halving assumes that the portfolio is already defined and given, and Hyperband provides such portfolio definitions. In contrast, our approach will just receive a sequence of candidates for validation, and this gives more flexibility to the approaches that want to use it.

The authors of [36] define the cost-sensitive training data allocation problem, which is related to the problem that we aim to address. They also introduce an algorithm that solves this problem, i.e., Data Allocation using Upper Bounds. Given a set of configurations, it first runs all configurations on two subsamples of the dataset, effectively building the initial segment of the learning curve. Based on this initial segment per configuration, it determines for each an optimistic upper bound. After that, it goes into the following loop: It runs the most promising configuration on a larger sample size and updates the performance bound. It reevaluates which configuration has the most potential at that moment, and continues with assigning more budget to the most promising configuration until one configuration has been run on the entire dataset. Similar to Hyperband and in contrast to LCCV, it needs to know the entire portfolio of learners at the start.

Learning Curve Prediction Models: A model can be trained based on the learning curve, predicting how it will evolve. The authors of [7] propose a set of parametric formulas which can be fitted to learning curve data so that they model the learning curve of a learner. They employ the method specifically to learning curves of neural networks, and the learning curve is constructed based on epochs, rather than instances. This allows for more information about earlier stages of the learning curve without the need to invest additional runtime. By selecting the best-fitting parametric model, they can predict the performance of a certain model for a hypothetical increased number of instances. The authors of [18] build upon this work by incorporating the concept of Bayesian neural networks. When having a set of historic learning curves, one can efficiently relate a given learning curve to an earlier-seen learning curve, and use that to make predictions about the learning curve at hand. The authors of [20] employ a k-NN-based model based on learning curves for a certain dataset to determine which datasets are similar to the dataset at hand. This approach was later extended by the authors of [39], to also select algorithms fast. The authors of [2] proposed a model-based version of Hyperband. This decision problem they aim to address is similar to the decision problem LCCV aims to address. They train a model to predict, based on the performance of the learner on the last-seen sample, whether the current model can still improve upon the best-seen model so far when provided with more data. This requires ample meta-data on earlier optimization

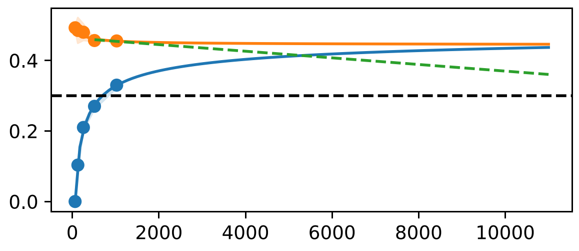

Fig. 1. Example of the pruning of LCCV. The black dashed line represents threshold value r. The orange dots represent performance observations on the validation set per anchor, and the orange curve is an MMF-model fitted through these points. The blue dots represent performance observations on the train set per anchor, and the blue curve is a fitted MMF model. The dashed green line represents the optimistic extrapolation from the last anchors on the validation curve. If this optimistic extrapolation does not improve over threshold value r, it is unlikely that the learner will actually improve over this value, and the evaluation of the learner can be stopped early.

procedures to train the meta-model. Additionally, model-based approaches impose a risk of terminating good candidates early. In contrast to these methods, the method that we propose is model-free and always selects the optimal algorithm if a set of reasonable assumptions are met.

IV. Learning-Curve-Based Cross-Validation

Learning-curve-based cross-validation (LCCV) estimates the performance of a learner based on empirical learning curves. LCCV evaluates the learner on ascendingly ordered so-called *anchor (points)* $S = (s_1, .., s_T)$, which are training set sizes. Several of such evaluations are made on each anchor to ensure a robust performance estimation. The idea is to early recognize whether or not a candidate can be competitive. Section IV-A explains how we use the convexity of the learning curve to decide on this question and Section IV-B details how we make sure that this process only discards candidates that are sub-optimal with high probability. Section IV-C explains how we use a learning curve model (in this case the Morgan-Mercer-Flodin (MMF)) to skip anchors and directly jump to the final anchor s_T for presumably competitive candidates. The algorithm is formalized in Section IV-D. That section also outlines that, even if the early stopping or skipping mechanisms fail, the runtime of LCCV is in the worst case twice as high as the one of a CV.

A. Aborting a Validation

The idea of LCCV is that we can use learning curves [25], [41] to early recognize that a candidate will not be competitive. For this, we can use both the train curves as well as the validation curves. Fig. 1 represents the general idea.

1) Aborting Based on the Validation Curve: We assume that the validation curves are convex. It is known that not all learning curves are convex, but we conducted an extensive empirical study (see Appendix F), available in the online supplemental material, justifying this assumption for the large majority of (the analyzed but presumably also other) cases. Mohr et al. [30] explores this topic in more detail. Convexity is important because it implies that the *slope* of the learning curve is an *increasing* function that, for increasing training data sizes, approximates 0 from below. Recall that we assume error curves in this article; for accuracy curves or F1 curves etc., the curve slopes would be decreasing of course. The convexity implies that we can take the observations of the last two anchors of our *empirical* learning curve, compute the slope, and extrapolate the learning curve with this slope. From this, we obtain a *bound* (not an estimate, as opposed to [7]) for the performance at the full data size and can prune if this bound will be worse than threshold value r [36].

More formally, denote the learning curve as a function $p(s)$, where p is the *true average* performance of the learner when trained with a train set of size s. Convexity of p implies that $p'(s) \leq p'(s')$ if $s' > s$, where p' is the derivative of p. Of course, as a learning curve is a sequence, it is not differentiable in the strict sense, so we rather sloppily refer to its slope p' as the slope of the line that connects a point with its predecessor, i.e., $p'(s) := p(s) - p(s-1)$ for all $s \in \mathbb{N}_+$; note that based on the convexity assumption $p' \leq 0$. If we find, for some s, that $p(s) + p'(s) \cdot (s_T - s) > r$, where s_T is the maximum size on which we can train, then we have evidence that the learner is not relevant.

The problem is of course that we can compute neither $p(s)$ nor $p'(s)$, because it refers to the *true* learning curve. That is, p realizes (1) for the hypotheses produced by a learner on different sample sizes. So we can only try to *estimate* the values of p and p' at selected anchors.

Given a sequence of anchors $s_1, .., s_t$ for which we already observed performance values of a learner, we need to estimate $p(s_t)$ and $p'(s_t)$ in order to determine whether the learner can be pruned based on the optimistic extrapolation of the learning curve. A canonical estimate of $p(s_t)$ is the empirical mean $\hat{\mu}_t$ of the observations at s_t. For the slope, we could rely on these estimates and use convexity to bound the slope with

$$p'(s_t) = p(s_t) - p(s_t - 1) \overset{conv}{\geq} \frac{p(s_t) - p(s_{t-1})}{s_t - s_{t-1}} \approx \frac{\hat{\mu}_t - \hat{\mu}_{t-1}}{s_t - s_{t-1}}. \tag{3}$$

However, replacing p' in the above decision rule by this bound is admissible *only if* the estimates $\hat{\mu}_t$ and $\hat{\mu}_{t-1}$ are accurate (enough). This calls for the usage of *confidence intervals* to obtain a notion of probabilities (here, from a frequentist viewpoint, the probability to observe a mean close to the true one). To compute such confidence bounds, we follow the common assumption that observations at each anchor follow a normal distribution [7], [17], [18]. Since the true standard deviation is not available for the estimate, we use the empirical one.

Collecting enough samples to shrink such confidence intervals to a minimum can be prohibitive, so we propose an alternative method, which estimates the slope in (3) optimistically from the confidence bounds. To this end, we connect the upper end of the interval at s_{t-1} with the lower end of the interval of s_t. This yields the slope of the most *optimistic* learning curve compatible with the observed performances up to a reasonable probability. Formally, let C_i be the confidence intervals at anchor s_i. Extrapolating from the *last* anchor s_t, the best possible performance of this particular learner at the final anchor s_T is

$$p(s_T) \geq \inf C_t - (s_T - s_t)\left(\frac{\sup C_{t-1} - \inf C_t}{s_{t-1} - s_t}\right), \quad (4)$$

which can now be compared against the performance of the best-evaluated learner so far r to decide on pruning.

2) Aborting Based on the Train Curve: While we are interested in the validation performance, early discarding can also be based on the performance of the model evaluated on the training data. We make the following assumptions, which are commonsense in machine learning literature [1]:

- the error on the train performances typically monotonically increases as the number of observations increases.
- the train performance at any anchor is typically better than the validation performance at any anchor.

The consequence is that, once the measured train error is already higher than the reference performance r (which is measured based on validation performance), it is unlikely that this learner will improve over r, and the evaluation can therefore be aborted. We consider this criterion once the observations at each anchor are complete and consider the *best* training performance observed at each anchor. If this value is worse than r at any anchor, the candidate is pruned. Appendix E elaborates on experiments that justify this, available in the online supplemental material.

We observed that these assumptions do not hold for tree-based learners. If the hyperparameter that controls the number of observations that are needed to end up in a leaf is set too high, these types of learners can only start learning at a given number of observations. As such, we excluded these learners from pruning based on the train performance.

B. Calibrating the Width of Confidence Intervals

Given the above approach based on confidence intervals, the next question is on how many samples these intervals should be based on. At least two observations are needed to obtain a standard deviation and hence to create a meaningful interval. However, apart from this restriction, the number of samples per anchor is a factor that is under our control. We should determine it carefully as setting this value too low might result in inaccurate or large confidence intervals while setting this value too high might unnecessarily slow down the validation process.

The behaviour of LCCV can here be controlled via four parameters. The first and the second parameters are simply the minimum and the maximum number of evaluations at each anchor, respectively. A minimum of other than 2 can make sense if we either want to allow greediness (1) or to be a bit more conservative about the reliability of small samples (> 2). To enable a dynamic approach between the two extremes, we consider a stability criterion ε. As soon as the confidence interval width drops under ε, we consider the observation set to be stable. Since error rates reside in the unit interval in which we consider results basically identical if they differ on an order of less than 10^{-4}, this value could be a good choice for ε. While this is indeed an arguably good choice at the last anchor, the "inner" anchors do not require such a high degree of certainty. The goal is not to approximate the learning curve at all anchors with high precision but only to get a rough grasp on its shape. To achieve this objective, a loose confidence interval of size even 0.1 might be perfectly fine. Otherwise, the validation routine might spend a lot of time on ultimately irrelevant anchors. Hence, we distinguish between the standard ε and a parameter ε_{max} applied only for the last anchor. Appendix E provides a sensitivity analysis over these hyperparameters, available in the online supplemental material.

Since the stability criterion ε could be rather loose, it makes sense to adopt an additional sanity check that assures that the observation set is *convexity-compatible*. We call a set of observations convexity-compatible if there exists a convex function that takes at each anchor a value that is in the confidence interval of the respective anchor. In other words, we do not consider a single empirical learning curve but *all* (convex) learning curves that are compatible with the current confidence bounds at the anchors.

There is an easy-to-check necessary and sufficient condition of the convexity-compatible property. To this end, one tries to construct a curve backward starting from the last two intervals and to decrease the slope as little as possible from each anchor to the next one. Formally, let $C_1, .., C_t$ be the confidence intervals at the t ascendingly ordered anchors, and σ_t be the slopes between the anchors. Let u_t be the minimal value that a convex-compatible curve can take at a given anchor t. If there is no value that u can take, there is no convex-compatible set of curves. We set $u_t = \sup C_t$ as the *worst* possible learning curve value at the *last* anchor and then consider a minimum possible negative slope σ_i at each anchor i. Starting with $\sigma_t = 0$, we can inductively define

$$u_{i-1} = \min\{x \in C_{i-1} \,|\, x \geq u_i - (s_i - s_{i-1})\sigma_i\} \quad \text{and}$$
$$\sigma_{i-1} = \frac{u_i - u_{i-1}}{s_i - s_{i-1}}. \quad (5)$$

Starting with the worst value in the last confidence interval is important to impose the least possible slope in the beginning, since the slope must keep (negatively) increasing.

If the set from which u_{i-1} is chosen is non-empty for all anchors, we have constructed a convex curve inside the confidence bands (necessary condition), and it is also easy to see that there cannot be a convex curve if we fail to produce one in this way (sufficient condition). In other words, one can simply check for convexity compatibility by checking whether the sequence of slopes of the lines that connect the upper bound of C_i with the lower bound of C_{i+1} is *increasing*.

If an observation set is not convexity compatible, we need to "repair" it before we can make decisions on pruning based on these results. Our repair technique is to simply step back one anchor. Stepping back from anchor s_t to s_{t-1}, the algorithm will typically only re-sample once at anchor s_{t-1} since the confidence bound is already small enough. Two cases are possible now. Either the convexity-compatibility is already lost at anchor s_{t-1}, in which case the same procedure is applied recursively stepping back to anchor s_{t-2}. Otherwise, it will return to anchor s_t and also re-sample a point at that anchor. In many cases, the convexity compatibility will have recovered at this time. If not, the procedure is repeated until the problem has been resolved or the maximum number of evaluations for the anchors has been reached.

C. Skipping Intermediate Evaluations

Unless we insist on evaluating all anchors for obtaining insight into the learning process, we should evaluate a learner on the full data once it becomes evident that it is competitive. This is clearly the case if we have a *stable* (small confidence interval) anchor score that is at least as good as the best-observed performance r; in particular, it is automatically true for the first candidate that is being evaluated. Note that, in contrast to aborting a validation procedure of a learner, we cannot lose a relevant candidate by jumping to the full evaluation. We might waste some computational time, but, unless a timeout applies, we cannot lose relevant candidates. Hence, we do not need strict guarantees for this decision.

With this observation in mind, it indeed seems reasonable to *estimate* the performance of the learner on the full data and to jump to the full dataset size if this estimate is at least as good as reference value r. A dozen of function classes have been proposed to model the behaviour of learning curves (see, e.g., [7]), but one of the most precise ones is the Morgan-Mercer-Flodin (MMF) model [3]. The MMF model captures the learning curve as a function $\hat{p}(s) = (ab + cs^d)/(b + s^d)$, where $a, b, c, d \in \mathbb{R}_+$. Given observations for at least four anchors, we can fit a non-linear regression model using, for example, the Levenberg-Marquardt algorithm [3]. After the sampling at an anchor has finished, we can fit the parameters of the above model and check whether $\hat{p}(s_T) \leq r$. In that case, we can immediately skip to the final anchor.

D. The LCCV Algorithm

The LCCV algorithm is sketched in Algorithm 1 and puts all of the above steps together. It initiates variables for the various anchor points (l. 1), the performance observations per anchor (l. 2, one set per anchor) and the various confidence intervals (l. 3, one interval per anchor). Additionally, t represents the index of the anchor that is being evaluated. The algorithm iterates over anchors in a fixed schedule S (in ascending order). At anchor s (lines 7-12), the learner is validated by drawing folds of training size s, training the learner on them, and computing its predictive performance on data not in those folds. These validations are repeated until a stopping criterion is met (cf. Section IV-B). In the pseudo-code, n represents the maximum number of evaluations per anchor. Each anchor also has a minimum number of evaluations, but this is left out for readability. Then it is checked whether the observations are currently compatible with a convex learning curve model. If this is not the case, the algorithm steps back one anchor and gathers more observations to get a better picture and "repair" the currently non-convex view (l. 15); details are described in Section IV-B. Otherwise, a decision is taken with respect to the four possible interpretations of the current learning curve. First, if the train performance at this anchor is already worse than the reference score r, we can safely assume that the

Algorithm 1: LCCV: LearningCurveCrossValidation.

```
1  $(s_1, .., s_T) \leftarrow$ initialize anchor points from min_exp and data size
2  $(O_1, .., O_T) \leftarrow (\emptyset, .., \emptyset)$ // sets of performance observations
3  $(C_1, .., C_T) \leftarrow$ initialize confidence intervals as $[0, 1]$ each
4  $t \leftarrow 1$
5  while $t \leq T \wedge (\sup C_T - \inf C_T > \varepsilon) \wedge |O_T| < n$ do
6      repair_convexity $\leftarrow$ false
       /* gather samples at current anchor point $s_t$ */
7      while $\sup C_t - \inf C_t > \varepsilon \wedge |O_t| < n \wedge \neg$ repair_convexity do
8          run classifier on a new train-validation split for anchor $s_t$
             training points, and add performance observation to $O_t$
9          update confidence interval $C_t$ based on values $O_t$
10         if $t > 1$ then $\sigma_{t-1} = (\sup C_{t-1} - \inf C_t)/(s_{t-1} - s_t)$
11         if $t > 2 \wedge \sigma_{t-1} < \sigma_{t-2} \wedge |O_{t-1}| < n$ then
12             repair_convexity $\leftarrow$ true
       /* Decide how to proceed from this anchor */
13     if average train error $> r$ then
           /* Not applicable for tree-based learners */
14         return $\perp$
15     else if repair_convexity then
16         $t \leftarrow t - 1$
17     else if projected bound for $s_T$ is $> r$ then
18         return $\perp$
19     else if $r = 1 \vee (t \geq 4 \ \wedge$ MMF_ESTIMATE$(s_T) \leq r)$ then
20         $t \leftarrow T$
21     else
22         $t \leftarrow t + 1$
23 return $\langle mean(C_T), (C_1, .., C_T)\rangle$
```

validation error will not improve over this, and the configuration can be pruned. In fact, this can be done before the convexity is repaired (l. 13). An exception to this are the tree-based learners, as explained in Section IV-A. Second, if it can be inferred that the validation performance at s_T will not be competitive with the best-observed performance r, LCCV returns $\perp$ (l. 17, cf. Section IV-A). Third, if extrapolation of the learning curve gives rise to the belief that the learner is competitive, then LCCV directly jumps to validation on the full dataset of the candidate (l. 19, cf. Section IV-C); the condition $t \geq 4$ is required because so many points are necessary to fit the MMF model. In any other case, we just keep evaluating at the next anchor (l. 21). In the end, the estimate for s_T is returned together with the confidence intervals for all anchors.

We observe that the runtime of LCCV is at most twice as much as the one of CV in the worst case, if we assume a computational complexity that is at least linear in the number of data points [29] (proof in Appendix A), available in the online supplemental material:

Theorem 1: Assume that the training times of learning algorithms grow at least linearly. Let d be a dataset, $T = \log_2 |d|$ and $\rho \in \mathbb{N}, \rho \leq T$. For anchors $S = (2^\rho, 2^{\rho+1} .., 2^{\lfloor T \rfloor}, 2^T)$ and with a maximum number of k samples per anchor, the runtime of LCCV is at most twice as high as the one of k-CV, where 2^d is the training size applied by k-CV.

A runtime bound of twice the runtime of CV seems not very exciting. However, this is a *worst case* bound, and empirical results will show that most of the time LCCV is much faster than CV due to its pruning mechanism. Besides, the quality of insights obtained from LCCV substantially augments what is learned from CV, because it allows us to make recommendations to the user on whether to acquire more instances or to find better features; we discuss this in more detail in Section V-B.

V. Evaluation

This evaluation addresses two main questions:

1) How does the runtime performance of a model selection process employing LCCV compare to the usage of 5-CV and 10-CV, and does the usage of LCCV produce competitive results? These questions are considered in Section V-A.
2) Can we use the collected empirical learning curves to make correct recommendations on the question of whether more data will improve the overall result? Section V-B elaborates on this.

To enable full reproducibility, the implementations of all experiments conducted here alongside the code to create the result figures and tables, detailed log files of the experiment runs, and result archives are available for the public.[1]

A. LCCV for Model Selection

In this evaluation, we study the effect of using LCCV instead of 5-CV or 10-CV as a validation method inside a simple AutoML tool based on random search. We also consider two other baselines based on the concept of racing and multi-armed bandits.

1) Benchmark Setup: Our evaluation measures the runtime of a random search AutoML tool that evaluates 256 classification pipelines from the scikit-learn library [32] using different validation techniques to assess candidate performance. The pipelines consist of a classification algorithm, which may be preceded by up to two pre-processing algorithms. The behaviour of the random search is simulated by creating a sequence of 256 random pipelines (including the hyperparameter values for the contained algorithms); the considered algorithms and the exact procedure are explained in Appendix B, available in the online supplemental material, and the resulting pipelines are published in the GitHub repository. We consider the two cases in which $\alpha \in \{80\%, 90\%\}$ of the available data d is used for training by ν.

We compare three sequential model validation techniques. For these techniques, a unified model selection technique iterates over the sequence of pipelines and retrieves, for each of them, a numeric score or $\perp$ from the respectively used validator and hence operates as described in Section II. The first technique to use here is ν itself for the candidate assessment (CV). The second technique (Wilcoxon) is a racing-inspired [22] variant of ν in which candidates are tested on various samples of the data, and are discarded as soon as the hypothesis that they beat the current baseline can be discarded based on a Wilcoxon signed rank test (p-value 0.05).[2] For both approaches, we use 5-CV if $\alpha = 0.8$ and 10-CV if $\alpha = 0.9$; for Wilcoxon we refer to 5-Wilcoxon and 10-Wilcoxon. The third candidate is LCCV, which we refer to as 80-LCCV or 90-LCCV depending on α. We explain the parametrization we used for LCCV in Section V-A2.

[1]github.com/fmohr/lccv/tree/master/publications/2022TPAMI

[2]Note that we could not use the official iRace implementation, as this is written in R and incompatible with our Python experiments.

In addition, we add a technique based on successive halving (SH) [16], which does not fit into the sequential model selection problem from Section II but rather defines its own model selector. Successive halving assumes knowing the full set of candidates from the beginning and halves this population each round while increasing the budget on which surviving candidates are evaluated. The budgets on which populations are evaluated form a set of anchors $\{64\eta^i \mid i \in \mathbb{N}, 64\eta^i \leq \alpha|d|\}$, where η is set maximally so that for the round T in which only two candidates are remaining it holds that $64\eta^T = \alpha|d|$. Successive halving does not track a tentative best candidate but simply returns the better of two candidates of the last round. Depending on α, we write 80-SH or 90-SH.

The *runtime* of each method on a given dataset is the average total runtime of the random search using this validation method. Of course, the concrete runtime can depend on the concrete set of candidates but also, in the case of LCCV, their order. Hence, over 10 seeds, we generate different classifier *portfolios* to be validated by the techniques, measure their overall runtime (of course using the same classifier portfolio and identical order for both techniques), and form the average over them.

The *performance* of the chosen learner of each run is computed by an exhaustive MCCV. To this end, we form 100 bi-partitions of split size 90%/10% and use the 90% for training and 10% for testing in all of the 100 repetitions (the split is 80%/20% if $\alpha = 80\%$). The average error rate observed over these 100 partitions is used as the validation score. Note that we do not need anything like “test” data in this evaluation, because we are only interested in whether the LCCV can reproduce the same model selection as CV. More precisely, we do not try to estimate the true learning curve value at the target train size but the value one would obtain in expectation when using only the *available* data. While this value tends to be an optimistically biased estimate of the true value, the true value itself is just not of interest to the validation mechanism. This is a problem that falls into the responsibility of an anti-over-fitting mechanism, which is not part of the validation mechanism and hence not part of the experiments.

As common in AutoML, we configured a timeout per validation. In these experiments, we set the timeout per validation to 30 minutes. This timeout refers to the *full* validation of a learner and not, for instance, to the validation of a single fold. Put differently, 30 minutes after the random search invoked the validation method, it interrupts validation and receives a partial result. For the case of CV this result is the mean performance among the evaluated folds or `nan` if not even the first fold could be evaluated. For LCCV, the partial result is the average results of the biggest evaluated anchor if any and `nan` otherwise.

To obtain insights over different types of datasets, we ran the above experiments on 75 datasets, a strict superset of the AutoML benchmark suite [13]. These datasets offer a broad spectrum of numbers of instances and attributes. All the datasets are published on the openml.org platform [10], [40]. Appendix C, available in the online supplemental material, shows a list of all datasets and characteristics that we used. To avoid label leakage, all pre-processing operators are fitted based on just the train set (using scikit-learns pipeline operator).

2) Parametrization of LCCV: The minimum number of samples per anchor was set to 3, as this intuitively seems a fair trade-off between confident extrapolation and performance, whereas the maximum number of samples per anchor was set to align with the CV number of samples (5 and 10, respectively). To be in line with successive halving, the minimum exponent ρ was set to 6 to train over at least 64 instances, which is also in line with the experiments conducted by van Rijn et al. [39]. The maximum width parameter ε for the confidence intervals was set according to the anchor. For the last anchor (i.e., of the maximum exponent), it was configured with $\varepsilon_{max} = 0.001$. For intermediate anchors, it was configured with $\varepsilon = 0.1$. Indeed, by configuring these parameters as such, we accept some uncertainty at intermediate anchors, which is arguably mitigated by the conservative extrapolation methodology, whereas at the final anchor we tolerate less uncertainty. While we have chosen these parameters for this set of experiments, many other configurations would have been valid as well. To further explore this, we have conducted a sensitivity analysis for some of the parameters, which can be found in Appendix E, available in the online supplemental material.

3) Technical Specification: The computations were executed in a compute center with Linux machines, each of them equipped with 2.6 Ghz Intel Xeon E5-2670 processors and 20 GB memory. For 13 datasets (see Appendix C for the concrete list), available in the online supplemental material, we used 60 GB of memory instead to avoid memory overflows. Despite the technical possibilities, we did *not* parallelize evaluations in the interest of minimizing potential confounding factors. That is, all the experiments were configured to run with a single CPU core. Experiments running for more than a week were killed for technical reasons, which however only applied in one case for CV and Wilcoxon.

4) Results: The results are summarized in Fig. 2. The boxplots capture, in this order, the average runtimes in minutes of the four methods on each of the 75 datasets (leftmost), the absolute reduction of runtimes of the last three methods compared to CV (left middle), the relative runtime of these three methods compared to the CV baseline (right middle), and the deviations in the error rate of the eventually chosen model of the methods compared to CV (rightmost). The absolute reduction (middle left) shows the saved minutes compared to CV, so more is better for the respective methods. The relative runtime is the quotient of a method's runtime and the runtime of CV, so less is better from that method's perspective. The top panes are for $\alpha = 80\%$, and the bottom panes for $\alpha = 90\%$. More detailed insights, e.g., visualizations per dataset, can be found in Appendix D, available in the online supplemental material.

The plots clearly show that CV and the Wilcoxon-based racing method perform very similarly and are the slowest methods. LCCV comes with significant speed-ups, leading to a median runtime of less than 50 % of CV (middle right plot, both settings). This translates to a median runtime reduction of 7 to 20 hours, with outliers leading up to a speed-up of more than 100 hours (middle left image, both settings). In some cases, LCCV requires more runtime than CV, however, on the larger datasets, where runtime reductions are more interesting, LCCV largely outperforms CV. Figs. 2, 3, 4, and 5 in Appendix D give more insight in this, available in the online supplemental material. Being even much more aggressive in pruning than LCCV, successive halving often (but not always) leads to additional runtime reductions. However, as the middle left plot shows, the absolute runtime reductions are mostly comparable with slight advantages for successive halving.

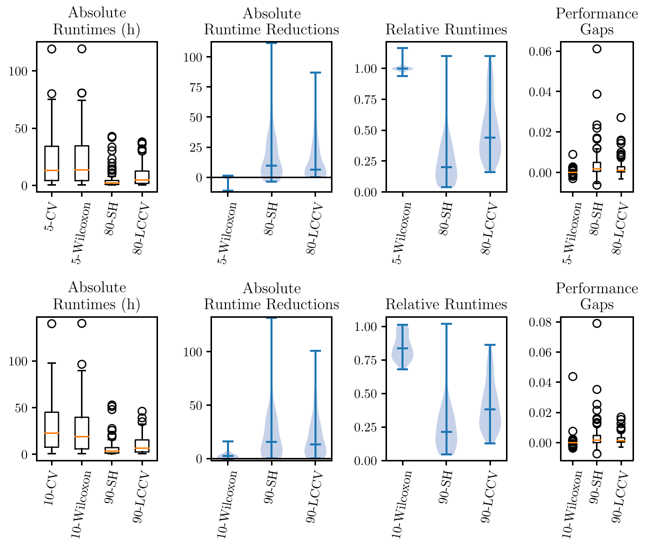

Fig. 2. Comparison between LCCV, vanilla cross-validation, a racing implementation and successive halving on 75 datasets, a superset of the AutoML benchmark. Top pane: Comparisons between methods that train the learner using 80% of the data, and evaluate the learner on the remaining 20%. Bottom pane: Comparison between methods that train using 90% of the data, and evaluate on the remaining 10%.

Comparing the methods in terms of accuracy, we see the opposite pattern (right-most image). The Wilcoxon-based racing method and CV perform very similarly (boxplot of Wilcoxon close to 0), whereas LCCV on average performs slightly worse. The difference in performance is less than 0.01 in over 90% of the cases and almost always less than 0.02. Successive halving performs a more aggressive pruning strategy and misses the optimal performance based on this in more cases.

To summarize, in our interpretation, LCCV is despite some exceptions preferable over CV concerning runtimes. LCCV is particularly strong on datasets with high training runtimes. While LCCV is also competitive in terms of performance most of the time, it occasionally performs worse than CV, which gives rise to further improvements in future work. While in terms of runtime it can not compete with successive halving, it is more robust in terms of performance. This leads to a wider Pareto front of methods, upon which all aforementioned methods take a specialized position. However, we remind that LCCV can be applied in the sequential model selection problem, which is not easily the case for successive halving.

B. LCCV for Data Acquisition Recommendations

We analyze whether the claim that LCCV gives more insights than CV in the data collection process is justified. We use the produced empirical learning curves to derive a power law model with which we answer the question of whether more instances will lead to *sufficiently* better results or not. Of course, in practice, we will be interested not in any marginal improvement at any cost but will require a reasonable ratio between additional instances and improvement.

1) Research Question: We want to assess how suitable the empirical learning curves collected with LCCV are for predicting whether more data would be useful for the performance of a learner. While more instances typically imply better results, additional instances also come at additional costs; as such, this question is not trivial to answer. In the most general case, we can try to find the optimal sample size in terms of a *utility* function [19], [43]. Since the return on investment decreases for increasing sample sizes, the utility assumes its global optimum at the *economic stopping point* and strictly decreases thereafter.

Since the concrete utility function depends on application contexts, we adopt a simplifying approach in this article to address the above question. Instead of introducing a concrete utility function, which is non-trivial [43], we reduce the problem to a binary decision situation: suppose that the user has the option to double the number of available instances and has to decide whether to do so or not. Since there are only two possible situations, we only need to express a condition under which accepting the additional instances is advantageous over staying with the current set. To this end, a threshold β expresses how much the performance (here error rate) must improve to have a utility gain by accepting the additional data. In other words, the utility function is, ordinally and only for the two possible states, characterized by β.

Importantly, the question about improvement does typically not refer to a specific learner but to the improvement that *any* learner of a considered portfolio could achieve over the *currently best* one if more data was available. That is, we are not necessarily interested in whether a learner can improve by at least β on its result but whether the result of the *best* learner can be improved by at least β by the *portfolio*.

We answer the following two research questions: Modelling learning curves with the inverse power law (IPL), i.e., $f(s) = a + bs^{-c}$ [25], fitted from the empirical curves produced by LCCV,

1) how well can we predict the *performance improvement* (regression problem) of individual learners as well as the portfolio as a whole, when the available data is doubled?
2) how well can we predict whether a certain performance improvement β can be achieved (binary classification problem) of individual learners as well as the portfolio as a whole when the available data is doubled? (for various values of β)

2) Realization: To assess the performance of a recommender for these questions on a dataset d, we hold a substantial portion of d apart that will then serve as "new" data. Of course, new data here does not mean untouched validation data but additional training instances that are pretended to be not available before. First, we sample two subsets of the data, d_{full} and d_{half}, of size $|d_{full}| = 10/9 \cdot 2^{\lfloor \log_2(0.9|d|) \rfloor}$, and $|d_{half}| = 10/9 \cdot 2^{\lfloor \log_2(0.9|d|) \rfloor - 1}$, respectively, where d_{half} is sampled as a subset from d_{full}. Choosing the sizes like this implies that using 90% of the data for training will correspond to a power of 2; here the highest power(s) possible for the train set of the dataset. Then, using only the data of d_{half}, we validate all classifiers in the portfolio with LCCV without cancellation and skipping (cf. Sections IV-A and IV-C), so that we have full evaluations at all anchors; recall that by doing so, the runtime is still bound by a factor of two over evaluating only once on all train data (cf. Section IV-D).

To make and qualify a prediction of whether doubling the number of instances will yield an improvement by at least β, we proceed as follows. We denote the performance of algorithm i trained on 90% and validated against 10% of the data in d_{half} and d_{full} as $P^i_{d_{half}}$ and $P^i_{d_{full}}$, respectively. The performance of the *best* performing algorithm is denoted as $P^*_{d_{half}}$ for d_{half} and $P^*_{d_{full}}$ for d_{full}. Pretending that d_{half} is the available data, the quantities $P^i_{d_{half}}$ and hence $P^*_{d_{half}}$ are known after having applied LCCV to the portfolio, but $P^i_{d_{full}}$ and $P^*_{d_{full}}$ are *unknown* and must be estimated. The correct answer is "yes" if $P^*_{d_{half}} - P^*_{d_{full}} \geq \beta$ and "no" otherwise. Given the learning curves of a portfolio of classifiers validated with LCCV, an IPL model is built for each of them using non-linear regression; this model is called the *IPL-regressor*. We denote the *predicted* performance of this model for algorithm i on d_{full} as $\hat{P}^i_{d_{full}}$. Then, we compute $\hat{P}^*_{d_{full}} := \min_i \hat{P}^i_{d_{full}}$ and return "yes" if this value is at most $P^*_{d_{half}} - \beta$ and "no" otherwise. This last decision rule is called the *IPL-classifier* for threshold β.

In this evaluation, we consider a simple portfolio of 17 classifiers. These are the same classifiers from scikit-learn library [32] as considered for the pipelines in Section V-A: BernoulliNB, GaussianNB, Decision Tree, Extra Trees (ensemble), Random Forest, GradientBoosting, kNN, SVC (with four different kernels), MLP, MultinomialNB, PassiveAggressive, LDA, QDA, SGD. We refer to Appendix B.1 for details, available in the online supplemental material.

3) Results: The results are presented in Figs. 3 and 4. In both figures, results in the left column refer to (actual and predicted) improvements for individual classifiers, and the results in the right column refer to improvements over the *best* classifier in the portfolio. For the left column plots, we have included only the insights of 5 out of the 17 classifiers for readability, which are Linear SVC (blue), Gradient Boosting (orange), LDA (green), QDA (red), Random Forest (purple).

To assess the first research question, we consider Fig. 3. We first focus on the top row of the figure, which relates the predicted improvement and the actual improvement. Each data point represents the performance of an algorithm on a dataset; all points of the same colour belong to the same algorithm. The x-value of a point is the *predicted improvement* for an algorithm i when increasing the data from d_{half} to d_{full}, i.e., $P^i_{d_{half}} - \hat{P}^i_{d_{full}}$. The y-axis shows the *actual improvement*, i.e., $P^i_{d_{half}} - P^i_{d_{full}}$. Hence, points close to the diagonal indicate reliable predictions. In the right plot, we show the same information for the portfolio setting. That is, the x-axis represents the *predicted* improvement of the *portfolio*, i.e., $P^*_{d_{half}} - \hat{P}^*_{d_{full}}$, and the y-axis representing

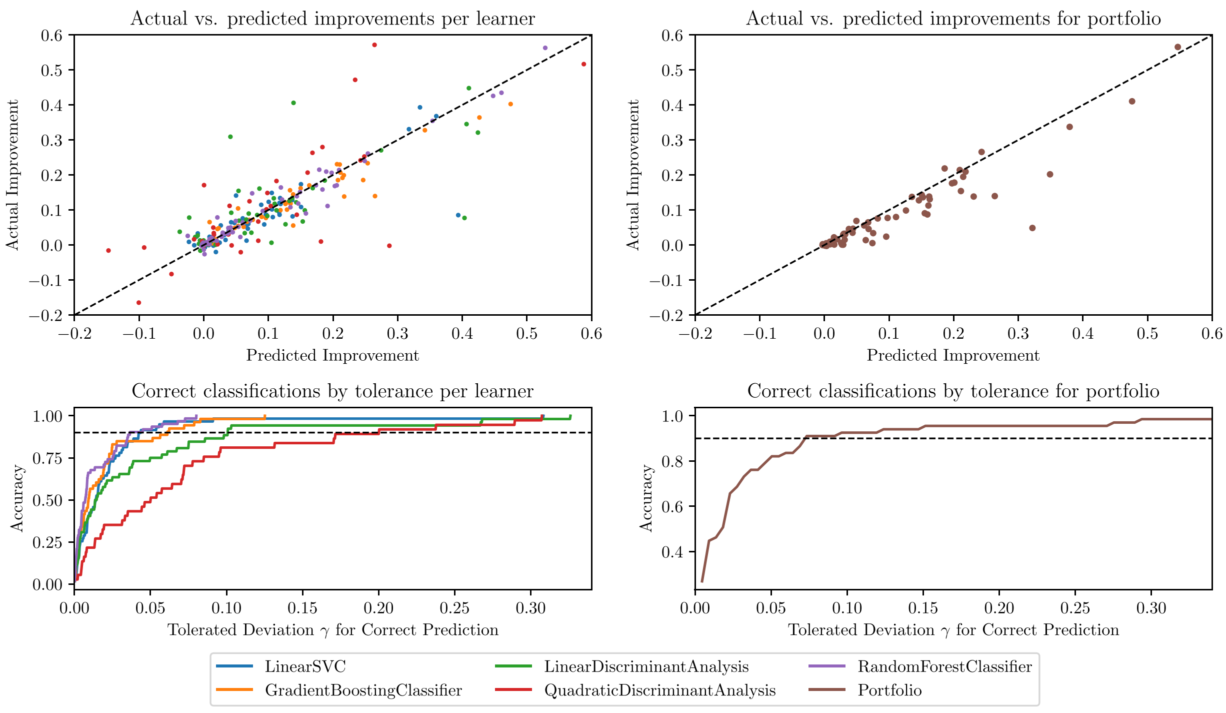

Fig. 3. Top: Predicted versus actual improvement when doubling the data. Bottom: Cumulative empirical distribution of the gap between these numbers.

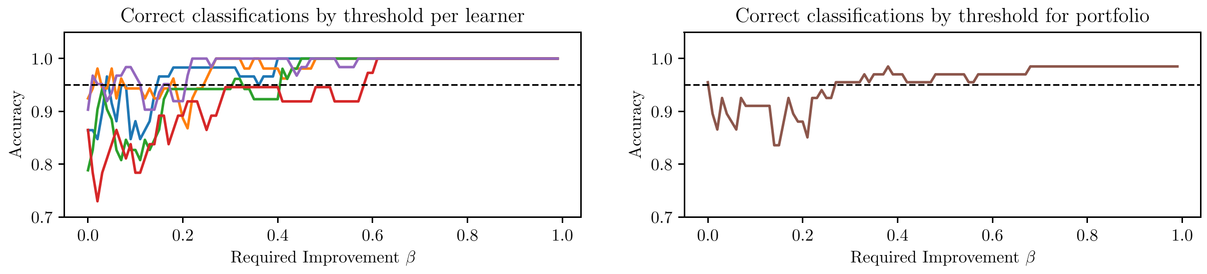

Fig. 4. Accuracy of the IPL classifier, which predicts whether performance will increase by at least β when doubling the data.

the *true* improvement of the portfolio, i.e., $P^*_{d_{half}} - P^*_{d_{full}}$. Here, the portfolio consists of all 17 classifiers.

The correlation plots show that for most classifiers the predicted improvement is correlated to the actual improvement. The points of the various classifiers are evenly spread around the diagonal, indicating that the individual learning curves of the various algorithms are predictable. This was also mentioned by various other authors, e.g., [7], [18]. The predictions per portfolio are often optimistic. The reason for this is that the portfolio always represents the best score at a given dataset size. When utilizing performance predictions, the highest prediction is always chosen, leading to a bias towards higher predictions.

The bottom row of Fig. 3 shows a more aggregated view of the same data. For different values of the tolerance parameter γ (x-axis), it shows the *portion* of points whose discrepancy between the predicted and the true value is at most γ; one could say that it shows the "accuracy" of the IPL-*regressor* with respect to tolerance γ. Formally, the y-axis show the percentage of cases where $|P^i_{d_{half}} - \hat{P}^i_{d_{full}}| \leq \gamma$ for the left plot and $|P^*_{d_{half}} - \hat{P}^*_{d_{full}}| \leq \gamma$ for the right plot. Relating this graph to the top row, it shows the relative frequency of points whose horizontal distance to the dashed line in the upper plot is at most γ. Note that these curves are by definition strictly increasing. The dashed horizontal line is a visual aid to indicate the 90% threshold.

The interpretation of the right plot under consideration is as follows. If we accept a mistake of up to 0.025 for the prediction of *how much* we can improve over the best learner at d_{half} when doubling the data, we will get a "correct" answer in roughly 50% of the cases. If we accept a mistake of 0.07 or more instead, the answer will be correct in over 90% of the cases.

With respect to the research question under consideration, namely, whether we can well predict the improvement achieved when doubling the data, the results are, depending on the expectations, a bit mediocre. On the positive side, the model is precise up to 0.01 on over 40% of the analyzed cases. On the downside, being precise up to 0.025 (which may be an acceptable range) in only 50% of the cases might seem an unacceptable gamble.

However, for the decision situation at hand, the above precision is unnecessarily strict. In fact, being able to accurately predict the learning curve value for data acquisition questions is only important if (i) we can control the exact number of instances to be gathered and (ii) indeed have a very detailed concept of utility, which is often not the case in practice. In our decision situation, we only need to know whether, based on the recommendation, we take the correct *decision* in accepting the cost for the additional batch of data or not. In other words, if the true improvement is much more than β and we predict just a value slightly above β, then we might make a rather big mistake from the regression point of view but *no* mistake from the classification point of view. The same holds for values below β. So one might argue that the prediction error is not so severe as long as we act in the correct way.

Inversely, great regression performance does not even necessarily imply substantially better decisions. If we make a mistake in the performance prediction, then the crucial question is whether this mistake will lead to a decision that is different from the decision one should take if the performance was *known*. But if the true performance improvement is very close to β, then even a very small mistake in the performance prediction can lead to a wrong decision. To summarize, both performances (classification and regression) are of interest and even though great regression results would be desirable, neither are they necessary nor sufficient for good classification performance.

Motivated by these observations, we can address the second research question with the plots in Fig. 4. This figure shows the accuracy of the IPL-*classifier* (y-axis) when the task is to predict whether the double amount of data will improve the performance at least by a certain percentage point β (x-axis). On the left, this is the percentage of cases where $(P^i_{d_{half}} - \hat{P}^i_{d_{full}} < \beta) \equiv (P^i_{d_{half}} - P^i_{d_{full}} < \beta)$ holds. In other words, the fraction of cases where the recommended action is indeed the one that maximizes the utility. In the right plot, this is done for the portfolio, i.e., the percentage of cases where $(P^*_{d_{half}} - \hat{P}^*_{d_{full}} < \beta) \equiv (P^*_{d_{half}} - P^*_{d_{full}} < \beta)$ holds. The dashed line is a visual aid for a score of 95%.

As can be seen, the results here are much more satisfying, specifically for the portfolio, which is of our interest. For *any* value of β, the accuracy is above 80%, *much* better than guessing. For most of the datasets, it is even above 90%. Note that, in contrast to the plots in Fig. 3, the curves in this figure are not monotone. The reason for this is that the response is likely to be correct if the required improvement β is very low (close to 0) or very high (close to 1). In the first case, always predicting "yes" will usually be correct while predicting "no" for high values of β is likewise a safe prediction. Hence, the difficult cases are those in which the true improvement is, incidentally, close to the required threshold β. In fact, in those cases, even a very small prediction error can cause a wrong binary prediction. For this reason, the accuracy drops a bit in the range of $\beta \in [0.05, 0.2]$; the reason why there are several and not just one valley in each curve is that the curves average over all the datasets.

To summarize, the empirical learning curves produced as a *side product* of LCCV, give great insights into the learning behaviour of the considered algorithms and can be an important resource for decision-making in data acquisition. Even though the potential improvements of a portfolio cannot always be predicted with high precision, the predictions are mostly still sufficiently exact to recommend correctly whether or not acquiring a fixed number of additional data points if the required improvement to obtain a positive net utility is defined.

VI. Conclusion

In this article, we presented LCCV, a technique to validate learning algorithms based on learning curves. In contrast to other evaluation methods that leverage learning curves, LCCV is designed to avoid pruning candidates that can potentially still outperform the current best candidate. Based on a convexity assumption, which turns out to hold almost always in practice, candidates are pruned once they are unlikely to improve upon the current best candidate.

This makes LCCV faster than vanilla cross-validation methods while usually delivering equally good results. We have empirically shown that LCCV outperforms both 5-CV and 10-CV in many cases in terms of the runtime of a model selection algorithm based on random search that employs these methods for validation. The median runtime reduction of LCCV compared to vanilla cross-validation is more than 50%, and for some datasets, a reduction of 85% is achieved, which corresponds to absolute reductions of 20 h (median) and 100 h (max) when compared to CV. We further compared LCCV to a Wilcoxon-based racing method as well as successive halving and found that all methods have their logical place on the Pareto front of runtime and performance. The greedy successive halving algorithm is usually even faster than LCCV, but LCCV is less prone to miss the optimal solution and can be used in *sequential* model selection, a strict superset of the situations to which successive halving can be applied in a straightforward way.

Moreover, the learning curves can be used to give additional information to the data owner. We showed that it gives good confidence recommendations ($> 80\%$ accuracy) of whether the acquisition of new data will enable a given desired reduction of the error rate.

Future work can focus on several refinements of the technique. First, it would be highly desirable to detect non-convexities to disable pruning in those cases. Second, it would be interesting to adopt a more dynamic and curve-dependent manner of choosing anchors instead of fixing these as constants before. Finally, it would be interesting to integrate LCCV in various AutoML toolboxes, such as Naive AutoML [26], ML-Plan [28], Auto-sklearn [9], [11], and Auto-Weka [38]. By utilizing iteration-based learning curves, it could even be integrated into deep learning toolboxes. Indeed, enabling these toolboxes to speed up the individual evaluations, has the potential to further push the state-of-the-art of machine learning research.

References

[1] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, *Learning From Data*, vol. 4. New York, NY, USA: AMLBook, 2012.

[2] B. Baker, O. Gupta, R. Raskar, and N. Naik, "Accelerating neural architecture search using performance prediction," in *Proc. 6th Int. Conf. Learn. Representations*, 2018.

[3] Y. Bard, *Nonlinear Parameter Estimation*. Cambridge, MA, USA: Academic Press, 1974.

[4] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, no. Feb., pp. 281–305, 2012.

[5] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, Curran Associates, Inc., 2011, pp. 2546–2554.

[6] P. Brazdil, J. N. van Rijn, C. Soares, and J. Vanschoren, *Metalearning: Applications to Automated Machine Learning and Data Mining*. Berlin, Germany: Springer, 2nd ed., 2022.

[7] T. Domhan, J. T. Springenberg, and F. Hutter, "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves," in *Proc. 24th Int. Joint Conf. Artif. Intell.*, 2015, pp. 3460–3468.

[8] M. Feurer and F. Hutter, "Towards further automation in AutoML," in *Proc. 5th ICML Workshop Autom. Mach. Learn.*, 2018.

[9] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 2962–2970.

[10] M. Feurer et al., "OpenML-Python: An extensible Python API for OpenML," *J. Mach. Learn. Res.*, vol. 22, no. 100, pp. 1–5, 2021.

[11] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter, "Auto-sklearn 2.0: Hands-free AutoML via meta-learning," *J. Mach. Learn. Res.*, vol. 23, no. 261, pp. 1–61, 2022.

[12] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Mach. Learn.*, vol. 63, no. 1, pp. 3–42, 2006.

[13] P. Gijsbers et al., "AMLB: An AutoML benchmark," 2022, *arXiv:2207.12560*.

[14] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: An update," *ACM SIGKDD Explorations*, vol. 11, pp. 10–18, 2009.

[15] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Proc. Conf. Learn. Intell. Optim.*, 2011, pp. 507–523.

[16] K. Jamieson and A. Talwalkar, "Non-stochastic best arm identification and hyperparameter optimization," in *Proc. 19th Int. Conf. Artif. Intell. Statist.*, 2016, pp. 240–248.

[17] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, "Fast Bayesian optimization of machine learning hyperparameters on large datasets," in *Proc. 20th Int. Conf. Artif. Intell. Statist.*, A. Singh and X. J. Zhu, Eds., PMLR, 2017, pp. 528–536.

[18] A. Klein, S. Falkner, J. T. Springenberg, and F. Hutter, "Learning curve prediction with Bayesian neural networks," in *Proc. Int. Conf. Learn. Representations*, 2017.

[19] M. Last, "Improving data mining utility with projective sampling," in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, Paris, France, 2009, pp. 487–496.

[20] R. Leite and P. Brazdil, "Active testing strategy to predict the best classification algorithm via sampling and metalearning," in *Proc. 19th Eur. Conf. Artif. Intell.*, 2010, pp. 309–314.

[21] L. Li, K. G. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *J. Mach. Learn. Res.*, vol. 18, pp. 185:1–185:52, 2017.

[22] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," *Operations Res. Perspectives*, vol. 3, pp. 43–58, 2016.

[23] I. Loshchilov and F. Hutter, "CMA-ES for hyperparameter optimization of deep neural networks," 2016, *arXiv:1604.07269*.

[24] F. Mohr and J. N. van Rijn, "Towards model selection using learning curve cross-validation," in *Proc. 8th ICML Workshop Autom. Mach. Learn.*, 2021.

[25] F. Mohr and J. N. van Rijn, "Learning curves for decision making in supervised machine learning–A survey," 2022, *arXiv:2201.12150*.

[26] F. Mohr and M. Wever, "Replacing the ex-def baseline in automl by naive AutoML," in *Proc. 8th ICML Workshop Automat. Mach. Learn. (AutoML)* 2021.

[27] F. Mohr and M. Wever, "Naive automated machine learning," *Mach. Learn.*, vol. 112, no. 4,pp. 1131–1170, 2022.

[28] F. Mohr, M. Wever, and E. Hüllermeier, "ML-Plan: Automated machine learning via hierarchical planning," *Mach. Learn.*, vol. 107, no. 8, pp. 1495–1515, 2018.

[29] F. Mohr, M. Wever, A. Tornede, and E. Hüllermeier, "Predicting machine learning pipeline runtimes in the context of automated machine learning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 9, pp. 3055–3066, Sep. 2021.

[30] F. Mohr, T. J. Viering, M. Loog, and J. N. van Rijn, "LCDB 1.0: A first learning curves database for classification tasks," in *Proc. Eur. Conf. Mach. Learn. Knowl. Discov. Databases*, Springer, 2022, pp. 3–19.

[31] R. S. Olson, N. Bartley, R. J. Urbanowicz, and J. H. Moore, "Evaluation of a tree-based pipeline optimization tool for automating data science," in *Proc. ACM Genet. Evol. Comput. Conf.*, 2016, pp. 485–492.

[32] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[33] J. Petrak, "Fast subsampling performance estimates for classification algorithm selection," in *Proc. ECML Workshop Meta- Learn.*, 2000.

[34] F. Pinto, C. Soares, and J. Mendes-Moreira, "Towards automatic generation of metafeatures," in *Proc. Pacific-Asia Conf. Knowl. Discov. Data Mining*, Springer, 2016, pp. 215–226.

[35] F. Provost, D. Jensen, and T. Oates, "Efficient progressive sampling," in *Proc. 5th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 1999, pp. 23–32.

[36] A. Sabharwal, H. Samulowitz, and G. Tesauro, "Selecting near-optimal learners via incremental data allocation," in *Proc. AAAI Conf. Artif. Intell.*, 2016, pp. 2007–2015.

[37] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Proc. Adv. Neural Inf. Process. Syst.*, Curran Associates, Inc., 2012, pp. 2951–2959.

[38] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2013, pp. 847–855.

[39] J. N. van Rijn, S. M. Abdulrahman, P. Brazdil, and J. Vanschoren, "Fast algorithm selection using learning curves," in *Proc. Int. Symp. Intell. Data Anal.*, Springer, 2015, pp. 298–309.

[40] J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo, "OpenML: Networked science in machine learning," *SIGKDD Explorations*, vol. 15, no. 2, pp. 49–60, 2013.

[41] T. J. Viering and M. Loog, "The shape of learning curves: A review," 2021, *arXiv:2103.10948*.

[42] D. Wang, J. Andres, J. Weisz, E. Oduor, and C. Dugan, "AutoDS: Towards human-centered automation of data science," 2021, *arXiv:2101.05273*.

[43] G. M. Weiss and Y. Tian, "Maximizing classifier utility when there are data acquisition and modeling costs," *Data Mining Knowl. Discov.*, vol. 17, no. 2, pp. 253–282, 2008.

Felix Mohr received the PhD degree from Paderborn University, Germany, in 2016. He is professor with the Faculty of Engineering, Universidad de la Sabana in Colombia. His research focus lies in the areas of stochastic tree search as well as automated software configuration with a particular specialization on automated machine learning.

Jan N. van Rijn is an assistant professor with the Leiden Institute of Advanced Computer Science (LIACS), Leiden University (The Netherlands). During his PhD, he (co-)founded and developed OpenML, an open science platform for machine learning. After obtaining his PhD, he worked as a postdoctoral researcher with the Machine Learning Lab, University of Freiburg (Germany), and Columbia University in the City of New York. His research interests include artificial intelligence, AutoML, and metalearning.