Oboe: Collaborative Filtering for AutoML Model Selection

Chengrun Yang, Yuji Akimoto, Dae Won Kim, Madeleine Udell
Cornell University
{cy438,ya242,dk444,udell}@cornell.edu

ABSTRACT

Algorithm selection and hyperparameter tuning remain two of the most challenging tasks in machine learning. Automated machine learning (AutoML) seeks to automate these tasks to enable widespread use of machine learning by non-experts. This paper introduces Oboe, a collaborative filtering method for time-constrained model selection and hyperparameter tuning. Oboe forms a matrix of the cross-validated errors of a large number of supervised learning models (algorithms together with hyperparameters) on a large number of datasets, and fits a low rank model to learn the low-dimensional feature vectors for the models and datasets that best predict the cross-validated errors. To find promising models for a new dataset, Oboe runs a set of fast but informative algorithms on the new dataset and uses their cross-validated errors to infer the feature vector for the new dataset. Oboe can find good models under constraints on the number of models fit or the total time budget. To this end, this paper develops a new heuristic for active learning in time-constrained matrix completion based on optimal experiment design. Our experiments demonstrate that Oboe delivers state-of-the-art performance faster than competing approaches on a test bed of supervised learning problems. Moreover, the success of the bilinear model used by Oboe suggests that AutoML may be simpler than was previously understood.

CCS CONCEPTS

• **Computing methodologies** → **Discrete space search**; **Continuous space search**; **Online learning settings**; **Active learning settings**; **Principal component analysis**; *Search with partial observations.*

KEYWORDS

AutoML, meta-learning, time-constrained, model selection, collaborative filtering

ACM Reference Format:
Chengrun Yang, Yuji Akimoto, Dae Won Kim, Madeleine Udell. 2019. Oboe: Collaborative Filtering for AutoML Model Selection. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3292500.3330909

KDD '19, August 4–8, 2019, Anchorage, AK, USA

ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330909

1 INTRODUCTION

It is often difficult to find the best algorithm and hyperparameter settings for a new dataset, even for experts in machine learning or data science. The large number of machine learning algorithms and their sensitivity to hyperparameter values make it practically infeasible to enumerate all configurations. Automated machine learning (AutoML) seeks to efficiently automate the selection of model (e.g., [8, 12, 14]) or pipeline (e.g., [11]) configurations, and has become more important as the number of machine learning applications increases.

We propose an algorithmic system, Oboe [1], that provides an initial tuning for AutoML: it selects a good algorithm and hyperparameter combination from a discrete set of options. The resulting model can be used directly, or the hyperparameters can be tuned further. Briefly, Oboe operates as follows.

During an offline training phase, it forms a matrix of the cross-validated errors of a large number of supervised-learning models (algorithms together with hyperparameters) on a large number of datasets. It then fits a low rank model to this matrix to learn latent low-dimensional meta-features for the models and datasets. Our optimization procedure ensures these latent meta-features best predict the cross-validated errors, among all bilinear models.

To find promising models for a new dataset, Oboe chooses a set of fast but informative models to run on the new dataset and uses their cross-validated errors to infer the latent meta-features of the new dataset. Given more time, Oboe repeats this procedure using a higher rank to find higher-dimensional (and more expressive) latent features. Using a low rank model for the error matrix is a very strong structural prior.

This system addresses two important problems: 1) *Time-constrained initialization*: how to choose a promising initial model under time constraints. Oboe adapts easily to short times by using a very low rank and by restricting its experiments to models that will run very fast on the new dataset. 2) *Active learning*: how to improve on the initial guess given further computational resources. Oboe uses extra time by allowing higher ranks and more expensive computational experiments, accumulating its knowledge of the new dataset to produce more accurate (and higher-dimensional) estimates of its latent meta-features.

Oboe uses collaborative filtering for AutoML, selecting models that have worked well on similar datasets, as have many previous methods including [1, 9, 12, 28, 38, 45]. In collaborative filtering, the critical question is how to characterize dataset similarity so that training datasets "similar" to the test dataset faithfully predict model performance. One line of work uses dataset meta-features — simple, statistical or landmarking metrics — to characterize datasets [9, 12–14, 31]. Other approaches (e.g., [43]) avoid meta-features. Our approach builds on both of these lines of work. Oboe relies

[1] The eponymous musical instrument plays the initial note to tune an orchestra.

on model performance to characterize datasets, and the low rank representations it learns for each dataset may be seen (and used) as latent meta-features. Compared to AutoML systems that compute meta-features of the dataset before running any models, the flow of information in Oboe is exactly opposite: Oboe uses only the performance of various models on the datasets to compute lower dimensional latent meta-features for models and datasets.

The active learning subproblem is to gain the most information to guide further model selection. Some approaches choose a function class to capture the dependence of model performance on hyperparameters; examples are Gaussian processes [3, 14, 17, 27, 33, 34, 36, 37], sparse Boolean functions [16] and decision trees [2, 20]. Oboe chooses the set of bilinear models as its function class: predicted performance is linear in each of the latent model and dataset meta-features.

Bilinearity seems like a rather strong assumption, but confers several advantages. Computations are fast and easy: we can find the global minimizer by PCA, and can infer the latent meta-features for a new dataset using least squares. Moreover, recent theoretical work suggests that this model class is more general than it appears: roughly, and under a few mild technical assumptions, any $m \times n$ matrix with independent rows and columns whose entries are generated according to a fixed function (here, the function computed by training the model on the dataset) has an approximate rank that grows as $\log(m+n)$ [40]. Hence large data matrices tend to look low rank.

Originally, the authors conceived of Oboe as a system to produce a good set of initial models, to be refined by other local search methods, such as Bayesian optimization. However, in our experiments, we find that Oboe's performance, refined by fitting models of ever higher rank with ever more data, actually improves faster than competing methods that use local search methods more heavily.

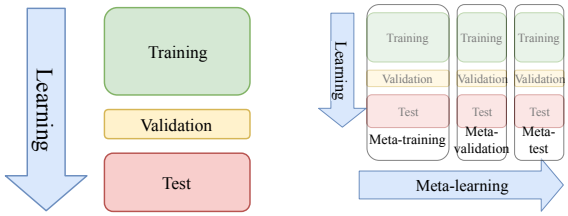
One key component of our system is the prediction of model runtime on new datasets. Many authors have previously studied algorithm runtime prediction using a variety dataset features [21], via ridge regression [18], neural networks [35], Gaussian processes [19], and more. Several measures have been proposed to trade-off between accuracy and runtime [4, 25]. We predict algorithm runtime using only the number of samples and features in the dataset. This model is particularly simple but surprisingly effective.

Classical experiment design (ED) [5, 22, 29, 32, 42] selects features to observe to minimize the variance of the parameter estimate, assuming that features depend on the parameters according to known, linear, functions. Oboe's bilinear model fits this paradigm, and so ED can be used to select informative models. Budget constraints can be added, as we do here, to select a small number of promising machine learning models or a set predicted to finish within a short time budget [24, 46].

This paper is organized as follows. Section 2 introduces notation and terminology. Section 3 describes the main ideas we use in Oboe. Section 4 presents Oboe in detail. Section 5 shows experiments.

2 NOTATION AND TERMINOLOGY

Meta-learning. Meta-learning is the process of learning across individual datasets or problems, which are subsystems on which



(a) Learning (b) Meta-learning

Figure 1: Standard vs meta-learning.

standard learning is performed [26]. Just as standard machine learning must avoid overfitting, experiments testing AutoML systems must avoid meta-overfitting! We divide our set of datasets into meta-training, meta-validation and meta-test sets, and report results on the meta-test set. Each of the three phases in meta-learning — meta-training, meta-validation and meta-test — is a standard learning process that includes training, validation and test.

Indexing. Throughout this paper, all vectors are column vectors. Given a matrix $A \in \mathbb{R}^{m\times n}$, $A_{i,:}$ and $A_{:,j}$ denote the ith row and jth column of A, respectively. i is the index over datasets, and j is the index over models. We define $[n] = \{1, \ldots, n\}$ for $n \in \mathbb{Z}$. Given an ordered set $\mathcal{S} = \{s_1, \ldots, s_k\}$ where $s_1 < \ldots < s_k \in [n]$, we write $A_{:\mathcal{S}} = \begin{bmatrix} A_{:,s_1} & A_{:,s_2} & \cdots & A_{:,s_k} \end{bmatrix}$.

Algorithm performance. A *model* $\mathcal{A}$ is a specific algorithm-hyperparameter combination, e.g. k-NN with $k = 3$. We denote by $\mathcal{A}(\mathcal{D})$ the expected cross-validation error of model $\mathcal{A}$ on dataset $\mathcal{D}$, where the expectation is with respect to the cross-validation splits. We refer to the model in our collection that achieves minimal error on $\mathcal{D}$ as the *best model* for $\mathcal{D}$. A model $\mathcal{A}$ is said to be *observed* on $\mathcal{D}$ if we have calculated $\mathcal{A}(\mathcal{D})$ by fitting (and cross-validating) the model. The *performance vector* e of a dataset $\mathcal{D}$ concatenates $\mathcal{A}(\mathcal{D})$ for each *model* $\mathcal{A}$ in our collection.

Meta-features. We discuss two types of meta-features in this paper. *Meta-features* refer to metrics used to characterize datasets or models. For example, the number of data points or the performance of simple models on a dataset can serve as meta-features of the dataset. As an example, we list the meta-features used in the AutoML framework auto-sklearn in Appendix B, Table 3. In constrast to standard meta-features, we use the term *latent meta-features* to refer to characterizations learned from matrix factorization.

Parametric hierarchy. We distinguish between three kinds of parameters:

- *Parameters* of a model (e.g., the splits in a decision tree) are obtained by training the model.
- *Hyperparameters* of an algorithm (e.g., the maximum depth of a decision tree) govern the training procedure. We use the word *model* to refer to an algorithm together with a particular choice of hyperparameters.
- *Hyper-hyperparameters* of a meta-learning method (e.g., the total time budget for Oboe) govern meta-training.

Time target and time budget. The time target refers to the anticipated time spent running models to infer latent features of each fixed dimension and can be exceeded. However, the runtime does not usually deviate much from the target since our model runtime

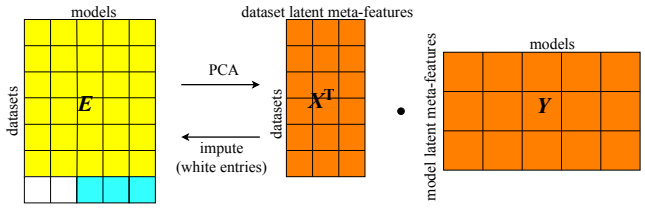


Figure 2: Illustration of model performance prediction via the error matrix E (yellow blocks only). Perform PCA on the error matrix (offline) to compute dataset (X) and model (Y) latent meta-features (orange blocks). Given a new dataset (row with white and blue blocks), pick a subset of models to observe (blue blocks). Use Y together with the observed models to impute the performance of the unobserved models on the new dataset (white blocks).

prediction works well. The time budget refers to the total time limit for Oboe and is never exceeded.

Midsize OpenML and UCI datasets. Our experiments use OpenML [41] and UCI [10] classification datasets with between 150 and 10,000 data points and with no missing entries.

3 METHODOLOGY

3.1 Model Performance Prediction

It can be difficult to determine *a priori* which meta-features to use so that algorithms perform similarly well on datasets with similar meta-features. Also, the computation of meta-features can be expensive (see Appendix C, Figure 11). To infer model performance on a dataset without any expensive meta-feature calculations, we use collaborative filtering to infer latent meta-features for datasets.

As shown in Figure 2, we construct an empirical error matrix $E \in \mathbb{R}^{m \times n}$, where every entry E_{ij} records the cross-validated error of model j on dataset i. Empirically, E has approximately low rank: Figure 3 shows the singular values $\sigma_i(E)$ decay rapidly as a function of the index i. This observation serves as foundation of our algorithm, and will be analyzed in greater detail in Section 5.2. The value E_{ij} provides a noisy but unbiased estimate of the true performance of a model on the dataset: $\mathbb{E}E_{ij} = \mathcal{A}_j(\mathcal{D}_i)$.

To denoise this estimate, we approximate $E_{ij} \approx x_i^\top y_j$ where x_i and y_j minimize $\sum_{i=1}^m \sum_{j=1}^n (E_{ij} - x_i^\top y_j)^2$ with $x_i, y_j \in \mathbb{R}^k$ for $i \in [M]$ and $j \in [N]$; the solution is given by PCA. Thus x_i and y_j are the latent meta-features of dataset i and model j, respectively. The rank k controls model fidelity: small ks give coarse approximations, while large ks may overfit. We use a doubling scheme to choose k within time budget; see Section 4.2 for details.

Given a new meta-test dataset, we choose a subset $\mathcal{S} \subseteq [N]$ of models and observe performance e_j of model j for each $j \in \mathcal{S}$. A good choice of S balances information gain against time needed to run the models; we discuss how to choose S in Section 3.3. We then infer latent meta-features for the new dataset by solving the least squares problem: minimize $\sum_{j \in \mathcal{S}} (e_j - \hat{x}^\top y_j)^2$ with $\hat{x} \in \mathbb{R}^k$. For all unobserved models, we predict their performance as $\hat{e}_j = \hat{x}^\top y_j$ for $j \notin \mathcal{S}$.

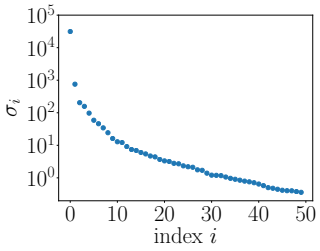


Figure 3: Singular value decay of an error matrix. The entries are calculated by 5-fold cross validation of machine models (listed in Appendix A, Table 2) on midsize OpenML datasets.

3.2 Runtime Prediction

Estimating model runtime allows us to trade off between running slow, informative models and fast, less informative models. We use a simple method to estimate runtimes, using polynomial regression on $n^{\mathcal{D}}$ and $p^{\mathcal{D}}$, the numbers of data points and features in $\mathcal{D}$, and their logarithms, since the theoretical complexities of machine learning algorithms we use are $O\big((n^{\mathcal{D}})^3, (p^{\mathcal{D}})^3, (\log(n^{\mathcal{D}}))^3\big)$. Hence we fit an independent polynomial regression model for each model:

$$f_j = \text{argmin}_{f_j \in \mathcal{F}} \sum_{i=1}^{M} \left(f_j(n^{\mathcal{D}_i}, p^{\mathcal{D}_i}, \log(n^{\mathcal{D}_i})) - t_j^{\mathcal{D}_i} \right)^2, \; j \in [n]$$

where $t_j^{\mathcal{D}}$ is the runtime of machine learning model j on dataset $\mathcal{D}$, and $\mathcal{F}$ is the set of all polynomials of order no more than 3. We denote this procedure by $f_j = \texttt{fit\_runtime}(n, p, t)$.

We observe that this model predicts runtime within a factor of two for half of the machine learning models on more than 75% midsize OpenML datasets, and within a factor of four for nearly all models, as shown in Section 5.2 and visualized in Figure 7.

3.3 Time-Constrained Information Gathering

To select a subset $\mathcal{S}$ of models to observe, we adopt an approach that builds on classical experiment design: we suppose fitting each machine learning model $j \in [n]$ returns a linear measurement $x^T y_j$ of x, corrupted by Gaussian noise. To estimate x, we would like to choose a set of observations y_j that span $\mathbb{R}^k$ and form a well-conditioned submatrix, but that corresponds to models which are fast to run. In passing, we note that the pivoted QR algorithm on the matrix Y (heuristically) finds a well conditioned set of k columns of Y. However, we would like to find a method that is runtime-aware.

Our experiment design (ED) procedure minimizes a scalarization of the covariance of the estimated meta-features $\hat{x}$ of the new dataset subject to runtime constraints [5, 22, 29, 32, 42]. Formally, define an indicator vector $v \in \{0,1\}^n$, where entry v_j indicates whether to fit model j. Let $\hat{t}_j$ denote the predicted runtime of model j on a meta-test dataset, and let y_j denote its latent meta-features, for $j \in [n]$. Now relax to allow $v \in [0,1]^n$ to allow for non-Boolean values and solve the optimization problem

$$\begin{aligned} \text{minimize} \quad & \log\det\left(\textstyle\sum_{j=1}^n v_j y_j y_j^\top\right)^{-1} \\ \text{subject to} \quad & \textstyle\sum_{j=1}^n v_j \hat{t}_j \le \tau \\ & v_j \in [0,1], \forall j \in [n] \end{aligned} \tag{1}$$

with variable $v \in \mathbb{R}^n$. We call this method ED (time). Scalarizing the covariance by minimizing the determinant is called D-optimal design. Several other scalarizations can also be used, including covariance norm (E-optimal) or trace (A-optimal). Replacing t_i by 1 gives an alternative heuristic that bounds the *number* of models fit by τ; we call this method ED (number).

Problem 1 is a convex optimization problem, and we obtain an approximate solution by rounding the largest entries of v up to 1 until the selected models exceed the time limit τ. Let $\mathcal{S} \subseteq [n]$ be the set of indices of e that we choose to observe, i.e. the set such that v_s rounds to 1 for $s \in \mathcal{S}$. We denote this process by $\mathcal{S} = \texttt{min\_variance\_ED}(\hat{t}, \{y_j\}_{j=1}^n, \tau)$.

4 THE OBOE SYSTEM

Shown in Figure 4, the Oboe system can be divided into offline and online stages. The offline stage is executed only once and explores the space of model performance on meta-training datasets. Time taken on this stage does not affect the runtime of Oboe on a new dataset; the runtime experienced by user is that of the online stage.

One advantage of Oboe is that the vast majority of the time in the online phase is spent training standard machine learning models, while very little time is required to decide which models to sample. Training these standard machine learning models requires running algorithms on datasets with thousands of data points and features, while the meta-learning task — deciding which models to sample — requires only solving a small least-squares problem.

4.1 Offline Stage

The (i, j)th entry of error matrix $E \in \mathbb{R}^{m \times n}$, denoted as E_{ij}, records the performance of the jth model on the ith meta-training dataset. We generate the error matrix using the *balanced error rate* metric, the average of false positive and false negative rates across different classes. At the same time we record runtime of machine learning models on datasets. This is used to fit runtime predictors described in Section 3. Pseudocode for the offline stage is shown as Algorithm 1.

Algorithm 1 Offline Stage

Require: meta-training datasets $\{\mathcal{D}_i\}_{i=1}^m$, models $\{\mathcal{A}_j\}_{j=1}^n$, algorithm performance metric $\mathcal{M}$
Ensure: error matrix E, runtime matrix T, fitted runtime predictors $\{f_j\}_{j=1}^n$

for $i = 1, 2, \ldots, m$ **do**
  $n^{\mathcal{D}_i}, p^{\mathcal{D}_i} \leftarrow$ number of data points and features in $\mathcal{D}_i$
  for $j = 1, 2, \ldots, n$ **do**
    $E_{ij} \leftarrow$ error of model $\mathcal{A}_j$ on dataset $\mathcal{D}_i$ according to metric $\mathcal{M}$
    $T_{ij} \leftarrow$ observed runtime for model $\mathcal{A}_j$ on dataset $\mathcal{D}_i$
  end for
end for
for $j = 1, 2, \ldots, n$ **do**
  fit $f_j = \texttt{fit\_runtime}(n, p, T_j)$
end for

4.2 Online Stage

Recall that we repeatly double the time target of each round until we use up the total time budget. Thus each round is a subroutine of the entire online stage and is shown as Algorithm 2, `fit_one_round`.

- **Time-constrained model selection (`fit_one_round`)** Our active learning procedure selects a fast and informative collection of models to run on the meta-test dataset. Oboe uses the results of these fits to estimate the performance of all other models as accurately as possible. The procedure is as follows. First predict model runtime on the meta-test dataset using fitted runtime predictors. Then use experiment design to select a subset $\mathcal{S}$ of entries of e, the performance vector of the test dataset, to observe. The observed entries are used to compute $\hat{x}$, an estimate of the latent meta-features of the test dataset, which in turn is used to predict every entry of e. We build an ensemble out of models predicted to perform well within the time target $\tilde{\tau}$ by means of greedy forward selection [6, 7]. We denote this subroutine as $\tilde{A} = \texttt{ensemble\_selection}(\mathcal{S}, e_{\mathcal{S}}, z_{\mathcal{S}})$, which takes as input the set of base learners $\mathcal{S}$ with their cross-validation errors $e_{\mathcal{S}}$ and predicted labels $z_{\mathcal{S}} = \{z_s | s \in \mathcal{S}\}$, and outputs ensemble learner $\tilde{A}$. The hyperparameters used by models in the ensemble can be tuned further, but in our experiments we did not observe substantial improvements from further hyperparameter tuning.

Algorithm 2 $\texttt{fit\_one\_round}(\{y_j\}_{j=1}^n, \{f_j\}_{j=1}^n, \mathcal{D}_{tr}, \tilde{\tau})$

Require: model latent meta-features $\{y_j\}_{j=1}^n$, fitted runtime predictors $\{f_j\}_{j=1}^n$, training fold of the meta-test dataset $\mathcal{D}_{\text{tr}}$, number of best models N to select from the estimated performance vector, time target for this round $\tilde{\tau}$
Ensure: ensemble learner $\tilde{A}$

for $j = 1, 2, \ldots, n$ **do**
  $\hat{t}_j \leftarrow f_j(n^{\mathcal{D}_{\text{tr}}}, p^{\mathcal{D}_{\text{tr}}})$
end for
$\mathcal{S} = \texttt{min\_variance\_ED}(\hat{t}, \{y_j\}_{j=1}^n, \tilde{\tau})$
for $k = 1, 2, \ldots, |\mathcal{S}|$ **do**
  $e_{\mathcal{S}_k} \leftarrow$ cross-validation error of model $\mathcal{A}_{\mathcal{S}_k}$ on $\mathcal{D}_{\text{tr}}$
end for
$\hat{x} \leftarrow (\begin{bmatrix} y_{\mathcal{S}_1} & y_{\mathcal{S}_2} & \cdots & y_{\mathcal{S}_{|\mathcal{S}|}} \end{bmatrix}^\top)^\dagger e_{\mathcal{S}}$
$\hat{e} \leftarrow \begin{bmatrix} y_1 & y_2 & \cdots & y_n \end{bmatrix}^\top \hat{x}$
$\mathcal{T} \leftarrow$ the N models with lowest predicted errors in $\hat{e}$
for $k = 1, 2, \ldots, |\mathcal{T}|$ **do**
  $e_{\mathcal{T}_k}, z_{\mathcal{T}_k} \leftarrow$ cross-validation error of model $\mathcal{A}_{\mathcal{T}_k}$ on $\mathcal{D}_{\text{tr}}$
end for
$\tilde{A} \leftarrow \texttt{ensemble\_selection}(\mathcal{T}, e_{\mathcal{T}}, z_{\mathcal{T}})$

- **Time target doubling** To select rank k, Oboe starts with a small initial rank along with a small time target, and then doubles the time target for `fit_one_round` until the elapsed time reaches half of the total budget. The rank k increments by 1 if the validation error of the ensemble learner decreases after doubling the time target, and otherwise does not change. Since the matrices returned by PCA with rank k are submatrices of those returned by PCA with rank l for $l > k$, we can compute the factors as submatrices of the m-by-n matrices returned by PCA with full rank $\min(m, n)$ [15]. The pseudocode is shown as Algorithm 3.

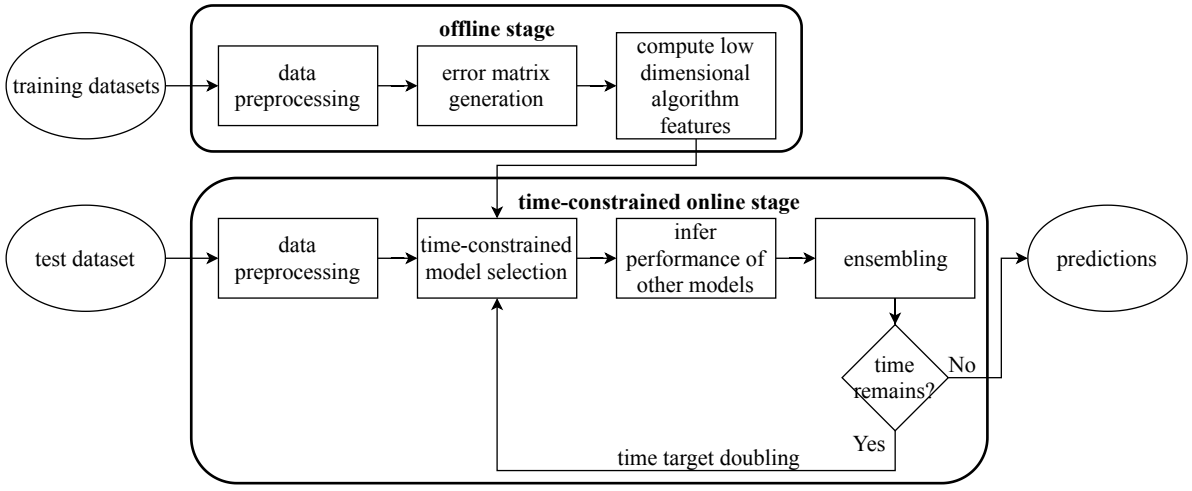


Figure 4: Diagram of data processing flow in the Oboe system.

Algorithm 3 Online Stage

Require: error matrix E, runtime matrix T, meta-test dataset $\mathcal{D}$, total time budget τ, fitted runtime predictors $\{f_j\}_{j=1}^n$, initial time target $\tilde{\tau}_0$, initial approximate rank k_0

Ensure: ensemble learner $\tilde{A}$

$x_i, y_j \leftarrow \arg\min \sum_{i=1}^m \sum_{j=1}^n (E_{ij} - x_i^\top y_j)^2$, $x_i \in \mathbb{R}^{\min(m,n)}$ for $i \in [M]$, $y_j \in \mathbb{R}^{\min(m,n)}$ for $j \in [N]$
$\mathcal{D}_{\text{tr}}, \mathcal{D}_{\text{val}}, \mathcal{D}_{\text{te}} \leftarrow$ training, validation and test folds of $\mathcal{D}$
$\tilde{\tau} \leftarrow \tilde{\tau}_0$
$k \leftarrow k_0$
while $\tilde{\tau} \leq \tau/2$ **do**
  $\{\tilde{y}_j\}_{j=1}^n \leftarrow k$-dimensional subvectors of $\{y_j\}_{j=1}^n$
  $\tilde{A} \leftarrow$ `fit_one_round`$(\{\tilde{y}_j\}_{j=1}^n, \{f_j\}_{j=1}^n, \mathcal{D}_{\text{tr}}, \tilde{\tau})$
  $e'_{\tilde{A}} \leftarrow \tilde{A}(\mathcal{D}_{\text{val}})$
  if $e'_{\tilde{A}} < e_{\tilde{A}}$ **then**
    $k \leftarrow k + 1$
  end if
  $\tilde{\tau} \leftarrow 2\tilde{\tau}$
  $e_{\tilde{A}} \leftarrow e'_{\tilde{A}}$
end while

5 EXPERIMENTAL EVALUATION

We ran all experiments on a server with 128 Intel® Xeon® E7-4850 v4 2.10GHz CPU cores. The process of running each system on a specific dataset is limited to a single CPU core. Code for the Oboe system is at https://github.com/udellgroup/oboe; code for experiments is at https://github.com/udellgroup/oboe-testing.

We test different AutoML systems on midsize OpenML and UCI datasets, using standard machine learning models shown in Appendix A, Table 2. Since data pre-processing is not our focus, we pre-process all datasets in the same way: one-hot encode categorical features and then standardize all features to have zero mean and unit variance. These pre-processed datasets are used in all the experiments.

5.1 Performance Comparison across AutoML Systems

We compare AutoML systems that are able to select among different algorithm types under time constraints: Oboe (with error matrix generated from midsize OpenML datasets), auto-sklearn [12], probabilistic matrix factorization (PMF) [14], and a *time-constrained* random baseline. The time-constrained random baseline selects models to observe randomly from those predicted to take less time than the remaining time budget until the time limit is reached.

5.1.1 Comparison with PMF. PMF and Oboe differ in the surrogate models they use to explore the model space: PMF incrementally picks models to observe using Bayesian optimization, with model latent meta-features from probabilistic matrix factorization as features, while Oboe models algorithm performance as bilinear in model and dataset meta-features.

PMF does not limit runtime, hence we compare it to Oboe using either QR or ED (number) to decide the set S of models (see Section 3.3). Figure 5 compares the performance of PMF and Oboe (using QR and ED (number) to decide the set S of models) on our collected error matrix to see which is best able to predict the smallest entry in each row. We show the regret: the difference between the minimal entry in each row and the one found by the AutoML method. In PMF, $N_0 = 5$ models are chosen from the best algorithms on similar datasets (according to dataset meta-features shown in Appendix B, Table 3) are used to warm-start Bayesian optimization, which then searches for the next model to observe. Oboe does not require this initial information before beginning its exploration. However, for a fair comparison, we show both "warm" and "cold" versions. The warm version observes both the models chosen by meta-features and those chosen by QR or ED; the number of observed entries in Figure 5 is the sum of all observed models. The cold version starts from scratch and only observes models chosen by QR and ED.

(Standard ED also performs well; see Appendix D, Figure 12.)

Figure 5 shows the surprising effectiveness of the low rank model used by Oboe:

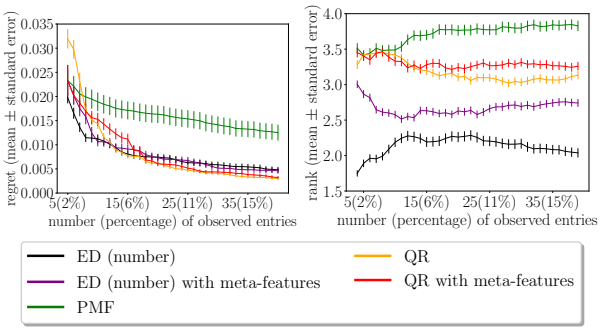


Figure 5: Comparison of sampling schemes (QR or ED) in Oboe and PMF. "QR" denotes QR decomposition with column pivoting; "ED (number)" denotes experiment design with number of observed entries constrained. The left plot shows the regret of each AutoML method as a function of number of entries; the right shows the relative rank of each AutoML method in the regret plot (1 is best and 5 is worst).

1 Meta-features are of marginal value in choosing new models to observe. For QR, using models chosen by meta-features helps when the number of observed entries is small. For ED, there is no benefit to using models chosen by meta-features.
2 The low rank structure used by QR and ED seems to provide a better guide to which models will be informative than the Gaussian process prior used by PMF: the regret of PMF does not decrease as fast as Oboe using either QR or ED.

5.1.2 Comparison with auto-sklearn. The comparison with PMF assumes we can use the labels for every point in the entire dataset for model selection, so we can compare the performance of every model selected and pick the one with lowest error. In contrast, our comparison with auto-sklearn takes place in a more challenging, realistic setting: when doing cross-validation on the meta-test dataset, we do not know the labels of the validation fold until we evaluate performance of the ensemble we built within time constraints on the training fold.

Figure 6 shows the error rate and ranking of each AutoML method as the runtime repeatedly doubles. Again, Oboe's simple bilinear model performs surprisingly well'[2]:

1 Oboe on average performs as well as or better than auto-sklearn (Figures 6c and 6d).
2 The quality of the initial models computed by Oboe and by auto-sklearn are comparable, but Oboe computes its first nontrivial model more than 8× faster than auto-sklearn (Figures 6a and 6b). In contrast, auto-sklearn must first compute meta-features for each dataset, which requires substantial computational time, as shown in Appendix C, Figure 11.
3 Interestingly, the rate at which the Oboe models improves with time is also faster than that of auto-sklearn: the improvement Oboe makes before 16s matches that of auto-sklearn from 16s to 64s. This indicates that the large time budget may be better spent in fitting more models than optimizing over hyperparameters, to which auto-sklearn devotes the remaining time.
4 Experiment design leads to better results than random selection in almost all cases.

[2] Auto-sklearn's GitHub Issue #537 says "Do not start auto-sklearn for time limits less than 60s". These plots should not be taken as criticisms of auto-sklearn, but are used to demonstrate Oboe's ability to select a model within a short time.

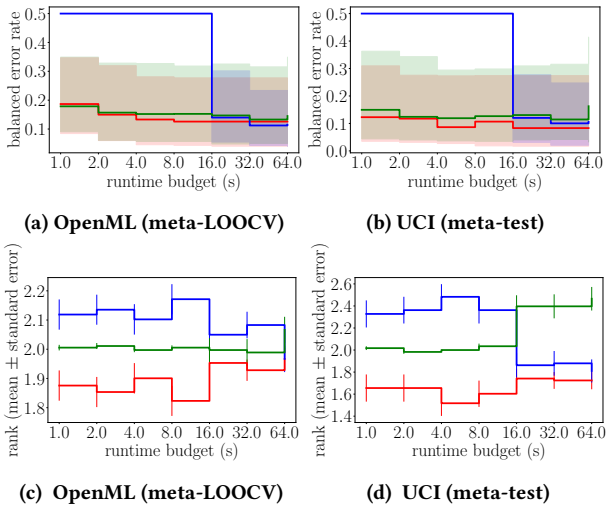


Figure 6: Comparison of AutoML systems in a time-constrained setting, including Oboe with experiment design (red), auto-sklearn (blue), and Oboe with time-constrained random initializations (green). OpenML and UCI denote midsize OpenML and UCI datasets. "meta-LOOCV" denotes leave-one-out cross-validation across datasets. In 6a and 6b, solid lines represent medians; shaded areas with corresponding colors represent the regions between 75th and 25th percentiles. Until the first time the system can produce a model, we classify every data point with the most common class label. Figures 6c and 6d show system rankings (1 is best and 3 is worst).

5.2 Why does Oboe Work?

Oboe performs well in comparison with other AutoML methods despite making a rather strong assumption about the structure of model performance across datasets: namely, bilinearity. It also requires effective predictions for model runtime. In this section, we perform additional experiments on components of the Oboe system to elucidate why the method works, whether our assumptions are warranted, and how they depend on detailed modeling choices.

Low rank under different metrics. Oboe uses balanced error rate to construct the error matrix, and works on the premise that the error matrix can be approximated by a low rank matrix. However, there is nothing special about the balanced error rate metric: most metrics result in an approximately low rank error matrix. For example, when using the AUC metric to measure error, the 418-by-219 error matrix from midsize OpenML datasets has only 38 eigenvalues greater than 1% of the largest, and 12 greater than 3%.

(Nonnegative) low rank structure of the error matrix. The features computed by PCA are dense and in general difficult to interpret. In contrast, nonnegative matrix factorization (NMF) produces sparse positive feature vectors and is thus widely used for

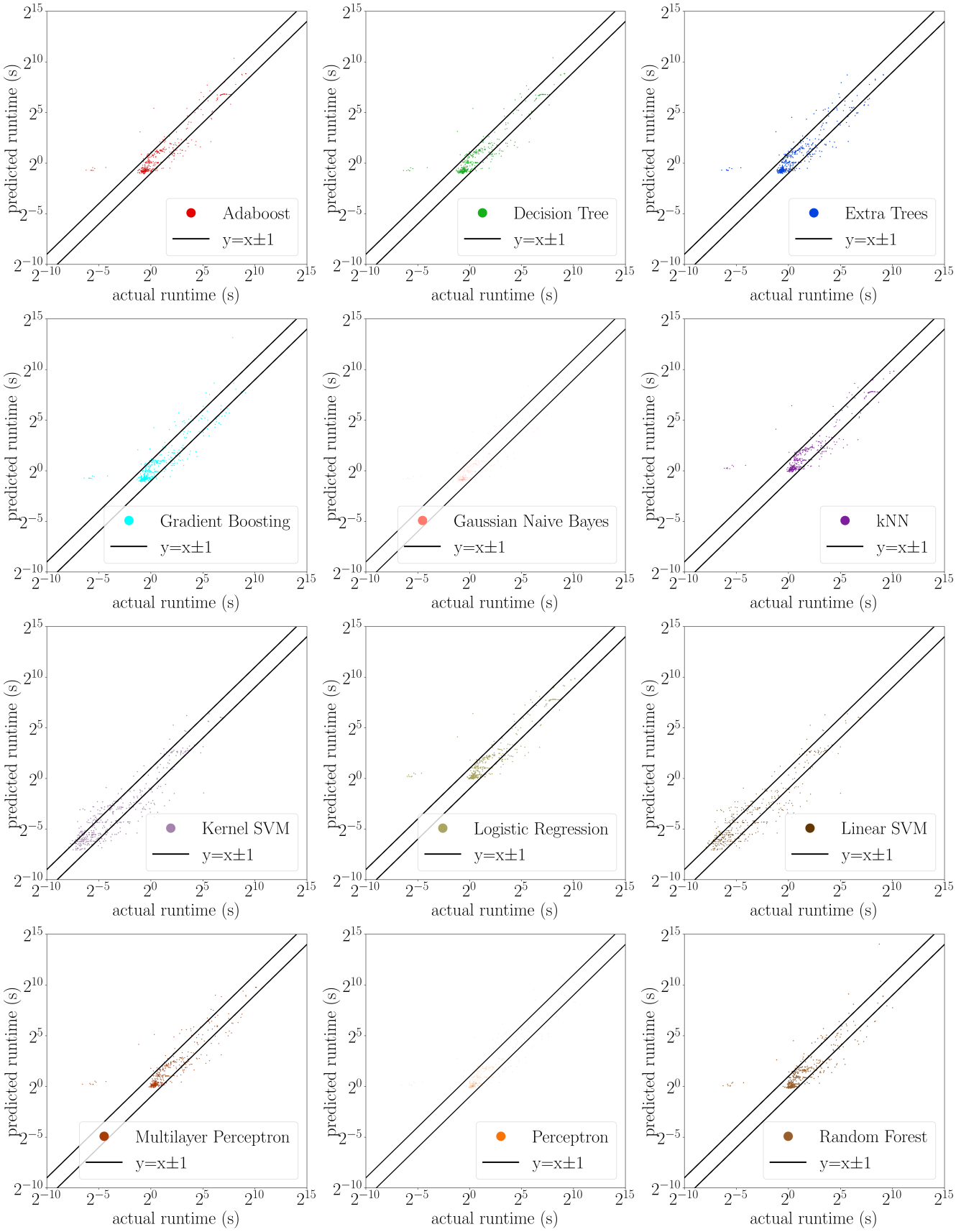


Figure 7: Runtime prediction performance on different machine learning algorithms, on midsize OpenML datasets.

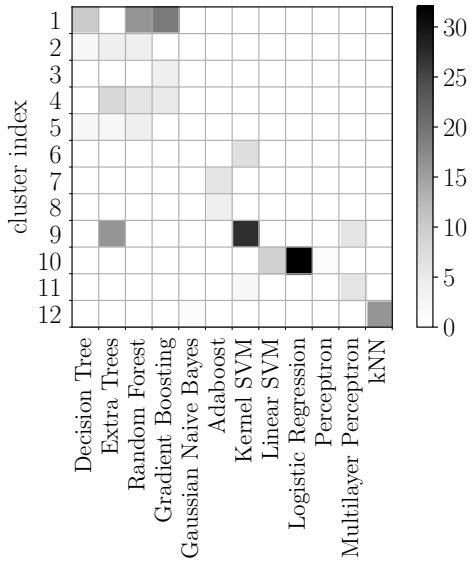


Figure 8: Algorithm heatmap in clusters. Each block is colored by the number of models of the corresponding algorithm type in that cluster. Numbers next to the scale bar refer to the numbers of models.

clustering and interpretability [23, 39, 44]. We perform NMF on the error matrix E to find nonnegative factors $W \in \mathbb{R}^{m\times k}$ and $H \in \mathbb{R}^{k\times n}$ so that $E \approx WH$. Cluster membership of each model is given by the largest entry in its corresponding column in H.

Figure 8 shows the heatmap of algorithms in clusters when $k = 12$ (the number of singular values no smaller than 3% of the largest one). Algorithm types are sparse in clusters: each cluster contains at most 3 types of algorithm. Also, models belonging to the same kinds of algorithms tend to aggregate into the same clusters: for example, Clusters 1 and 4 mainly consist of tree-based models; Cluster 10 of linear models; and Cluster 12 of neighborhood models.

Runtime prediction performance. Runtimes of linear models are among the most difficult to predict, since they depend strongly on the conditioning of the problem. Our runtime prediction accuracy on midsize OpenML datasets is shown in Table 1 and in Figure 7. We can see that our empirical prediction of model runtime is roughly unbiased. Thus the sum of predicted runtimes on multiple models is a roughly good estimate.

Cold-start. Oboe uses D-optimal experiment design to cold-start model selection. In Figure 9, we compare this choice with A- and E-optimal design and nonlinear regression in Alors [28], by means of leave-one-out cross-validation on midsize OpenML datasets. We measure performance by the relative RMSE $\|e - \hat{e}\|_2/\|e\|_2$ of the predicted performance vector and by the number of correctly predicted best models, both averaged across datasets. The approximate rank of the error matrix is set to be the number of eigenvalues larger than 1% of the largest, which is 38 here. The time limit in experiment design implementation is set to be 4 seconds; the nonlinear regressor used in Alors implementation is the default `RandomForestRegressor` in scikit-learn 0.19.2 [30].

Table 1: Runtime prediction accuracy on OpenML datasets

Algorithm type	Runtime prediction accuracy within factor of 2	within factor of 4
Adaboost	83.6%	94.3%
Decision tree	76.7%	88.1%
Extra trees	96.6%	99.5%
Gradient boosting	53.9%	84.3%
Gaussian naive Bayes	89.6%	96.7%
kNN	85.2%	88.2%
Logistic regression	41.1%	76.0%
Multilayer perceptron	78.9%	96.0%
Perceptron	75.4%	94.3%
Random Forest	94.4%	98.2%
Kernel SVM	59.9%	86.7%
Linear SVM	30.1%	73.2%

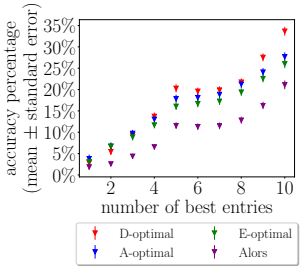


Figure 9: Comparison of cold-start methods.

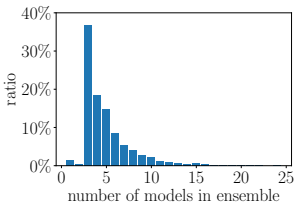


Figure 10: Histogram of Oboe ensemble size. The ensembles were built in executions on midsize OpenML datasets in Section 5.1.2.

The horizontal axis is the number of models selected; the vertical axis is the percentage of best-ranked models shared between true and predicted performance vectors. D-optimal design robustly outperforms.

Ensemble size. As shown in Figure 10, more than 70% of the ensembles constructed on midsize OpenML datasets have no more than 5 base learners. This parsimony makes our ensembles easy to implement and interpret.

6 SUMMARY

Oboe is an AutoML system that uses collaborative filtering and optimal experiment design to predict performance of machine learning models. By fitting a few models on the meta-test dataset, this system transfers knowledge from meta-training datasets to select a promising set of models. Oboe naturally handles different algorithm and hyperparameter types and can match state-of-the-art performance of AutoML systems much more quickly than competing approaches.

This work demonstrates the promise of collaborative filtering approaches to AutoML. However, there is much more left to do. Future work is needed to adapt Oboe to different loss metrics, budget types, sparsely observed error matrices, and a wider range of machine learning algorithms. Adapting a collaborative filtering approach to search for good machine learning *pipelines*, rather than

individual algorithms, presents a more substantial challenge. We also hope to see more approaches to the challenge of choosing hyper-hyperparameter settings subject to limited computation and data: meta-learning is generally data(set)-constrained. With continuing efforts by the AutoML community, we look forward to a world in which domain experts seeking to use machine learning can focus on data quality and problem formulation, rather than on tasks — such as algorithm selection and hyperparameter tuning — which are suitable for automation.

ACKNOWLEDGMENTS

This work was supported in part by DARPA Award FA8750-17-2-0101. The authors thank Christophe Giraud-Carrier, Ameet Talwalkar, Raul Astudillo Marban, Matthew Zalesak, Lijun Ding and Davis Wertheimer for helpful discussions, and thank Jack Dunn for a script to parse UCI Machine Learning Repository datasets.

REFERENCES

[1] Rémi Bardenet, Mátyás Brendel, Balázs Kégl, and Michele Sebag. 2013. Collaborative hyperparameter tuning. In *ICML*. 199–207.
[2] Thomas Bartz-Beielstein and Sandor Markon. 2004. Tuning search algorithms for real-world applications: A regression tree based approach. In *Congress on Evolutionary Computation*, Vol. 1. IEEE, 1111–1118.
[3] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*. 2546–2554.
[4] Bernd Bischl, Jakob Richter, Jakob Bossek, Daniel Horn, Janek Thomas, and Michel Lang. 2017. mlrMBO: A modular framework for model-based optimization of expensive black-box functions. *arXiv preprint arXiv:1703.03373* (2017).
[5] Stephen Boyd and Lieven Vandenberghe. 2004. *Convex optimization*. Cambridge University Press.
[6] Rich Caruana, Art Munson, and Alexandru Niculescu-Mizil. 2006. Getting the most out of ensemble selection. In *ICDM*. IEEE, 828–833.
[7] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. 2004. Ensemble selection from libraries of models. In *ICML*. ACM, 18.
[8] Boyuan Chen, Harvey Wu, Warren Mo, Ishanu Chattopadhyay, and Hod Lipson. 2018. Autostacker: A compositional evolutionary learning system. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 402–409.
[9] Tiago Cunha, Carlos Soares, and André C. P. L. F. de Carvalho. 2018. CF4CF: Recommending Collaborative Filtering Algorithms Using Collaborative Filtering. In *Proceedings of the 12th ACM Conference on Recommender Systems (RecSys '18)*. ACM, New York, NY, USA, 357–361. https://doi.org/10.1145/3240323.3240378
[10] Dua Dheeru and Efi Karra Taniskidou. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml
[11] Iddo Drori, Yamuna Krishnamurthy, Remi Rampin, Raoni de Paula Lourenco, Jorge Piazentin Ono, Kyunghyun Cho, Claudio Silva, and Juliana Freire. 2018. AlphaD3M: Machine learning pipeline synthesis. In *AutoML Workshop at ICML*.
[12] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*. 2962–2970.
[13] Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. 2014. Using meta-learning to initialize Bayesian optimization of hyperparameters. In *International Conference on Meta-learning and Algorithm Selection*. Citeseer, 3–10.
[14] Nicolo Fusi, Rishit Sheth, and Melih Elibol. 2018. Probabilistic matrix factorization for automated machine learning. In *Advances in Neural Information Processing Systems*. 3352–3361.
[15] Gene H Golub and Charles F Van Loan. 2012. *Matrix computations*. JHU Press.
[16] Elad Hazan, Adam Klivans, and Yang Yuan. 2018. Hyperparameter optimization: a spectral approach. In *ICLR*. https://openreview.net/forum?id=H1zriGeCZ
[17] Ralf Herbrich, Neil D Lawrence, and Matthias Seeger. 2003. Fast sparse Gaussian process methods: The informative vector machine. In *Advances in Neural Information Processing Systems*. 625–632.
[18] Ling Huang, Jinzhu Jia, Bin Yu, Byung-Gon Chun, Petros Maniatis, and Mayur Naik. 2010. Predicting execution time of computer programs using sparse polynomial regression. In *Advances in Neural Information Processing Systems*. 883–891.
[19] Frank Hutter, Youssef Hamadi, Holger H Hoos, and Kevin Leyton-Brown. 2006. Performance prediction and automated tuning of randomized and parametric algorithms. In *International Conference on Principles and Practice of Constraint Programming*. Springer, 213–228.
[20] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2011. Sequential Model-Based Optimization for General Algorithm Configuration. *LION* 5 (2011), 507–523.
[21] Frank Hutter, Lin Xu, Holger H Hoos, and Kevin Leyton-Brown. 2014. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence* 206 (2014), 79–111.
[22] RC St John and Norman R Draper. 1975. D-optimality for regression designs: a review. *Technometrics* 17, 1 (1975), 15–23.
[23] Jingu Kim and Haesun Park. 2008. *Sparse nonnegative matrix factorization for clustering*. Technical Report. Georgia Institute of Technology.
[24] Andreas Krause, Ajit Singh, and Carlos Guestrin. 2008. Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research* 9, Feb (2008), 235–284.
[25] Rui Leite, Pavel Brazdil, and Joaquin Vanschoren. 2012. Selecting classification algorithms with active testing. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*. Springer, 117–131.
[26] Christiane Lemke, Marcin Budka, and Bogdan Gabrys. 2015. Metalearning: a survey of trends and technologies. *Artificial Intelligence Review* 44, 1 (2015), 117–130.
[27] David JC MacKay. 1992. Information-based objective functions for active data selection. *Neural Computation* 4, 4 (1992), 590–604.
[28] Mustafa Mısır and Michèle Sebag. 2017. Alors: An algorithm recommender system. *Artificial Intelligence* 244 (2017), 291–314.
[29] Alexander M Mood et al. 1946. On Hotelling's weighing problem. *The Annals of Mathematical Statistics* 17, 4 (1946), 432–446.
[30] 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. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
[31] Bernhard Pfahringer, Hilan Bensusan, and Christophe G Giraud-Carrier. 2000. Meta-Learning by Landmarking Various Learning Algorithms. In *ICML*. 743–750.
[32] Friedrich Pukelsheim. 1993. *Optimal design of experiments*. Vol. 50. SIAM.
[33] Carl Edward Rasmussen and Christopher KI Williams. 2006. *Gaussian processes for machine learning*. the MIT Press.
[34] Paola Sebastiani and Henry P Wynn. 2000. Maximum entropy sampling and optimal Bayesian experimental design. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 62, 1 (2000), 145–157.
[35] Kate Smith-Miles and Jano van Hemert. 2011. Discovering the suitability of optimisation algorithms by learning from evolved instances. *Annals of Mathematics and Artificial Intelligence* 61, 2 (2011), 87–104.
[36] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*. 2951–2959.
[37] Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. 2010. Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. In *ICML*. 1015–1022.
[38] David H Stern, Horst Samulowitz, Ralf Herbrich, Thore Graepel, Luca Pulina, and Armando Tacchella. 2010. Collaborative Expert Portfolio Management. In *AAAI*. 179–184.
[39] Ali Caner Türkmen. 2015. A review of nonnegative matrix factorization methods for clustering. *arXiv preprint arXiv:1507.03194* (2015).
[40] Madeleine Udell and Alex Townsend. 2018. Why are Big Data Matrices Approximately Low Rank? *SIAM Mathematics of Data Science (SIMODS), to appear* (2018). arXiv:1705.07474
[41] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. 2013. OpenML: Networked Science in Machine Learning. *SIGKDD Explorations* 15, 2 (2013), 49–60. https://doi.org/10.1145/2641190.2641198
[42] Abraham Wald. 1943. On the efficient design of statistical investigations. *The Annals of Mathematical Statistics* 14, 2 (1943), 134–140.
[43] M. Wistuba, N. Schilling, and L. Schmidt-Thieme. 2015. Learning hyperparameter optimization initializations. In *IEEE International Conference on Data Science and Advanced Analytics*. 1–10. https://doi.org/10.1109/DSAA.2015.7344817
[44] Wei Xu, Xin Liu, and Yihong Gong. 2003. Document clustering based on non-negative matrix factorization. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*. ACM, 267–273.
[45] Dani Yogatama and Gideon Mann. 2014. Efficient transfer learning method for automatic hyperparameter tuning. In *Artificial Intelligence and Statistics*. 1077–1085.
[46] Yuyu Zhang, Mohammad Taha Bahadori, Hang Su, and Jimeng Sun. 2016. FLASH: fast Bayesian optimization for data analytic pipelines. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2065–2074.

Table 2: Base Algorithm and Hyperparameter Settings

Algorithm type	Hyperparameter names (values)
Adaboost	`n_estimators (50,100), learning_rate (1.0,1.5,2.0,2.5,3)`
Decision tree	`min_samples_split (2,4,8,16,32,64,128,256,512,1024,0.01,0.001,0.0001,1e-05)`
Extra trees	`min_samples_split (2,4,8,16,32,64,128,256,512,1024,0.01,0.001,0.0001,1e-05), criterion (gini,entropy)`
Gradient boosting	`learning_rate (0.001,0.01,0.025,0.05,0.1,0.25,0.5), max_depth (3, 6), max_features (null,log2)`
Gaussian naive Bayes	-
kNN	`n_neighbors (1,3,5,7,9,11,13,15), p (1,2)`
Logistic regression	`C (0.25,0.5,0.75,1,1.5,2,3,4), solver (liblinear,saga), penalty (l1,l2)`
Multilayer perceptron	`learning_rate_init (0.0001,0.001,0.01), learning_rate (adaptive), solver (sgd,adam), alpha (0.0001, 0.01)`
Perceptron	-
Random forest	`min_samples_split (2,4,8,16,32,64,128,256,512,1024,0.01,0.001,0.0001,1e-05), criterion (gini,entropy)`
Kernel SVM	`C (0.125,0.25,0.5,0.75,1,2,4,8,16), kernel (rbf,poly), coef0 (0,10)`
Linear SVM	`C (0.125,0.25,0.5,0.75,1,2,4,8,16)`

For reproducibility, please refer to our GitHub repositories (the Oboe system: https://github.com/udellgroup/oboe; experiments: https://github.com/udellgroup/oboe-testing). Additional information is as follows.

A MACHINE LEARNING MODELS

Shown in Table 2, the hyperparameter names are the same as those in scikit-learn 0.19.2.

B DATASET META-FEATURES

Dataset meta-features used throughout the experiments are listed in Table 3 (next page).

C META-FEATURE CALCULATION TIME

On a number of not very large datasets, the time taken to calculate meta-features in the previous section are already non-negligible, as shown in Figure 11. Each dot represents one midsize OpenML dataset.

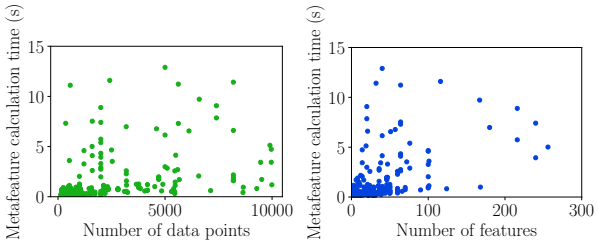


Figure 11: Meta-feature calculation time and corresponding dataset sizes of the midsize OpenML datasets. The collection of meta-features is the same as that used by auto-sklearn [12]. We can see some calculation times are not negligible.

D COMPARISON OF EXPERIMENT DESIGN WITH DIFFERENT CONSTRAINTS

In Section 5.1.1, we compared experiment design (ED) with constraint on the number of observed entries. This version is more comparable to QR and PMF than the version with runtime constraint (Equation 1). However, the time-constrained version has decent performance, as shown in Figure 12.

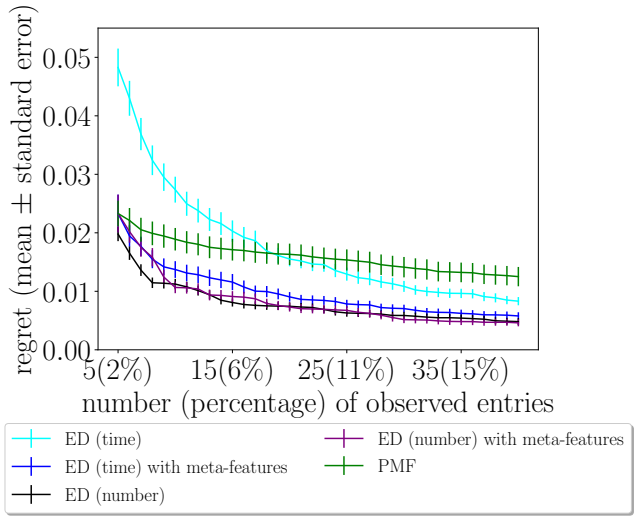


Figure 12: Comparison of different versions of ED with PMF. "ED (time)" denotes ED with runtime constraint, with time limit set to be 10% of the total runtime of all available models; "ED (number)" denotes ED with the number of entries constrained.

Table 3: Dataset Meta-features

Meta-feature name	Explanation
number of instances	number of data points in the dataset
log number of instances	the (natural) logarithm of number of instances
number of classes	
number of features	
log number of features	the (natural) logarithm of number of features
number of instances with missing values	
percentage of instances with missing values	
number of features with missing values	
percentage of features with missing values	
number of missing values	
percentage of missing values	
number of numeric features	
number of categorical features	
ratio numerical to nominal	the ratio of number of numerical features to the number of categorical features
ratio numerical to nominal	
dataset ratio	the ratio of number of features to the number of data points
log dataset ratio	the natural logarithm of dataset ratio
inverse dataset ratio	
log inverse dataset ratio	
class probability (min, max, mean, std)	the (min, max, mean, std) of ratios of data points in each class
symbols (min, max, mean, std, sum)	the (min, max, mean, std, sum) of the numbers of symbols in all categorical features
kurtosis (min, max, mean, std)	
skewness (min, max, mean, std)	
class entropy	the entropy of the distribution of class labels (logarithm base 2)
landmarking [31] meta-features	
LDA	
decision tree	decision tree classifier with 10-fold cross validation
decision node learner	10-fold cross-validated decision tree classifier with `criterion="entropy", max_depth=1, min_samples_split=2, min_samples_leaf=1, max_features=None`
random node learner	10-fold cross-validated decision tree classifier with `max_features=1` and the same above for the rest
1-NN	
PCA fraction of components for 95% variance	the fraction of components that account for 95% of variance
PCA kurtosis first PC	kurtosis of the dimensionality-reduced data matrix along the first principal component
PCA skewness first PC	skewness of the dimensionality-reduced data matrix along the first principal component