OBOE: Collaborative Filtering for AutoML Initialization

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. The number of machine learning applications is growing much faster than the number of machine learning experts, hence we see an increasing demand for efficient automation of learning processes. Here, we introduce OBOE, an algorithm for time-constrained model selection and hyperparameter tuning. Taking advantage of similarity between datasets, OBOE finds promising algorithm and hyperparameter configurations through collaborative filtering. Our system explores these models under time constraints, so that rapid initializations can be provided to warm-start more fine-grained optimization methods. One novel aspect of our approach is 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.

1 Introduction

The large number of machine learning algorithms and their sensitivity to hyperparameter values make it practically infeasible to enumerate all configurations. The field of AutoML seeks to efficiently automate the selection of model configurations, and has attracted increasing attention in recent years.

We propose an algorithmic system, OBOE [1], that provides an initial tuning for AutoML. It addresses two important problems: 1) *time-constrained initialization*: how to choose promising models under time constraints, and 2) *active learning*: how to improve given further computational resources.

The first subproblem is important in that hyperparameter spaces have heterogenous shapes and are poorly studied. One solution is collaborative filtering [1, 32, 36, 22], in which it is important to characterize dataset similarity. One line of work uses dataset meta-features [25, 10, 9]; the other (e.g., [35]) avoids meta-features. Our approach builds on both of these lines by treating low rank representations of model performance vectors as latent meta-features, and thus rely exclusively on model performance for dataset similarity.

The active learning subproblem aims to select the best model or 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 [27, 30, 3, 11, 28, 13, 21, 31], sparse Boolean functions [12] and decision trees [2, 16]. Our OBOE system uses the set of low rank matrices as surrogate model, which is a flexible and general model class that enjoys the simplicity and speed of well-developed numerical linear algebra algorithms.

One key component of our system is a model which predicts model runtime on datasets. Many authors have previously studied algorithm runtime as a function of feature vectors [17], via ridge regression [14], neural networks [29], Gaussian processes [15], etc. Several measures have been proposed to trade-off between accuracy and runtime [20, 4]. We fit algorithm runtime onto dataset size, which is particularly simple but surprisingly effective.

[1] This name comes from the musical instrument oboe that plays an initial note in an orchestra; the other instruments use this note to tune to the right frequency before the performance begins.

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

In a linear model, classical experiment design [34, 23, 18, 26, 5] selects features that minimize variance of the parameter estimate. Budget constraints can be added [19, 37]. We adopt this approach to select promising machine learning models believed to finish within time budget.

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

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.

All vectors in this paper are column vectors. Given a matrix $A \in \mathbb{R}^{m \times n}$, $A_{i,:}$ and $A_{:,j}$ denote the *i*th row and *j*th column of A, respectively. 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 $A_{:\mathcal{S}} = [A_{:,s_1} \quad A_{:,s_2} \quad \cdots \quad A_{:,s_k}]$.

$\mathcal{A}(\mathcal{D})$ is the cross-validation error of model $\mathcal{A}$ on dataset $\mathcal{D}$. The expectation is with respect to the way we split dataset into folds. The *best model* on $\mathcal{D}$ is the model in our collection of models that achieves minimum error on $\mathcal{D}$. A model $\mathcal{A}$ is said to be *observed* on $\mathcal{D}$ if we actually calculate $\mathcal{A}(\mathcal{D})$.

Hyper-hyperparameters refer to OBOE settings, e.g., the rank used to approximate the error matrix.

3 Methodology

Unsupervised inference of model performance While the distance between meta-feature vectors is informative of dataset similarity, it is often difficult to determine *a priori* which meta-features to use. Also, the computation of meta-features can be expensive (see Appendix H, Figure 7). To infer model performance on a dataset without any meta-feature calculation that takes more time than it takes to read in the dataset, we use collaborative filtering.

We construct an error matrix $E \in \mathbb{R}^{M \times N}$, where rows are indexed by datasets and columns by models. Empirically, E has approximately low rank: Figure 1 shows the singular values $\sigma_i(E)$ as a function of the index i. This serves as foundation of our algorithm. Hence we approximate $E_{ij} \approx x_i^T y_j$ where x_i and y_j are the minimizers of $\sum_{i=1}^{m} \sum_{j=1}^{n} (E_{ij} - x_i^T y_j)^2$ with $x_i, y_j \in \mathbb{R}^k$ for $i \in [M]$ and $j \in [N]$; the solution is given by PCA. In collaborative filtering, each x_i and y_j are the latent meta-features of dataset i and model j, respectively.

Given a meta-test dataset, we choose a subset $\mathcal{S} \subseteq [N]$ of models and observe the performance e_j of model j for each $j \in \mathcal{S}$, where $\mathcal{S}$ is determined by the dataset and runtime budget τ. We then infer latent meta-features of that dataset by solving the least squares problem: minimize $\sum_{j \in \mathcal{S}} (e_j - \hat{x}^T y_j)^2$ with $\hat{x} \in \mathbb{R}^k$. For all unobserved models, we predict their performance as $\hat{e}_j = \hat{x}^T y_j$ for $j \notin \mathcal{S}$.

The remaining challenge is to choose $\mathcal{S}$ and k, which will be addressed in the following sections.

Runtime prediction Estimated model runtime allows us to tradeoff between slow, informative and fast, less informative models. We observe that we are able to predict runtime of half of the machine learning models within a factor of two on more than 75% midsize OpenML classification datasets, as shown in Appendix E and visualized in Figures 5 and 6. Our method uses polynomial regression on $n^{\mathcal{D}}$ and $p^{\mathcal{D}}$, the numbers of data points and features in $\mathcal{D}$, 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 $f_j = \texttt{fit\_runtime}(n, p, t)$.

Time-constrained information gathering To select model subset $\mathcal{S}$, 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. Then we choose which models we should fit by defining an indicator vector $v \in \{0, 1\}^n$, where entry v_j indicates whether to fit model j, and minimizing a scalarization of the covariance of the estimated meta-features $\hat{x}$ of the new dataset subject to runtime constraints. 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 we relax $v \in \mathbb{R}^n$ to allow for non-Boolean values and solve the optimization problem

$$\begin{aligned} \underset{v_j}{\text{minimize}} \quad & \log\det\Big(\sum_{j=1}^{n} v_j y_j y_j^T\Big)^{-1} \\ \text{subject to} \quad & \sum_{j=1}^{n} v_j \hat{t}_j \le \tau \\ & v_j \in [0,1] \qquad \forall j \in [n]. \end{aligned}$$

in which the scalarization by means of covariance determinant minimization is called D-optimal design. Several other scalarizations can also be used, e.g., covariance norm (E-optimal) or trace (A-optimal). This relaxed problem is a convex optimization problem, and we obtain an approximate solution via rounding. 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 OBOE

The OBOE system can be divided into two stages: offline and online, whose pseudocode is shown in Appendix B. The offline stage is executed only once and stores information of meta-training datasets. Time taken on this stage does not affect the prediction of OBOE on a new dataset; the runtime experienced by the user is that of the online stage.

Offline stage We generate the error matrix under *balanced error rate*, the average of false positive and false negative rates across different classes, and record runtime of machine learning model on datasets. This is used to fit runtime predictors described in Section 3.

Online stage As shown below, `fit_one_round` is a subroutine of time target doubling.

- **Time-constrained model selection (`fit_one_round`)** We first predict model runtime on the meta-test dataset using fitted runtime predictors. Then we 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 estimation of the latent meta-features of the test dataset. We then use $\hat{x}$ to predict every entry of e. We build an ensemble out of models predicted to perform well within time target $\tilde{\tau}$, by means of greedy forward selection [7, 6]. This step outputs a classifier, and can be placed after further hyperparameter tuning. 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 $\tilde{A}$.
- **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. Rank k increments by 1 if the validation error of the ensemble learner decreases after doubling the time target, and otherwise does not change.

5 Experimental evaluation

Code for the OBOE system is at `https://github.com/udellgroup/oboe`; code for experiments is at `https://github.com/udellgroup/oboe-testing`. Experimental setup and minor results can be found in Appendix D.

Cold-start functionality OBOE uses D-optimal experiment design to cold-start model selection. In Figure 2, we compare this with A- and E-optimal design and nonlinear regression in Alors [22], by means of leave-one-out cross-validation on midsize OpenML datasets. The horizontal axis is the number of models selected; the vertical axis is the average percentage of best-ranked models shared between true and predicted performance vectors. D-optimal design robustly outperforms.

Performance comparison across AutoML systems We compare AutoML systems that are able to select among different algorithm types under time constraints: OBOE (whose error matrix was generated by OpenML datasets), auto-sklearn [9], and a *time-constrained* random baseline. Figure 3 shows the percentile and ranking changes of prediction errors as runtime repeatedly doubles.

Several observations on the experimental results are [2]:

1 OBOE on average performs as well as or better than competing approaches (Figures 3c and 3d).

[2] Auto-sklearn's GitHub Issue #537 says "Do not start Auto-sklearn for time limits less than 60s". We compare OBOE with auto-sklearn, just to demonstrate OBOE's ability of model selection within a short time.

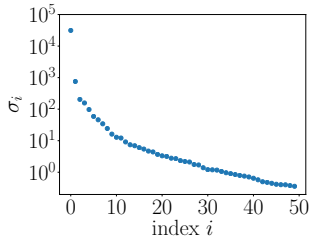


Figure 1: Singular value decay of an error matrix.

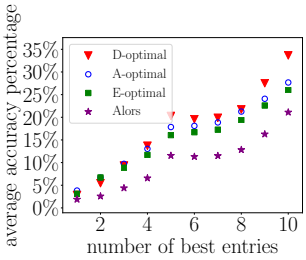
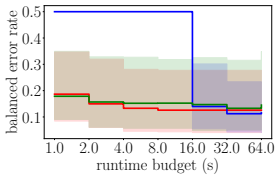
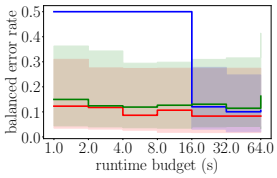


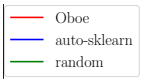
Figure 2: Comparison of cold-start methods.

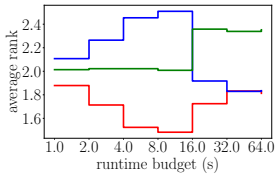


(a) midsize OpenML datasets (meta-training)

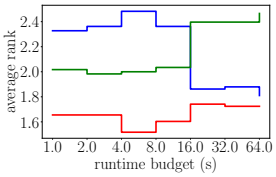


(b) midsize UCI datasets (meta-test)





(c) midsize OpenML datasets (meta-training)



(d) midsize UCI datasets (meta-test)

Figure 3: Time-constrained performance of AutoML systems. In 3a and 3b, 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 3c and 3d show system rankings (1 is best and 3 is worst).

2 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 3a and 3b). In contrast, auto-sklearn must first compute meta-features for each dataset, which requires substantial computational time, as shown in Appendix H, Figure 7.

3 Interestingly, the rate at which the OBOE models improves with time is faster than that of auto-sklearn: the level of improvements OBOE makes before 16s matches that of auto-sklearn from 16s to 64s. This indicates that increased computational time 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.

6 Summary

OBOE is an AutoML system that uses ideas from 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, many improvements are possible. Future work to adapt OBOE to different loss metrics, budget types, sparse error matrices and a wider range of machine learning algorithms, as well as to augment the initializations given by OBOE with fine tuning by any state-of-the-art hyperparameter optimization method, may yield substantial improvements. Furthermore, we look forward to seeing more approaches to the challenge of choosing hyper-hyperparameter settings subject to limited computation and data. 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.

References

[1] Rémi Bardenet, Mátyás Brendel, Balázs Kégl, and Michele Sebag. Collaborative hyperparameter tuning. In *ICML*, pages 199–207, 2013.

[2] Thomas Bartz-Beielstein and Sandor Markon. Tuning search algorithms for real-world applications: A regression tree based approach. In *Congress on Evolutionary Computation*, volume 1, pages 1111–1118. IEEE, 2004.

[3] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pages 2546–2554, 2011.

[4] Bernd Bischl, Jakob Richter, Jakob Bossek, Daniel Horn, Janek Thomas, and Michel Lang. 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. *Convex optimization*. Cambridge University Press, 2004.

[6] Rich Caruana, Art Munson, and Alexandru Niculescu-Mizil. Getting the most out of ensemble selection. In *ICDM*, pages 828–833. IEEE, 2006.

[7] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble selection from libraries of models. In *ICML*, page 18. ACM, 2004.

[8] Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017.

[9] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*, pages 2962–2970, 2015.

[10] Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Using meta-learning to initialize Bayesian optimization of hyperparameters. In *International Conference on Meta-learning and Algorithm Selection*, pages 3–10. Citeseer, 2014.

[11] Nicolo Fusi and Huseyn Melih Elibol. Probabilistic matrix factorization for automated machine learning. *arXiv preprint arXiv:1705.05355*, 2017.

[12] Elad Hazan, Adam Klivans, and Yang Yuan. Hyperparameter Optimization: A Spectral Approach. *arXiv preprint arXiv:1706.00764*, 2017.

[13] Ralf Herbrich, Neil D Lawrence, and Matthias Seeger. Fast sparse Gaussian process methods: The informative vector machine. In *Advances in Neural Information Processing Systems*, pages 625–632, 2003.

[14] Ling Huang, Jinzhu Jia, Bin Yu, Byung-Gon Chun, Petros Maniatis, and Mayur Naik. Predicting execution time of computer programs using sparse polynomial regression. In *Advances in Neural Information Processing Systems*, pages 883–891, 2010.

[15] Frank Hutter, Youssef Hamadi, Holger H Hoos, and Kevin Leyton-Brown. Performance prediction and automated tuning of randomized and parametric algorithms. In *International Conference on Principles and Practice of Constraint Programming*, pages 213–228. Springer, 2006.

[16] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential Model-Based Optimization for General Algorithm Configuration. *LION*, 5:507–523, 2011.

[17] Frank Hutter, Lin Xu, Holger H Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, 2014.

[18] RC St John and Norman R Draper. D-optimality for regression designs: a review. *Technometrics*, 17(1):15–23, 1975.

[19] Andreas Krause, Ajit Singh, and Carlos Guestrin. Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, 9(Feb):235–284, 2008.

[20] Rui Leite, Pavel Brazdil, and Joaquin Vanschoren. Selecting classification algorithms with active testing. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 117–131. Springer, 2012.

[21] David JC MacKay. Information-based objective functions for active data selection. *Neural Computation*, 4(4):590–604, 1992.

[22] Mustafa Mısır and Michèle Sebag. Alors: An algorithm recommender system. *Artificial Intelligence*, 244:291–314, 2017.

[23] Alexander M Mood et al. On Hotelling's weighing problem. *The Annals of Mathematical Statistics*, 17(4):432–446, 1946.

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

[25] Bernhard Pfahringer, Hilan Bensusan, and Christophe G Giraud-Carrier. Meta-Learning by Landmarking Various Learning Algorithms. In *ICML*, pages 743–750, 2000.

[26] Friedrich Pukelsheim. *Optimal design of experiments*, volume 50. SIAM, 1993.

[27] Carl Edward Rasmussen and Christopher KI Williams. *Gaussian processes for machine learning*. the MIT Press, 2006.

[28] Paola Sebastiani and Henry P Wynn. Maximum entropy sampling and optimal Bayesian experimental design. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 62(1):145–157, 2000.

[29] Kate Smith-Miles and Jano van Hemert. Discovering the suitability of optimisation algorithms by learning from evolved instances. *Annals of Mathematics and Artificial Intelligence*, 61(2):87–104, 2011.

[30] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012.

[31] Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*, 2009.

[32] David H Stern, Horst Samulowitz, Ralf Herbrich, Thore Graepel, Luca Pulina, and Armando Tacchella. Collaborative Expert Portfolio Management. In *AAAI*, pages 179–184, 2010.

[33] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. OpenML: Networked Science in Machine Learning. *SIGKDD Explorations*, 15(2):49–60, 2013.

[34] Abraham Wald. On the efficient design of statistical investigations. *The Annals of Mathematical Statistics*, 14(2):134–140, 1943.

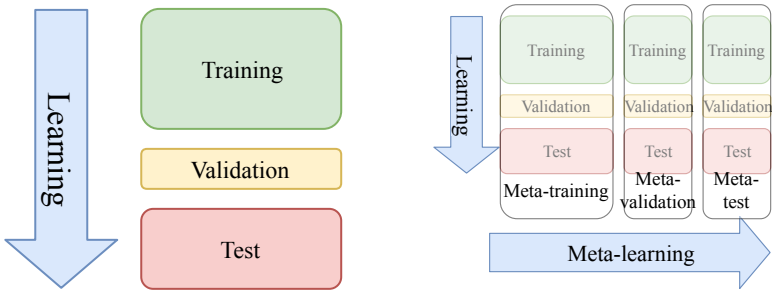
[35] M. Wistuba, N. Schilling, and L. Schmidt-Thieme. Learning hyperparameter optimization initializations. In *IEEE International Conference on Data Science and Advanced Analytics*, pages 1–10, Oct 2015.

[36] Dani Yogatama and Gideon Mann. Efficient transfer learning method for automatic hyperparameter tuning. In *Artificial Intelligence and Statistics*, pages 1077–1085, 2014.

[37] Yuyu Zhang, Mohammad Taha Bahadori, Hang Su, and Jimeng Sun. FLASH: fast Bayesian optimization for data analytic pipelines. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2065–2074. ACM, 2016.

A Visualization of learning vs meta-learning

Figure 4 visualizes the idea of meta-learning, as well as how we split datasets for experimental evaluation. Each colored block is a set of datasets.



(a) Learning (b) Meta-learning

Figure 4: Standard vs meta learning.

B Algorithm pseudocode

B.1 Offline stage

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 $\mathcal{D}_i$ in terms of metric $\mathcal{M}$
    $T_{ij} \leftarrow$ runtime of fitting 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

B.2 Online stage

Algorithm 2 is a subroutine of Algorithm 3; the latter of which shows the whole online stage.

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 $\mathcal{D}_{tr}$, time target for this round $\tilde{\tau}$
Ensure: ensemble learner $\tilde{A}$

for $j = 1, 2, \ldots, n$ **do**
  $\hat{t}_j \leftarrow f_j(n^{\mathcal{D}_{tr}}, p^{\mathcal{D}_{tr}})$
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**
  $e_{\mathcal{S}_k}, z_{\mathcal{S}_k} \leftarrow$ cross-validation error and predicted labels of model $\mathcal{A}_{\mathcal{S}_k}$ on $\mathcal{D}_{tr}$
end for
$\tilde{A} = \texttt{ensemble\_selection}(\mathcal{S}, e_{\mathcal{S}}, z_{\mathcal{S}})$

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 $\tilde{A}$

$x_i, y_j \leftarrow \arg\min \sum_{i=1}^m \sum_{j=1}^n (E_{ij} - x_i^T y_j)^2$, $x_i \in \mathbb{R}^{\min(m,n)}$ for $i \in [M]$, $y_j \in \mathbb{R}^{\min(m,n)}$ for $j \in [N]$

$\mathcal{D}_{tr}, \mathcal{D}_{val}, \mathcal{D}_{te} \leftarrow$ 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`$(\{\tilde{y}_j\}_{j=1}^n, \{f_j\}_{j=1}^n, \mathcal{D}_{tr}, \tilde{\tau})$

  $e'_{\tilde{A}} \leftarrow \tilde{A}(\mathcal{D}_{val})$

  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

C Models used for error matrix generation

Table 1 shows all the algorithms (in alphabetical order; the same below) together with their hyperparameter settings that we have considered to date. We run these algorithms on datasets using scikit-learn 0.19.2 [24]. Hyperparameter settings not listed in this table are set to be default values in the scikit-learn library. Hyperparameter names in Table 1 are consistent with scikit-learn classifier arguments.

Table 1: Base algorithms 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)`

D Experimental evaluation details

Datasets We test different AutoML systems on OpenML [33] and UCI [8] classification datasets, with between 150 and 10,000 data points and with no missing entries, which we call midsize OpenML and UCI datasets. 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.

D.1 Why we build an ensemble in the online stage?

In the online stage, OBOE can return to the user either: 1) *a collection of promising models*, which can be used to initialize any other AutoML system; or 2) *an ensemble of machine learning models*, so as to offer one single prediction for each test point. A collection of models is required to build an ensemble of models. However, it is difficult to characterize the quality of a collection of models, since there is no widely acknowledged performance metric for model collections. Hence we focus on building an ensemble of models in experimental evaluations.

D.2 Hyper-hyperparameter choice

Cold-start functionality We consider performance measured by 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 meta-features used in the Alors implementation are listed in Appendix F, Table 3; the nonlinear regressor used in the Alors implementation is the default `RandomForestRegressor` in scikit-learn 0.19.2 [24].

Error metric 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%.

D.3 Performance comparison across AutoML systems

D.3.1 Approaches

OBOE As columns of the error matrix, we use the algorithm types and hyperparameter ranges listed in Appendix C, Table 1. We chose to vary hyperparameters people usually tune, and picked their ranges to contain the values people usually use. We have not optimized over them. The datasets we use as rows are the midsize OpenML datasets. In our implementation, each entry of E is generated using 5-fold cross-validation. The (decaying) eigenvalues of this error matrix are shown in Figure 1 of the main paper.

auto-sklearn We compare with auto-sklearn+meta-learning+ensemble, using the method `classification.AutoSklearnClassifier` in auto-sklearn 0.4.0 [9].

Random Models chosen by a completely random baseline may take longer to complete than the time budget, in which case no result would be reported. Instead, we adopt a *time-constrained* random baseline that replaces the experiment design subroutine of OBOE with a time-constrained random selection method: we randomly select a subset of models that we predict will complete within the time target of each doubling round.

D.3.2 Experimental setup

Our system does not perform further hyperparameter optimization after selecting a subset of models and forming an ensemble, while we do allow auto-sklearn to perform hyperparameter optimization after model selection to ensure it makes full use of the time budget. This choice gives auto-sklearn a slight advantage over OBOE.

We ran all experiments on a server with 128 Intel® Xeon® E7-4850 v4 2.10GHz CPU cores and 1056GB memory. The process of running each system on a specific dataset is limited to a single CPU core.

E Runtime prediction performances on individual machine learning algorithms

Runtime prediction accuracy on OpenML datasets with between 150 and 10,000 data points and with no missing entries is shown in Table 2 and visualized in Figures 5 and 6.

Table 2: Runtime prediction accuracy on OpenML datasets

Algorithm type	Runtime prediction accuracy within factor of 2	Runtime prediction accuracy 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%

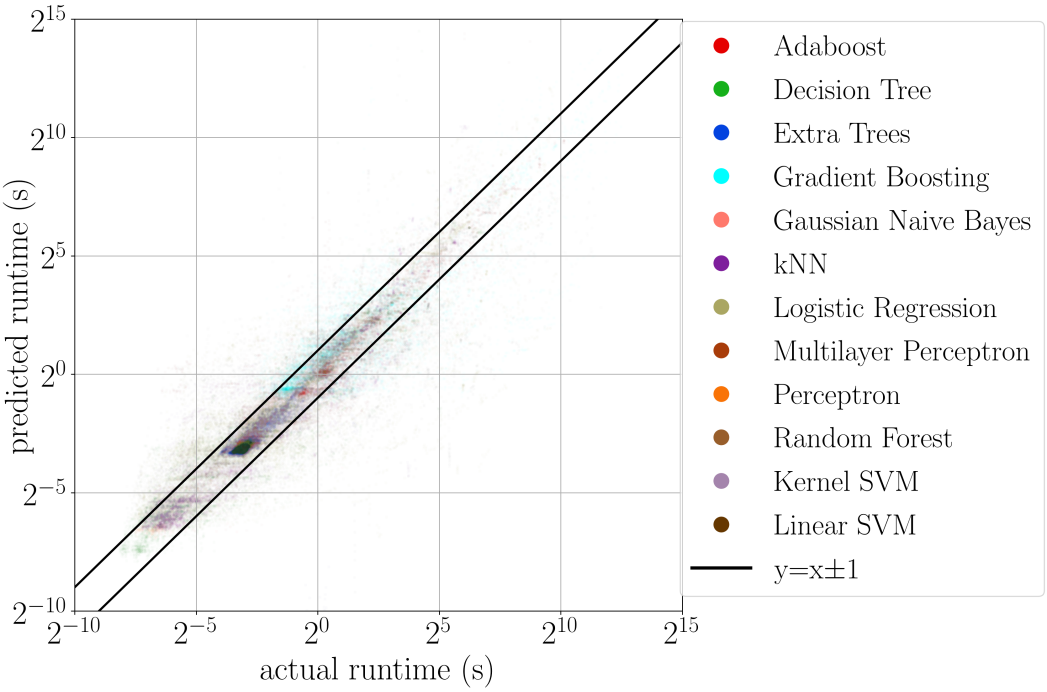


Figure 5: Performance of runtime prediction on midsize OpenML datasets; the algorithm-wise performance is shown in Figure 6.

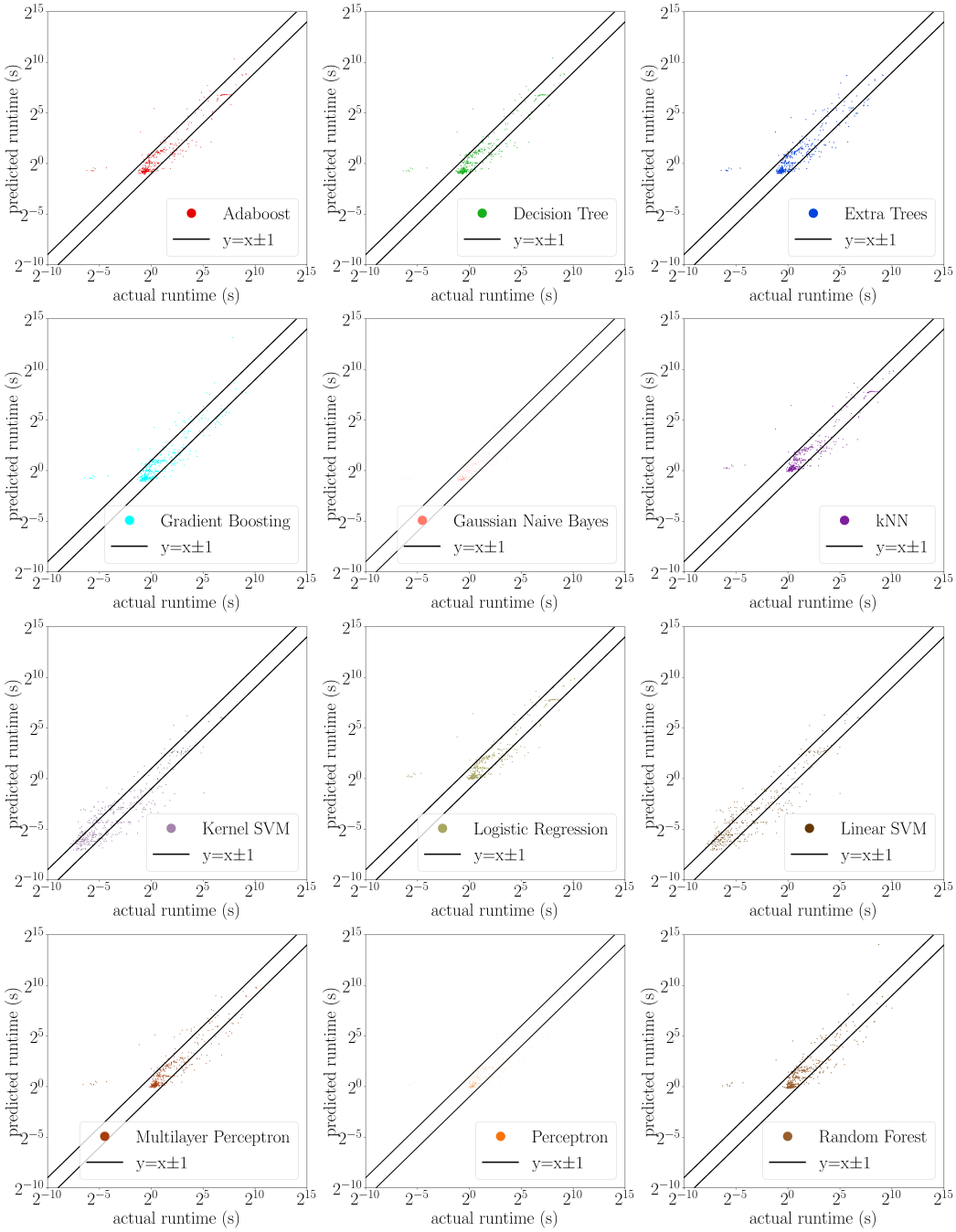


Figure 6: Runtime prediction performance on different machine learning algorithms, on midsize OpenML datasets.

F Dataset meta-features

Dataset meta-features (used in Figure 2 of the main paper for the Alors implementation, and in Figure 3 for the auto-sklearn [9] implementation) are listed in Table 3.

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 [25] 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

G Cold-start comparison

As a single time point in Figure 2 of the main paper, relative RMSE and number of best entry overlaps given by different approaches are shown in Table 4.

Table 4: Results in Figure when number of models selected equals 5.

Metric	Cold-start method			
	D	A	E	Alors
Relative RMSE	18%	43%	31%	85%
Overlap of best 5 models	0.89	0.89	0.80	0.58

H 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 7. Each dot represents one midsize OpenML dataset.

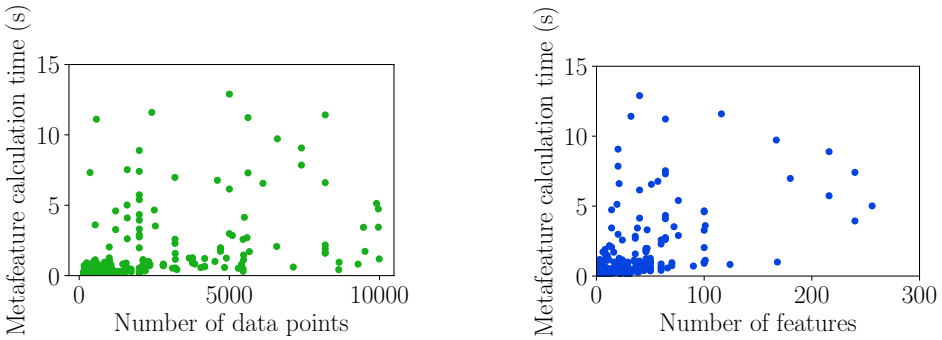


Figure 7: Meta-feature calculation time and corresponding dataset sizes of the midsize OpenML datasets. The collection of meta-features is the same as that used by auto-sklearn [9]. We can see some calculation times are not negligible.