Probabilistic Matrix Factorization for Automated Machine Learning

Nicolo Fusi [1] **Rishit Sheth** [1 2] **Melih Huseyn Elibol** [1]

Abstract

In order to achieve state-of-the-art performance, modern machine learning techniques require careful data pre-processing and hyperparameter tuning. Moreover, given the ever increasing number of machine learning models being developed, model selection is becoming increasingly important. Automating the selection and tuning of machine learning pipelines, consisting of data pre-processing methods and machine learning models, has long been one of the goals of the machine learning community. In this paper, we propose to solve this meta-learning task by combining ideas from collaborative filtering and Bayesian optimization. Specifically, we exploit experiments performed on hundreds of different datasets via probabilistic matrix factorization and then use an acquisition function to guide the exploration of the space of possible pipelines. In our experiments, we show that our approach quickly identifies high-performing pipelines across a wide range of datasets, significantly outperforming the current state-of-the-art.

1. Introduction

Machine learning models often depend on hyperparameters that require extensive fine-tuning in order to achieve optimal performance. For example, state-of-the-art deep neural networks have highly tuned architectures and require careful initialization of the weights and learning algorithm (for example, by setting the initial learning rate and various decay parameters). These hyperparameters can be learned by cross-validation (or holdout set performance) over a grid of values, or by randomly sampling the hyperparameter space (Bergstra & Bengio, 2012); but, these approaches do not take advantage of any continuity in the parameter space. More recently, *Bayesian optimization* has emerged as

[1]Microsoft Research, Cambridge, MA, USA [2]Department of Computer Science, Tufts University, Medford, MA, USA. Correspondence to: Nicolo Fusi <fusi@microsoft.com>, Rishit Sheth <rishit.sheth@tufts.edu>.
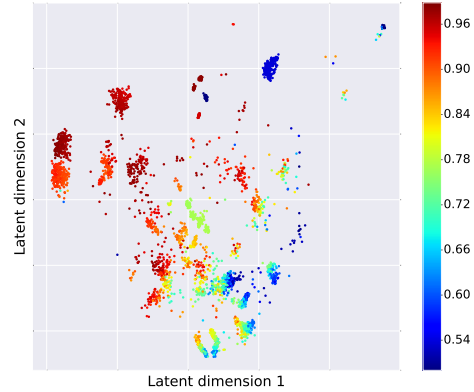
Figure 1: Two-dimensional embedding of 5,000 ML pipelines across 576 OpenML datasets. Each point corresponds to a pipeline and is colored by the AUROC obtained by that pipeline in one of the OpenML datasets (OpenML dataset id 943).

a promising alternative to these approaches (Srinivas et al., 2009; Hutter et al., 2011; Osborne et al., 2009; Bergstra et al., 2011; Snoek et al., 2012; Bergstra et al., 2013). In Bayesian optimization, the loss (*e.g.* root mean square error) is modeled as a function of the hyperparameters. A regression model (usually a Gaussian process) and an acquisition function are then used to iteratively decide which hyperparameter setting should be evaluated next. More formally, the goal of Bayesian optimization is to find the vector of hyperparameters $\boldsymbol{\theta}$ that corresponds to

$$\arg \min_{\boldsymbol{\theta}} \mathscr{L}(\mathcal{M}(\mathbf{x}; \boldsymbol{\theta}), \mathbf{y}),$$

where $\mathcal{M}(\mathbf{x} \,|\, \boldsymbol{\theta})$ are the predictions generated by a machine learning model $\mathcal{M}$ (*e.g.* SVM, random forest, etc.) with hyperparameters $\boldsymbol{\theta}$ on some inputs $\mathbf{x}$; $\mathbf{y}$ are the targets/labels and $\mathscr{L}$ is a loss function. Usually, the hyperparameters are a subset of $\mathbb{R}^D$, although in practice many hyperparameters can be discrete (*e.g.* the number of layers in a neural network) or categorical (*e.g.* the loss function to use in a gradient boosted regression tree).

Bayesian optimization techniques have been shown to be

very effective in practice and sometimes identify better hyperparameters than human experts, leading to state-of-the-art performance in computer vision tasks (Snoek et al., 2012). One drawback of these techniques is that they are known to suffer in high-dimensional hyperparameter spaces, where they often perform comparably to random search (Li et al., 2016b). This limitation has both been shown in practice (Li et al., 2016b), as well as studied theoretically (Srinivas et al., 2009; Grünewälder et al., 2010) and is due to the necessity of sampling enough hyperparameter configurations to get a good estimate of the predictive posterior over a high-dimensional space. In practice, this is not an insurmountable obstacle to the fine-tuning of a handful of parameters in a single model, but it is becoming increasingly impractical as the focus of the community shifts from tuning individual hyperparameters to identifying entire ML pipelines consisting of data pre-processing methods, machine learning models *and* their parameters (Feurer et al., 2015).

Our goal in this paper is indeed not only to tune the hyperparameters of a given model, but also to identify which model to use and how to pre-process the data. We do so by leveraging experiments already performed across different datasets $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_D\}$ to solve the optimization problem

$$\underset{\mathcal{M}, \mathcal{P}, \theta_m, \theta_p}{\arg \min} \ \mathscr{L}(\mathcal{M}(\mathcal{P}(\mathbf{x}; \boldsymbol{\theta}_p); \boldsymbol{\theta}_m), \mathbf{y}),$$

where $\mathcal{M}$ is the ML model with hyperparameters $\boldsymbol{\theta}_m$ and $\mathcal{P}$ is the pre-processing method with hyperparameters $\boldsymbol{\theta}_p$. In the rest of the paper, we refer to the combination of pre-processing method, machine learning model and their hyperparameters as a *machine learning pipeline*. Some of the dimensions in ML pipeline space are continuous, some are discrete, some are categorical (e.g. the model dimension can be a choice between a random forest or an SVM), and some are conditioned on another dimension (e.g. the number of trees dimension in a random forest). The mixture of discrete, continuous and conditional dimensions in ML pipelines makes modeling continuity in this space particularly challenging. For this reason, unlike previous work, we consider instantiations of pipelines, meaning that we fix the set of pipelines ahead of training. For example, an instantiated pipeline can consist in computing the top 5 principal components of the input data and then applying a random forest with 1000 trees. Extensive experiments in section 4 demonstrate that this discretization of the space actually leads to better performance than models that attempt to model continuity.

We show that the problem of predicting the performance of ML pipelines on a new dataset can be cast as a collaborative filtering problem that can be solved with probabilistic matrix factorization techniques. The approach we follow in the rest of this paper, based on Gaussian process latent variable models (Lawrence & Urtasun, 2009; Lawrence, 2005), embeds different pipelines in a latent space based on their performance across different datasets. For example, Figure 1 shows the first two dimensions of the latent space of ML pipelines identified by our model on OpenML (Vanschoren et al., 2013) datasets. Each dot corresponds to an ML pipeline and is colored depending on the AUROC achieved on a holdout set for a given OpenML dataset. Since our probabilistic approach produces a full predictive posterior distribution over the performance of the ML pipelines considered, we can use it in conjunction with acquisition functions commonly used in Bayesian optimization to guide the exploration of the ML pipeline space. Through extensive experiments, we show that our method significantly outperforms the current state-of-the-art in automated machine learning in the vast majority of datasets we considered.

2. Related work

The concept of leveraging experiments performed in previous problem instances has been explored in different ways by two different communities. In the Bayesian optimization community, most of the work revolves around either casting this problem as an instance of multi-task learning or by selecting the first parameter settings to evaluate on a new dataset by looking at what worked in related datasets (we will refer to this as meta-learning for cold-start). In the multi-task setting, Swersky et al. (2013) have proposed a multi-task Bayesian optimization approach leveraging multiple related datasets in order to find the best hyperparameter setting for a new task. For instance, they suggested using a smaller dataset to tune the hyperparameters of a bigger dataset that is more expensive to evaluate. Schilling et al. (2015) also treat this problem as an instance of multi-task learning, but instead of treating each dataset as a separate task (or output), they effectively consider the tasks as conditionally independent given an indicator variable specifying which dataset was used to run a given experiment. Springenberg et al. (2016) do something similar with Bayesian neural networks, but instead of passing an indicator variable, their approach learns a dataset-specific embedding vector. Perrone et al. (2017) also effectively learn a task-specific embedding, but instead of using Bayesian neural networks end-to-end like in (Springenberg et al., 2016), they use feedforward neural networks to learn the basis functions of a Bayesian linear regression model.

Other approaches address the cold-start problem by evaluating parameter settings that worked well in previous datasets. The most successful attempt to do so for automated machine learning problems (*i.e.* in very high-dimensional and structured parameter spaces) is the work by Feurer et al. (2015). In their paper, the authors compute meta-features of both the dataset under examination as well as a vari-

ety of OpenML (Vanschoren et al., 2013) datasets. These meta-features include for example the number of classes or the number of samples in each dataset. They measure similarity between datasets by computing the L1 norm of the meta-features and use the optimization runs from the nearest datasets to warm-start the optimization. Reif et al. (2012) also use meta-features of the dataset to cold-start the optimization performed by a genetic algorithm. Wistuba et al. (2015) focus on hyperparameter optimization and extend the approach presented in (Feurer et al., 2015) by also taking into account the performance of hyperparameter configurations evaluated on the new dataset. In the same paper, they also propose to carefully pick these evaluations such that the similarity between datasets is more accurately represented, although they found that this doesn't result in improved performance in their experiments.

Other related work has been produced in the context of algorithm selection for satisfiability problems. In particular, Stern et al. (2010) tackled constraint solving problems and combinatorial auction winner determination problems using a latent variable model to select which algorithm to use. Their model performs a joint linear embedding of problem instances and experts (*e.g.* different SAT solvers) based on their meta-features and a sparse matrix containing the results of previous algorithm runs. Malitsky & O'Sullivan (2014) also proposed to learn a latent variable model by decomposing the matrix containing the performance of each solver on each problem. They then develop a model to project commonly used hand-crafted meta-features used to select algorithms onto the latent space identified by their model. They use this last model to do one-shot (*i.e.* non-iterative) algorithm selection. This is similar to what was done by Mısır & Sebag (2017), but they do not use the second regression model and instead perform one-shot algorithm selection directly.

Our work is most related to (Feurer et al., 2015) in terms of scope (*i.e.* joint automated pre-processing, model selection and hyperparameter tuning), but we discretize the space and set up a multi-task model, while they capture continuity in parameter space in a single-task model with a smart initialization. Our approach is also loosely related to the work of Stern et al. (2010), but we perform sequential model based optimization with a non-linear mapping between latent and observed space in an unsupervised model, while they use a supervised linear model trained on ranks for one-shot algorithm selection. The application domain of their model also required a different utility function and a time-based feedback model.

3. AutoML as probabilistic matrix factorization

In this paper, we develop a method that can draw information from *all* of the datasets for which experiments are available, whether they are immediately related (*e.g.* a smaller version of the current dataset) or not. The idea behind our approach is that if two datasets have similar (*i.e.* correlated) results for a few pipelines, it's likely that the remaining pipelines will produce results that are similar as well. This is somewhat reminiscent of a collaborative filtering problem for movie recommendation, where if two users liked the same movies in the past, it's more likely that they will like similar ones in the future.

More formally, given N machine learning pipelines and D datasets, we train each pipeline on part of each dataset and we evaluate it on an holdout set. This gives us a matrix $\mathbf{Y} \in \mathbb{R}^{N \times D}$ summarizing the performance of each pipeline in each dataset. In the rest of the paper, we will assume that $\mathbf{Y}$ is a matrix of balanced accuracies (see e.g., (Guyon et al., 2015)), and that we want to maximize the balanced accuracy for a new dataset; but, our approach can be used with any loss function (*e.g.* RMSE, balanced error rate, etc.). Having observed the performance of different pipelines on different datasets, the task of predicting the performance of any of them on a new dataset can be cast as a matrix factorization problem.

Specifically, we are seeking a low rank decomposition such that $\mathbf{Y} \approx \mathbf{XW}$, where $\mathbf{X} \in \mathbb{R}^{N \times Q}$ and $\mathbf{W} \in \mathbb{R}^{Q \times D}$, where Q is the dimensionality of the latent space. As done in Lawrence & Urtasun (2009) and Salakhutdinov & Mnih (2008), we consider the probabilistic version of this task, known as *probabilistic matrix factorization*

$$p(\mathbf{Y} \,|\, \mathbf{X}, \mathbf{W}, \sigma^2) = \prod_{n=1}^{N} \mathcal{N}(\mathbf{y}_n \,|\, \mathbf{x}_n \mathbf{W}, \sigma^2 \mathbb{I}), \quad (1)$$

where $\mathbf{x}_n$ is a row of the latent variables $\mathbf{X}$ and $\mathbf{y}_n$ is a vector of measured performances for pipeline n. In this setting both $\mathbf{X}$ and $\mathbf{W}$ are unknown and must be inferred.

3.1. Non-linear matrix factorization with Gaussian process priors

The probabilistic matrix factorization approach just introduced assumes that the entries of Y are linearly related to the latent variables. In nonlinear probabilistic matrix factorization (Lawrence & Urtasun, 2009), the elements of $\mathbf{Y}$ are given by a *nonlinear function* of the latent variables, $y_{n,d} = f_d(\mathbf{x}_n) + \epsilon$, where ϵ is independent Gaussian noise. This gives a likelihood of the form

$$p\left(\mathbf{Y} \,|\, \mathbf{X}, \mathbf{f}, \sigma^2\right) = \prod_{n=1}^{N} \prod_{d=1}^{D} \mathcal{N}\left(y_{n,d} | f_d\left(\mathbf{x}_n\right), \sigma^2\right), \quad (2)$$

Following Lawrence & Urtasun (2009), we place a Gaussian process prior over $f_d(\mathbf{x}_n)$ so that any vector $\mathbf{f}$ is governed by a joint Gaussian density, $p(\mathbf{f} \mid \mathbf{X}) = \mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{K})$, where $\mathbf{K}$ is a covariance matrix, and the elements $\mathbf{K}_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$ encode the degree of correlation between two samples as a function of the latent variables. If we use the covariance function $k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^\top \mathbf{x}_j$, which is a prior corresponding to linear functions, we recover a model equivalent to (1). Alternatively, we can choose a prior over non-linear functions, such as a squared exponential covariance function with automatic relevance determination (ARD, one length-scale per dimension),

$$k(\mathbf{x}_i, \mathbf{x}_j) = \alpha \exp\left(-\frac{\gamma_q}{2}||\mathbf{x}_i - \mathbf{x}_j||^2\right), \qquad (3)$$

where α is a variance (or amplitude) parameter and γ_q are length-scales. The squared exponential covariance function is infinitely differentiable and hence is a prior over very smooth functions. In practice, such a strong smoothness assumption can be unrealistic and is the reason why the Matern class of kernels is sometimes preferred (Williams & Rasmussen, 2006). In the rest of this paper we use the squared exponential kernel and leave the investigation of the performance of Matern kernels to future work.

After specifying a GP prior, the marginal likelihood is obtained by integrating out the function f under the prior

$$p(\mathbf{Y} \mid \mathbf{X}, \boldsymbol{\theta}, \sigma^2) = \int p(\mathbf{Y} \mid \mathbf{X}, \mathbf{f}) \, p(\mathbf{f} \mid \mathbf{X}) \, d\mathbf{f}$$

$$= \prod_{d=1}^{D} \mathcal{N}(\mathbf{y}_{:,d} \mid \mathbf{0}, \mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbb{I}), \quad (4)$$

where $\boldsymbol{\theta} = \{\alpha, \gamma_1, \ldots, \gamma_q\}$.

In principle, we could add metadata about the pipelines and/or the datasets by adding additional kernels. As we discuss in section 4 and show in Figures 2 and 3, we didn't find this to help in practice, since the latent variable model is able to capture all the necessary information even in the fully unsupervised setting.

3.2. Inference with missing data

Running multiple pipelines on multiple datasets is an embarrassingly parallel operation, and our proposed method readily takes advantage of these kinds of computationally cheap observations. However, in applications where it is expensive to gather such observations, $\mathbf{Y}$ will be a sparse matrix, and it becomes necessary to be able to perform inference with missing data. Given that the marginal likelihood in (4) follows a multivariate Gaussian distribution, marginalizing over missing values is straightforward and simply requires "dropping" the missing observations from the mean and covariance. More formally, we define an indexing function $e(d) : \mathbb{N} \to \mathbb{N}^m$ that given a dataset index

d returns the list of m pipelines that have been evaluated on d. We can then rewrite (4) as

$$p(\mathbf{Y} \mid \mathbf{X}, \boldsymbol{\theta}, \sigma^2) = \prod_{d=1}^{D} \mathcal{N}(\mathbf{y}_{e(d),d} \mid \mathbf{0}, \mathbf{C}_d), \qquad (5)$$

where $\mathbf{C}_d = \mathbf{K}(\mathbf{X}_{e(d)}, \mathbf{X}_{e(d)}) + \sigma^2 \mathbb{I}$.

As done in Lawrence & Urtasun (2009), we infer the parameters $\boldsymbol{\theta}, \sigma$ and latent variables $\mathbf{X}$ by minimizing the log-likelihood using stochastic gradient descent. We do so by presenting the entries $\mathbf{Y}_{e(d),d}$ one at a time and updating $\mathbf{X}_{e(d)}, \boldsymbol{\theta}$ and σ for each dataset d. The negative log-likelihood of the model can be written as

$$\mathbf{L} = \sum_{d=1}^{D} -\text{const.} - \frac{N_d}{2}\log|\mathbf{C}_d| - \frac{1}{2}(\mathbf{y}_{e(d),d}^\top \mathbf{C}_d^{-1} \mathbf{y}_{e(d),d}),$$
$$(6)$$

where N_d is the number of pipelines evaluated for dataset d. For every dataset d we update the global parameters $\boldsymbol{\theta}$ as well as the latent variables $\mathbf{X}_{e(d)}$ by evaluating at the t-th iteration:

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \eta\frac{\partial \mathbf{L}}{\partial \boldsymbol{\theta}} \qquad (7)$$

$$\mathbf{X}_{e(d)}^{t+1} = \mathbf{X}_{e(d)}^t - \eta\frac{\partial \mathbf{L}}{\partial \mathbf{X}_{e(d)}}, \qquad (8)$$

where η is the learning rate.

3.3. Predictions

Predictions from the model can be easily computed by following the standard derivations for Gaussian process (Williams & Rasmussen, 2006) regression. The predicted performance $y_{m,j}^*$ of pipeline m for a new dataset d is given by

$$p(y_{m,d}^* \mid \mathbf{X}, \boldsymbol{\theta}, \sigma) = \mathcal{N}(y_{m,d}^* \mid \mu_{m,d}, v_{m,d}) \qquad (9)$$

$$\mu_{m,d} = \mathbf{k}_{e(d),m}^\top \mathbf{C}_d^{-1} \mathbf{y}_{e(d),d}$$

$$v_{m,d} = k_{m,m} + \sigma^2 - \mathbf{k}_{e(d),m}^\top \mathbf{C}_d^{-1}\mathbf{k}_{e(d),m},$$

remembering that $\mathbf{C}_d = \mathbf{K}(\mathbf{X}_{e(d)}, \mathbf{X}_{e(d)}) + \sigma^2 \mathbb{I}$ and defining $\mathbf{k}_{e(d),m} = \mathbf{K}(\mathbf{X}_{e(d)}, \mathbf{X}_m)$ and $k_{m,m} = \mathbf{K}(\mathbf{X}_m, \mathbf{X}_m)$.

The computational complexity for generating these predictions is largely determined by the number of pipelines already evaluated for a test dataset and is due to the inversion of a $N_j \times N_j$ matrix. This is not particularly onerous because the typical number of evaluations is likely to be in the hundreds, given the cost of training each pipeline and the risk of overfitting to the validation set if too many pipelines are evaluated.
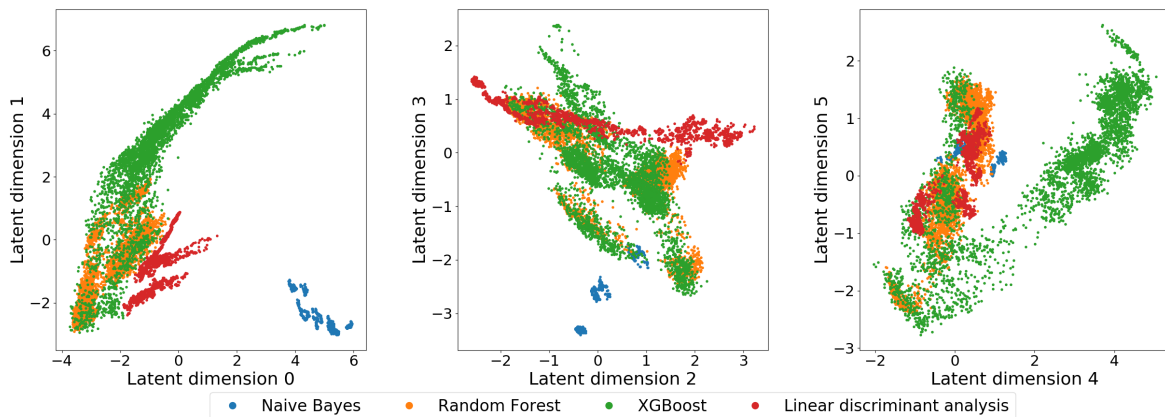
Figure 2: Latent embeddings of 42,000 machine learning pipelines colored according to which model was included in each pipeline. These are paired plots of the first 5 dimensions of our 20-dimensional latent space. The latent space effectively captures structure in the space of models.

3.4. Acquisition functions

The model described so far can be used to predict the expected performance of each ML pipeline as a function of the pipelines already evaluated, but does not yet give any guidance as to which pipeline should be tried next. A simple approach to pick the next pipeline to evaluate is to iteratively pick the maximum predicted performance arg $\max_{m} (\mu_{m,d})$, but such a utility function, also known as an acquisition function, would discard information about the uncertainty of the predictions. One of the most widely used acquisition functions is expected improvement (EI) (Močkus, 1975), which is given by the expectation of the improvement function

$$I(y^*_{m,d}, y_{best}) \triangleq (y^*_{m,d} - y_{best})\mathbb{I}(y^*_{m,d} > y_{best})$$

$$\mathtt{EI}_{m,d} \triangleq \mathbb{E}[I(y^*_{m,d}, y_{best})],$$

where y_{best} is the best result observed. Since $y^*_{m,j}$ is Gaussian distributed (see (9)), this expectation can be computed analytically

$$\mathtt{EI}_{m,d} = v_{m,d}\left[\gamma_{m,d}\Phi(\gamma_{m,d} + \mathcal{N}(\gamma_{m,d} \,|\, 0, 1))\right],$$

where Φ is the cumulative distribution function of the standard normal and $\gamma_{m,j}$ is defined as

$$\gamma_{m,d} = \frac{\mu_{m,d} - y_{best} - \xi}{v_{m,d}},$$

where ξ is a free parameter to encourage exploration. After computing the expected improvement for each pipeline, the next pipeline to evaluate is simply given by
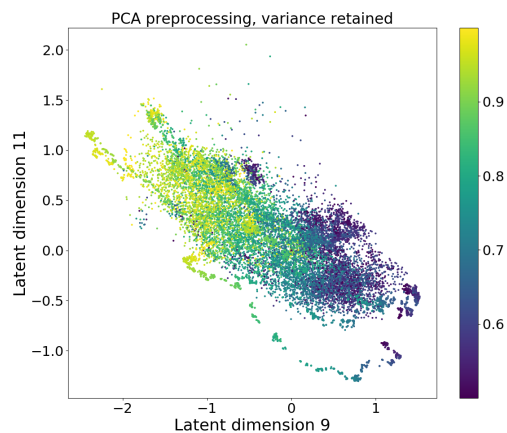
$$\arg \max_{m} (\mathtt{EI}_{m,d}).$$

Figure 3: Latent embedding of all the pipelines in which PCA is included as a pre-processor. Each point is colored according to the percentage of variance retained by PCA (*i.e.* the hyperparameter of interest when tuning PCA in ML pipelines).

The expected improvement is just one of many possible acquisition functions, and different problems may require different acquisition functions. See (Shahriari et al., 2016) for a review.

4. Experiments

In this section, we compare our method to a series of baselines as well as to auto-sklearn (Feurer et al., 2015), the current state-of-the-art approach and overall winner of the ChaLearn AutoML competition (Guyon et al., 2016).

We ran all of the experiments on 553 OpenML (Vanschoren et al., 2013) datasets, selected by filtering for binary and multi-class classification problems with no more than 10, 000 samples and no missing values, although our method is capable of handling datasets which cause ML pipeline runs to be unsuccessful (described below).

4.1. Generation of training data

We generated training data for our method by splitting each OpenML dataset in 80% training data, 10% validation data and 10% test data, running 42, 000 ML pipelines on each dataset and measuring the balanced accuracy (*i.e.* accuracy rescaled such that random performance is 0 and perfect performance is 1.0).

We generated the pipelines by sampling a combination of pre-processors $\mathcal{P} = \{P^1, P^2, ..., P^n\}$, machine learning models $\mathcal{M} = \{M^1, M^2, ..., M^m\}$, and their corresponding hyperparameters $\Theta_P = \{\theta_P^1, ..., \theta_P^n\}$ and $\Theta_M = \{\theta_M^1, ..., \theta_M^m\}$ from the entries in Supplementary Table 1. All the models and pre-processing methods we considered were implemented in scikit-learn (Pedregosa et al., 2011). We sampled the parameter space by using functions provided in the auto-sklearn library (Feurer et al., 2015). Similar to what was done in (Feurer et al., 2015), we limited the maximum training time of each individual model within a pipeline to 30 seconds and its memory consumption to 16GB. Because of network failures and the cluster occasionally running out of memory, the resulting matrix $\mathbf{Y}$ was not fully sampled and had approximately 21% missing entries. As pointed out in the previous section, this is expected in realistic applications and is not a problem for our method, since it can easily handle sparse data.

Out of the 553 total datasets, we selected 100 of them as a held out test set. We found that some of the OpenML datasets are so easy to model, that most of the machine learning pipelines we tried worked equally well. Since this could swamp any difference between the different methods we were evaluating, we chose our test set taking into consideration the difficulty of each dataset. We did so by randomly drawing without replacement each dataset with probabilities proportional to how poorly random selection performed on it. Specifically, for each dataset, we ran random search for 300 iterations and recorded the regret. The probability of selecting a dataset was then proportional to the regret on that dataset, averaged over 100 trials of random selection. After removing OpenML datasets that were used to train auto-sklearn, the final size of the held out test set was 89. The training set consisted of the remaining 464 datasets (the IDs of both training and test sets are provided in the supplementary material).
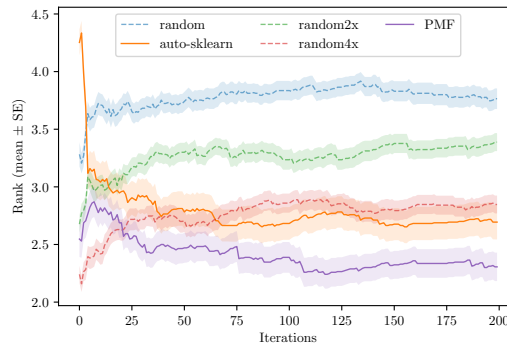

Figure 4: Average rank of all the approaches we considered as a function of the number of iterations. For each holdout dataset, the methods are ranked based on the balanced accuracy obtained on the validation set at each iteration. The ranks are then averaged across datasets. Lower is better. The shaded areas represent the standard error for each method.

4.2. Parameter settings

We set the number of latent dimensions to $Q = 20$, learning rate to $\eta = 1e^{-7}$, and (column) batch-size to 50. The latent space was initialized using PCA, and training was run for 300 epochs (corresponding to approximately 3 hours on a 16-core Azure machine). Finally, we configured the acquisition function with $\xi = 0.01$.

4.3. Results

We compared the model described in this paper, **PMF**, to the following methods:

- **Random**. For each test dataset, we performed a random search by sampling each pipeline to be evaluated from the set of 42,000 at random without replacement.

- **Random 2x**. Same as above, but with twice the budget. This simulates parallel evaluation of pipelines and is a strong baseline (Li et al., 2016a).

- **Random 4x**. Same as a above but with 4 times the budget.

- **auto-sklearn** (Feurer et al., 2015). We ran auto-sklearn for 4 hours per dataset and set to optimize balanced accuracy on a holdout set. We disabled the automated ensembling of models in order to obtain a fair comparison to the other non-ensembling methods.

Our method uses the same procedure used in (Feurer et al., 2015) to "warm-start" the process by selecting the first 5 pipelines, after which the acquisition function selects subsequent pipelines.
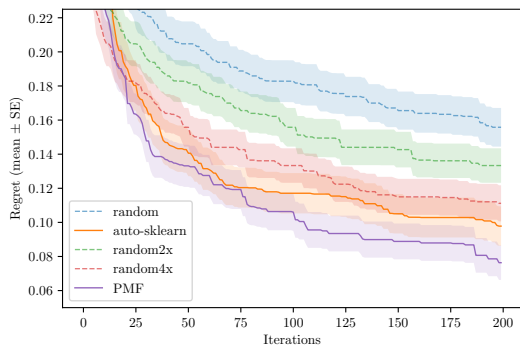
Figure 5: Difference between the maximum balanced accuracy observed on the test set and the balanced accuracy obtained by each method at each iteration. Lower is better. The shaded areas represent the standard error for each method.
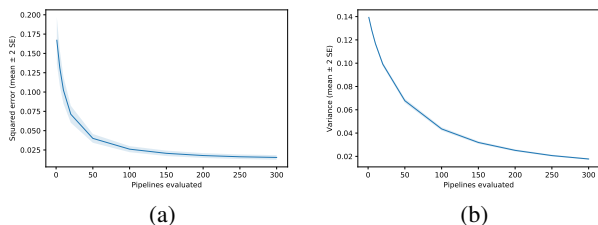

Figure 6: (a) Mean squared error (MSE) between predicted and observed balanced accuracies in the test set as a function of the number of iterations. Lower is better. MSE is averaged across all test datasets. (b) Posterior predictive variance as a function of the number of iterations and averaged across all test datasets. Shaded area shows two standard errors around the mean.

Figure 4 shows the average rank for each method as a function of the number of iterations (*i.e.* the number of pipelines evaluated). Starting from the first iteration, our approach consistently achieves the best average rank. Auto-sklearn is the second best model, outperforming random 2x and almost matched by random 4x. Please note that random 2x and random 4x are only intended as baselines that are easy to understand and interpret, but that in no way can be considered practical solutions, since they both have a much larger computational budget than the non-baseline methods.

Rank plots such as Figure 4 are useful to understand the relative performance of a set of models, but they don't give any information about the magnitude of the difference in performance. For this reason, we measured the difference between the maximum balanced accuracy obtained by any pipeline in each dataset and the one obtained by the pipeline

selected at each iteration. The results summarized in Figure 5 show that our method still outperforms all the others. We also investigated how well our method performs when fewer observations/training datasets are available. In the first experiment, we ran our method in the setting where 90% of the entries in $\mathbf{Y}$ are missing. Supplementary Figures 1 and 2 demonstrate our method degrades in performance only slightly, but still results in the best performance amongst competitors. In the second experiment, we matched the number (and the identity, for the most part) of datasets that auto-sklearn uses to initialize its Bayesian optimization procedure. The results, shown in Supplementary Figures 3 and 4, confirm that our model outperforms competing approaches even when trained on a subset of the data.

Next, we investigated how quickly our model is able to improve its predictions as more pipelines are evaluated. Figure 6a shows the mean squared error computed across the test datasets as a function of the number of evaluations. As expected the error monotonically decreases and appears to asymptote after 200 iterations. Figure 6b shows the uncertainty of the model (specifically, the posterior variance) as a function of the number of evaluations. Overall, Figure 6 a and b support that as more evaluations are performed, the model becomes less uncertain and the accuracy of the predictions increases.

Including pipeline metadata. Our approach can easily incorporate information about the composition and the hyperparameters of the pipelines considered. This metadata could for example include information about which model is used within each pipeline or which pre-processor is applied to the data before passing it to the model. Empirically, we found that including this information in our model didn't improve performance (data not shown). Indeed, our model is able to effectively capture most of this information in a completely unsupervised fashion, just by observing the sparse pipelines-dataset matrix $\mathbf{Y}$. This is visible in Figure 2, where we show the latent embedding colored according to which model was included in which pipeline. On a finer scale, the latent space can also capture different settings of an individual hyperparameter. This is shown in Figure 3, where each pipeline is embedded in a 2-dimensional space and colored by the value of the hyperparameter of interest, in this case the percent of variance retained by a PCA preprocessor. Overall, our findings indicate that pipeline metadata is not needed by our model if enough experimental data (*i.e.* enough entries in matrix $\mathbf{Y}$) is available.

5. Discussion

We have presented a new approach to automatically build predictive ML pipelines for a given dataset, automating the selection of data pre-processing method and machine learning model as well as the tuning of their hyperparameters.

Our approach combines techniques from collaborative filtering and ideas from Bayesian optimization to intelligently explore the space of ML pipelines, exploiting experiments performed in previous datasets. We have benchmarked our approach against the state-of-the-art in 89 OpenML datasets with different sample sizes, number of features and number of classes. Overall, our results show that our approach outperforms both the state-of-the-art as well as a set of strong baselines.

One potential concern with our method is that it requires sampling (*i.e.* instantiating pipelines) from a potentially high-dimensional space and thus could require exponentially many samples in order to explore all areas of this space. We have found this not to be a problem for three reasons. First, many of the dimensions in the space of pipelines are conditioned on the choice of other dimensions. For example, the number of trees or depth of a random forest are parameters that are only relevant if a random forest is chosen in the "model" dimension. This reduces the effective search space significantly. Second, in our model we treat every pipeline as an additional sample, so increasing the sampling density also results in an increase in sample size (and similarly, adding a dataset also increases the effective sample size). Finally, very dense sampling of the pipeline space is only needed if the performance is very sensitive to small parameter changes, something that we haven't observed in practice. If this is a concern, we advise using our approach in conjunction with traditional Bayesian optimization methods (such as (Snoek et al., 2012)) to further fine-tune the parameters.

We are currently investigating several extensions of this work. First, we would like to include dataset-specific information in our model. As discussed in section 3, the only data taken into account by our model is the performance of each method in each dataset. Similarity between different pipelines is induced by having correlated performance across multiple datasets, and ignores potentially relevant metadata about datasets, such as the sample size or number of classes. We are currently working on including such information by extending our model using additional kernels and dual embeddings (*i.e.* embedding both pipelines and dataset in separate latent spaces). Second, we are interested in using acquisition functions that include a factor representing the computational cost of running a given pipeline (Snoek et al., 2012) to handle instances when datasets have a large number of samples. The machine learning models we used for our experiments were constrained not to exceed a certain runtime, but this could be impractical in real applications. Finally, we are planning to experiment with different probabilistic matrix factorization models based on variational autoencoders.

References

Bergstra, James and Bengio, Yoshua. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

Bergstra, James, Yamins, Daniel, and Cox, David D. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. *Proceedings of the International Conference on Machine Learning*, 28:115–123, 2013.

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

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

Grünewälder, Steffen, Audibert, Jean-Yves, Opper, Manfred, and Shawe-Taylor, John. Regret bounds for Gaussian process bandit problems. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, pp. 273–280, 2010.

Guyon, Isabelle, Bennett, Kristin, Cawley, Gavin, Escalante, Hugo Jair, Escalera, Sergio, Ho, Tin Kam, Macia, Núria, Ray, Bisakha, Saeed, Mehreen, Statnikov, Alexander, et al. Design of the 2015 chalearn automl challenge. In *Neural Networks (IJCNN), 2015 International Joint Conference on*, pp. 1–8. IEEE, 2015.

Guyon, Isabelle, Chaabane, Imad, Escalante, Hugo Jair, Escalera, Sergio, Jajetic, Damir, Lloyd, James Robert, Maci, Nria, Ray, Bisakha, Romaszko, Lukasz, Sebag, Michle, Statnikov, Alexander, Treguer, Sbastien, and Viegas, Evelyne. A brief review of the chalearn automl challenge: Any-time any-dataset learning without human intervention. In Hutter, Frank, Kotthoff, Lars, and Vanschoren, Joaquin (eds.), *Proceedings of the Workshop on Automatic Machine Learning*, volume 64 of *Proceedings of Machine Learning Research*, pp. 21–30, New York, New York, USA, 24 Jun 2016. PMLR. URL http://proceedings.mlr.press/v64/guyon_review_2016.html.

Hutter, Frank, Hoos, Holger H, and Leyton-Brown, Kevin. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pp. 507–523. Springer, 2011.

Lawrence, Neil. Probabilistic non-linear principal component analysis with Gaussian process latent variable

models. *Journal of Machine Learning Research*, 6(Nov): 1783–1816, 2005.

Lawrence, Neil and Urtasun, Raquel. Non-linear matrix factorization with Gaussian processes. *Proceedings of the International Conference on Machine Learning*, 2009.

Li, Lisha, Jamieson, Kevin, DeSalvo, Giulia, Rostamizadeh, Afshin, and Talwalkar, Ameet. Hyperband: A novel Bandit-Based approach to hyperparameter optimization. March 2016a.

Li, Lisha, Jamieson, Kevin, DeSalvo, Giulia, Rostamizadeh, Afshin, and Talwalkar, Ameet. Efficient hyperparameter optimization and infinitely many armed bandits. *arXiv preprint arXiv:1603.06560*, 2016b.

Malitsky, Yuri and O'Sullivan, Barry. Latent features for algorithm selection. In *Seventh Annual Symposium on Combinatorial Search*, July 2014.

Mısır, Mustafa and Sebag, Michèle. Alors: An algorithm recommender system. *Artif. Intell.*, 244:291–314, March 2017.

Močkus, J. On Bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*, pp. 400–404. Springer, 1975.

Osborne, Michael A, Garnett, Roman, and Roberts, Stephen J. Gaussian processes for global optimization. In *3rd International Conference on Learning and Intelligent Optimization (LION3)*, pp. 1–15, 2009.

Pedregosa, Fabian, Varoquaux, Gaël, Gramfort, Alexandre, Michel, Vincent, Thirion, Bertrand, Grisel, Olivier, Blondel, Mathieu, Prettenhofer, Peter, Weiss, Ron, Dubourg, Vincent, Vanderplas, Jake, Passos, Alexandre, Cournapeau, David, Brucher, Matthieu, Perrot, Matthieu, and Duchesnay, Édouard. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830, 2011.

Perrone, Valerio, Jenatton, Rodolphe, Seeger, Matthias, and Archambeau, Cedric. Multiple adaptive bayesian linear regression for scalable bayesian optimization with warm start. December 2017.

Reif, Matthias, Shafait, Faisal, and Dengel, Andreas. Meta-learning for evolutionary parameter optimization of classifiers. *Mach. Learn.*, 87(3):357–380, June 2012.

Salakhutdinov, Ruslan and Mnih, Andriy. Bayesian probabilistic matrix factorization using Markov chain Monte Carlo. In *Proceedings of the 25th International Conference on Machine Learning*, pp. 880–887, 2008.

Schilling, Nicolas, Wistuba, Martin, Drumond, Lucas, and Schmidt-Thieme, Lars. Hyperparameter optimization with factorized multilayer perceptrons. In *Machine Learning and Knowledge Discovery in Databases*, Lecture Notes in Computer Science, pp. 87–103. Springer, Cham, September 2015.

Shahriari, Bobak, Swersky, Kevin, Wang, Ziyu, Adams, Ryan P, and de Freitas, Nando. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

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

Springenberg, Jost Tobias, Klein, Aaron, Falkner, Stefan, and Hutter, Frank. Bayesian optimization with robust bayesian neural networks. In Lee, D D, Sugiyama, M, Luxburg, U V, Guyon, I, and Garnett, R (eds.), *Advances in Neural Information Processing Systems 29*, pp. 4134–4142. Curran Associates, Inc., 2016.

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

Stern, D H, Samulowitz, H, Herbrich, R, Graepel, T, and others. Collaborative expert portfolio management. *AAAI*, 2010.

Swersky, Kevin, Snoek, Jasper, and Adams, Ryan P. Multi-task Bayesian optimization. In *Advances in Neural Information Processing Systems*, pp. 2004–2012, 2013.

Vanschoren, Joaquin, van Rijn, Jan N., Bischl, Bernd, and Torgo, Luis. OpenML: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013.

Williams, Christopher KI and Rasmussen, Carl Edward. Gaussian processes for machine learning. *The MIT Press, Cambridge, MA, USA*, 2006.

Wistuba, Martin, Schilling, Nicolas, and Schmidt-Thieme, Lars. Learning data set similarities for hyperparameter optimization initializations. In *MetaSel@ PKDD/ECML*, pp. 15–26, 2015.

ML / PP Algorithm	Parameter	Range
Polynomial Features	`degree`	`[2, 3]`
Polynomial Features	`interaction_only`	`{False, True}`
Polynomial Features	`include_bias`	`{True, False}`
Principal Component Analysis	`keep_variance`	`[0.5, 0.9999]`
Principal Component Analysis	`whiten`	`{False, True}`
Linear Discriminant Analysis	`shrinkage`	`{None, auto, manual}`
Linear Discriminant Analysis	`n_components`	`[1, 250]`
Linear Discriminant Analysis	`tol`	`[1e-05, 0.1]`
Linear Discriminant Analysis	`shrinkage_factor`	`[0.0, 1.0]`
Extreme Gradient Boosting	`max_depth`	`[1, 10]`
Extreme Gradient Boosting	`learning_rate`	`[0.01, 1.0]`
Extreme Gradient Boosting	`n_estimators`	`[50, 500]`
Extreme Gradient Boosting	`subsample`	`[0.01, 1.0]`
Extreme Gradient Boosting	`min_child_weight`	`[1, 20]`
Quadratic Discriminant Analysis	`reg_param`	`[0.0, 10.0]`
Extra Trees	`criterion`	`{gini, entropy}`
Extra Trees	`max_features`	`[0.5, 5.0]`
Extra Trees	`min_samples_split`	`[2, 20]`
Extra Trees	`min_samples_leaf`	`[1, 20]`
Extra Trees	`bootstrap`	`{True, False}`
Decision Tree	`criterion`	`{gini, entropy}`
Decision Tree	`max_depth`	`[0.0, 2.0]`
Decision Tree	`min_samples_split`	`[2, 20]`
Decision Tree	`min_samples_leaf`	`[1, 20]`
Gradient Boosted Decision Trees	`learning_rate`	`[0.01, 1.0]`
Gradient Boosted Decision Trees	`n_estimators`	`[50, 500]`
Gradient Boosted Decision Trees	`max_depth`	`[1, 10]`
Gradient Boosted Decision Trees	`min_samples_split`	`[2, 20]`
Gradient Boosted Decision Trees	`min_samples_leaf`	`[1, 20]`
Gradient Boosted Decision Trees	`subsample`	`[0.01, 1.0]`
Gradient Boosted Decision Trees	`max_features`	`[0.5, 5.0]`
K Neighbors	`n_neighbors`	`[1, 100]`
K Neighbors	`weights`	`{uniform, distance}`
K Neighbors	`p`	`{1, 2}`
Multinomial Naive Bayes	`alpha`	`[0.01, 100.0]`
Multinomial Naive Bayes	`fit_prior`	`{True, False}`
Support Vector Machine	`C`	`[0.03125, 32768.0]`
Support Vector Machine	`kernel`	`{rbf, poly, sigmoid}`
Support Vector Machine	`gamma`	`[3.05176e-05, 8.0]`
Support Vector Machine	`shrinking`	`{True, False}`
Support Vector Machine	`tol`	`[1e-05, 0.1]`
Support Vector Machine	`coef0`	`[-1.0, 1.0]`
Support Vector Machine	`degree`	`[1, 5]`
Random Forest	`criterion`	`{gini, entropy}`
Random Forest	`max_features`	`[0.5, 5.0]`
Random Forest	`min_samples_split`	`[2, 20]`
Random Forest	`min_samples_leaf`	`[1, 20]`
Random Forest	`bootstrap`	`{True, False}`
Bernoulli Naive Bayes	`alpha`	`[0.01, 100.0]`
Bernoulli Naive Bayes	`fit_prior`	`{True, False}`

Table 1: List of preprocessing methods, ML models/algorithms and parameters considered.

The following is the list of OpenML dataset IDs used to train the proposed method in the main paper:

```
[3, 6, 10, 11, 12, 14, 16, 18, 20, 21, 22, 26, 28, 30, 31, 32, 36, 39, 41, 43,
44, 46, 50, 54, 59, 60, 61, 62, 151, 155, 161, 162, 164, 180, 181, 182, 183, 184,
187, 189, 209, 223, 225, 227, 230, 275, 277, 287, 292, 294, 298, 300, 307, 310,
```

```
312, 313, 329, 333, 334, 335, 336, 338, 339, 343, 346, 375, 377, 383, 385, 386,
387, 389, 391, 392, 395, 400, 401, 444, 446, 448, 450, 457, 458, 461, 462, 463,
464, 465, 467, 468, 469, 472, 476, 477, 478, 479, 480, 679, 682, 685, 694, 713,
715, 716, 717, 718, 719, 720, 721, 722, 723, 725, 727, 728, 729, 730, 732, 734,
735, 737, 741, 742, 743, 744, 745, 746, 747, 748, 749, 751, 752, 754, 755, 756,
758, 759, 761, 762, 765, 766, 767, 768, 769, 770, 772, 775, 776, 777, 778, 779,
780, 782, 784, 785, 787, 788, 790, 791, 792, 793, 794, 795, 796, 797, 801, 803,
804, 805, 807, 808, 811, 813, 814, 815, 816, 817, 818, 819, 820, 821, 823, 824,
827, 828, 829, 830, 832, 833, 834, 835, 837, 841, 843, 845, 846, 847, 848, 849,
850, 853, 855, 857, 859, 860, 863, 864, 865, 866, 867, 868, 870, 871, 872, 873,
874, 875, 877, 878, 879, 880, 881, 882, 884, 885, 886, 889, 890, 892, 894, 895,
900, 901, 903, 905, 910, 912, 913, 914, 915, 916, 917, 919, 921, 922, 923, 924,
925, 928, 932, 933, 934, 935, 936, 937, 938, 941, 942, 943, 946, 947, 950, 951,
952, 953, 954, 955, 956, 958, 959, 962, 964, 965, 969, 970, 971, 973, 974, 976,
977, 978, 979, 980, 983, 987, 988, 991, 994, 995, 997, 1004, 1005, 1006, 1009,
1011, 1013, 1014, 1015, 1016, 1019, 1020, 1021, 1022, 1025, 1026, 1038, 1040,
1041, 1043, 1044, 1045, 1046, 1048, 1055, 1056, 1059, 1060, 1061, 1062, 1063,
1064, 1065, 1066, 1068, 1069, 1075, 1079, 1081, 1082, 1104, 1106, 1107, 1115,
1116, 1120, 1121, 1122, 1123, 1124, 1125, 1126, 1127, 1129, 1131, 1132, 1133,
1135, 1136, 1137, 1140, 1141, 1143, 1144, 1145, 1147, 1148, 1149, 1150, 1151,
1152, 1153, 1154, 1155, 1156, 1157, 1158, 1160, 1162, 1163, 1165, 1167, 1169,
1217, 1236, 1237, 1238, 1413, 1441, 1442, 1443, 1444, 1446, 1448, 1449, 1450,
1451, 1452, 1454, 1455, 1457, 1459, 1460, 1464, 1467, 1471, 1475, 1481, 1482,
1486, 1488, 1489, 1496, 1498, 1500, 1501, 1505, 1507, 1508, 1509, 1510, 1516,
1517, 1519, 1520, 1527, 1528, 1529, 1530, 1531, 1532, 1533, 1534, 1535, 1536,
1537, 1538, 1539, 1540, 1541, 1542, 1544, 1545, 1546, 1556, 1557, 1561, 1562,
1563, 1564, 1565, 1567, 1568, 1569, 4134, 4135, 4153, 4340, 4534, 4538, 40474,
40475, 40476, 40477, 40478, 1050, 1067, 740, 398, 23, 1036, 1049, 799, 822, 904,
806]
```

The list of OpenML dataset IDs used in the held out test set was:

```
[733, 812, 731, 929, 1600, 475, 726, 197, 394, 1472, 1159, 763, 1483, 1080, 836,
851, 911, 459, 37, 40, 927, 887, 783, 1012, 764, 714, 285, 1117, 384, 888, 1447,
1100, 789, 48, 1054, 1164, 838, 869, 931, 876, 1073, 1071, 750, 1518, 948, 736,
896, 1503, 278, 279, 908, 724, 996, 891, 926, 337, 909, 826, 800, 1487, 1512,
945, 825, 949, 753, 774, 906, 902, 1473, 8, 862, 920, 1078, 683, 1084, 1412, 53,
276, 1543, 907, 397, 918, 771, 773, 1077, 1453, 893, 1513, 388]
```

For the first additional experiment, we add results from (i) the Factorized MLP of (Schilling et al., 2015), and (ii) training the proposed method on an observation matrix with 90% of the entries missing. For the latter, we start with the original observation matrix for the training data, which has 20% missing entries, drop an additional 70% uniformly at random, and train with Adam using a learning rate of 10^{-2} and $Q = 5$ for the latent dimensionality. The test set remains unchanged from the experiment in the main paper. Figures 1 and 2 show the results overlaid onto the results from the experiment in the main paper. We can see the performance of the factorized MLP to be between random and random-2x, even after tuning its hyperparameters to improve performance. This is significantly worse than both auto-sklearn and the proposed method. Our method degrades in performance only slightly when 10% of the observations are available versus when training on 80% available observations, but still out-performs auto-sklearn demonstrating very good robustness to missing data. Although not shown, we also ran the proposed method with $\xi = 0$ in the acquisition function, and this did not produce any distinguishable effect in our results. Our method even with $\xi = 0$ still achieves the best regret and rank.
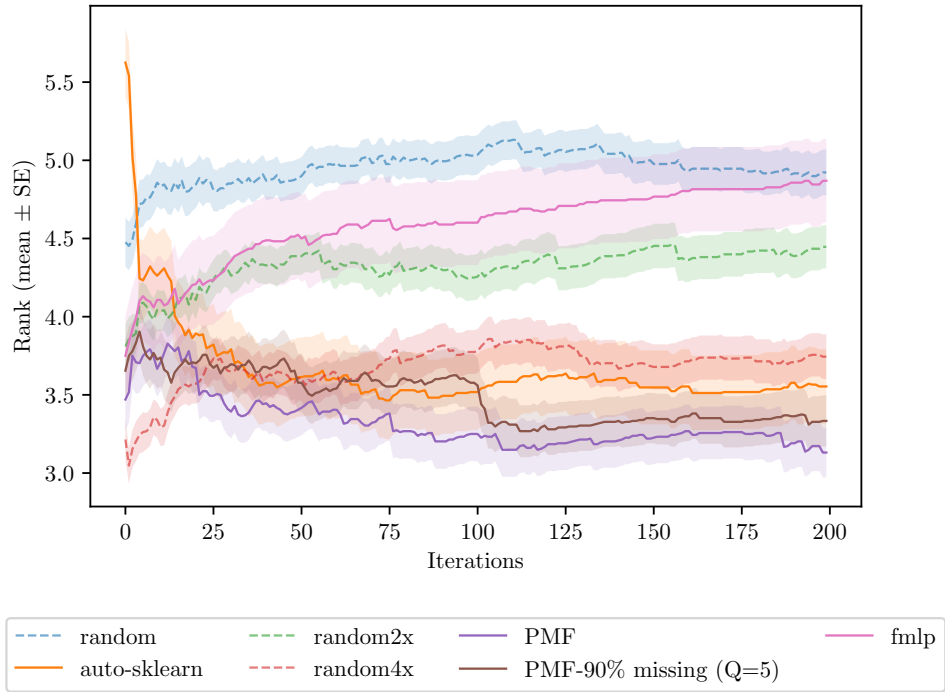
Figure 1: Average rank of all the approaches we considered as a function of the number of iterations. For each holdout dataset, the methods are ranked based on the balanced accuracy obtained on the validation set at each iteration. The ranks are then averaged across datasets. Lower is better. The shaded areas represent the standard error for each method.
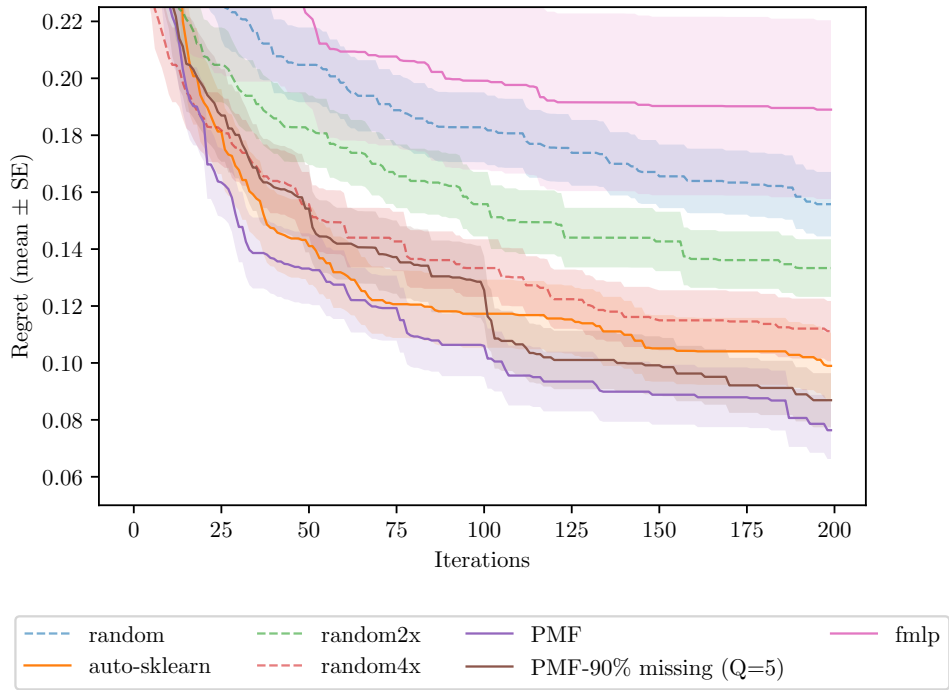

Figure 2: Difference between the maximum balanced accuracy observed on the test set and the balanced accuracy obtained by each method at each iteration. Lower is better. The shaded areas represent the standard error for each method.

For the second additional experiment, we trained the proposed method on 93 of the datasets used to train auto-sklearn and an additional 47 datasets selected uniformly at random from the training set of the main paper. The model and training parameters were 20 latent dimensions and a learning rate of 10^{-5}. The evaluation was performed on the same held out test set. Figures 3 and 4 show the outcome of this experiment and demonstrate that our method still outperforms auto-sklearn. For this experiment, the list of OpenML dataset IDs used to train our method was

```
[3, 6, 12, 14, 16, 18, 21, 22, 26, 28, 30, 31, 32, 36, 44, 46, 60, 180, 181, 182,
184, 300, 389, 391, 392, 395, 401, 679, 715, 718, 720, 722, 723, 727, 728, 734,
735, 737, 741, 743, 751, 752, 761, 772, 797, 803, 807, 813, 816, 819, 821, 823,
833, 837, 843, 845, 846, 847, 849, 866, 871, 881, 901, 903, 910, 912, 913, 914,
917, 923, 934, 953, 958, 959, 962, 971, 976, 977, 978, 979, 980, 991, 995, 1019,
1020, 1021, 1040, 1041, 1056, 1068, 1069, 1116, 1120, 310, 1132, 685, 824, 1015,
1541, 50, 890, 1014, 1446, 747, 875, 1459, 721, 900, 878, 1236, 40478, 1562,
1079, 1496, 1449, 988, 796, 162, 811, 1145, 776, 457, 476, 1482, 1529, 1127, 952,
740, 1043, 1546, 4135, 1022, 853, 1237, 758, 827, 814, 450, 155, 462]
```
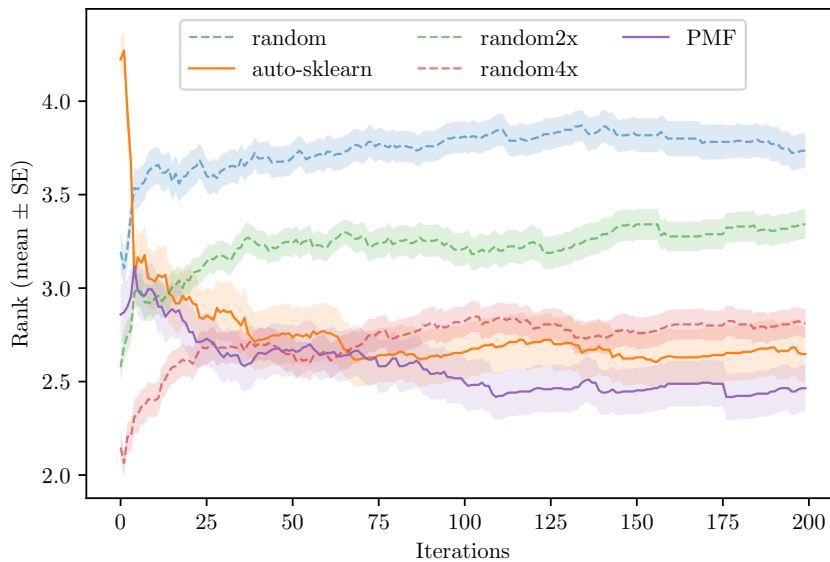

Figure 3: Average rank of all the approaches we considered as a function of the number of iterations. For each holdout dataset, the methods are ranked based on the balanced accuracy obtained on the validation set at each iteration. The ranks are then averaged across datasets. Lower is better. The shaded areas represent the standard error for each method.
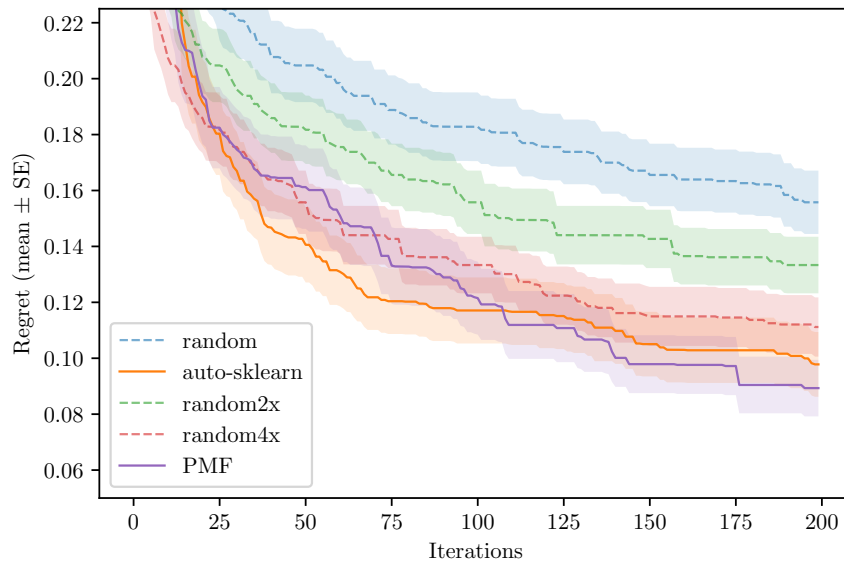
Figure 4: Difference between the maximum balanced accuracy observed on the test set and the balanced accuracy obtained by each method at each iteration. Lower is better. The shaded areas represent the standard error for each method.