HyperImpute:
Generalized Iterative Imputation with Automatic Model Selection

Daniel Jarrett 1 2, Bogdan Cebere 1 2, Tennison Liu 1, Alicia Curth 1, Mihaela van der Schaar 1 2

Abstract
Consider the problem of imputing missing values in a dataset. On the one hand, conventional approaches using iterative imputation benefit from the simplicity and customizability of learning conditional distributions directly, but suffer from the practical requirement for appropriate model specification of each and every variable. On the other hand, recent methods using deep generative modeling benefit from the capacity and efficiency of learning with neural network function approximators, but are often difficult to optimize and rely on stronger data assumptions. In this work, we study an approach that marries the advantages of both: We propose HyperImpute, a generalized iterative imputation framework for adaptively and automatically configuring column-wise models and their hyperparameters. Practically, we provide a concrete implementation with out-of-the-box learners, optimizers, simulators, and extensible interfaces. Empirically, we investigate this framework via comprehensive experiments and sensitivities on a variety of public datasets, and demonstrate its ability to generate accurate imputations relative to a strong suite of benchmarks. Contrary to recent work, we believe our findings constitute a strong defense of the iterative imputation paradigm.

https://github.com/vanderschaarlab/hyperimpute

1. Introduction
Missing data is a ubiquitous problem in real-life data collection. For instance, certain characteristics of a patient may not have been recorded properly during their visit, but we may nevertheless be interested in knowing the values those variables most likely took on [1–4]. Here, we consider precisely this problem of imputing the missing values in a dataset. Specifically, consider the general case where different records in a dataset may contain missing values for different variables, so no columns are assumed to be complete.

Most popular approaches fall into two main categories. On the one hand, conventional approaches using iterative imputation operate by estimating the conditional distributions of each feature on the basis of all the others, and missing values are imputed using such univariate models in a round-robin fashion until convergence [5–10]. In theory, this approach affords great customizability in creating multivariate models: by simply specifying univariate models, one can easily and implicitly work with joint models outside any known parametric multivariate density [10–12]. In practice, however, this strategy often suffers from the requirement that models for every single variable be properly specified. In particular, for each column with missing values, one needs to choose the functional form for the model, select the set of regressors as input, include any interaction terms of interest, add appropriate regularization, or handle derived variables separately to prevent collinearity in the common case of linear models; this is time-consuming and dependent on human expertise.

On the other hand, recent methods using deep generative models operate by estimating a joint model of all features together, from which missing values can be queried [13–22]. In theory, these approaches more readily take advantage of the capacity and efficiency of learning using deep function approximators and the ability to capture correlations among covariates by amortizing the parameters [19,23]. In practice, however, this strategy comes at the price of much more challenging optimization—GAN-based imputers [16–18,24] are often prone to the usual difficulties in adversarial training [25,26], and VAE-based imputers [19,20] are subject to the usual limitations of training latent-variable models through variational bounds [27,28]; empirically, these methods may often be outperformed by iterative imputation [29,30]. Further, most of such techniques—with the notable exception to [22]—either separately require fully-observed datasets during training [13–15], or operate on the strong assumption that missingness patterns are entirely independent of both observed and unobserved data [16,21], which is not realistic.

Three Desiderata: Can we do better? In light of the preceding discussion, we argue that a good baseline solution to the imputation problem should satisfy the following criteria:
HyperImpute

Table 1: Comparison with Related Work. ¹Denotes the most general regime under which each method is appropriate. (But note that methods are often empirically tested in all three regimes, not necessarily with theoretical motivation). ²Except MIWAE [22]. ³Only for HI-VAE [19].

<table>
<thead>
<tr>
<th>Technique</th>
<th>Examples</th>
<th>Missing Pattern¹</th>
<th>Required Data</th>
<th>Data Types</th>
<th>Column-wise</th>
<th>Auto Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean Imputation</td>
<td>[31]</td>
<td>MCAR-only</td>
<td>Incomplete</td>
<td>Continuous</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Discriminative, 1-Shot</td>
<td>[32,33]</td>
<td>MAR</td>
<td>Incomplete</td>
<td>Mixed</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Discriminative, Iterative</td>
<td>[34,35]</td>
<td>MAR</td>
<td>Incomplete</td>
<td>Mixed</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Generative, Implicit</td>
<td>[36,37]</td>
<td>MAR</td>
<td>Incomplete</td>
<td>Mixed</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Gen., Explicit (Full Input)</td>
<td>[40,43]</td>
<td>MAR</td>
<td>Incomplete</td>
<td>Mixed</td>
<td>Yes³</td>
<td>No</td>
</tr>
<tr>
<td>Gen., Explicit (Incomplete Input)</td>
<td>[40,43]</td>
<td>MAR</td>
<td>Incomplete</td>
<td>Mixed</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>HyperImpute (Ours)</td>
<td>MAR</td>
<td>Incomplete</td>
<td>Mixed</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

- **Flexibility**: It should combine the flexibility of conditional specification with the capacity of deep approximators.
- **Optimization**: It should relieve the burden of complete specification, and be easily and automatically optimized.
- **Assumptions**: It should be trainable without complete data, but not assume missingness is completely random.

**Contributions** In this work, we present a simple but effective method that satisfies these criteria, facilitates accessibility and reproducibility in imputation research, and constitutes a strong defense of the iterative imputation paradigm. Our contributions are three-fold. First, we formalize the imputation problem and describe HyperImpute, a generalized iterative framework for adaptively and automatically configuring column-wise models and their hyperparameters (Section 3). Second, we give a practical implementation with out-of-the-box learners, optimizers, simulators, and extensible interfaces (Section 4). Third, we empirically investigate this method via comprehensive experiments and sensitivities, and demonstrate its ability to generate accurate imputations relative to strong benchmarks (Section 5). Contrary to what recent work suggests, we find that iterative imputation—done right—consistently outperforms more complex alternatives.

2. Background

By way of preface, two key distinctions warrant emphasis: First, we are focusing on imputing missing values as an end in and of itself—that is, to estimate what those values probably looked like. In particular, we are not focusing on imputing missing values as a means to obtain input for some known downstream task—such as regression models for predicting labels [34,35], generative models for synthetic data [17,36], or active sensing models for information acquisition [37,38]; these motivate concerns fundamentally entangled with each downstream task, and often call for joint training to directly minimize the objectives of those end goals [17,39]. Here, we focus solely on the imputation problem itself.

Second, we restrict our discussion to the (most commonly studied) setting where missingness patterns depend only on the observed components of the data, and not the missing components themselves. Briefly, data may be classified as “missing completely at random” (MCAR), where the missingness does not depend on the data at all; “missing at random” (MAR), where the missingness depends only on the observed components; or “missing not at random” (MNAR), where the missingness depends on the missing components themselves as well [40,43]. (These notions are formalized mathematically in Section 5). In MCAR and MAR settings, the non-response is “ignorable” in the sense that inferences do not require modeling the missingness mechanism itself [42,43]. This is not the case in the MNAR setting, where the missing data distribution is generally impossible to identify without imposing domain-specific assumptions, constraints, or parametric forms for the missingness mechanism [44,50]. Here, we limit our attention to MCAR and MAR settings.

**Related Work** Table 1 presents relevant work in our setting, and summarizes the key advantages of HyperImpute over prevailing techniques. State-of-the-art methods can be categorized as discriminative or generative. In the former, iterative methods are the most popular, and consist in specifying a univariate model for each feature on the basis of all others, and performing a sequence of regressions by cycling through each such target variable until all models converge; well-known examples include the seminal MissForest and MICE, and their imputations are valid in the MAR setting [3,10,42]. Less effectively, one-shot methods first train regression models using fully-observed training data, which are then applied to incomplete testing data for imputation, and are only appropriate under the more limited MCAR setting [32,33].

On the generative side, implicit models consist of imputers trained as generators in GAN-based frameworks; despite their popularity, imputations they produce are only valid under the MCAR assumption [16,18]. Alternatively, explicit models refer to deep latent-variable models trained to approximate joint densities using variational bounds; as noted in Section 1, most either rely on having fully-observed training data [18,45], or otherwise are only appropriate for use under the MCAR assumption [19,21]. The only exception is MIWAE [22], which adapts the objective of importance-weighted autoencoders [51] to approximate maximum likelihood in MAR settings; but their bound is only tight in the limit of infinite computational power. Further, save methods that fit separate decoders for each feature [19], generative methods do not accommodate column-specific modeling.
Finally, for completeness there are also traditional methods based on mean substitution \cite{31}, hot deck imputation \cite{52}, k-nearest neighbors \cite{53}, EM-based joint models \cite{54}, matrix completion using low-rank assumptions \cite{55}, as well as a recently proposed technique based on optimal transport \cite{29}.

3. HyperImpute

We begin by formalizing our imputation problem and setting (Section 3.1). Motivated by our three criteria, we propose performing generalized iterative imputation (Section 3.2), which we solve by automatic model selection (Section 3.3), yielding our proposed HyperImpute algorithm (Section 3.4).

3.1. Problem Formulation

Let $X := (X_1, ..., X_D) \in \mathcal{X} = \mathcal{X}_1 \times ... \times \mathcal{X}_D$ denote a $D$-dimensional random variable, where $\mathcal{X}_d \subseteq \mathbb{R}$ (continuous), and $\mathcal{X}_d = \{1, ..., K_d\}$ (categorical), for $d \in \{1, ..., D\}$. We shall adopt notation similar to recent work (see e.g. \cite{16,43}).

Incomplete Data We do not have complete observational access to $X$; instead, access is mediated by random masks $M := (M_1, ..., M_D) \in \{0, 1\}^D$, such that $X_d$ is observable precisely when $M_d = 1$. Formally, let $\tilde{X}_d := X_d \cup \{\ast\}$ augment the space for each $d$, where “\ast” denotes an unobserved value. Then the incomplete random variable that we observe is given by $X := (\tilde{X}_1, ..., \tilde{X}_D) \in \tilde{\mathcal{X}} = \tilde{\mathcal{X}}_1 \times ... \times \tilde{\mathcal{X}}_D$, with

$$\tilde{X}_d := \begin{cases} X_d, & \text{if } M_d = 1 \\ \ast, & \text{if } M_d = 0 \end{cases} \quad (1)$$

Imputation Problem Suppose we are given an (incomplete) dataset $D := \{(X^n, M^n)\}_{n=1}^N$ of $N$ records. (In the sequel, we shall drop indices $n$ unless otherwise necessary). We wish to impute the missing values for any and all records $X$—that is, to approximately reverse the corruption process of Equation (1) by generating $X := (X_1, ..., \tilde{X}_D) \in \tilde{\mathcal{X}}$. For each $d$ with $M_d = 0$, let $\tilde{X}_d$ denote its imputed value. Then

$$\tilde{X}_d := \begin{cases} X_d, & \text{if } M_d = 1 \\ \ast, & \text{if } M_d = 0 \end{cases} \quad (2)$$

Missingness Mechanism Let $S_M := \{d : M_d = 1\}$ be the set of indices picked out by $M$, and define the selector projection $f_M : X \mapsto f_M(X) = (X_d)_{d \in S_M}$ from $\mathcal{X}$ onto the subspace $\Pi_{d \in S_M} \mathcal{X}_d$. Note that $f_M$ induces a partition of $X = (X_{\text{obs}}, X_{\text{mis}})$, where $X_{\text{obs}} := f_M(X)$ is the observed component and $X_{\text{mis}} := f_{1-M}(X)$ the missing component. The missingness mechanism for the incomplete variable $X$ is

- MCAR, if $\forall M, X, X' : p(M|X) = p(M|X')$
- MAR, if $\forall M, X, X' : X_{\text{obs}} = X_{\text{obs}} \Rightarrow p(M|X) = p(M|X')$
- MNAR, if neither of the above conditions hold.

Throughout, we assume our data $D$ is MCAR or MAR. While recent literature often uses similar classifications in discussion \cite{16,18,19,22,29}, most are not rigorously scoped or otherwise use definitions that have been shown to be ambiguous (see e.g. discussion of \cite{43} on \cite{10,42,56,57}).

3.2. Generalized Iterative Imputation

Recall our criteria from Section 1. Consider parsimony in assumptions (viz. criterion 3): Neither do we wish to assume complete data for training, nor assume the data is MCAR. This immediately rules out most of Table 1 leaving only iterative imputation (e.g. MICE \cite{10}, deep generation (i.e. MIWAE \cite{22}), and optimal transport (i.e. Sinkhorn \cite{29}). Next, consider flexibility (viz. criterion 1): Only iterative imputation allows specifying different models for each feature, and is important in practice: Conditional specifications span a much larger space than the space of known joint models, and uniquely permit incorporating design-specific considerations such as bounds and interactions—difficult to do so with a single joint density, parametric or otherwise \cite{8,10,12}.

Let $A$ be some space of univariate models and hyperparameters. Classic iterative imputation requires a specification $a_d \in A$ for each column (e.g. linear regression), with corresponding hypothesis space $\mathcal{H}_d$ (e.g. regression coefficients). Now, it is known that conditionally-specified models may not always induce valid joint distributions \cite{8,58}, and that poorly-fitting conditional models may lead to biased results \cite{12,59}. Let $h_d \in \mathcal{H}_d$ be a hypothesis for the $d$-th model:

$$p(X_d|X_1, ..., X_{d-1}, X_{d+1}, ..., X_D; h_d) \quad (4)$$

and let $\mathcal{H}_{\text{com}}$ be the space of tuples $(h_1, ..., h_D)$ that induce valid joint distributions. Augmenting the capacity of univariate models makes it more likely $\mathcal{H}_{\text{com}} \subset \Pi_{d=1}^D \mathcal{H}_d$ so that the true joint distribution is embedded in the parameter space of the conditionals—thereby improving results.\footnote{We defer to treatment in \cite{8,11,13,42,60} for detailed discussion of consistency and convergence properties of iterative imputation.} So as our first step, we propose to generalize the iterative method beyond learning hypotheses $h_d \in \mathcal{H}_d$ that correspond to a specific $a_d$, but instead to search over all models and hyperparameters in $A$ itself—which allows us to incorporate the capacity of state-of-the-art function approximators such as deep neural networks and modern boosting techniques.

3.3. Automatic Model Selection

Prima facie, we have only made things harder w.r.t. optimization (viz. criterion 2). We have now added the complexity of multiple flexible classes of learners and their hyperparameters. Choosing the best set of specifications is highly non-trivial: It depends on the characteristics of each feature, the relationships among them, the number of training samples, feature dimensionalities, and the rates and patterns of data missingness. However, this has received little attention in related work: The burden is often placed on the user, for whom domain expertise is assumed sufficient \cite{59}. Instead, can these be automatically selected, configured, and optimized?
In leveraging AutoML \cite{61, 62} to the rescue, we first consider the standard “top-down” search strategy (i.e. with a single global optimizer; see e.g. \cite{63, 64} as applied in practice). In the following, let $A$ denote the cardinality of the space of models and hyperparameters (for each univariate model), let $K$ denote an upper bound on the number of iterations for iterative procedure to converge (under any specification), and recall that $D$ denotes the number of feature dimensions.

- **Top-Down Search**: Search over the entire space of combinations of univariate models and hyperparameters. So, the iterative imputation procedure (run to completion) is called within a global search loop. The size of the search space is $A^D$, and each evaluation calls the iterative procedure once, which runs $O(KD)$ regressions. For search algorithms that reduce complexity by a constant factor, the overall complexity of the optimization is $O(KDA^D)$.

- **Concurrent Search**: What if we optimized all columns in parallel, on a per-column basis? This can be done by repeatedly calling iterative imputation (run to completion) within a global search loop, but evaluating and optimizing within each univariate search space independently of others. The size of each search space is $A$, and as before each search evaluation requires $O(KD)$ regressions. Under the same assumptions, the overall complexity is $O(KDA)$.

Computationally, the former is clearly intractable. But the latter is also undesirable, since each univariate model is optimized on its own—and is thus unaware of how the models and hyperparameters being selected for the other columns may potentially affect the best model and hyperparameters for the current column. Can we do better? Instead of calling an (inner) iterative procedure within an (outer) search procedure, we propose an “inverted” strategy that calls an (inner) search procedure within an (outer) iterative procedure:

- **Inside-Out Search**: Begin with an iterative imputation procedure, which runs $O(KD)$ regressions. Each regression is carried out by searching over the space of univariate models and hyperparameters, for which the size of the search space is $A$. The overall iterative imputation procedure is run to completion. For search algorithms that reduce complexity by a constant factor, the overall complexity is $O(KDA)$. We call this strategy **HyperImpute**.

On the one hand, performing the iterative loop on the outside allows us to inherit the usual properties of classic iterative imputation—specifically, that (i.) the imputations are valid in general under the MAR assumption \cite{2, 10}; that (ii.) they asymptotically pull toward the consistent model when the joint distribution is realizable in the parameter space of the conditional specifications \cite{11, 12}; and that (iii.) concerns about incompatibility or non-convergence are seldom serious in practice \cite{42, 59}. On the other hand, performing the search procedure on the inside allows us to benefit from automatic model selection among flexible function approximators and their hyperparameters, while obtaining the same lower complexity of the (more naive) concurrent strategy.

### 3.4. The HyperImpute Algorithm

Algorithm 1 presents HyperImpute. We begin by performing a baseline imputation, such as simple mean substitution. Next, we iterate through the dataset column by column, refining the imputations for each feature as we go. Specifically, at each iteration (i.e. for each feature dimension $d$), we first locate all records for which that feature is observed, and collect the observed target $D_{d}^{obs}$ and its regressors $D_{d}^{obs}$. (Note that some components of the regressors may themselves be imputations). We perform model selection to find the best model (and hyperparameter) $a_d \in A$ for the observed target. This is used to learn a hypothesis $h_d \in H_d$ for imputing the target value of all records for which that feature is missing, i.e. the missing target $D_{d}^{mis}$, on the basis of regressors $D_{-d}^{obs}$.

The outer procedure is performed until an imputation stopping criterion $\gamma$ is met, e.g. based on the incremental change in imputation quality. The inner procedure is performed until a selection stopping criterion $\sigma$ is met, e.g. to make use of cached information from previous searches for heuristic speedup. Note that if we removed the model selection procedure from HyperImpute entirely, instead passing a fixed set of conditional specifications $\{a_d\}_{d=1}^{D}$ into the algorithm, we would recover conventional iterative imputation. As part of our experiments, we investigate and illustrate the sources of gain that stem from column-wise specification, automatic model selection, adaptive selection across iterations, as well as having a flexible catalogue of base learners (Section 5).

---

The target $D_{d}^{obs} := \{X_{d}^{n}\}_{n=1}^{N}$ (6) contains the entries of that column ($d$) whose value is not missing ($M_{d}^{n} = 1$). The regressors $D_{-d}^{obs} := \{X_{-d}^{n}\}_{n=1}^{N}$ (5) contains all other columns ($-d$) for those same rows. Importantly, we should be clear that the “$M_{d}^{n} = 1$” is picking out rows where targets (not regressors!) are observed.
4. Practical Implementation

In addition to the HyperImpute algorithm itself, our goal is also to facilitate accessibility and reproducibility in imputation research. Concretely, our implementation consists of:

- **Learners**: These are candidate classes for each univariate model, and include conditional specifications for both classification and regression—such as linear models, deep neural networks, and bagging and boosting methods. In Algorithm 1, the global set of models and hyperparameters \( A \) includes the configuration space of all candidates selected by the user to be searched over for each variable. The “plugin” interface makes this trivially extensible: Additional learners simply need to conform to the \texttt{fit-predict} paradigm and expose a well-defined hyperparameter set.

- **Optimizers**: These are candidate algorithms that implement the \texttt{ModelSearch} in Algorithm 1. Given an objective function, these focus on configuration selection (e.g. bayesian optimization \cite{63,64}), or on configuration evaluation (e.g. adaptive computation \cite{65,66}). Among the implemented options, we default to our adaptation of Hyperband \cite{66} that accommodates configuration spaces \( A \) spanning different learner classes, and to using totals of RMSE (continuous) or negative AUROC (categorical) as objective. As above, the interface is extensible as desired.

- **Imputers**: These are candidate imputation methods that serve two purposes: First, any existing imputer can fulfill the \texttt{BaselineImpute} function in Algorithm 1 to seed \( \hat{D} \) for the first iteration; in our experiments, we default to mean substitution \cite{31}. Second, any imputer constitutes a benchmark algorithm in performance comparison experiments—such as any method in Table 1 (see Section 5). Like above, any new imputer simply needs to conform to the \texttt{fit-transform} paradigm. We have implemented comprehensive benchmark algorithms from recent literature.

Finally, we also implement modules and interfaces for simulation of missing data (according to different missing mechanisms), evaluation of imputed data (according to different performance metrics), and comparison of imputation methods via seeded and systematically cross-validated experiments. HyperImpute is implemented as an \texttt{sklearn} transformer, so it is fully compatible with \texttt{sklearn-pipelines}, and can be easily integrated as a component of an existing pipeline (e.g. for a downstream prediction task \cite{64,67,69}).

5. Empirical Investigation

Four aspects of HyperImpute deserve empirical investigation, and our goal in this section is to highlight them in turn:

1. **Performance**: Bottom-line—Does HyperImpute work? Section 5.1 compares the performance of HyperImpute with respect to a variety of state-of-the-art benchmarks.

2. **Gains**: Why does it work? Section 5.2 deconstructs various aspects of HyperImpute to investigate its sources of performance gain relative to classic iterative imputation.

3. **Selection**: What does it learn? Section 5.3 gives insight into the types of models that end up being selected, illustrating the process of adaptive and automatic selection.

4. **Convergence**: Does HyperImpute converge? Section 5.4 performs diagnostics on the iterative process of the method, illustrating its internal convergence behavior.

**Benchmarks** We test HyperImpute against the following:

- Mean substitution, which imputes the column-wise unconditional mean (Mean) \cite{31}: Imputation by chained equations, which is an iterative imputation method using linear/logistic models for conditional expectations (ICE); we follow the implementation in \cite{29} using \cite{67} based on \cite{10}; MissForest, a non-parametric iterative imputation algorithm using random forests as base learners (MissForest) \cite{3}; Generative adversarial imputation networks, an adaptation of generative adversarial networks \cite{70,71} for missing data imputation, where the discriminator is now trained to classify the generator’s output in an element-wise fashion (GAIN) \cite{16}; Missing data importance-weighted autoencoders, a deep latent variable model fit to missing data by optimizing a variational bound \cite{51} adapted to the presence of missing data (MIWAE) \cite{22}; SoftImpute, which performs imputation through soft-thresholded singular value decomposition, based on a low-rank assumption on the data (SoftImpute) \cite{55}; Imputation models trained through optimal transport...
metrics, which leverages the assumption that two random batches of samples extracted from the same dataset should be similarly distributed, and uses Sinkhorn divergences between batches to quantify that objective (Sinkhorn) [29]; and a recent method using causal learning as a regularizer for progressive refinement of imputations by concurrently modeling the missing data mechanism itself (MIRACLE) [48].

Datasets We employ 12 real-world datasets from the UCI machine learning repository [72], similar to the experiment setup in recent works [16, 22, 29]. To simulate MCAR data, the mask variable for each data point is realized according to a Bernoulli random variable with fixed mean. To simulate MAR data, a random subset of features is first set aside to be non-missing, and on the basis of which the remaining features are then masked: The masking mechanism takes the form of a logistic model that uses the non-missing features as inputs, and is parameterized by randomly chosen weights, with the bias term determined by the required rate of missingness. For completeness, we also simulate MNAR data for experiments—although this is not the focus of our work: This is either done by further masking the input features of the MAR mechanism according to a Bernoulli random variable with fixed mean, or by directly self-masking values using interval-censoring. In either MNAR mechanism, the missingness now depends on the missing values themselves.

Evaluation Methods are evaluated according to how well the imputed values align with their ground-truth values, measured by the root-mean-square error (RMSE); as well as how well the imputed distribution matches that of the ground-truth distribution, measured by the Wasserstein distance (WD), similar to [29]. For each dataset, benchmark, and experiment setting, evaluations are performed using 10 different random seeds, and we report the mean and standard deviations of the resulting performance metrics. HyperImpute is trained to output the conditional expectation of missing values; we defer an investigation of multiple imputation to future work. Throughout our experiments, we perform various sensitivities by assessing how relative performance varies according to the (a) number of samples used: “observed data size”; (b) number of features present: “feature count”; (c) proportion of missing values: “missingness rate”; and (d) missingness mechanism: MCAR, MAR, MNAR. The following subsections contain the most relevant results for the MAR setting; see Appendix A for further details on datasets and implementations, and see Appendix B for complete experiments, sensitivity analyses, and ablation studies.
Table 2: Source of Gains. Experiments under the MAR setting at a missingness rate of 0.3. Results shown in terms of mean ± standard deviation of RMSE for different sensitivities. See Table 3 for legend. All numbers are scaled by a factor of 10 for readability. Best is bold.

<table>
<thead>
<tr>
<th>Setting</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>airfoil</th>
<th>california</th>
<th>compression</th>
<th>letter</th>
<th>wine_white</th>
</tr>
</thead>
<tbody>
<tr>
<td>ice_lr</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>2.349 ± 0.483</td>
<td>0.789 ± 0.324</td>
<td>1.429 ± 0.215</td>
<td>1.087 ± 0.149</td>
<td>0.919 ± 0.060</td>
</tr>
<tr>
<td>ice_rf</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>2.365 ± 0.533</td>
<td>0.777 ± 0.339</td>
<td>1.615 ± 0.141</td>
<td>1.118 ± 0.148</td>
<td>0.987 ± 0.035</td>
</tr>
<tr>
<td>ice_cb</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>2.078 ± 0.482</td>
<td>0.726 ± 0.327</td>
<td>1.251 ± 0.204</td>
<td>0.750 ± 0.099</td>
<td>0.877 ± 0.054</td>
</tr>
<tr>
<td>global_search</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>1.599 ± 0.322</td>
<td>0.745 ± 0.326</td>
<td>1.031 ± 0.175</td>
<td>0.527 ± 0.067</td>
<td>0.853 ± 0.049</td>
</tr>
<tr>
<td>column_naive</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>1.861 ± 0.446</td>
<td>0.713 ± 0.333</td>
<td>1.094 ± 0.167</td>
<td>0.525 ± 0.065</td>
<td>0.845 ± 0.081</td>
</tr>
<tr>
<td>wo_flexibility_rf</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2.689 ± 0.755</td>
<td>0.762 ± 0.331</td>
<td>1.802 ± 0.186</td>
<td>1.117 ± 0.152</td>
<td>0.982 ± 0.031</td>
</tr>
<tr>
<td>wo_flexibility_cb</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>1.594 ± 0.367</td>
<td>0.740 ± 0.336</td>
<td>1.082 ± 0.157</td>
<td>0.757 ± 0.100</td>
<td>0.876 ± 0.050</td>
</tr>
<tr>
<td>wo_adaptivity</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>1.665 ± 0.360</td>
<td>0.721 ± 0.354</td>
<td>1.047 ± 0.176</td>
<td>0.526 ± 0.066</td>
<td>0.890 ± 0.074</td>
</tr>
<tr>
<td>HyperImpute</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>1.479 ± 0.294</td>
<td>0.704 ± 0.336</td>
<td>1.013 ± 0.145</td>
<td>0.524 ± 0.067</td>
<td>0.801 ± 0.053</td>
</tr>
</tbody>
</table>

Figure 4: Model Selections. Experiments conducted under the MAR setting on 12 UCI datasets. Likelihood of different learner classes being selected for use as univariate models at various (a) missingness rates, (b) number of samples used, and (c) across iterations of the algorithm, with selection counts tallied across all columns and datasets; when not perturbed for analysis, the missingness rate is fixed at 0.3.

5.1. Overall Performance

Figure 3 shows the performance of HyperImpute and benchmark algorithms on all 12 datasets under the MAR setting at 30% missingness rate. We observe that HyperImpute very consistently performs at or above the level of all benchmarks: In particular, it outperforms all of them on 10 out of the 12 datasets with respect to both the RMSE and WD metrics. In Appendix B, we include much more comprehensive results collected in MCAR, MAR, and MNAR simulations at four levels of missingness rates, demonstrating that HyperImpute consistently performs better across a wide range of settings.

Moreover, to better evaluate HyperImpute’s performance, we conduct a sensitivity analysis by varying the number of samples used, number of features present, and the missingness rate of the dataset. Figure 3 shows the performance of HyperImpute within these experiments against the five closest competitors (ICE, MissForest, GAIN, MIWAE, and Sinkhorn). Firstly, we see that as the number of samples increases, the performance improvement of HyperImpute relative to that of its benchmarks also increases. Secondly, we see that the advantage of HyperImpute is more noticeable with larger numbers of features (i.e. more than five); this is likely a byproduct of the iterative imputation scheme, since discriminative training of column-wise models become more challenging at lower-dimensions. Nonetheless, for feature counts above five, HyperImpute enjoys notable advantages. Thirdly, HyperImpute demonstrates significant improvement over benchmarks across the entire range of different rates of missingness: Importantly, it achieves low WD, suggesting it is less prone to overfitting to sparser datasets.

5.2. Source of Gains

HyperImpute is designed with a number of characteristics in mind (Section 3). Having empirically demonstrated strong overall results, an immediate question is how important these characteristics are for performance. Specifically, consider the source of gains from: (A) column-wise specification, (B) automatic model and/or hyperparameter selection, (C) adaptive selection across imputation iterations, and (D) having a flexible catalogue of base learners. To disentangle the contributions of each of these to the final imputation performance of HyperImpute, here we deliberately “switch off” different properties and examine the resulting performance. Table 3 summarizes the possible combinations of settings. For settings ice and wo_flexibility, we consider linear models, random forest, and catboost for the learner class.

Results are shown in Table 2. Observe that all four aspects of HyperImpute are crucial for good performance: Specifically, we note an average performance gain (i.e. decrease in RMSE) of 18% as compared to the best ice models. Compared to results obtained when HyperImpute is run with a restricted class of base learners (wo_flexibility), there is similarly an average of 11% performance improvement. Compared to results obtained when models are only selected and applied globally (global_search), there is an average of 4% performance gain. Lastly, compared with naive column-wise selection (column_naive) and the inclusion of a flexible pool of base learners without adaptive selection across iterations (wo_adaptivity), HyperImpute sees an average of 7% and 5% performance gains respectively. WD results for different settings can be found in Appendix B.
Table 3: Legend for Source of Gains. A “learner class” is a subset of \( \mathcal{A} \) (e.g., all random forest models). Note that “wo_adaptivity” selects both models and hyperparameters, whereas “column_naive” selects only the model class and uses its default hyperparameters.

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICE</td>
<td>ice</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Single model ( a \in \mathcal{A} ) for all columns is auto-selected initially and then fixed</td>
<td>global_search</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Individual models ( a_d \in \mathcal{A} ) for each column are auto-selected initially and then fixed</td>
<td>column_naive</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Like HyperImpute, but each column’s model is limited to a single learner class</td>
<td>wo_flexibility</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Like HyperImpute, but each column’s model is fixed after initial auto-selection</td>
<td>wo_adaptivity</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>HyperImpute</td>
<td>HyperImpute</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

5.3. Model Selections

Next, we illustrate what the HyperImpute algorithm tells us about different varieties of imputation problems. Across the same experimental settings and datasets as described in Section 5.2, we investigate how the *actual selections* for column-wise models differ across the missingness rates, the number of samples used, and across iterations of the algorithm. Figure 4 summarizes the relative proportions of different classes of learners being selected across all experiments. Each time a class of learners is ultimately selected to impute a column within an iteration, it is included once within the tally. Interestingly, a diverse mix of models is selected across all experiments, and the composition of which varies systematically with the characteristics of the underlying datasets.

While varying the missingness rate, we observe that the empirical selection likelihood of neural networks (NN) and boosting (GB) decrease substantially with increasing missingness rates, whereas random forests (RF) and linear regressions (LR) become relatively more likely to be selected. This could reflect the tendency for highly expressive methods (NN, GB) to overfit when only given smaller rates of observable data, thus favoring simpler methods with lower-variance. We observe a similar trend when we vary dataset sizes—LR and RF appear to dominate when fewer than 500 samples are available; in contrast, NN and GB are substantially more commonly selected for larger datasets where more data is available to optimize their larger parameter sets.

Lastly, we study how adaptive model selection behaviors across imputation iterations. As the number of iterations required for convergence varies by dataset, we let HyperImpute run for 30 iterations across all datasets to obtain comparable results. The changes in selection patterns indicate that HyperImpute indeed selects *different types* of models across iterations as the baseline imputations get updated and improve over time. In particular, we observe that GB—and, to a lesser extent, NNs—are more commonly selected for later rounds when imputations stabilize, presumably because more emphasis can be placed on correctly modeling more difficult conditional imputations. In Appendix B.4, we present similar analyses for the MCAR and MNAR settings.

5.4. Convergence

Finally, we verify that HyperImpute successfully *converges*: We compare the model’s “internal” view of the imputation performance—measured by the value of the objective function during model selection, to the ground-truth imputation performance metrics (RMSE and WD). In Figure 5 we show results on representative datasets iris and wine_white, with convergence results on additional datasets presented in Appendix B.5. We observe that HyperImpute converges to a plateau quickly, generally within 4 iterations of the start, and that improvements in the model’s internal objective correspond well to its ground-truth imputation performance.

6. Conclusion

Recent imputation research have often neglected iterative methods, relegating it to a trivial benchmarking exercise. To the contrary, our findings furnish a strong argument that a well-configured conditional specification easily produces state-of-the-art performance—an insight that may shape directions for future research. We introduced HyperImpute, a generalized iterative framework to automatically and adaptively configure column-wise models from expressive function approximators. We provide a practical implementation of the algorithm and comprehensive benchmarks, and an integrated suite of component tools for accessibility and reproducibility in imputation research. Finally, we demonstrated its use as an investigative platform for studying the characteristics and solutions to different imputation problems.
Acknowledgments

We would like to thank all reviewers for all their invaluable feedback. This work was supported by AstraZeneca, Alzheimer’s Research UK, the National Science Foundation (grant no. 1722516), and the US Office of Naval Research.

References


A. Experiment Details

In this section, we discuss further experimental details. We first give an overview of dataset details (Section A.1) and simulation details (Section A.2). Then we discuss the configuration space (Section A.3) and search strategies (Section A.4) for ModelSearch. Finally, we describe the termination criterion (Section A.5) and a clarification on data types (Section A.6).

A.1. Dataset Details

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of Instances</th>
<th>Number of Features</th>
<th>Experiment Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Airfoil Self-Noise Dataset</td>
<td>1503</td>
<td>6</td>
<td>airfoil</td>
</tr>
<tr>
<td>Blood Transfusion Service Center Dataset</td>
<td>748</td>
<td>5</td>
<td>blood</td>
</tr>
<tr>
<td>California Housing Dataset</td>
<td>20640</td>
<td>9</td>
<td>california</td>
</tr>
<tr>
<td>Concrete Compressive Strength Dataset</td>
<td>1030</td>
<td>9</td>
<td>compression</td>
</tr>
<tr>
<td>Diabetes Dataset</td>
<td>442</td>
<td>10</td>
<td>diabetes</td>
</tr>
<tr>
<td>Ionosphere Dataset</td>
<td>351</td>
<td>34</td>
<td>ionosphere</td>
</tr>
<tr>
<td>Iris Dataset</td>
<td>150</td>
<td>4</td>
<td>iris</td>
</tr>
<tr>
<td>Letter Recognition Dataset</td>
<td>20000</td>
<td>16</td>
<td>letter</td>
</tr>
<tr>
<td>Libras Movement Dataset</td>
<td>360</td>
<td>91</td>
<td>libras</td>
</tr>
<tr>
<td>Spambase Dataset</td>
<td>4601</td>
<td>57</td>
<td>spam</td>
</tr>
<tr>
<td>Wine Quality Dataset(Red)</td>
<td>1599</td>
<td>12</td>
<td>wine_red</td>
</tr>
<tr>
<td>Wine Quality Dataset(White)</td>
<td>4898</td>
<td>12</td>
<td>wine_white</td>
</tr>
</tbody>
</table>

Table 4: Datasets for Evaluation.

A.2. Simulation Details

The procedures for simulating missingness in each dataset are adapted directly from the experiment setup and implementation of [29], and are exactly replicable using the source code.

- **MCAR**: Each value is removed according to the realization of a Bernoulli random variable with a fixed parameter.
- **MAR**: First, a subset of variables is randomly selected to be fully-observed, so only the remaining variables can have values that are missing. Second, these remaining variables have values removed according to a logistic model with random weights, using the fully-observed variables as regressors. The desired rate of missingness is achieved by adjusting the bias term.
- **MNAR**: This is done by either further removing the values of the input features in the MAR mechanism above, or by directly removing values using interval-censoring.

In all three cases MCAR, MAR, and MNAR, experiments are performed using 10%, 30%, 50%, and 70% missingness.

A.3. Configuration Space

In Table Table 5 we present the full configuration space (models and associated hyperparameter ranges) we consider for the column-wise model selection within HyperImpute. We use linear/logistic regressions and random forests as implemented in sklearn, XGBoost from the xgboost python package, catboost from the catboost python package and neural nets implemented using pytorch.

A.4. Search Strategies

**Objective Function**   Any ModelSearch strategy requires a column-wise objective function which is optimized through cross-validation when choosing the best conditional imputation model for a given column. In our implementation, we differentiate between column types; depending on the label type, the evaluation objective would be to minimize RMSE (continuous labels) or to maximize AUROC (categorical labels).

**Search Strategies**   To efficiently explore configuration spaces \( \mathcal{A} \) of models (linear models, gradient boosting, random forests, neural nets etc.) with disjoint sets of hyperparameters, we implemented a number of search strategies that take as input \( \mathcal{A} \) and a pre-defined objective function and output the best model found under computational constraints.
1. Naive Search Strategy
   - Each model in the search pool is evaluated using its default hyperparameters, on the objective function.
   - The method returns the best model across evaluations.
   - **Pros:** Fast. **Cons:** No exploration of hyperparameters.

2. Bayesian Optimization Strategy (Model-Specific)
   - For each model in the search pool, we run a dedicated Bayesian Optimization (BO) call to suggest which hyperparameters to test to improve the objective function.
   - As a final model, we use the model and hyperparameters associated with the best score across all BO runs.
   - **Pros:** Good exploration. **Cons:** Slow, the Bayesian Optimization needs to be separately executed for each model in the pool.

3. Adapted HyperBand Strategy
   - For each model in the search pool we define a special parameter, `iterations`, translated to epochs in linear/neural nets, or estimators in forests/gradient boosting.
   - In a preprocessing step, we learn a scaling mapping between the iterations of the different models, by evaluating each model for [1, 5, 10] iterations, and evaluating their learning rate.
   - We apply the standard HyperBand [66] search algorithm on the model search pool and the objective function while scaling the `iterations` values using the mapping learned in the preprocessing step.
   - **Pros:** Good exploration/performance balance. **Cons:** The model performance mappings might be imprecise.
A.5. Termination Criterion

The termination criterion $\gamma$ in Algorithm 1 is met if: (1) the total number of iterations (i.e. loops over all columns $d \in \pi$) exceeds a pre-specified limit, or (2) changes in imputed values fall below a norm-based threshold (here we use the max norm), or (3) the optimization objective (i.e. the “imputation quality”) stops improving over multiple consecutive rounds. For the exact implementation, these conditions are specified directly within `plugin_hyperimpute.py` in the source code.

A.6. Data Types

Unlike most popular recent works that treat all inputs as real-valued (see e.g. [16-18, 22, 29, 48, 55]), HyperImpute appropriately handles both categorical and continuous variables. Specifically, HyperImpute automatically (1) defines separate tasks for each variable (categorical/continuous), (2) maintains corresponding classes of candidates (classifiers/regressions), and (3) searches in their respective hyperparameter domains using distinct loss/objective functions. Specifically, see Table 5.

B. Additional Results

For step-by-step experiment code for generating the following results in Sections B.1—B.5, please refer to the corresponding notebooks in the `experiments/` directory in the source code.

B.1. Overall Performance

In this section, we provide additional results to highlight HyperImpute’s imputation performance across a range of different missingness scenarios. To be exact, we report imputation performance on 12 UCI datasets as measured by RMSE and WD across three missingness scenarios, {MCAR, MAR, and MNAR} and four missingness rates, {0.1, 0.3, 0.5, 0.7}. The experiments are performed on the same datasets and compared to the same benchmarks using the procedures described in the experimental setup.

Figure 6, Figure 7, and Figure 8 plots the performance for MCAR, MAR, and MNAR respectively. Notably, HyperImpute out-performs the majority of benchmarks in terms of both RMSE and WD across different scenarios and missingness rates.

B.2. Sensitivity Analysis

Next, we quantitatively evaluate the robustness of HyperImpute to different missingness scenarios and missingness characteristics (i.e. observed data size, feature count, missingness rate) on the letter dataset. We perform sensitivity analysis by independently varying each of those parameters and plot the results in Figure 9.

We see a few trends common across scenarios:

- HyperImpute outperforms all benchmarks when lower number of samples are available, with the performance improvement more significant as data sizes increase,
- For settings with low feature counts, HyperImpute does not demonstrate markedly better imputation performance. This is likely due to the difficulty in the discriminatively training with less available regressors. However, at higher feature counts, HyperImpute demonstrates consistently superior performance,
- Lastly, HyperImpute achieves superior performance across all missingness scenarios and missingness rates. This advantage is more obvious at higher missingness rates, when performances of the benchmarks become significantly worse, but HyperImpute is able to minimise performance loss.

B.3. Source of Gains

Here, we attach the complete results for our source of gains study, including RMSE and WD metrics on five UCI datasets. The RMSE scores for different settings are shown in Table 6 and WD scores in Table 7. We first note that all components of our algorithm, including (1) column-wise imputation, (2) automatic model and hyperparameter tuning, (3) adaptive imputer selection across iterations, and (4) flexible suite of base imputers, all contribute to performance improvements.

Specifically, we note an average performance gain of 18% compared to the best ice models. Compared to results obtained when HyperImpute is run with a restricted class of base imputers, there is similarly a 11% performance improvement.
Additionally, in contrast to results obtained when an imputer is selected and applied globally, i.e. global_search, there is a 4% performance gain. Lastly, column-wise imputer selection and the inclusion of a flexible catalogue of base learners affords 7% and 5% performance gain, respectively.

Subsequently, we look at the distance between imputed data and the underlying ground truth. There is an average gain of 40% in WD when compared to the best ice models. This becomes 27% when we compare against the best results obtained using restricted base imputers, i.e. wo_flexibility. In contrast to global_search, the WDs are an average of 17% lower. Lastly, column-wise imputer selection and flexible base learner classes present 14% and 12% additional performance improvement.

Table 6: Source of Gains. Experiments under the MAR setting at a missingness rate of 0.3. Results shown in terms of mean ± std of RMSE across different settings on 5 datasets. All values are scaled by a factor of 10 for readability. Best results are emboldened.

<table>
<thead>
<tr>
<th></th>
<th>airfoil</th>
<th>california</th>
<th>compression</th>
<th>letter</th>
<th>wine_white</th>
</tr>
</thead>
<tbody>
<tr>
<td>ice_ir</td>
<td>2.349 ± 0.483</td>
<td>0.789 ± 0.324</td>
<td>1.429 ± 0.215</td>
<td>1.087 ± 0.149</td>
<td>0.919 ± 0.060</td>
</tr>
<tr>
<td>ice_rf</td>
<td>2.365 ± 0.533</td>
<td>0.777 ± 0.339</td>
<td>1.615 ± 0.141</td>
<td>1.118 ± 0.148</td>
<td>0.987 ± 0.034</td>
</tr>
<tr>
<td>ice_cb</td>
<td>2.078 ± 0.482</td>
<td>0.726 ± 0.327</td>
<td>1.251 ± 0.204</td>
<td>0.750 ± 0.099</td>
<td>0.877 ± 0.054</td>
</tr>
<tr>
<td>global_search</td>
<td>1.599 ± 0.322</td>
<td>0.745 ± 0.326</td>
<td>1.683 ± 0.175</td>
<td>0.527 ± 0.067</td>
<td>0.853 ± 0.049</td>
</tr>
<tr>
<td>column_naive</td>
<td>1.861 ± 0.446</td>
<td>0.713 ± 0.333</td>
<td>1.094 ± 0.167</td>
<td>0.525 ± 0.065</td>
<td>0.845 ± 0.081</td>
</tr>
<tr>
<td>wo_flexibility_rf</td>
<td>2.689 ± 0.755</td>
<td>0.762 ± 0.331</td>
<td>1.807 ± 0.186</td>
<td>1.117 ± 0.152</td>
<td>0.982 ± 0.031</td>
</tr>
<tr>
<td>wo_flexibility_cb</td>
<td>1.594 ± 0.367</td>
<td>0.740 ± 0.336</td>
<td>1.082 ± 0.157</td>
<td>0.757 ± 0.100</td>
<td>0.876 ± 0.050</td>
</tr>
<tr>
<td>wo_adaptivity</td>
<td>1.665 ± 0.360</td>
<td>0.721 ± 0.354</td>
<td>1.047 ± 0.176</td>
<td>0.526 ± 0.066</td>
<td>0.890 ± 0.074</td>
</tr>
<tr>
<td>HyperImpute</td>
<td>1.479 ± 0.294</td>
<td>0.704 ± 0.336</td>
<td>1.013 ± 0.145</td>
<td>0.524 ± 0.067</td>
<td>0.801 ± 0.053</td>
</tr>
</tbody>
</table>

Table 7: Source of Gains. Experiments under the MAR setting at a missingness rate of 0.3. Results shown in terms of mean ± std of WD across different settings on 5 datasets. All values are scaled by a factor of 10 for readability. Best results are emboldened.

<table>
<thead>
<tr>
<th></th>
<th>airfoil</th>
<th>california</th>
<th>compression</th>
<th>letter</th>
<th>wine_white</th>
</tr>
</thead>
<tbody>
<tr>
<td>ice_ir</td>
<td>1.066 ± 0.307</td>
<td>0.330 ± 0.192</td>
<td>0.781 ± 0.219</td>
<td>0.526 ± 0.060</td>
<td>0.584 ± 0.093</td>
</tr>
<tr>
<td>ice_rf</td>
<td>1.133 ± 0.307</td>
<td>0.354 ± 0.173</td>
<td>1.010 ± 0.040</td>
<td>0.799 ± 0.106</td>
<td>0.648 ± 0.047</td>
</tr>
<tr>
<td>ice_cb</td>
<td>0.790 ± 0.272</td>
<td>0.196 ± 0.107</td>
<td>0.537 ± 0.053</td>
<td>0.400 ± 0.045</td>
<td>0.491 ± 0.093</td>
</tr>
<tr>
<td>global_search</td>
<td>0.375 ± 0.105</td>
<td>0.253 ± 0.171</td>
<td>0.329 ± 0.045</td>
<td>0.257 ± 0.010</td>
<td>0.410 ± 0.088</td>
</tr>
<tr>
<td>column_naive</td>
<td>0.516 ± 0.278</td>
<td>0.155 ± 0.085</td>
<td>0.315 ± 0.030</td>
<td>0.261 ± 0.016</td>
<td>0.390 ± 0.172</td>
</tr>
<tr>
<td>wo_flexibility_rf</td>
<td>1.217 ± 0.366</td>
<td>0.329 ± 0.153</td>
<td>1.103 ± 0.079</td>
<td>0.807 ± 0.117</td>
<td>0.645 ± 0.060</td>
</tr>
<tr>
<td>wo_flexibility_cb</td>
<td>0.443 ± 0.142</td>
<td>0.188 ± 0.097</td>
<td>0.369 ± 0.057</td>
<td>0.394 ± 0.043</td>
<td>0.495 ± 0.096</td>
</tr>
<tr>
<td>wo_adaptivity</td>
<td>0.327 ± 0.092</td>
<td>0.170 ± 0.111</td>
<td>0.307 ± 0.025</td>
<td>0.256 ± 0.009</td>
<td>0.485 ± 0.164</td>
</tr>
<tr>
<td>HyperImpute</td>
<td>0.291 ± 0.103</td>
<td>0.114 ± 0.060</td>
<td>0.289 ± 0.041</td>
<td>0.256 ± 0.010</td>
<td>0.426 ± 0.141</td>
</tr>
</tbody>
</table>

B.4. Model Selections

We also wish to further investigate how the imputer selection changes under different missingness mechanisms. To do so, we tally the imputers chosen across columns, datasets and iterations. We plot how the likelihood of model selection varies with different missingness rates, number of samples and iterations in Figure 10. While missingness mechanisms differ, similar insights can be drawn.

Most notably, boosting algorithms and neural nets (NNs), which are highly expressive algorithms but require larger amounts of data to train, are more prevalent in low missingness rates and larger datasets. By contrast, linear regression (LR) and bagging algorithms, which are less expressive but lower variance predictors, are more common in low data regimes. We also note that HyperImpute tends to opt for more powerful algorithms in later iterations as it chooses to focus on more challenging imputations.

B.5. Convergence

Lastly, we present convergence results for MAR simulations at 0.3 missingness for the 12 UCI datasets employed in our experiments. We are interested in comparing the rates of convergence across imputation tasks. Evidently, convergence is generally achieved after 4 iterations, with the model’s internal objective corresponding well to ground-truth imputation performance. Of the 12 experiments, 3 does not show converging behaviour even after > 10 iterations. Firstly, we note that while the plots appear non-convergent, the actual fluctuations are very small due to the scale, i.e. < 0.01 in RMSE. We additionally note that imputation performance is still superior to all benchmark methods in terms of both RMSE and WD on those datasets (see Figure 8).
Figure 6: Overall Performance. Experiments on 12 UCI datasets under MCAR simulations at four levels of missingness—{0.1, 0.3, 0.5, 0.7}. Results shown as mean ± std of RMSE and WD.
Figure 7: Overall Performance. Experiments on 12 UCI datasets under MAR simulations at four levels of missingness—{0.1, 0.3, 0.5, 0.7}. Results shown as mean ± std of RMSE and WD.
Figure 8: Overall Performance. Experiments on 12 UCI datasets under MNAR simulations at four levels of missingness—{0.1, 0.3, 0.5, 0.7}. Results shown as mean ± std of RMSE and WD.
Figure 9: Sensitivity Analysis. Experiments performed on the letter dataset under the MCAR, MAR and MNAR simulations. Results shown in terms of mean ± std of RMSE and WD with sensitivities according to (a) observed data size, (b) feature count, and (c) missingness rate. When not perturbed for analysis, the observed data size is fixed at \( N = 20,000 \), feature count at \( D = 14 \), and missingness rate at 0.3.
Figure 10: Model Selections. Experiments conducted under the MCAR, MAR, and MNAR setting on 12 UCI datasets. Likelihood of different learner classes being selected for use as univariate models at various (a) missingness rates, (b) number of samples used, and (c) across iterations of the algorithm, with selection counts tallied across all columns and datasets. When not perturbed for analysis, the missingness rate is fixed at 0.3.
Figure 11: Model Convergence. Experiments performed using MAR at 0.3 missingness rate on 12 UCI datasets. Mean ± std of different metrics, including (a) objective error, (b) RMSE, (c) WD.
C. Hyperparameters

When speaking of “hyperparameter optimization”, we must—importantly—first distinguish between (1) the hyperparameters of an imputation method (e.g. GAIN, MIWAE, Sinkhorn), or (2) the hyperparameters of each column-wise model (i.e. various regression/classification models) within an iterative procedure (e.g. ICE, MissForest, HyperImpute).

Regarding (1), since we work in the (realistic) setting where we are not given access to completely-observed data during training, it is theoretically impossible to perform hyperparameter optimization for an imputation method per se: To do so requires “ground truths” of the missing values themselves (for measuring imputation quality), which we do not have. (Note that cross-validating based on imputations of observed values is futile: The identity \( f(D) = D \) would appear globally optimal despite imputing nothing at all. And without knowing the true missingness pattern, naively adding artificial missingness for the purposes of hyperparameter optimization would optimize for an incorrect objective). In fact, GAIN, MIWAE, and Sinkhorn all operate in this setting: Their authors simply prescribe sensible defaults for hyperparameters—which we use.

Regarding (2), however, it is entirely possible to perform hyperparameter optimization for the column-wise models within iterative imputation, using observed values (because the iterative procedure essentially reduces the original problem to a series of column-wise “prediction” problems): This is precisely what HyperImpute takes advantage of, and is what makes it a strict generalization of ICE, MissForest, or—for that matter—any iterative method that relies on a pre-selected set of conditional specifications. (Of course, in order to report final test-time benchmarking results, we must employ non-missing held-out data for performance evaluation, but—again—the point is that such complete data is not available at training-time).

Finally, note that HyperImpute is sklearn-compatible, and so it can be easily integrated as a component of an existing sklearn/AutoML pipeline (e.g. for a downstream prediction task [64, 67–69]).

D. Running Time

For some running time comparisons, see (left) figure below for an example on the spam dataset at various MAR missingness. The main takeaway is that HyperImpute is far from being the most time-intensive. Interpreting wall-clock times requires care, but a key remark is that model training tends to dominate (e.g. only using random forests often slows down MissForest), whereas HyperImpute’s model selection chooses/re-uses models from all classes—which can end up faster. Laptop hardware: 32GB RAM, Intel Core i7-6700HQ, GeForce GTX 950M. All algorithms take order of seconds/minutes for convergence.

In addition, we examine the effect of feature dimension on running time: See (right) figure above for an example on the largest dataset (libras) with various feature counts and missingness. In varying the feature count, features are subsetted from left to right in their original order of appearance in the raw dataset. Missingness is reported for 10%, 30%, and 50%. Results are roughly consistent with the \( O(D) \) complexity in number of features, and with our observation that model training dominates.