HyperImpute: Generalized Iterative Imputation with Automatic Model Selection

Daniel Jarrett [*1] **Bogdan Cebere** [*1] **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:

[*]Equal contribution. [1]Department of Applied Mathematics & Theoretical Physics, University of Cambridge, UK. [2]Department of Electrical Engineering, University of California, Los Angeles, USA. Correspondence to: Bogdan Cebere <bcc38@cam.ac.uk>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

Table 1: *Comparison with Related Work*. [1]Denotes 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). [2]Except MIWAE [22]. [3]Only for HI-VAE [19].

Technique	Examples	Missing Pattern[1]	Required Data	Data Types	Column-wise	Auto Selection
Mean Imputation	[31]	MCAR-only	Incomplete	Continuous	-	-
Discriminative, 1-Shot	[32, 33]	MCAR-only	Fully-Observed	Mixed	Yes	No
Discriminative, Iterative	[3, 5–9]	MAR	Incomplete	Mixed	Yes	No
Generative, Implicit	[16–18]	MCAR-only	Incomplete	Mixed	No	No
Gen., Explicit (Full Input)	[13–15]	MAR	Fully-Observed	Continuous	No	No
Gen., Explicit (Incomplete Input)	[19–22]	MCAR-only[2]	Incomplete	Mixed	Yes[3]	No
Optimal Transport	[29]	MAR	Incomplete	Mixed	No	No
HyperImpute	**(Ours)**	**MAR**	**Incomplete**	**Mixed**	**Yes**	**Yes**

- ***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 [36–38]; 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 3). 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 [13–15], 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 [31], hot deck imputation [52], k-nearest neighbors [53], EM-based joint models [54], matrix completion using low-rank assumptions [55], as well as a recently proposed technique based on optimal transport [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 $\mathbf{X} := (X_1, ..., X_D) \in \mathcal{X} = \mathcal{X}_1 \times ... \times \mathcal{X}_D$ 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. [16,43]).

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

$$\tilde{X}_d := \begin{cases} X_d, & \text{if } M_d = 1 \\ *\ , & \text{if } M_d = 0 \end{cases} \tag{1}$$

Imputation Problem Suppose we are given an (incomplete) dataset $\mathcal{D} := \{(\tilde{\mathbf{X}}^n, \mathbf{M}^n)\}_{n=1}^N$ 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 $\tilde{\mathbf{X}}$—that is, to approximately reverse the corruption process of Equation 1 by generating $\hat{\mathbf{X}} := (\hat{X}_1, ..., \hat{X}_D) \in \mathcal{X}$. For each d with $M_d = 0$, let $\bar{X}_d$ denote its imputed value. Then

$$\hat{X}_d := \begin{cases} \tilde{X}_d, & \text{if } M_d = 1 \\ \bar{X}_d, & \text{if } M_d = 0 \end{cases} \tag{2}$$

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

$$\begin{array}{lll} \text{MCAR}, & \text{if } \forall \mathbf{M}, \mathbf{X}, \mathbf{X}' : & p(\mathbf{M}|\mathbf{X}) = p(\mathbf{M}|\mathbf{X}') \\ \text{MAR}\ \ , & \text{if } \forall \mathbf{M}, \mathbf{X}, \mathbf{X}' : & \mathbf{X}_{\text{obs}} = \mathbf{X}'_{\text{obs}} \Rightarrow \\ & & p(\mathbf{M}|\mathbf{X}) = p(\mathbf{M}|\mathbf{X}') \\ \text{MNAR}, & \text{if neither of the above conditions hold.} & \end{array} \tag{3}$$

Throughout, we assume our data $\mathcal{D}$ is MCAR or MAR. While recent literature often uses similar classifications in discussion [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 [43] on [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 [10]), deep generation (i.e. MIWAE [22]), and optimal transport (i.e. Sinkhorn [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 [8,10,12].

Let $\mathcal{A}$ be some space of univariate models and hyperparameters. Classic iterative imputation requires a specification $a_d \in \mathcal{A}$ 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 [8, 58], and that poorly-fitting conditional models may lead to biased results [12, 59]. Let $h_d \in \mathcal{H}_d$ be a hypothesis for the d-th model:

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

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 $\mathcal{H}_{\text{com}} \subset \prod_{d=1}^D \mathcal{H}_d$ so that the true joint distribution is embedded in the parameter space of the conditionals—thereby improving results.[4] 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 $\mathcal{A}$ 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 nontrivial: 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 [5–9]. Instead, can these be *automatically* selected, configured, and optimized?

[4] We defer to treatment in [8, 11, 12, 42, 60] for detailed discussion of consistency and convergence properties of iterative imputation.

In leveraging AutoML [61, 62] to the rescue, we first consider the standard "top-down" search strategy (i.e. with a single global optimizer; see e.g. [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 [9, 10]; that (ii.) they asymptotically pull toward the consistent model when the joint distribution is realizable in the parameter space of the conditional specifications [11, 12]; and that (iii.) concerns about incompatibility or non-convergence are seldom serious in practice [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: HyperImpute

Parameters: Global set of models & hyperparameters $\mathcal{A}$, `ModelSearch` function, `BaselineImpute` function, Imputation stop criterion γ, Selection skip criterion σ, Column visitation order π
Input: Incomplete dataset $\mathcal{D} := \{(\tilde{\mathbf{X}}^n, \mathbf{M}^n)\}_{n=1}^N$
Output: Imputed dataset $\hat{\mathcal{D}} := \{\hat{\mathbf{X}}^n\}_{n=1}^N$
Initialize: $\hat{\mathcal{D}} \leftarrow$ `BaselineImpute`$(\mathcal{D})$
while γ is False **do** ▷ keep imputing?
  for column $d \in$ visitation order π **do**
    $\hat{\mathcal{D}}_{-d}^{\text{obs}} := \{\hat{\mathbf{X}}_{-d}^n\}_{n:M_d^n=1}$ (5)
    $\mathcal{D}_d^{\text{obs}} := \{\hat{X}_d^n\}_{n:M_d^n=1}$ (6)
    if σ is False **then** ▷ keep selecting?
      $a_d \leftarrow$ `ModelSearch`$(\mathcal{D}_d^{\text{obs}}, \hat{\mathcal{D}}_{-d}^{\text{obs}}, \mathcal{A})$
    $h_d \leftarrow a_d$`.train`$(\mathcal{D}_d^{\text{obs}}, \hat{\mathcal{D}}_{-d}^{\text{obs}}, \mathcal{H}_d)$
    $\hat{\mathcal{D}}_{-d}^{\text{mis}} := \{\hat{\mathbf{X}}_{-d}^n\}_{n:M_d^n=0}$ (7)
    $\hat{\mathcal{D}}_d^{\text{mis}} := \{\hat{X}_d^n\}_{n:M_d^n=0} \leftarrow h_d$`.impute`$(\hat{\mathcal{D}}_{-d}^{\text{mis}})$ (8)
  return $\hat{\mathcal{D}}$

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* $\mathcal{D}_d^{\text{obs}}$ and its regressors $\hat{\mathcal{D}}_{-d}^{\text{obs}}$. (Note that some components of the regressors may themselves be imputations). We perform *model selection* to find the best model (and hyperparameter) $a_d \in \mathcal{A}$ for the observed target. This is used to learn a hypothesis $h_d \in \mathcal{H}_d$ for imputing the target value of all records for which that feature is missing, i.e. the *missing target* $\hat{\mathcal{D}}_d^{\text{mis}}$, on the basis of regressors $\hat{\mathcal{D}}_{-d}^{\text{mis}}$.[5]

The outer procedure is performed until an *imputation stopping criterion* γ is met, e.g. based on the incremental change in imputation quality. The inner procedure is performed until a *selection stopping criterion* σ 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).

[5] The *target* $\mathcal{D}_d^{\text{obs}} := \{\hat{X}_d^n\}_{n:M_d^n=1}$ (6) contains the entries of that column (d) whose value is not missing ($M_d^n = 1$). The *regressors* $\hat{\mathcal{D}}_{-d}^{\text{obs}} := \{\hat{\mathbf{X}}_{-d}^n\}_{n:M_d^n=1}$ (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.

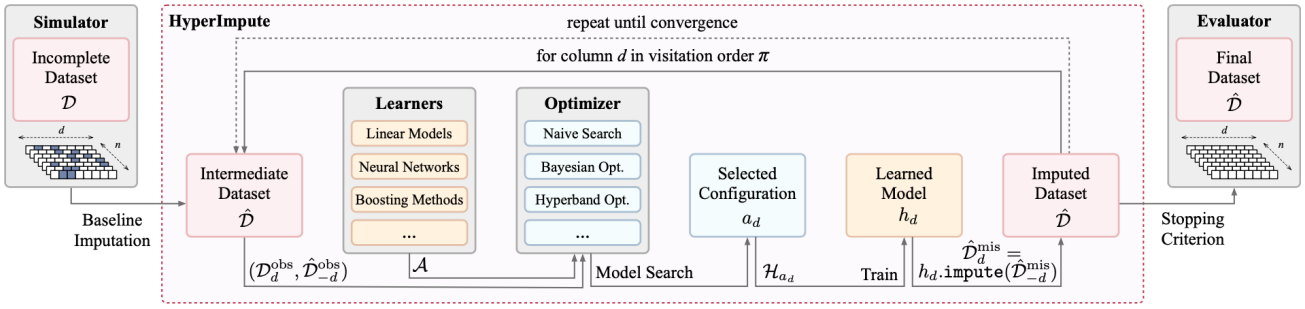


Figure 1: *High-level overview of HyperImpute.* Blue indicates model selection algorithms and their selected output. Orange indicates candidate models and their trained output. Red indicates datasets and imputations. Gray indicates component modules in HyperImpute.

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 $\mathcal{A}$ 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 *fit-predict* paradigm and expose a well-defined hyperparameter set.
- **Optimizers**: These are candidate algorithms that implement the `ModelSearch` in Algorithm 1. Given an objective function, these focus on configuration *selection* (e.g. bayesian optimization [63, 64]), or on configuration *evaluation* (e.g. adaptive computation [65, 66]). Among the implemented options, we default to our adaptation of *Hyperband* [66] that accommodates configuration spaces $\mathcal{A}$ 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 `BaselineImpute` function in Algorithm 1 to seed $\hat{\mathcal{D}}$ for the first iteration; in our experiments, we default to mean substitution [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 *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 `sklearn` transformer, so it is fully compatible with `sklearn`-pipelines, and can be easily integrated as a component of an existing pipeline (e.g. for a downstream prediction task [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**) [31]; Imputation by chained equations, which is an iterative imputation method using linear/logistic models for conditional expectations (**ICE**); we follow the implementation in [29] using [67] based on [10]; MissForest, a non-parametric iterative imputation algorithm using random forests as base learners (**MissForest**) [3]; Generative adversarial imputation networks, an adaptation of generative adversarial networks [70, 71] for missing data imputation, where the discriminator is now trained to classify the generator's output in an element-wise fashion (**GAIN**) [16]; Missing data importance-weighted autoencoders, a deep latent variable model fit to missing data by optimizing a variational bound [51] adapted to the presence of missing data (**MIWAE**) [22]; SoftImpute, which performs imputation through soft-thresholded singular value decomposition, based on a low-rank assumption on the data (**SoftImpute**) [55]; Imputation models trained through optimal transport

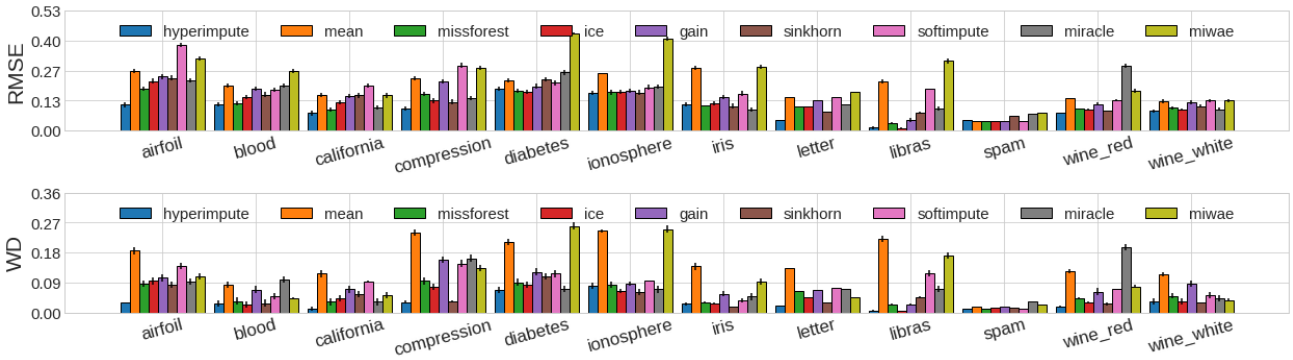


Figure 2: *Overall Performance.* Experiments on 12 UCI datasets under MAR at 0.3 missingness. Results shown as mean ± standard deviation of RMSE and WD. HyperImpute outperforms all benchmarks on both metrics in 10 of the 12, and at least one metric on all 12.

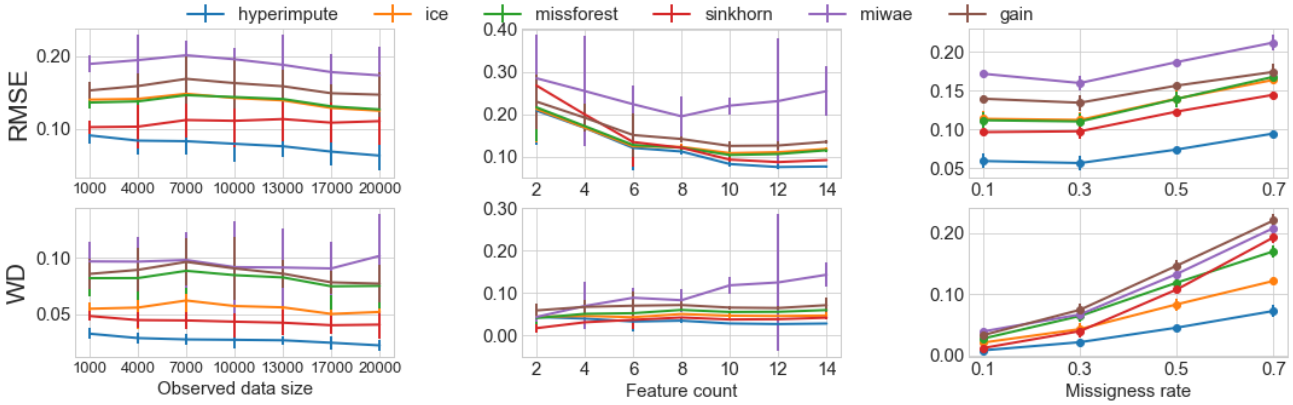


Figure 3: *Sensitivity Analysis.* Experiments performed on the `letter` dataset under the MAR setting. Results shown in terms of mean ± standard deviation of RMSE and WD, with sensitivities according to (a) observed data size, (b) feature count, and (c) missingness rate; when not perturbed for analysis, the observed data size is fixed at $N = 20{,}000$, the feature count at $D = 14$, and missingness rate at 0.3.

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.

Setting	**A**	**B**	**C**	**D**	*airfoil*	*california*	*compression*	*letter*	*wine_white*
`ice_lr`	×	×	×	×	2.349 ± 0.483	0.789 ± 0.324	1.429 ± 0.215	1.087 ± 0.149	0.919 ± 0.060
`ice_rf`	×	×	×	×	2.365 ± 0.533	0.777 ± 0.339	1.615 ± 0.141	1.118 ± 0.148	0.987 ± 0.034
`ice_cb`	×	×	×	×	2.078 ± 0.482	0.726 ± 0.327	1.251 ± 0.204	0.750 ± 0.099	0.877 ± 0.054
`global_search`	×	✓	×	✓	1.599 ± 0.322	0.745 ± 0.326	1.031 ± 0.175	0.527 ± 0.067	0.853 ± 0.049
`column_naive`	✓	×	×	✓	1.861 ± 0.446	0.713 ± 0.333	1.094 ± 0.167	0.525 ± 0.065	0.845 ± 0.081
`wo_flexibility_rf`	✓	✓	✓	×	2.689 ± 0.755	0.762 ± 0.331	1.807 ± 0.186	1.117 ± 0.152	0.982 ± 0.031
`wo_flexibility_cb`	✓	✓	✓	×	1.594 ± 0.367	0.740 ± 0.336	1.082 ± 0.157	0.757 ± 0.100	0.876 ± 0.050
`wo_adaptivity`	✓	✓	×	✓	1.665 ± 0.360	0.721 ± 0.354	1.047 ± 0.176	0.526 ± 0.066	0.890 ± 0.074
`HyperImpute`	✓	✓	✓	✓	**1.479 ± 0.294**	**0.704 ± 0.336**	**1.013 ± 0.145**	**0.524 ± 0.067**	**0.801 ± 0.053**

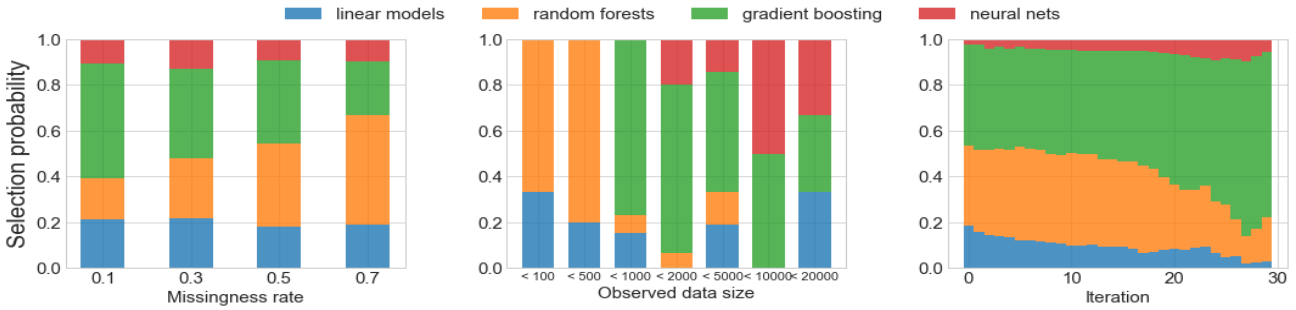


Figure 4: *Model Selections.* Experiments conducted under the MAR setting on 12 UCI datasets. Likelihood of different learner classes being selected for use as univariate models at various (a) missingness rates, (b) number of samples used, and (c) across iterations of the algorithm, with selection counts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

5.1. Overall Performance

Figure 2 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.

Description	*Name*	A	B	C	D
ICE	`ice`	×	×	×	×
Single model $a \in \mathcal{A}$ for all columns is auto-selected initially and then fixed	`global_search`	×	✓	×	✓
Individual models $a_d \in \mathcal{A}$ for each column are auto-selected initially and then fixed	`column_naive`	✓	×	×	✓
Like HyperImpute, but each column's model is limited to a single learner class	`wo_flexibility`	✓	✓	✓	×
Like HyperImpute, but each column's model is fixed after initial auto-selection	`wo_adaptivity`	✓	✓	×	✓
HyperImpute	`HyperImpute`	✓	✓	✓	✓

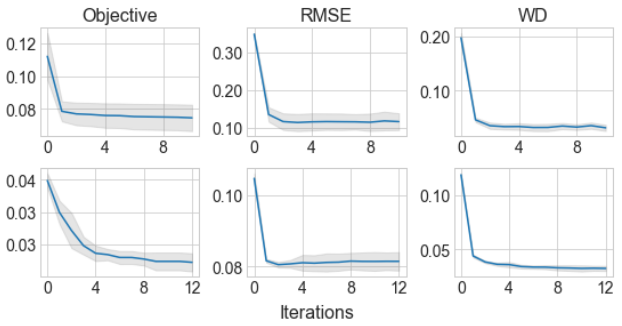


Figure 5: *Convergence.* Mean ± standard deviation of (a) the value of the objective function, (b) the RMSE metric, and (c) the WD metric, across iterations in experiments performed on `iris` (top) and `wine_white` (bottom) datasets under the MAR setting and at 0.3 missingness. Note that iteration 0 denotes results *after* the initial round of mean imputations, and iteration 1 denotes results *after* an additional complete round-robin of conditional models are selected and trained by HyperImpute. We observe that HyperImpute significantly speeds up convergence within the iterative framework.

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 5.1, 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 behaves 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

[1] John Barnard and Xiao-Li Meng. Applications of multiple imputation in medical studies: from aids to nhanes. *Statistical methods in medical research*, 8(1):17–36, 1999.

[2] A Mackinnon. The use and reporting of multiple imputation in medical research–a review. *Journal of internal medicine*, 268(6):586–593, 2010.

[3] Daniel J Stekhoven and Peter Bühlmann. Missforest—non-parametric missing value imputation for mixed-type data. *Bioinformatics*, 28(1):112–118, 2012.

[4] Ahmed M Alaa, Jinsung Yoon, Scott Hu, and Mihaela Van der Schaar. Personalized risk scoring for critical care prognosis using mixtures of gaussian processes. *IEEE Transactions on Biomedical Engineering*, 65(1):207–218, 2017.

[5] Jaap Brand. *Development, implementation and evaluation of multiple imputation strategies for the statistical analysis of incomplete data sets*. 1999.

[6] David Heckerman, David Maxwell Chickering, Christopher Meek, Robert Rounthwaite, and Carl Kadie. Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research*, 1(Oct):49–75, 2000.

[7] Trivellore E Raghunathan, James M Lepkowski, John Van Hoewyk, Peter Solenberger, et al. A multivariate technique for multiply imputing missing values using a sequence of regression models. *Survey methodology*, 27(1):85–96, 2001.

[8] Andrew Gelman. Parameterization and bayesian modeling. *Journal of the American Statistical Association*, 99(466):537–545, 2004.

[9] Stef Van Buuren, Jaap PL Brand, Catharina GM Groothuis-Oudshoorn, and Donald B Rubin. Fully conditional specification in multivariate imputation. *Journal of statistical computation and simulation*, 76(12):1049–1064, 2006.

[10] Stef Van Buuren and Karin Groothuis-Oudshoorn. mice: Multivariate imputation by chained equations in r. *Journal of statistical software*, 45:1–67, 2011.

[11] Jingchen Liu, Andrew Gelman, Jennifer Hill, Yu-Sung Su, and Jonathan Kropko. On the stationary distribution of iterative imputations. *Biometrika*, 101(1):155–173, 2014.

[12] Jian Zhu and Trivellore E Raghunathan. Convergence properties of a sequential regression multiple imputation algorithm. *Journal of the American Statistical Association*, 110(511):1112–1124, 2015.

[13] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pages 1278–1286. PMLR, 2014.

[14] Lovedeep Gondara and Ke Wang. Multiple imputation using deep denoising autoencoders. *arXiv preprint arXiv:1705.02737*, 2017.

[15] Pierre-Alexandre Mattei and Jes Frellsen. Leveraging the exact likelihood of deep latent variable models. *arXiv preprint arXiv:1802.04826*, 2018.

[16] Jinsung Yoon, James Jordon, and Mihaela Schaar. Gain: Missing data imputation using generative adversarial nets. In *International Conference on Machine Learning*, pages 5689–5698. PMLR, 2018.

[17] Steven Cheng-Xian Li, Bo Jiang, and Benjamin Marlin. Misgan: Learning from incomplete data with generative adversarial networks. *arXiv preprint arXiv:1902.09599*, 2019.

[18] Seongwook Yoon and Sanghoon Sull. Gamin: generative adversarial multiple imputation network for highly missing data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8456–8464, 2020.

[19] Alfredo Nazabal, Pablo M Olmos, Zoubin Ghahramani, and Isabel Valera. Handling incomplete heterogeneous data using vaes. *Pattern Recognition*, 107:107501, 2020.

[20] Oleg Ivanov, Michael Figurnov, and Dmitry Vetrov. Variational autoencoder with arbitrary conditioning. *arXiv preprint arXiv:1806.02382*, 2018.

[21] Trevor W Richardson, Wencheng Wu, Lei Lin, Beilei Xu, and Edgar A Bernal. Mcflow: Monte carlo flow models for data imputation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14205–14214, 2020.

[22] Pierre-Alexandre Mattei and Jes Frellsen. Miwae: Deep generative modelling and imputation of incomplete data sets. In *International Conference on Machine Learning*, pages 4413–4423. PMLR, 2019.

[23] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[24] Zongyu Dai, Zhiqi Bu, and Qi Long. Multiple imputation via generative adversarial network for high-dimensional blockwise missing value problems. *arXiv preprint arXiv:2112.11507*, 2021.

[25] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in neural information processing systems*, 29:2234–2242, 2016.

[26] Hoang Thanh-Tung and Truyen Tran. Catastrophic forgetting and mode collapse in gans. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10. IEEE, 2020.

[27] Shengjia Zhao, Jiaming Song, and Stefano Ermon. Towards deeper understanding of variational autoencoding models. *arXiv preprint arXiv:1702.08658*, 2017.

[28] James Lucas, George Tucker, Roger Grosse, and Mohammad Norouzi. Understanding posterior collapse in generative latent variable models. 2019.

[29] Boris Muzellec, Julie Josse, Claire Boyer, and Marco Cuturi. Missing data imputation using optimal transport. In *International Conference on Machine Learning*, pages 7130–7140. PMLR, 2020.

[30] Sebastian Jäger, Arndt Allhorn, and Felix Bießmann. A benchmark for data imputation methods. *Frontiers in big Data*, page 48, 2021.

[31] Graeme Hawthorne, Graeme Hawthorne, and Peter Elliott. Imputing cross-sectional missing data: comparison of common techniques. *Australian & New Zealand Journal of Psychiatry*, 39(7):583–590, 2005.

[32] Esther-Lydia Silva-Ramírez, Rafael Pino-Mejías, Manuel López-Coello, and María-Dolores Cubiles-de-la Vega. Missing value imputation on missing completely at random data using multilayer perceptrons. *Neural Networks*, 24(1):121–129, 2011.

[33] Euredit. Interim report on evaluation criteria for statistical editing and imputation. *Euredit Project*, 3, 2005.

[34] Marine Le Morvan, Julie Josse, Thomas Moreau, Erwan Scornet, and Gaël Varoquaux. Neumiss networks: differentiable programming for supervised learning with missing values. *arXiv preprint arXiv:2007.01627*, 2020.

[35] Alexandre Perez-Lebel, Gaël Varoquaux, Marine Le Morvan, Julie Josse, and Jean-Baptiste Poline. Benchmarking missing-values approaches for predictive models on health databases. *GigaScience*, 2022.

[36] Chao Ma, Sebastian Tschiatschek, José Miguel Hernández-Lobato, Richard Turner, and Cheng Zhang. Vaem: a deep generative model for heterogeneous mixed type data. *arXiv preprint arXiv:2006.11941*, 2020.

[37] Chao Ma, Sebastian Tschiatschek, Konstantina Palla, José Miguel Hernández-Lobato, Sebastian Nowozin, and Cheng Zhang. Eddi: Efficient dynamic discovery of high-value information with partial vae. *arXiv preprint arXiv:1809.11142*, 2018.

[38] Sarah Lewis, Tatiana Matejovicova, Yingzhen Li, Angus Lamb, Yordan Zaykov, Miltiadis Allamanis, and Cheng Zhang. Accurate imputation and efficient data acquisitionwith transformer-based vaes. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*, 2021.

[39] Marine Le Morvan, Julie Josse, Erwan Scornet, and Gaël Varoquaux. What's a good imputation to predict with missing values? *arXiv preprint arXiv:2106.00311*, 2021.

[40] Donald B Rubin. Inference and missing data. *Biometrika*, 63(3):581–592, 1976.

[41] Donald B Rubin. *Multiple imputation for nonresponse in surveys*, volume 81. John Wiley & Sons, 1987.

[42] Stef Van Buuren. *Flexible imputation of missing data*. CRC press, 2018.

[43] Shaun Seaman, John Galati, Dan Jackson, and John Carlin. What is meant by "missing at random"? *Statistical Science*, 28(2):257–268, 2013.

[44] Hyekyung Jung, Joseph L Schafer, and Byungtae Seo. A latent class selection model for nonignorably missing data. *Computational statistics & data analysis*, 55(1):802–812, 2011.

[45] Roderick JA Little. Pattern-mixture models for multivariate incomplete data. *Journal of the American Statistical Association*, 88(421):125–134, 1993.

[46] Karthika Mohan, Felix Thoemmes, and Judea Pearl. Estimation with incomplete data: The linear case. In *Proceedings of the International Joint Conferences on Artificial Intelligence Organization*, 2018.

[47] Aude Sportisse, Claire Boyer, and Julie Josses. Estimation and imputation in probabilistic principal component analysis with missing not at random data. *Advances in Neural Information Processing Systems*, 33, 2020.

[48] Trent Kyono, Yao Zhang, Alexis Bellot, and Mihaela van der Schaar. Miracle: Causally-aware imputation via learning missing data mechanisms. *Advances in Neural Information Processing Systems*, 34, 2021.

[49] Niels Bruun Ipsen, Pierre-Alexandre Mattei, and Jes Frellsen. not-miwae: Deep generative modelling with missing not at random data. *arXiv preprint arXiv:2006.12871*, 2020.

[50] Chao Ma and Cheng Zhang. Identifiable generative models for missing not at random data imputation. *Advances in Neural Information Processing Systems*, 34, 2021.

[51] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

[52] David A Marker, David R Judkins, and Marianne Winglee. Large-scale imputation for complex surveys. *Survey nonresponse*, 329341, 2002.

[53] Olga Troyanskaya, Michael Cantor, Gavin Sherlock, Pat Brown, Trevor Hastie, Robert Tibshirani, David Botstein, and Russ B Altman. Missing value estimation methods for dna microarrays. *Bioinformatics*, 17(6):520–525, 2001.

[54] Pedro J García-Laencina, José-Luis Sancho-Gómez, and Aníbal R Figueiras-Vidal. Pattern classification with missing data: a review. *Neural Computing and Applications*, 19(2):263–282, 2010.

[55] Trevor Hastie, Rahul Mazumder, Jason D Lee, and Reza Zadeh. Matrix completion and low-rank svd via fast alternating least squares. *The Journal of Machine Learning Research*, 16(1):3367–3402, 2015.

[56] Roderick JA Little and Donald B Rubin. *Statistical analysis with missing data*, volume 793. John Wiley & Sons, 2002.

[57] Garrett M Fitzmaurice, Nan M Laird, and James H Ware. *Applied longitudinal analysis*, volume 998. John Wiley & Sons, 2012.

[58] Patrizia Berti, Emanuela Dreassi, and Pietro Rigo. Compatibility results for conditional distributions. *Journal of Multivariate Analysis*, 125:190–203, 2014.

[59] Jörg Drechsler and Susanne Rässler. Does convergence really matter? In *Recent advances in linear models and related areas*, pages 341–355. Springer, 2008.

[60] Barry C Arnold, Enrique Castillo, and Jose Maria Sarabia. Conditionally specified distributions: an introduction (with comments and a rejoinder by the authors). *Statistical Science*, 16(3):249–274, 2001.

[61] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

[62] Xin He, Kaiyong Zhao, and Xiaowen Chu. Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622, 2021.

[63] Zi Wang, Chengtao Li, Stefanie Jegelka, and Pushmeet Kohli. Batched high-dimensional bayesian optimization via structural kernel learning. In *International Conference on Machine Learning*, pages 3656–3664. PMLR, 2017.

[64] Ahmed Alaa and Mihaela Schaar. Autoprognosis: Automated clinical prognostic modeling via bayesian optimization with structured kernel learning. In *International conference on machine learning*, pages 139–148. PMLR, 2018.

[65] Tammo Krueger, Danny Panknin, and Mikio L Braun. Fast cross-validation via sequential testing. *J. Mach. Learn. Res.*, 16(1):1103–1155, 2015.

[66] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization, 2018.

[67] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[68] Changhee Lee, William Zame, Ahmed Alaa, and Mihaela Schaar. Temporal quilting for survival analysis. In *The 22nd international conference on artificial intelligence and statistics*, pages 596–605. PMLR, 2019.

[69] Daniel Jarrett, Jinsung Yoon, Ioana Bica, Zhaozhi Qian, Ari Ercole, and Mihaela van der Schaar. Clairvoyance: A pipeline toolkit for medical time series. In *International Conference on Learning Representations*, 2020.

[70] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[71] Ian Goodfellow. Generative adversarial networks. *NIPS 2016 Tutorial*, 2016.

[72] Dheeru Dua and Casey Graff. Uci machine learning repository. *University of California, Irvine, School of Information and Computer Sciences*, 2017.

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

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

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.

Model Class	Regression Task	Classification Task
Linear Models	- max_iter $\in [100, 1000, 10000]$. - solver $\in$ [“auto", “svd", “cholesky", “lsqr", “sparse_cg", “sag", “saga"]	-solver $\in$ [“newton-cg", “lbfgs", “sag", “saga"] - C $\in [1e-3, 1e-2]$ -multi_class $\in$ [“auto", “ovr", “multinomial"] - class_weight $\in$ [“balanced", None]
XGBoost	- reg_lambda $\in [1e-3, 10.0]$ - reg_alpha $\in [1e-3, 10.0]$ - colsample_bytree $\in [0.1, 0.9]$ - colsample_bynode $\in [0.1, 0.9]$ - colsample_bylevel $\in [0.1, 0.9]$ - subsample $\in [0.1, 0.9]$ - lr $\in [1e-4, 1e-3, 1e-2]$ - max_depth $\in [2, 9]$ - n_estimators $\in [10, 100]$	- reg_lambda $\in [1e-3, 10.0]$ - reg_alpha $\in [1e-3, 10.0]$ - colsample_bytree $\in [0.1, 0.9]$ - colsample_bynode $\in [0.1, 0.9]$ - colsample_bylevel $\in [0.1, 0.9]$ - subsample $\in [0.1, 0.9]$ - lr $\in [1e-4, 1e-3, 1e-2]$ - max_depth $\in [2, 9]$ - min_child_weight $\in [0, 300]$ - n_estimators $\in [10, 100]$ - max_bin $\in [256, 512]$ - booster $\in$ [“gbtree", “gblinear", “dart"]
CatBoost	- depth $\in [1, 5]$ - n_estimators $\in [10, 100]$ - grow_policy $\in$ [None,“Depthwise", “SymmetricTree",“Lossguide"]	- depth $\in [1, 5]$ - n_estimators $\in [10, 100]$ - grow_policy $\in$ [None,“Depthwise", “SymmetricTree",“Lossguide"]
Random Forest	- criterion $\in$ [“mse", “mae"] - max_features $\in$ [“auto", “sqrt", “log2"] - min_samples_split $\in [2, 5, 10]$ - min_samples_leaf $\in [2, 5, 10]$ - max_depth $\in [1, 4]$	- criterion $\in$ [“gini", “entropy"] - max_features $\in$ [“auto", “sqrt", “log2"] - min_samples_split $\in [2, 5, 10]$ - min_samples_leaf $\in [2, 5, 10]$ - max_depth $\in [1, 4]$
Neural Nets	- n_layers_hidden $\in [1, 2]$ - n_units_hidden $\in [10, 100]$ - lr $\in [1e-4, 1e-3]$ - weight_decay $\in [1e-4, 1e-3]$ - dropout $\in [0., 0.2]$ - clipping_value $\in [0, 1]$	- n_layers_hidden $\in [1, 2]$ - n_units_hidden $\in [10, 100]$ - lr $\in [1e-4, 1e-3]$ - weight_decay $\in [1e-4, 1e-3]$ - dropout $\in [0., 0.2]$ - clipping_value $\in [0, 1]$

Table 5: Hyperparameter domain for each model class, and for each task type.

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 γ 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 $\pm$ std of RMSE across different settings on 5 datasets. All values are scaled by a factor of 10 for readability. Best results are emboldened.

	airfoil	**california**	**compression**	**letter**	**wine_white**
`ice_lr`	2.349 ± 0.483	0.789 ± 0.324	1.429 ± 0.215	1.087 ± 0.149	0.919 ± 0.060
`ice_rf`	2.365 ± 0.533	0.777 ± 0.339	1.615 ± 0.141	1.118 ± 0.148	0.987 ± 0.034
`ice_cb`	2.078 ± 0.482	0.726 ± 0.327	1.251 ± 0.204	0.750 ± 0.099	0.877 ± 0.054
`global_search`	1.599 ± 0.322	0.745 ± 0.326	1.031 ± 0.175	0.527 ± 0.067	0.853 ± 0.049
`column_naive`	1.861 ± 0.446	0.713 ± 0.333	1.094 ± 0.167	0.525 ± 0.065	0.845 ± 0.081
`wo_flexibility_rf`	2.689 ± 0.755	0.762 ± 0.331	1.807 ± 0.186	1.117 ± 0.152	0.982 ± 0.031
`wo_flexibility_cb`	1.594 ± 0.367	0.740 ± 0.336	1.082 ± 0.157	0.757 ± 0.100	0.876 ± 0.050
`wo_adaptivity`	1.665 ± 0.360	0.721 ± 0.354	1.047 ± 0.176	0.526 ± 0.066	0.890 ± 0.074
`HyperImpute`	$\mathbf{1.479 \pm 0.294}$	$\mathbf{0.704 \pm 0.336}$	$\mathbf{1.013 \pm 0.145}$	$\mathbf{0.524 \pm 0.067}$	$\mathbf{0.801 \pm 0.053}$

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

	airfoil	**california**	**compression**	**letter**	**wine_white**
`ice_lr`	1.066 ± 0.307	0.330 ± 0.192	0.781 ± 0.219	0.526 ± 0.069	0.584 ± 0.093
`ice_rf`	1.135 ± 0.307	0.354 ± 0.173	1.010 ± 0.040	0.799 ± 0.106	0.648 ± 0.047
`ice_cb`	0.790 ± 0.272	0.196 ± 0.107	0.537 ± 0.053	0.400 ± 0.045	0.491 ± 0.093
`global_search`	0.375 ± 0.105	0.253 ± 0.171	0.329 ± 0.045	0.257 ± 0.010	0.410 ± 0.088
`column_naive`	0.516 ± 0.278	0.155 ± 0.085	0.315 ± 0.030	0.261 ± 0.016	0.390 ± 0.172
`wo_flexibility_rf`	1.217 ± 0.366	0.329 ± 0.153	1.103 ± 0.079	0.807 ± 0.117	0.645 ± 0.060
`wo_flexibility_cb`	0.443 ± 0.142	0.188 ± 0.097	0.369 ± 0.057	0.394 ± 0.043	0.495 ± 0.096
`wo_adaptivity`	0.327 ± 0.092	0.170 ± 0.111	0.307 ± 0.025	0.256 ± 0.009	0.485 ± 0.164
`HyperImpute`	$\mathbf{0.291 \pm 0.103}$	$\mathbf{0.114 \pm 0.060}$	$\mathbf{0.289 \pm 0.041}$	$\mathbf{0.256 \pm 0.010}$	$\mathbf{0.426 \pm 0.141}$

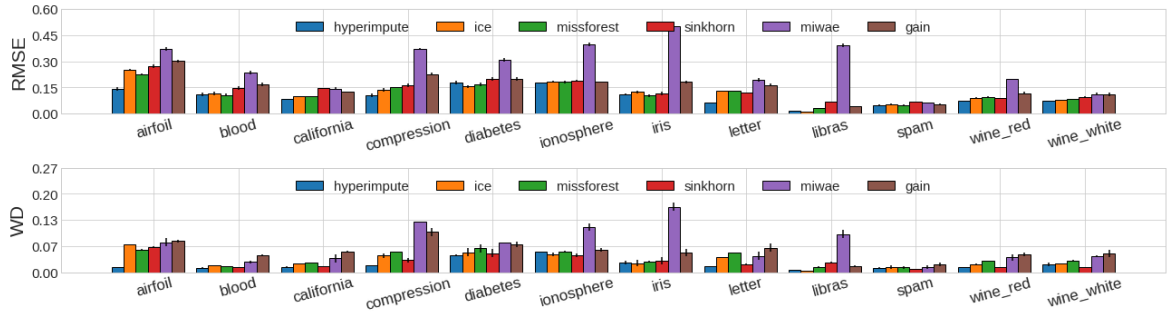
B.4. Model Selections

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.

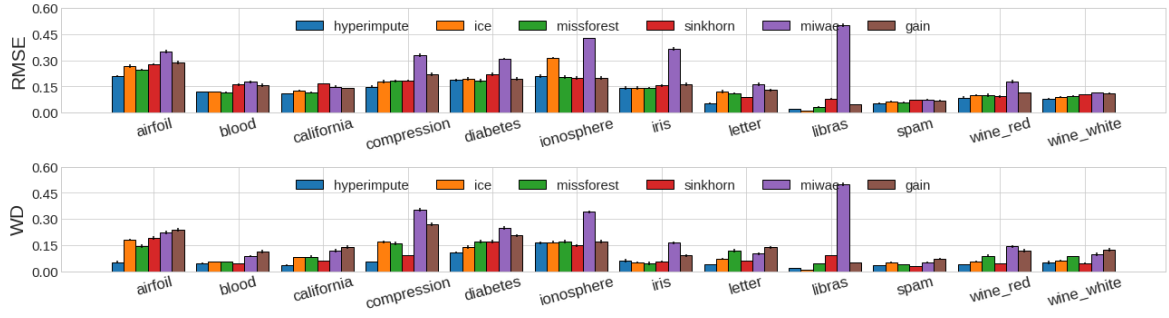
Most notably, boosting algorithms and neural nets (NNs), which are highly expressive algorithms but require larger amounts of data to train, are more prevalent in low missingness rates and larger datasets. By contrast, linear regression (LR) and bagging algorithms, which are less expressive but lower 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

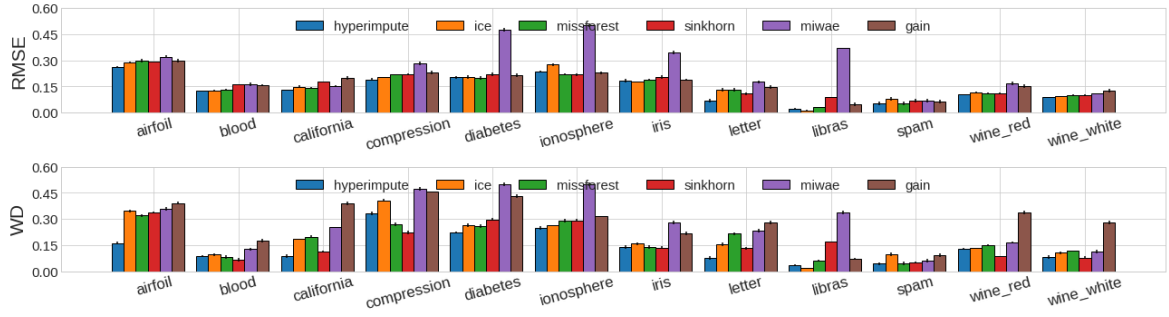
Lastly, we present convergence results for MAR simulations at 0.3 missingness for the 12 UCI datasets employed in our experiments. We are interested in comparing the rates of convergence across imputation tasks. Evidently, convergence is generally achieved after 4 iterations, with the model's internal 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).



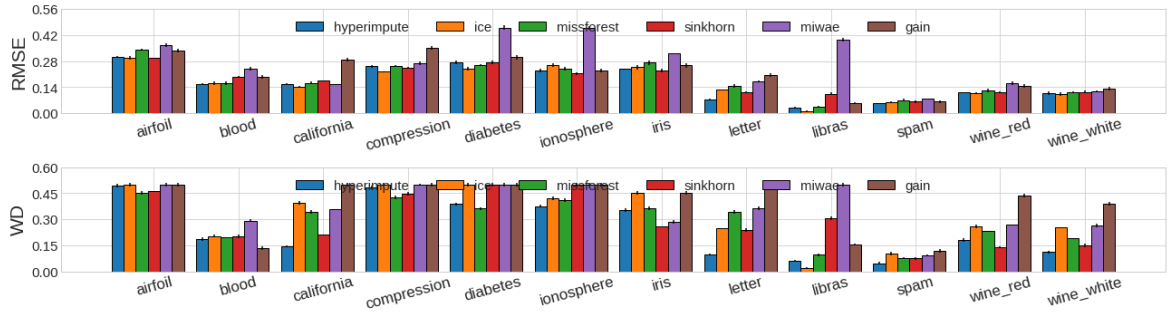
(a) MCAR-0.1



(b) MCAR-0.3

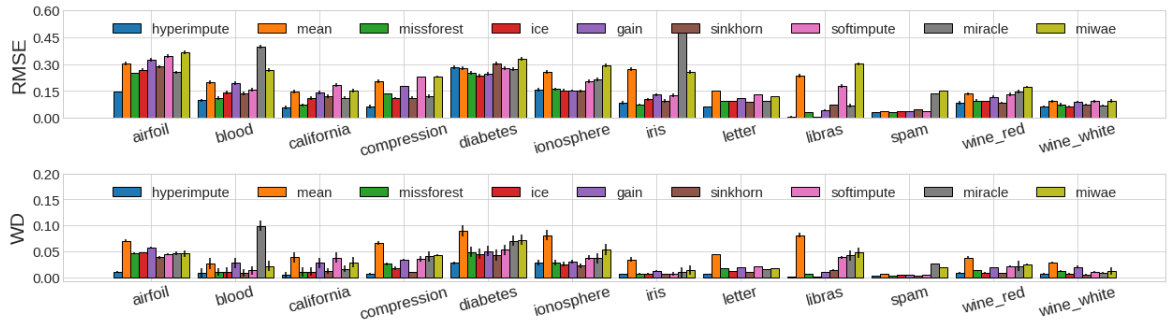


(c) MCAR-0.5

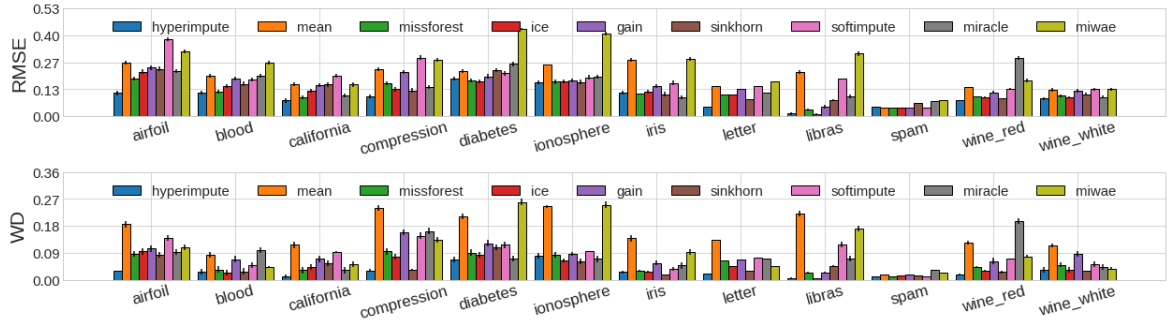


(d) MCAR-0.7

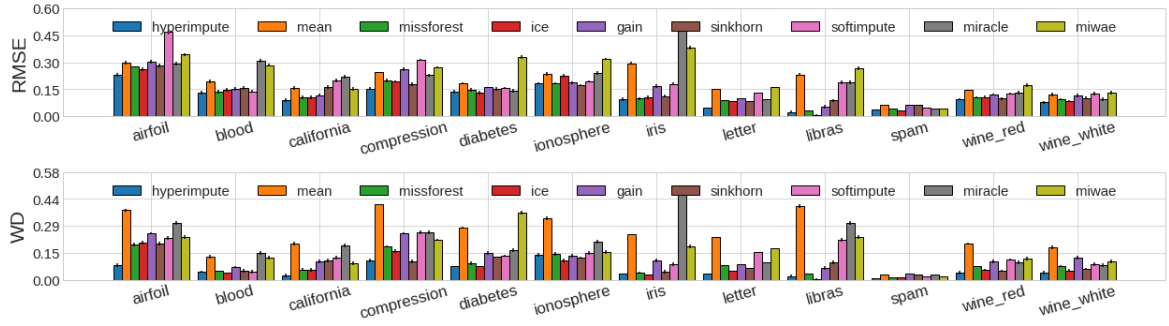
Figure 6: *Overall Performance.* Experiments on 12 UCI datasets under MCAR simulations at four levels of missingness—{0.1, 0.3, 0.5, 0.7}. Results shown as mean $\pm$ std of RMSE and WD.



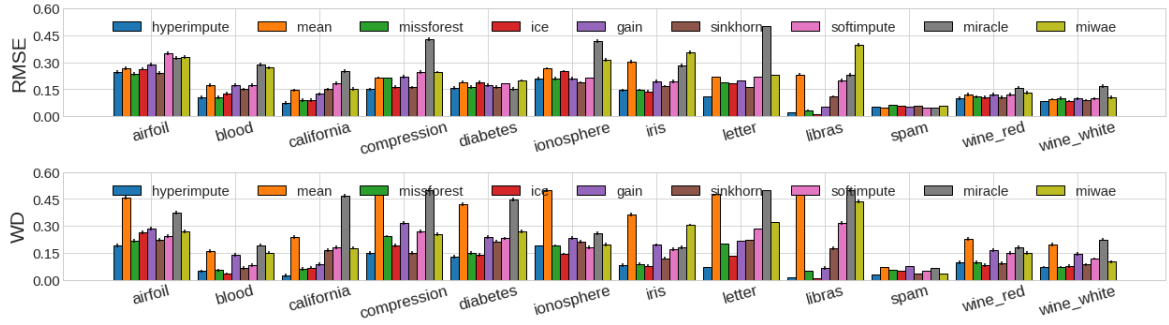
(a) MAR-0.1



(b) MAR-0.3

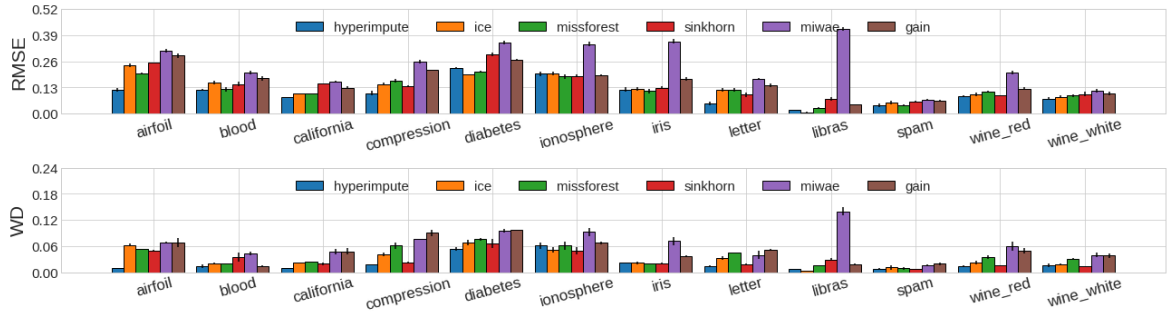


(c) MAR-0.5

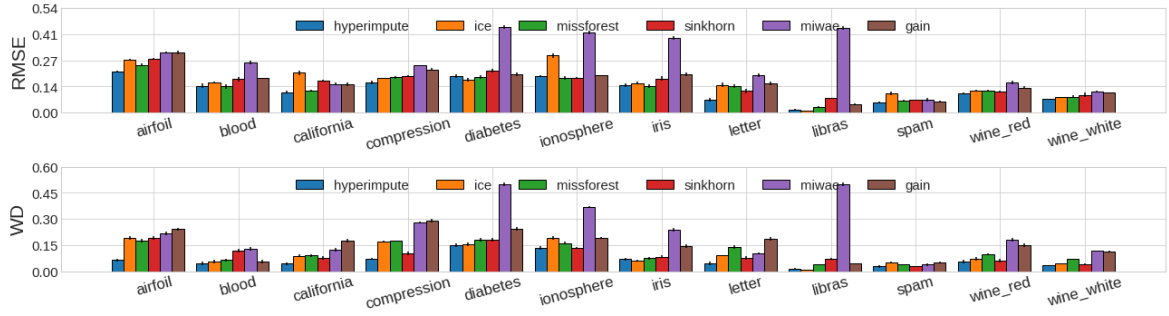


(d) MAR-0.7

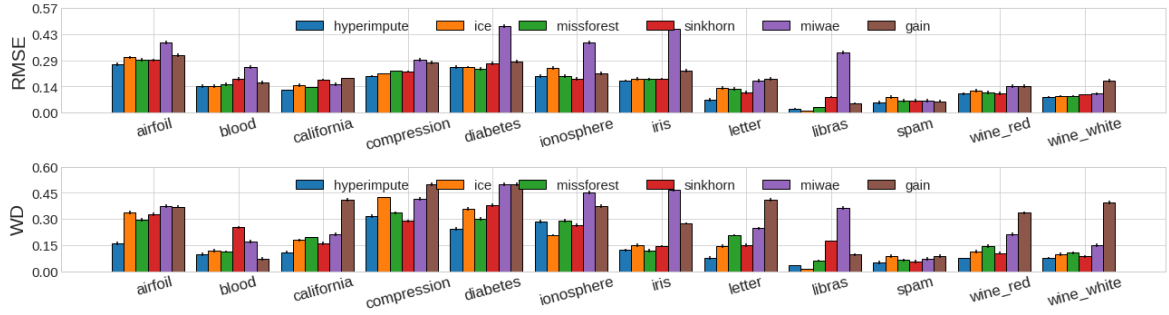
Figure 7: *Overall Performance.* Experiments on 12 UCI datasets under MAR simulations at four levels of missingness—{0.1, 0.3, 0.5, 0.7}. Results shown as mean $\pm$ std of RMSE and WD.



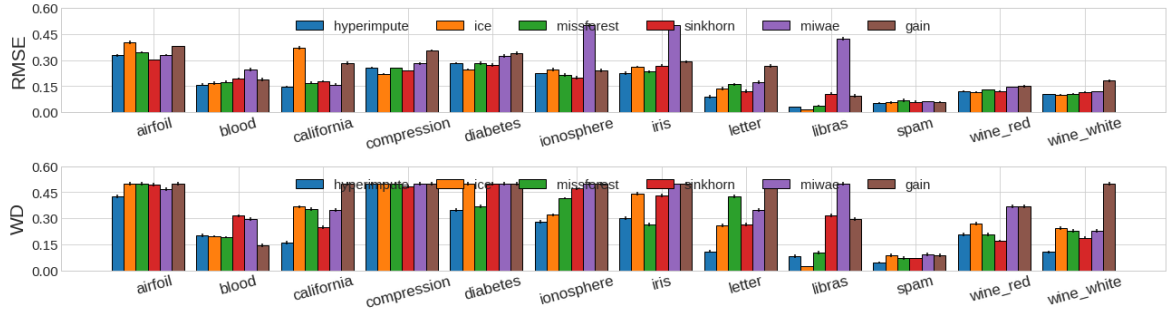
(a) MNAR-0.1



(b) MNAR-0.3

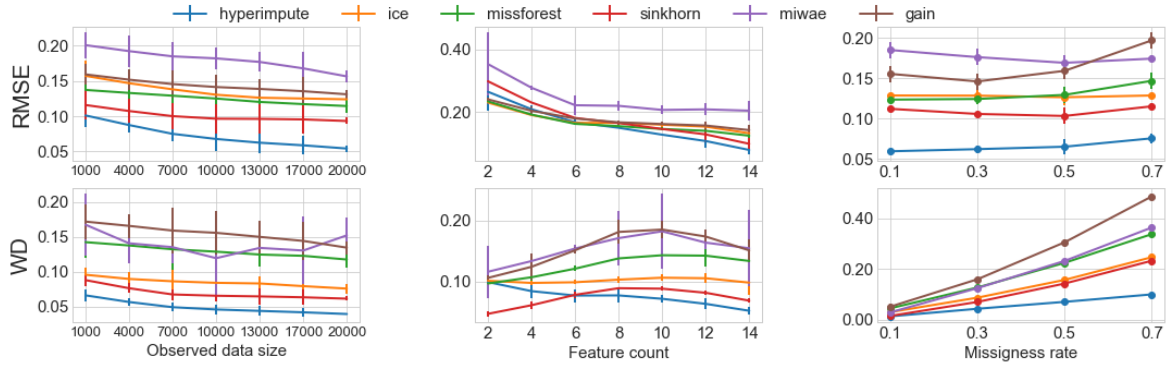


(c) MNAR-0.5

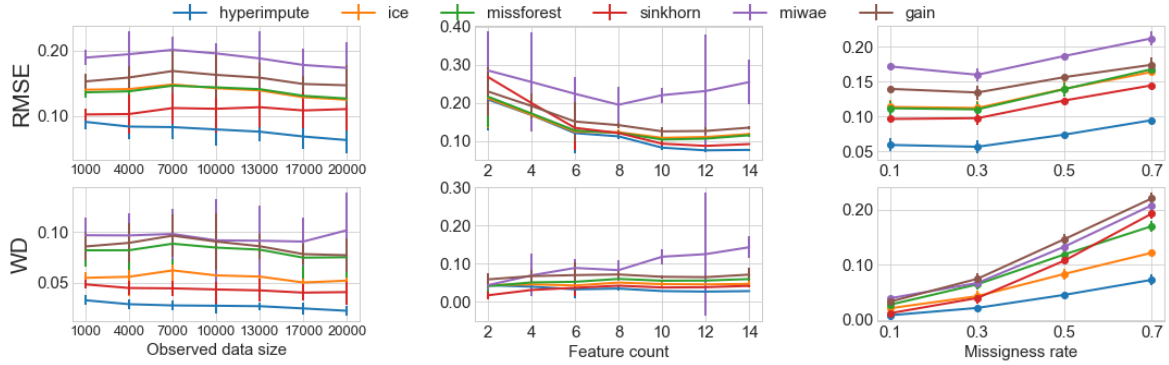


(d) MNAR-0.7

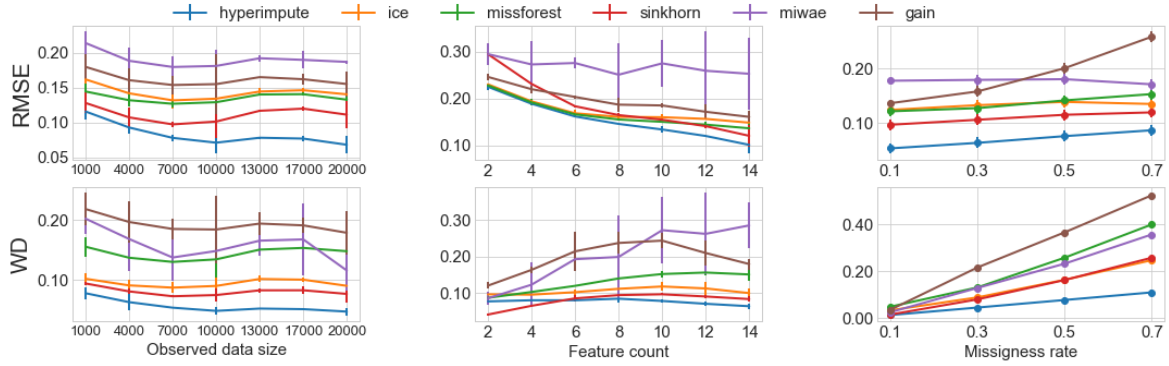
Figure 8: *Overall Performance.* Experiments on 12 UCI datasets under MNAR simulations at four levels of missingness—{0.1, 0.3, 0.5, 0.7}. Results shown as mean $\pm$ std of RMSE and WD.



(a) MCAR

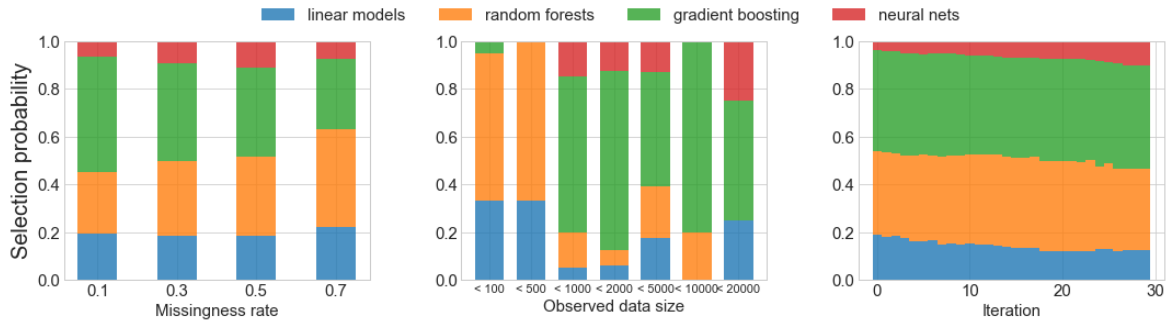


(b) MAR

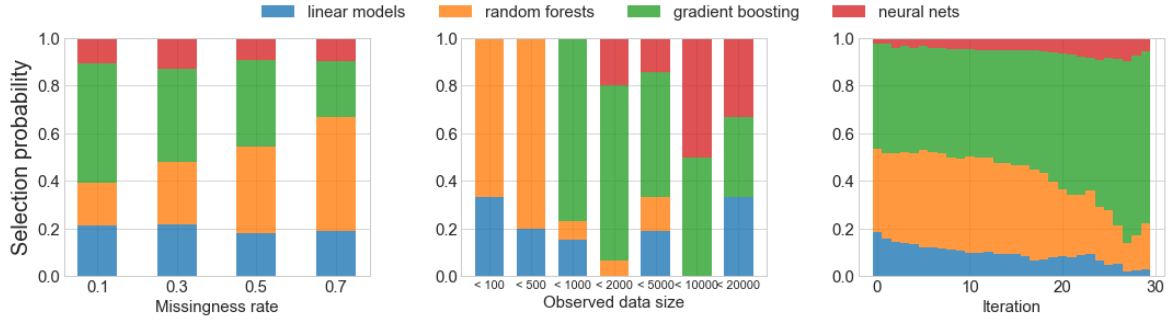


(c) MNAR

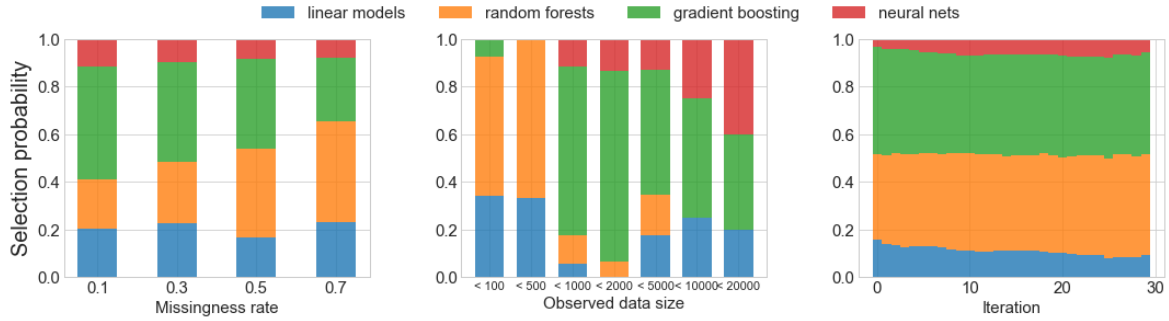
Figure 9: *Sensitivity Analysis.* Experiments performed on the `letter` dataset under the MCAR, MAR and MNAR simulations. Results shown in terms of mean $\pm$ std of RMSE and WD with sensitivities according to (a) observed data size, (b) feature count, and (c) missingness rate. When not perturbed for analysis, the observed data size is fixed at $N = 20,000$, feature count at $D = 14$, and missingness rate at 0.3.



(a) MCAR



(b) MAR



(c) MNAR

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.

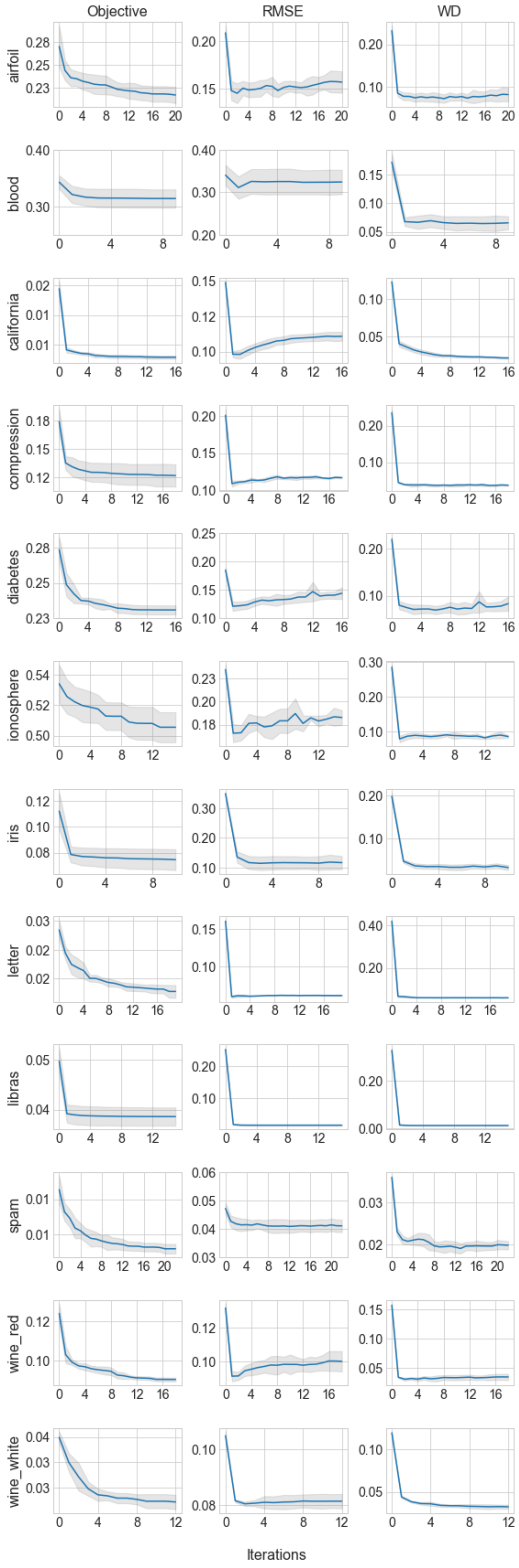


Figure 11: *Model Convergence.* Experiments performed using MAR at 0.3 missingness rate on 12 UCI datasets. Mean $\pm$ std of different metrics, including (a) objective error, (b) RMSE, (c) WD.

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).

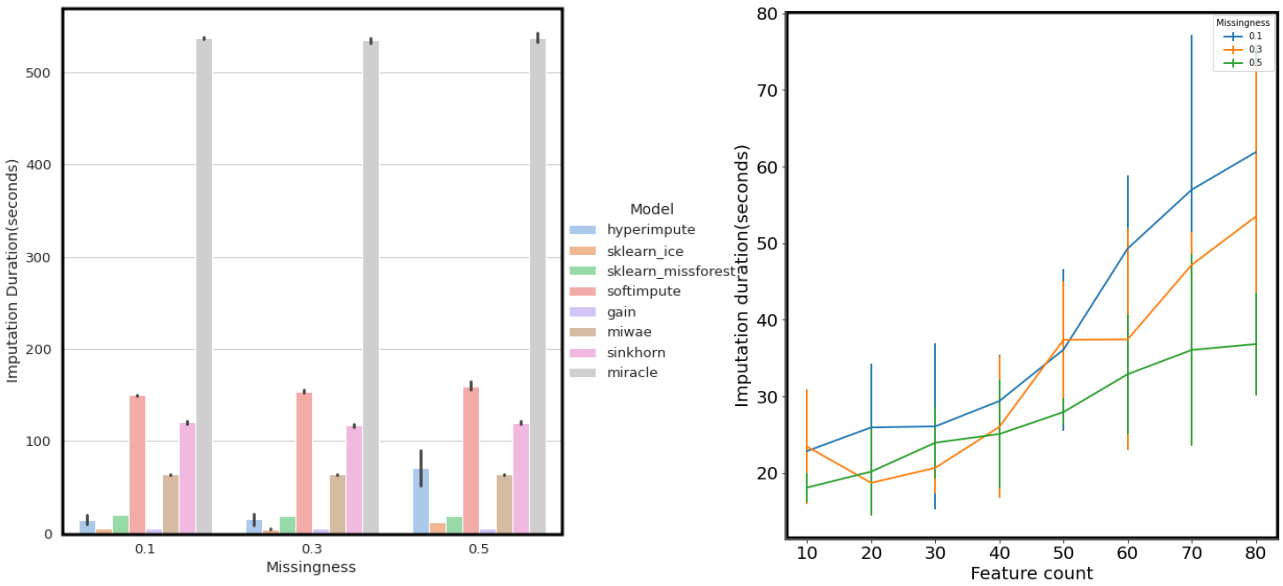
Regarding (1), since we work in the (realistic) setting where we are *not* given access to completely-observed data during training, it is theoretically **impossible** to perform hyperparameter optimization for an imputation method *per se*: To do so requires "ground truths" of the *missing* values themselves (for measuring imputation quality), which we do not have. (Note that cross-validating based on imputations of *observed* values is futile: The identity $f(\mathcal{D}):=\mathcal{D}$ would appear globally optimal despite imputing nothing at all. And without knowing the true missingness pattern, naively adding *artificial* missingness for the puposes 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.