Missing value imputation with MERCS: a faster alternative to MissForest[⋆]

Elia Van Wolputte[1,2][0000−0002−6502−0809] and
Hendrik Blockeel[1,2][0000−0003−0378−3699]

[1] KU Leuven, Department of Computer Science, Belgium
[2] Leuven.AI, Belgium
`firstname.lastname@kuleuven.be`

Abstract. Fundamentally, many problems in Machine Learning are understood as some form of *function approximation*; given a dataset $\mathcal{D}$, learn a function $f_{\boldsymbol{X} \rightarrow \boldsymbol{Y}}$. However, this overlooks the ubiquitous problem of missing data. E.g., if afterwards an unseen instance has missing input variables, we actually need a function $f_{\boldsymbol{X'} \rightarrow \boldsymbol{Y}}$ with $\boldsymbol{X}' \subset \boldsymbol{X}$ to predict its label. Strategies to deal with missing data come in three kinds: naive, probabilistic and iterative. The naive case replaces missing values with a fixed value (e.g. the mean), then uses $f_{\boldsymbol{X} \rightarrow \boldsymbol{Y}}$ as if nothing was ever missing. The probabilistic case has a generative model $\mathcal{M}$ of $\mathcal{D}$ and uses probabilistic inference to find the most likely value of $\boldsymbol{Y}$, given values for any subset of $\boldsymbol{X}$. The iterative approach consists of a loop: according to some model $\mathcal{M}$, fill in all the missing values based on the given ones, retrain $\mathcal{M}$ on the completed data and redo your predictions, until these converge. `MissForest` is a well-known realization of this idea using Random Forests. In this work, we establish the connection between `MissForest` and `MERCS` (a multi-directional generalization of Random Forests). We go on to show that under certain (realistic) conditions where the retraining step in `MissForest` becomes a bottleneck, `MERCS` (which is trained only once) offers at-par predictive performance at a fraction of the time cost.

Keywords: Missing value imputation · Ensemble methods · Multi-directional models · decision trees

1 Introduction

Many machine learning methods assume there are no missing values in the data, or missing values are relatively infrequent. Under this assumption, a variety of techniques has been proposed to handle missing data. It is useful to maintain a clear distinction between two cases: *missing values at training time* (relevant during learning) and *missing values at prediction time* (making a prediction, using a given model, for an instance that lacks certain information needed by the

[⋆] Code available at `github.com/eliavw/missmercs` model). First, when missing values occur at training time, the learning procedure may deal with them by ignoring all instances with missing values, ignoring attributes that have missing values, guessing the missing value (imputation) before proceeding with the computations, or using other techniques. The second case, missing values at prediction time, is quite a different problem: a model is given, but the model needs information that is not available. Nevertheless, some techniques for handling missing values during prediction resemble those for the training phase, e.g. imputation can be used, if some model for imputation is available.

In this paper, we focus specifically on missing values at prediction time. There are contexts where missing values at prediction time may be much more frequent, and possibly also more systematic, than typically assumed by many learners. To illustrate, consider two practical examples;

– First, **machine learning in industrial contexts** often depends on sensor data. Consider an AI-system (e.g. a predictive maintenance application) which makes automatic decisions based upon input information coming from sensors. When a single sensor breaks down and no longer provides information, the AI-system needs to carry on and perform as well as possible, although less input information is now available.
– Second, consider a common **spreadsheet**. Suppose a user filling in data in a spreadsheet or a web form: ML methods exist to assist users by predicting information to be inserted in certain cells. Ideally, these predictions use as much as possible information filled in elsewhere, regardless of exactly which cells are already filled in and which ones are not. So, at prediction time, robustness with regard to missing input information is crucial.

In both cases, at prediction time we need a model $\mathcal{M}$ that can predict some output variable(s) $\boldsymbol{Y}$ from input variable(s) $\boldsymbol{X}$, so we can regard $\mathcal{M}$ as a function from $\boldsymbol{X}$ to $\boldsymbol{Y}$. However, the actual input that is available for a particular prediction often consists of values for a strict subset $\boldsymbol{X}' \subset \boldsymbol{X}$. In the first example, this is caused by malfunctioning sensors, in the second one by empty cells in the spreadsheet. Thus, handling missing values at prediction time boils down to the task of deriving from $\mathcal{M} : \boldsymbol{X} \to \boldsymbol{Y}$ another function $\mathcal{M}' : \boldsymbol{X}' \to \boldsymbol{Y}$ with $\boldsymbol{X}' \subset \boldsymbol{X}$ which still makes maximally accurate predictions.

In a nutshell, we propose to solve this problem as follows: use a tree-based approach such as `MissForest`[17], but avoid its multiple training iterations. Given some robust prediction strategies, so-called `MERCS` models[20] could do just that. So, our proposal decomposes into two research questions:

Q1 Can a `MERCS` model be made robust to missing values at prediction time?
Q2 How does `MERCS` compare against `MissForest`, a well-established tree-based technique to deal with missing data?

First, section 2 pinpoints the current gap in knowledge, and thus provides further context and motivation for our solution strategy (i.e. **Q1** and **Q2**). On one hand, we find `MissForest`[17]: a powerful tree-based approach for missing data, which is iterative and thus ill-suited for missing values at prediction time. On the other hand, we find `MERCS`[20]: a somewhat similar, but non-iterative, tree-based model, which currently lacks prediction strategies to deal with missing data effectively.

Section 3 outlines our proposal to answer **Q1** (how to extend the `MERCS` framework with robust prediction strategies) which constitutes our algorithmic contribution. Two key ideas matter here. First, *attribute importance*: this quantifies the relevance of trees for a given prediction task. Second: *chaining*: inspired by `MissForest`, `MERCS` can be made to use outputs of some trees as inputs for others.

Lastly, sections 4 and 5 contain experimental evaluations of **Q1** and **Q2** respectively. Ultimately, the answer to both **Q1** and **Q2** is positive: when dealing with missing input values at prediction time, `MERCS` models are a viable alternative to `MissForest`.

2 Related Work and Background

We focus on missing value handling at prediction time: given a model $\mathcal{M}$ that represents a function $\mathcal{M} : \boldsymbol{X} \rightarrow \boldsymbol{Y}$ and a *query-instance* x_q which has only values for a strict subset $\boldsymbol{X}' \subset \boldsymbol{X}$, how can we still use $\mathcal{M}$ to predict the value of $\boldsymbol{Y}$?

The discussion of related work is organized into four parts, each covering a specific approach for missing value handling. First, we consider naive approaches for handling missing values. Second, we discuss probabilistic graphical models, which handle unobserved values so naturally that the term "missing values handling" is typically not even used in that context. Third, we discuss iterative approaches in general, and `MissForest` in particular. `MissForest` is a popular tree-based technique for missing value imputation at training time. Lastly, we discuss `MERCS`, another tree-based framework in some ways similar to `MissForest`, but which could be more suitable in the specific case of missing values at prediction time.

2.1 Naive Methods

A generally applicable approach is what we call naive methods: guess the missing values. Concretely, this comes down to a one-size-fits-all strategy: simply fill in the mean, median or mode of the variable. We call this "naive", as it just fills in the same value for all instances.

The **advantage** of this technique is its low cost, both in time and memory. The obvious **disadvantage** is the limited accuracy of a naive approach. In our context, i.e. where predictive accuracy matters, naive methods can still serve as a baseline to compare other methods to, but nothing more.

2.2 Probabilistic Methods

Probabilistic graphical models (PGMs), such as Bayesian networks or Markov random fields[8, 12, 13], model the probability distribution $P_{\boldsymbol{X}}$ over all variables. From this distribution, any marginal distribution $P_{\boldsymbol{X}'}$ with $\boldsymbol{X}' \subset \boldsymbol{X}$ can be computed, as well as any conditional distribution $P_{\boldsymbol{Y}|\boldsymbol{X}'}$ with $\boldsymbol{X}', \boldsymbol{Y} \subseteq \boldsymbol{X}$. As the joint distribution uniquely defines all marginal and conditional distributions, the target variable $\boldsymbol{Y}$ can be predicted from any subset $\boldsymbol{X}'$ that is equal to the set of all known variables.

Their versatility is the main **advantage** of probabilistic methods. In a sense, the "problem of missing values" simply does not even exist in this context: the optimal way of handling them follows naturally from the probabilistic model itself.

The **disadvantage** are the computational costs involved. Explicitly deriving the marginal/conditional probabilities in a PGM is NP-hard in the general case. Performing probabilistic inference in the original PGM is NP-hard too. In practice, approximate inference in the original PGM is used at prediction time, but even that can be costly. Another issue are data-types: in practice, PGMs work best on nominal data. Numeric data or mixtures of nominal/numeric data can be challenging for probabilistic approaches.

2.3 Iterative Approaches

Iterative approaches[5, 18] gradually refine their imputations by means of a simple loop. First, for each variable in your dataset, you learn a predictive model, using the other variables of the dataset as inputs. Second, you use that model to fill in any missing values of that variable. This too, is repeated for each variable in the dataset. Third, you repeat this entire process (both training and prediction) until you reach a stopping criterion which indicates when no more progress is being made.

`MissForest`[17] is a specific implementation of the aforementioned idea. In `MissForest`, the underlying predictive models are Random Forests. The stopping criterion is dual; the loop is stopped when the resulting change from iteration i to iteration $i + 1$ is less then a user-defined parameter γ or when a certain maximum number of iterations n is exceeded.

The **advantages** of `MissForest` are twofold. First, Random Forests are non-parametric and make relatively few assumptions about the underlying data distributions. Second, they work well on both numeric and nominal data, or on mixtures of the two. This versatility with regard to data-types is often highlighted[21] as the "killer-feature" which makes `MissForest` such an attractive option in real-world scenarios.

The **disadvantage** of `MissForest` (or any iterative approach for that matter) is that, essentially by definition, it is geared towards missing data at training time: this is not the problem we set out to solve. Indeed, the training phase at each step of the iteration involves significant costs in time (i.e. you need several training rounds) and memory (i.e. you always need to have data to train on).

When an incomplete dataset is given, iterative approaches are perfectly equipped to fill in the gaps introduced by the missing data. But at prediction time, when unseen, incomplete query-instances x_q come in one by one and need a prediction for $\boldsymbol{Y}$ right away, this double cost of keeping a training set in memory and retraining your model each time a new query-instance pops up, quickly becomes a bottleneck.

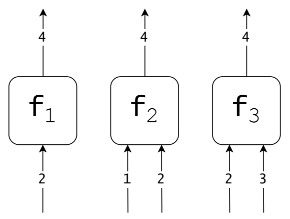
2.4 `MERCS`

`MERCS`[20] is a method for learning *multi-directional* ensembles of decision trees. This as opposed to classical ensembles of decision trees, which are *uni-directional*: a single function $f_{\boldsymbol{X}\to\boldsymbol{Y}}$ is learned that predicts some output variable(s) $\boldsymbol{Y}$ from input variable(s) $\boldsymbol{X}$, and it is known at training time what $\boldsymbol{X}$ and $\boldsymbol{Y}$ are. Bagging[3], Random Forests[4] and Gradient Boosted Trees[7] are all examples of methods that learn such uni-directional ensembles. In a multi-directional ensemble, a single tree may have multiple target variables (so-called multi-target trees), and different trees may have different sets of target variables. Such ensembles can be learned using a method that is quasi identical to the learning algorithm for Random Forests: the only difference is that for each new tree $T^i_{\boldsymbol{X}^i\to\boldsymbol{Y}^i}$ that is learned, a new set of target variables $\boldsymbol{Y}^i$ is chosen. Learning methods for `MERCS` models differ mostly in terms of how they choose $\boldsymbol{Y}^i$ for each tree. For instance, using one target variable per tree often gives slightly higher accuracy for individual trees, but having many target variables in one tree can reduce the size of the ensemble without reducing the number of trees available for predicting a given variable. Cf. Van Wolputte et al.[20] for more details.

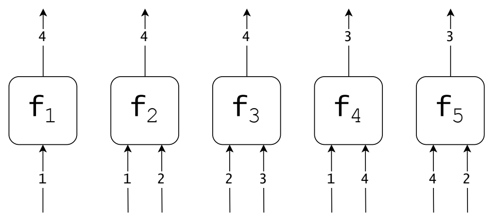
What is interesting here is that `MERCS` is somewhat similar to `MissForest`: both are multi-directional ensembles of decision trees, where any variable of the dataset can be predicted by at least one tree of the ensemble. But whereas `MissForest` was originally conceived to do missing value imputation in a given dataset, `MERCS` was not. `MERCS` originated as a fast, tree-based alternative to PGMs: learn a *model* $\mathcal{M}$ from dataset $\mathcal{D}$.

This begs the question: could `MERCS`, like `MissForest`, become a powerful tool for missing value imputation? We believe it does. The **advantages** of `MERCS` in this context are twofold. First, like probabilistic approaches, `MERCS` learns a model $\mathcal{M}$ from training data $\mathcal{D}$. Afterwards, there is no need to keep this training data around, all the necessary knowledge is encoded in the model itself. As a consequence, `MERCS` would be particularly interesting for missing value imputation at prediction time, a regime where iterative approaches struggle. Second, like `MissForest`, `MERCS` is a tree-based approach, which means a.o. that `MERCS` can also deal with (mixtures of) nominal and numeric variables.

At this point, the main **disadvantage** is that it remains unclear whether `MERCS` can handle missing values effectively. In order to be proficient in such a regime, `MERCS` needs a prediction strategy which is *robust to missing data*: given an unseen instance x_q, `MERCS` should still able to do a high quality prediction for the value of $\boldsymbol{Y}$, even if x_q has some missing values. How to achieve this will be the topic of section 3.



(a) A Random Forest (Eq. 1)[4]. The input attributes $\boldsymbol{X}^i$ of the component trees $T^i_{\boldsymbol{X}^i \rightarrow \boldsymbol{Y}}$ are random subsets of $\boldsymbol{A} \setminus \boldsymbol{Y}$.



(b) MERCS (Eq. 2) [20]. MERCS generalizes Random Forests and also selects output attributes $\boldsymbol{Y}^i$ at random.

Fig. 1: Random Forests and MERCS. Attributes $A_j \in \boldsymbol{A}$ are depicted as lines annotated with their respective indices j. A decision tree, $T^i_{\boldsymbol{X}^i \rightarrow \boldsymbol{Y}^i}$, is depicted as a box connecting its input ($\boldsymbol{X}^i$) to its output ($\boldsymbol{Y}^i$) attributes.

3 Robust Prediction Strategies for MERCS

This section outlines our answer to research question **Q1**: how to make the MERCS framework robust to missing values at prediction time. The motivation behind this approach is explained in section 2, whereas the experimental evaluation happens in section 4.

In the following, we use T^i $(i = 1 \ldots k)$ to denote the different trees in the model $\mathcal{M}$. $\boldsymbol{X}^i$ refers to the set of input attributes used by tree T^i, and $\boldsymbol{Y}^i$ to the set of output (or target) attributes of T^i. Similarly, we use $q_{\boldsymbol{I} \rightarrow \boldsymbol{O}}$ to denote a particular prediction task or *query*, where $\boldsymbol{I}$ denotes the set of attributes whose value is given (i.e. input attributes of $q_{\boldsymbol{I} \rightarrow \boldsymbol{O}}$) and $\boldsymbol{O}$ the set of attributes to be predicted (i.e. output attributes of $q_{\boldsymbol{I} \rightarrow \boldsymbol{O}}$). Furthermore, $\boldsymbol{A}$ simply refers to the set of all the attributes of a given dataset $\mathcal{D}$.

Take a Random Forest (Fig. 1a),

$$RF(\boldsymbol{X}, \boldsymbol{Y}) = \{T^i_{\boldsymbol{X}^i \rightarrow \boldsymbol{Y}} | \boldsymbol{X}^i \subset \boldsymbol{A} \setminus \boldsymbol{Y}\}, \tag{1}$$

and introduce randomness in the target attributes. In this way, $RF(\boldsymbol{X}, \boldsymbol{Y})$ generalizes to a multi-directional ensemble of decision trees, or a MERCS model (Fig. 1b),

$$M(\boldsymbol{A}) = \{\, T^i_{\boldsymbol{X}^i \rightarrow \boldsymbol{Y}^i} | \boldsymbol{X}^i, \boldsymbol{Y}^i \subset \boldsymbol{A},\, \boldsymbol{X}^i \cap \boldsymbol{Y}^i = \emptyset\}. \tag{2}$$

Now, to answer an arbitrary query $q_{\boldsymbol{I} \rightarrow \boldsymbol{O}}$ a MERCS model needs a *prediction strategy.* This has two reasons. First, note that $q_{\boldsymbol{I} \rightarrow \boldsymbol{O}}$ is not known at training time, and second, learning a dedicated decision tree for every possible $q_{\boldsymbol{I} \rightarrow \boldsymbol{O}}$ is simply not feasible. Thus, this prediction strategy decides how to optimally use the available T^i, present in the MERCS model (Eq. 2), to answer any incoming $q_{\boldsymbol{I} \rightarrow \boldsymbol{O}}$ as accurately as possible.

Rather than a single prediction strategy, we define a naive baseline and three, increasingly complex, strategies. These subdivide into two groups. First, *single-layer* strategies, where *attribute importance* quantifies the relevance of T^i for a given prediction task $q_{\boldsymbol{I}\rightarrow\boldsymbol{O}}$. Second, *multi-layer* strategies that, like `MissForest`, use *chaining*: take the prediction of one tree T^i as the input for another tree T^j.

3.1 Attribute Importance and Single-Layer Prediction Strategies

Assume that for a given tree T^i, only some of its input attributes $\boldsymbol{X^i}$ are known. The more inputs are missing, the less accurate we expect the predictions of T^i to be. But not all attributes in $\boldsymbol{X}^i$ are equally important for the prediction. One way to measure this is **attribute importance**[10]:

$$I(A_j, T^i) \propto \sum_{\{a(\tau)=A_j\}} p(\tau)\Delta i(\tau) \tag{3}$$

where τ ranges over all nodes of the tree, $p(\tau)$ is the proportion of instances sorted into τ, $a(\tau)$ is the attribute tested at τ, and $\Delta i(\tau)$ is the expected reduction of impurity achieved by that node. So, the attribute importance $I(A_j, T^i)$ is essentially the normalized sum of the impurity decreases achieved by splitting on attribute A_j.

Consider a query $q_{\boldsymbol{I}\rightarrow\boldsymbol{O}}$, meaning that attributes $\boldsymbol{I}$ are given. The less important the missing input attributes ($\boldsymbol{X^i} \setminus \boldsymbol{I}$) of T^i are, the more accurate T^i likely is. Therefore, we use the sum of importances of the known attributes ($\boldsymbol{I}$) to quantify the relevance of T^i to make predictions in this context.

We call this sum the **input relevance** of T^i for a set of given attributes $\boldsymbol{I}$:

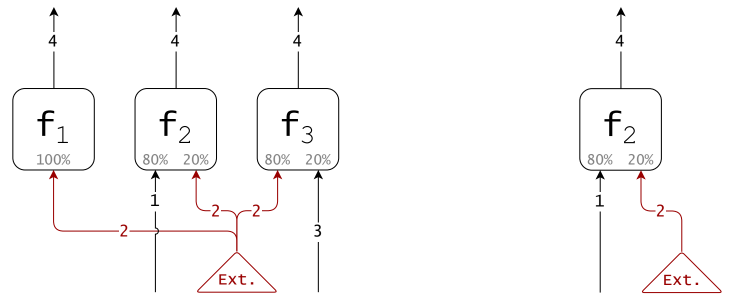
$$IR(T^i, \boldsymbol{I}) = \sum_{A_j \in \boldsymbol{X}^i \cap \boldsymbol{I}} I(A_j, T^i). \tag{4}$$

Now that we have established the notion of input relevance, we define our two single-layer strategies. We distinguish between a naive Random Forest baseline (*RF-prediction*) and *MRAI-prediction* which does exploit input relevance.

Random Forest (baseline) The most basic strategy is as follows: each T^i that predicts some attributes in $\boldsymbol{O}$, that is, $\boldsymbol{Y}_i \cap \boldsymbol{O} \neq \emptyset$, is regarded as equally relevant. $\mathcal{M}$'s prediction of an individual target attribute is obtained by aggregating the predictions of all T^i in $\mathcal{M}$ that predict that attribute. A standard aggregation (majority vote, mean, ...) is used, without taking input relevance into account. (Fig. 2a)

MRAI-prediction A second strategy, MRAI-prediction[3], does take input relevance into account. T^i is considered relevant if $\boldsymbol{Y}_i \cap \boldsymbol{O} \neq \emptyset$ and $IR(T^i, \boldsymbol{I}) \geq \theta$, for some threshold θ. That is, trees that rely too strongly on attributes whose values are missing are not included in the set of predictors. (Fig. 2b)

[3] MRAI stands for **m**ost **r**elevant **a**ttribute **i**mportance.



(a) **RF-prediction**. Selects all $T^i_{\boldsymbol{X}^i \rightarrow \boldsymbol{Y}^i}$ that predict an attribute in $\boldsymbol{O}$ regardless of missing input attributes ($\boldsymbol{X}^i \setminus \boldsymbol{I}$).

(b) **MRAI-prediction**. Selects only the most relevant trees, based on their *input relevance* (Eq. 4), which takes into account *attribute importance* (Eq. 3).

Fig. 2: RF-prediction (baseline) and MRAI-prediction build an ad hoc ensemble of relevant trees. A naive fallback procedure (depicted as red triangle) takes care of missing inputs if necessary. Attribute importances are indicated in gray.

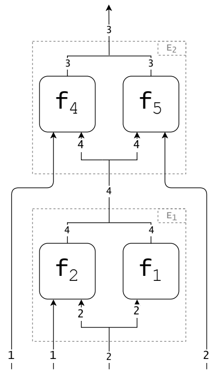
MRAI-prediction can be understood as a refinement of the MA-prediction strategy introduced in Van Wolputte et al.[20] This MA-prediction essentially is MRAI-prediction minus attribute importance: each attribute is deemed equally important. Preliminary experiments revealed MRAI-prediction to consistently outperform MA-prediction. Therefore, the old MA-prediction strategy is omitted from subsequent experiments in favor of its superior cousin: the novel MRAI-prediction strategy.

3.2 Chaining and Multi-Layer Prediction Strategies

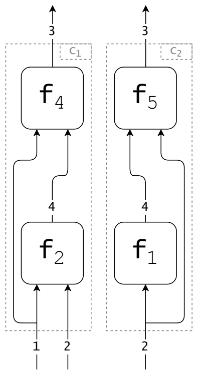
Assume $\boldsymbol{Y}^i \cap \boldsymbol{O} \neq \emptyset$ for some tree T^i, but $IR(T^i, \boldsymbol{I}) < \theta$. Now, if more input attributes of T^i had been known, $IR(T^i, \boldsymbol{I})$ might have met the threshold θ. In fact, this can readily be achieved. After all, a `MERCS` model is multi-directional and thus contains at least one predictor for each attribute. Concretely, we can make T^i meet this threshold θ by predicting some of its missing input attributes ($\boldsymbol{X}^i \setminus \boldsymbol{I}$), using other trees T^j with $\boldsymbol{Y}^j \cap (\boldsymbol{X}^i \setminus \boldsymbol{I})$. Afterwards, we treat these predictions of T^j as known values. To decide which T^j to use, we can use exactly the same criterion as we did before: T^j is a suitable predictor if it predicts some of the missing input attributes of T^i and if $IR(T^j, \boldsymbol{I}) > \theta$. If some of T^j's input attributes are missing, the same procedure can be repeated.

This principle is known as *chaining*[14]. In our multi-layer algorithms, chaining is exploited in two different manners: bottom-up and top-down. Consequently, we distinguish between *BU-prediction* and *TD-prediction* respectively.

BU-prediction The BU-prediction strategy is a recursive application of MRAI-prediction. It works in a bottom-up fashion: we keep a set of known attributes



(a) **BU-prediction**. In each layer, an ensemble of relevant T^i is selected according to $IR(T^i, \boldsymbol{K})$. This makes more attributes available to the next layer, which means other T^i become relevant. Eventually, we obtain a chain of ensembles to predict $\boldsymbol{O}$.



(b) **TD-prediction**. Builds a chain of relevant T^i, the probability of including a model (Eq. 6) is proportional to $IR(T^i, \boldsymbol{I})$. Repeated application of this idea yields an ensemble of chains to predict $\boldsymbol{O}$.

Fig. 3: BU-prediction and TD-prediction use *chaining*.

$\boldsymbol{K}$, whose initial value is $\boldsymbol{I}$. For each T^i with $IR(T^i, \boldsymbol{K}) > \theta$, add the variables in $\boldsymbol{Y}^i$ to $\boldsymbol{K}$. That concludes one step. Repeat this until $\boldsymbol{O} \subseteq \boldsymbol{K}$. (Fig. 3a)

If, at a given step, the threshold θ is set too high, there may not be any trees with a sufficiently high input relevance. This means no progress is made and the procedure ends with $\boldsymbol{O} \not\subseteq \boldsymbol{K}$. In order to make progress, simply repeat that step with a lower value for θ, and proceed.

TD-prediction TD-prediction exploits the MRAI-principle in a top-down manner. Rather than extending a set of known attributes $\boldsymbol{K}$ until it covers $\boldsymbol{O}$ (as BD-prediction does), TD-prediction starts from the output attributes $\boldsymbol{O}$ instead.

First, we define a set of unknown attributes of interest $\boldsymbol{U}$, whose initial value is $\boldsymbol{O}$. Then, take the subset of trees which predict at least one attribute in $\boldsymbol{U}$,

$$C = \{T^i_{\boldsymbol{X}^i \rightarrow \boldsymbol{Y}^i} | \boldsymbol{Y}^i \cap \boldsymbol{U} \neq \emptyset\} \tag{5}$$

and continue by defining a probability distribution,

$$p(i) = \frac{IR(T^i, \boldsymbol{I})}{\sum_{T^i \in C} IR(T^i, \boldsymbol{I})}, \tag{6}$$

which assigns to each $T^i \in C$ a probability proportional to $IR(T^i, \boldsymbol{I})$. Using p, we can randomly choose a tree T^j such that more suitable trees are more likely to get chosen. This concludes one step.

For the next step, first adjust $\boldsymbol{U}$ accordingly, i.e. $\boldsymbol{U} = \boldsymbol{X}^j \setminus \boldsymbol{I}$. That means that $\boldsymbol{U}$ now contains the missing input attributes of the tree T^j selected in the previous step. Repeat the procedure from the definition of C on. (Fig. 3b)

Essentially, this procedure does a random walk through the random forest. Starting from $\boldsymbol{U} = \boldsymbol{O}$, it randomly chooses a tree T^i that predicts (part of) $\boldsymbol{O}$; more suitable trees are more likely to get chosen. If that tree has missing inputs, choose a tree that predicts some of those inputs. Keep repeating this up to some maximum depth or until there are no missing inputs left. As the TD-procedure is randomized, it can be repeated multiple times. Each time, a different path through the random forest is followed.

It is instructive to compare BU-prediction and TD-prediction by viewing them as searches through a graph. Let G be a bipartite graph with nodes being attributes and trees; trees have incoming edges from their input attributes and outgoing edges to their output attributes. BU constructs a subgraph of G that connects $\boldsymbol{I}$ to $\boldsymbol{O}$ using only tree nodes whose IR is above some threshold. TD is a randomized search for paths that end in $\boldsymbol{O}$ but may begin at any point, and tends to contain tree nodes with high IR. Neither BU nor TD entirely avoid the use of external procedures for missing value imputation. BU only avoids them when $\theta = 1$ leads to a solution. TD only avoids them on paths where each tree happens to predict all the missing inputs of the tree that comes behind it in the path.

4 Comparison of Prediction Strategies in MERCS

This experiment is set up to answer our first research question **Q1**: how to make the MERCS framework robust to missing values at prediction time? Here, we compare all the prediction strategies for MERCS we introduced in section 3, across different degrees of missing data. This allows us to see which prediction strategies are actually robust to missing data at prediction time. As an external baseline, we also add a PGM.

Datasets Our experiments use a standard benchmark suite[4] of 28 real-world datasets. Our focus on multi-directionality requires adequate datasets in the sense that it should be possible to think of several potentially interesting prediction tasks. Prior appearance in studies on structure learning[9], make this benchmark a natural fit for our current setting. Lastly, PGMs are less flexible with regard to data-types (cf. section 2), but in this benchmark, that will not be an issue, since all variables in these datasets are binary.

Methodology For each dataset, we train both a MERCS model and a PGM. For PGMs, we rely on the SMILE-engine[5] for structure learning and inference.

[4] Cf. `github.com/UCLA-StarAI/Density-Estimation-Datasets` and [1, 11, 19]

[5] The SMILE-engine is a part of the powerful and widely used BayesFusion system, cf. `bayesfusion.com/publications`.

For structure learning, we use the *greedy thick thinning* algorithm. For inference, we use the approximate *EPIS-sampling* algorithm. In `MERCS`, trees are randomly assigned 60% of attributes as inputs, 2 output attributes and are limited to a maximum depth of 16. We ensure each attribute occurs 4 times as an output attribute, meaning we have $2m$ trees in total, m being the amount of attributes in the dataset. The Random Forest baseline, essentially `MERCS` with a trivial prediction strategy, uses the exact same trees to ensure consistency.

To see the effect of missing input attributes on performance, we consider an extensive set of queries $q_{\boldsymbol{I} \rightarrow \boldsymbol{O}}$. For each dataset, we randomly pick 10 output attributes. For each of those, we build a series of 10 increasingly difficult queries; the first one has no missing input attributes, and in each consecutive query of the series, we omit (at random) an additional 10% of its input attributes. In the end, this amounts to 2800 distinct prediction tasks.

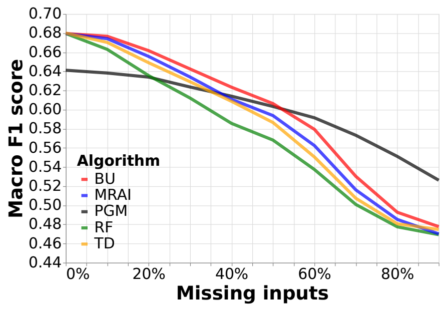
Evaluation Criteria For predictive performance, we look at F1-score[6, 15] on a test set. The random selection of output attributes ($\boldsymbol{O}$) in our queries $q_{\boldsymbol{I} \rightarrow \boldsymbol{O}}$ means we cannot exclude very unbalanced targets. For these, high predictive accuracy is meaningless. F1-score is not susceptible to this kind of effect[16] and therefore more suitable for our needs. For runtime, we report prediction times, relative to the PGM-baseline.

Results BU-prediction is the most robust prediction strategy in `MERCS` (Fig. 4a). When less than half of the inputs is missing, PGMs exhibit lower predictive performance than `MERCS`. MRAI-prediction outperforms the naive Random Forest baseline (RF), indicating that *input relevance* (Eq. 4) works, and consequently that *attribute importance* (Eq. 3) is a useful heuristic. In its turn, BU-prediction improves upon MRAI, showing that *chaining* indeed improves robustness. However, TD-prediction does not, and additionally is much slower than BU-prediction (Fig. 4b), which indicates that bottom-up chaining is recommended.

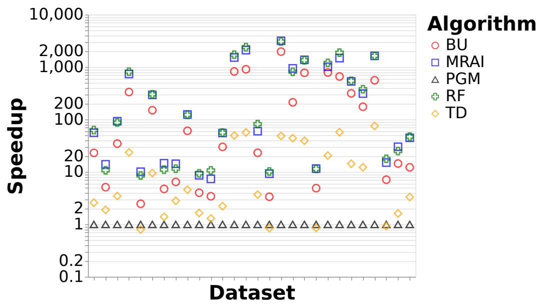
In terms of runtime, note that roughly speaking, all prediction strategies in `MERCS` do offer order(s) of magnitude of speedup over PGMs (Fig. 4b) across the board. PGMs rely on probabilistic inference. This makes them very robust to missing values, but also comes at a significant overhead in prediction time.

5 `MERCS` vs. `MissForest`

This experiment is set up to answer the second research question **Q2**: how does `MERCS` compare to `MissForest`? Concretely, we try to evaluate whether `MERCS` can succeed where `MissForest` struggles, namely when missing values are only introduced at prediction time. We expect `MissForest` to experience a bottleneck, since it its iterative nature requires retraining for each new query-instance q_k. The question is whether `MERCS`, can offer similar predictive performance, without the need to retrain.



(a) F1-score (avg. over datasets and queries) vs. % of missing inputs. In MERCS, BU-prediction is most robust to missing inputs and outperforms PGMs up until 50% of inputs are missing.



(b) At prediction time, MERCS offers order(s) of magnitude speedup over PGMs (which rely on probabilistic inference). In MERCS, TD-prediction is the most costly.

Fig. 4: Avg. F1-scores, and relative prediction times of all prediction strategies. When at least half the inputs are given and prediction time matters, MERCS (and in particular, BU-prediction) works.

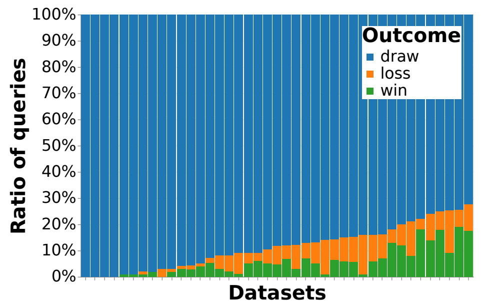
Datasets This experiment uses a curated benchmark suite of classification problems, known as OpenML-CC18[2]. Here, we are interested not so much in the multi-directional aspect (as in section 4), but really on our core problem: handling missing values at prediction time. Therefore, this benchmark, with well-defined (categorical) target variables, is ideally suited.

Methodology Each dataset is divided into a train set and a test set. This division is already defined in the OpenML-CC18 itself, which enhances reproducibility. Now, since we are interested in how MERCS and MissForest handle missing values *at prediction time*, we use the test set to generate query-instances x_q. This happens as follows: from the test set, take an instance x. From this instance x, omit a fixed number input variables at random (i.e. make those missing). This defines a query-instance x_q. This is repeated 100 times, yielding 100 query-instances per dataset. The pattern of which attributes are missing can vary from instance to instance.

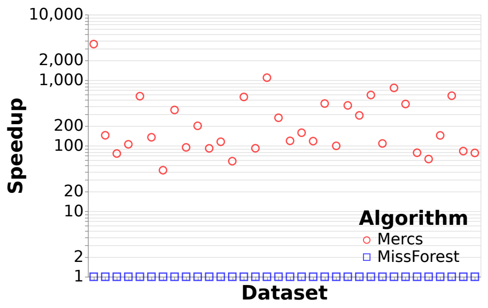
Both for MERCS and MissForest the goal is, given a query-instance x_q, predict the value of its output attribute $\boldsymbol{Y}$.

In the case of MissForest, we add the query-instance x_q to the entire training set, and run the MissForest algorithm on all these instances. Since the target variable of the query-instance is unknown and therefore missing, it will also be imputed. For each query-instance, this loop has to be repeated in full.

In the case of MERCS, we can clearly distinguish between a training phase and a testing phase. First, we train a MERCS model $\mathcal{M}$ on the training set. Second, given a query-instance x_q, we can ask $\mathcal{M}$ to predict the target variable $\boldsymbol{Y}$, from the non-missing input variables. We can repeat this for all 100 query-instances,



(a) `MERCS` vs. `MissForest`, win-draw-loss comparison. The overwhelming majority of draws shows that `MERCS` and `MissForest` perform at par under the conditions examined here.



(b) `MERCS` consistently offers multiple orders of magnitude speedup over `MissForest`. The iterative nature of `MissForest` means that it needs to retrain, whereas the `MERCS` model only trains once.

Fig. 5: `MERCS` vs. `MissForest`.

without the need of retraining. We use the BU-prediction strategy, since it is the most robust to missing values (cf. section 3 and Fig. 4).

Evaluation Criteria Our primary interest here is to determine of either `MERCS` or `MissForest` is clearly superior to its competitor, and if so, at which cost. Since we are dealing with classification problems, a prediction for a single query-instance is either correct or incorrect. Thus, if approach A is correct and approach B is incorrect, that constitutes a *win* for approach A on that query-instance (and vice-versa a *loss* for approach B). If both approaches are (in)correct, that constitutes a *draw*. In terms of cost, we simply measure prediction times, averaged across queries.

Results In terms of predictive performance, it is clear from the amount of draws (Fig. 5a) that in the overwhelming majority of queries and datasets, it really does not matter whether you choose `MERCS` or `MissForest`. Both predict the same value in the large majority of cases, and when they differ, each is about equally likely to win, taken over all datasets.

In terms of runtime, although the robust BU-prediction strategy in `MERCS` is slower than the naive Random Forest baseline (Fig. 4b), it still entails a significant speedup (up to 3 orders of magnitude in some cases) over `MissForest` (Fig. 5b), across all datasets.

6 Conclusions

To conclude, let us simply answer our original two research questions, **Q1** and **Q2**.

6.1 Q1: How to make `MERCS` robust to missing values at prediction time?

In section 3, we extend the original `MERCS` framework with three new prediction strategies: MRAI, BU and TD. All of these rely on *attribute importance* (Eq. 3) to select the most relevant trees for the task at hand. Additionally, BU and TD make use of *chaining*: the outputs of one decision tree can serve as inputs for another one. In section 4, these proposed prediction strategies are compared experimentally.

The answer to **Q1** (and consequently, our contribution to the original `MERCS`-framework[20]) is that both attribute importance and chaining can improve robustness, and BU-prediction is found to be the best strategy (Fig. 4a) for `MERCS`. Additionally, the computational costs associated with the BU-prediction strategy are acceptable. (Fig. 4b)

6.2 Q2: How does `MERCS` compare against `MissForest`?

In section 2, we made the argument that `MERCS` would make an interesting replacement for `MissForest` when dealing with missing values at prediction time. The reason being that an iterative approach such as `MissForest` is really geared towards dealing with the missing value problem at training time, since the iterative procedure requires multiple training rounds. Of course, this first required `MERCS` itself to be somewhat robust against missing values, which was dealt with in research question **Q1**. What remains is to see whether `MERCS` can actually *improve* upon `MissForest`.

The answer to **Q2** is that, when query-instances x_q come in one by one, `MERCS` improves upon `MissForest`. In terms of predictive performance, both approaches yield similar results (Fig. 5a). But, in terms of runtime, `MERCS` is orders of magnitude faster than `MissForest` (Fig. 5b). The iterative nature of `MissForest` makes it particularly ill-suited to tackle missing data at prediction time: it needs to retrain for each query-instance. `MERCS`, which never needs to retrain, is thus orders of magnitude faster when these query-instances come in one by one. (Fig. 5b)

7 Acknowledgments

This research received funding from the Flemish Government under the '*Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen*" programme. Elia Van Wolputte is supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreement No. 694980 "`SYNTH`: Synthesising Inductive Data Models". Lastly, we wish to acknowledge Simon Slangen, who in his Msc. thesis was the first to investigate advanced prediction strategies in `MERCS`.

References

1. Bekker, J., Davis, J., Choi, A., Darwiche, A., Van den Broeck, G.: Tractable Learning for Complex Probability Queries. In: Advances in Neural Information Processing Systems 28 (NIPS) (Dec 2015)
2. Bischl, B., Casalicchio, G., Feurer, M., Hutter, F., Lang, M., Mantovani, R.G., van Rijn, J.N., Vanschoren, J.: Openml benchmarking suites. arXiv preprint arXiv:1708.03731 (2017)
3. Breiman, L.: Bagging predictors. Machine learning **24**(2), 123–140 (1996)
4. Breiman, L.: Random forests. Machine learning **45**(1), 5–32 (2001)
5. Buuren, S.v., Groothuis-Oudshoorn, K.: mice: Multivariate imputation by chained equations in r. Journal of statistical software pp. 1–68 (2010)
6. Chinchor, N.: MUC-4 evaluation metrics. In: Proceedings of the 4th conference on Message understanding. pp. 22–29 (1992)
7. Friedman, J.H.: Greedy function approximation: a gradient boosting machine. Annals of statistics pp. 1189–1232 (2001)
8. Koller, D., Friedman, N., Getoor, L., Taskar, B.: Graphical models in a nutshell. Introduction to statistical relational learning **43** (2007)
9. Liang, Y., Bekker, J., Van den Broeck, G.: Learning the Structure of Probabilistic Sentential Decision Diagrams. In: Proceedings of the 33rd Conference on Uncertainty in Artificial Intelligence (UAI) (Aug 2017)
10. Louppe, G., Wehenkel, L., Sutera, A., Geurts, P.: Understanding variable importances in forests of randomized trees. In: Advances in neural information processing systems. pp. 431–439 (2013)
11. Lowd, D., Davis, J.: Learning Markov network structure with decision trees. In: 2010 IEEE International Conference on Data Mining. pp. 334–343 (2010)
12. MacKay, D.J.: Information theory, inference and learning algorithms. Cambridge university press (2003)
13. Neapolitan, R.E.: Learning bayesian networks, vol. 38. Pearson Prentice Hall Upper Saddle River, NJ (2004)
14. Read, J., Pfahringer, B., Holmes, G., Frank, E.: Classifier chains for multi-label classification. Machine Learning **85**(3), 333–359 (Dec 2011)
15. Rijsbergen, C.J.V.: Information Retrieval. Butterworth-Heinemann, Newton, MA, USA, 2nd edn. (1979)
16. Sasaki, Y.: The truth of the F-measure. Teach Tutor mater p. 5 (2007)
17. Stekhoven, D.J., Bühlmann, P.: Missforest—non-parametric missing value imputation for mixed-type data. Bioinformatics **28**(1), 112–118 (2012)
18. Van Buuren, S.: Flexible imputation of missing data. CRC press (2018)
19. Van Haaren, J., Davis, J.: Markov network structure learning: A randomized feature generation approach. In: Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence. AAAI Publications (2012)
20. Van Wolputte, E., Korneva, E., Blockeel, H.: MERCS: Multi-Directional Ensembles of Regression and Classification Trees. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence. pp. 4276 – 4283. AAAI Publications, New Orleans, Louisiana, USA (2018)
21. Waljee, A.K., Mukherjee, A., Singal, A.G., Zhang, Y., Warren, J., Balis, U., Marrero, J., Zhu, J., Higgins, P.D.: Comparison of imputation methods for missing laboratory data in medicine. BMJ open **3**(8), e002847 (2013)