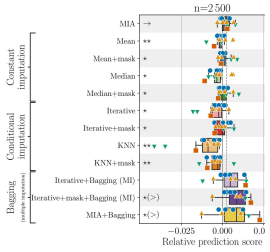
Jan 11, 2022 Version 2

# Benchmarking missing-values approaches for predictive models on health databases V.2




DOI
**dx.doi.org/10.17504/protocols.io.b3nfqmbn**

Alexandre Perez-Lebel[1], Gaël Varoquaux[1], Marine Le Morvan[1], Julie Josse[2], Jean-Baptiste Poline[3]

[1]Inria, Palaiseau, France; [2]Inria, Montpellier, France; [3]McGill University, Montreal, Canada

Missing values analysis

Alexandre Perez-Lebel

OPEN ACCESS

**DOI: dx.doi.org/10.17504/protocols.io.b3nfqmbn**

**Protocol Citation:** Alexandre Perez-Lebel, Gaël Varoquaux, Marine Le Morvan, Julie Josse, Jean-Baptiste Poline 2022. Benchmarking missing-values approaches for predictive models on health databases. **protocols.io** **https://dx.doi.org/10.17504/protocols.io.b3nfqmbn**Version created by **Alexandre Perez-Lebel**



**Protocol status:** Working
**We use this protocol and it's working**

**Created:** January 10, 2022

**Last Modified:** January 11, 2022

**Protocol Integer ID:** 56743

**Keywords:** Missing Values, Machine Learning, Supervised Learning, Benchmark, Imputation, Multiple Imputation, Bagging,

## Abstract

BACKGROUND
As databases grow larger, it becomes harder to fully control their collection, and they frequently come with missing values: incomplete observations. These large databases are well suited to train machine-learning models, for instance for forecasting or to extract biomarkers in biomedical settings. Such predictive approaches can use discriminative --rather than generative-- modeling, and thus open the door to new missing-values strategies. Yet existing empirical evaluations of strategies to handle missing values have focused on inferential statistics.

RESULTS
Here we conduct a systematic benchmark of missing-values strategies in predictive models with a focus on large health databases: four electronic health record datasets, a population brain imaging one, a health survey and two intensive care ones. Using gradient-boosted trees, we compare native support for missing values with simple and state-of-the-art imputation prior to learning. We investigate prediction accuracy and computational time. For prediction after imputation, we find that adding an indicator to express which values have been imputed is important, suggesting that the data are missing not at random. Elaborate missing values imputation can improve prediction compared to simple strategies but requires longer computational time on large data. Learning trees that model missing values --with missing incorporated attribute-- leads to robust, fast, and well-performing predictive modeling.

CONCLUSIONS
Native support for missing values in supervised machine learning predicts better than state-of-the-art imputation with much less computational cost. When using imputation, it is important to add indicator columns expressing which values have been imputed.

## Guidelines

This protocol details the experiments run in the GigaScience article *Benchmarking missing-values approaches for predictive models on health databases*, Perez-Lebel et al. 2022. The code used for running the experiments and plotting the results is available on GitHub: https://github.com/aperezlebel/benchmark_mv_approaches.

Accessing the databases can be time consuming. We published our detailed results in CSV files to allow further analysis of our results without needing to access the data: https://github.com/aperezlebel/benchmark_mv_approaches/blob/2ed30c0ffffa93f0398731b11b9202523c4da96f/scores/merged_scores.csv.

## Materials

Computing cluster

## Safety warnings

No safety warnings.

## Introduction

1 This protocol details the experiments run in the GigaScience article *Benchmarking missing-values approaches for predictive models on health databases*, Perez-Lebel et al. 2022. The code used for running the experiments and plotting the results is available on GitHub:

Software

**Benchmarking missing-values approaches for predictive models...** NAME

Alexandre Perez-Lebel DEVELOPER

https://github.com/alexprz/article-benchmark_mv_approaches SOURCE LINK

And can be installed through the following steps:

Command

**Download and install the code reproducing the experiments.**

```
git clone https://github.com/aperezlebel/benchmark_mv_approaches.git
cd benchmark_mv_approaches
conda install --file requirements.txt
```

## Data

2 We benchmarked 12 supervised predictive methods on 13 prediction tasks taken from 4 health databases.

Each one of the 4 databases needs to be downloaded separately from their respective source project. Access to Traumabase, UK BioBank and MIMIC-III, requires an application. NHIS is freely available. Once downloaded, data path of each database can be updated in the **TB.py**, **UKBB.py**, **MIMIC.py** and **NHIS.py** files which are in the *database/* folder of the project.

2.1

Dataset

NAME

Traumabase

LINK

http://www.traumabase.eu/en_US

The Traumabase Group (TB) is a collaboration studying major trauma. The database gathers information from 20 French trauma centers on more than 20 000 trauma cases from admission until discharge from critical care. Data collection started in 2010 and is still ongoing in 2020. We used records spanning from 2010 to 2019. We defined 5 prediction tasks on this database, 4 classifications and 1 regression.

Data can be obtained by **contacting the team on the Traumabase website**.

2.2

Dataset

NAME

UKBB

LINK

https://www.ukbiobank.ac.uk/

UK Biobank (UKBB) is a major prospective epidemiology cohort with biomedical measurements. It provides health information on more than 500 000 United-Kingdom participants aged between 40 to 69 years from 2006 to 2010. We defined 5 tasks on this database, 4 classifications and 1 regression.

The data are available upon application **as detailed on the UK BioBank website**.

2.3

Dataset

MIMIC-III (v1.4) NAME

https://mimic.physionet.org/ LINK

The Medical Information Mart for Intensive Care (MIMIC) database is an Intensive Care Unit (ICU) dataset developed by the MIT Lab for Computational Physiology. It comprises deidentified health data associated with about 60 000 ICU admissions recorded at the Beth Israel Deaconess Medical Center of Boston, United States, between 2001 and 2012. It includes demographics, vital signs, laboratory tests, medications, and more. We defined 2 classification tasks on this database.

The data can be accessed via **an application described on the MIMIC website.** Note that, as of the time of writing, the completion of an online MIT course is required for the application. We used the 1.4 version of the data in the project.

2.4

Dataset

NHIS (2017) NAME

https://www.cdc.gov/nchs/nhis/nhis_2017_data_release.htm LINK

The National Health Interview Survey (NHIS) is a major data collection program of the National Center for Health Statistics (NCHS), part of the Centers for Disease Control and Prevention (CDC) in the United States. It aims to monitor the health of the population. Since 1957, it collects data from United-States population. We used the 2017 edition, summing up to approximately 35 000 households containing about 87 500 persons. We defined 1 regression task on this database.

It is **freely-accessible on the NHIS website.**

## Prediction tasks

3 From these databases, we defined 13 prediction tasks. That is, a set of input features and an outcome to predict. All features of each task belong to the same database.

Available tasks can be obtained with:

Command

**List the names of all the available tasks.**

```
python main.py info available -t
```

Names of the available tasks are:

Expected result

TB/death_pvals
TB/platelet_pvals
TB/hemo
TB/hemo_pvals
TB/septic_pvals
UKBB/breast_25
UKBB/breast_pvals
UKBB/skin_pvals
UKBB/parkinson_pvals
UKBB/fluid_pvals
MIMIC/septic_pvals
MIMIC/hemo_pvals
NHIS/income_pvals

## Predictive methods

4 36 predictive methods are available. The list of their IDs and names can be obtained running:

Command

**List the IDs and names of all the available methods.**

```
python main.py info available -m
```

IDs and names of the available methods are:

**Expected result**

0: Classification
1: Classification_Logit
2: Regression
3: Regression_Ridge
4: Classification_imputed_Mean
5: Classification_Logit_imputed_Mean
6: Regression_imputed_Mean
7: Regression_Ridge_imputed_Mean
8: Classification_imputed_Mean+mask
9: Classification_Logit_imputed_Mean+mask
10: Regression_imputed_Mean+mask
11: Regression_Ridge_imputed_Mean+mask
12: Classification_imputed_Med
13: Classification_Logit_imputed_Med
14: Regression_imputed_Med
15: Regression_Ridge_imputed_Med
16: Classification_imputed_Med+mask
17: Classification_Logit_imputed_Med+mask
18: Regression_imputed_Med+mask
19: Regression_Ridge_imputed_Med+mask
20: Classification_imputed_Iterative
21: Classification_Logit_imputed_Iterative
22: Regression_imputed_Iterative
23: Regression_Ridge_imputed_Iterative
24: Classification_imputed_Iterative+mask
25: Classification_Logit_imputed_Iterative+mask
26: Regression_imputed_Iterative+mask
27: Regression_Ridge_imputed_Iterative+mask
28: Classification_imputed_KNN
29: Classification_Logit_imputed_KNN
30: Regression_imputed_KNN
31: Regression_Ridge_imputed_KNN
32: Classification_imputed_KNN+mask
33: Classification_Logit_imputed_KNN+mask
34: Regression_imputed_KNN+mask
35: Regression_Ridge_imputed_KNN+mask

*Classification* and *Regression* code respectively for HistGradientBoostingClassifier and HistGradientBoostingRegressor from scikit-learn. *Classification_Logit* and *Regression_Ridge* code respectively for linear models Logit and Ridge used in the supplementary experiment. To each of these 4 base codes can be appended the name of an imputer (eg *_imuted_Mean, _Imputed_Med, ...)* with or without the mask (eg *_imuted_Mean, _Imputed_Mean+mask, ...).* Whether to use Bagging can be specified later as explained in the *Prediction* section of this protocol.

## Feature selection

5 11 tasks have their features automatically selected with a simple ANOVA-based univariate test of the link of each feature to the outcome (task name ends with "_pvals" in the code and "_screening" in the article).
The 2 remaining tasks have their feature manually defined following the choices of experts in prior studies.

5.1 **ANOVA-based feature selection**

Categorical features are first one-hot encoded. Then, the ANOVA-based univariate test is performed on one third of the samples which are then discarded. We kept the 100 encoded features having the smallest 100 p-values. Once the features are selected, the cross-validated prediction is performed on the remaining two thirds of the samples.
For these tasks, there are 5 trials during which the samples on which the selection test is performed are redrawn, and the prediction each time fitted on the new remaining samples and the new selected features.

We used *f_classif* and *f_regression* from the *feature_selection* module of scikit-learn.

For each of these tasks, p-values of the test can be computed for each trial by running:

Command

**Compute p-values of ANOVA-based test to select features**

```
python main.py select {task_name} --T {T}
```

Be careful to replace placeholders {task_name} and {T} by the name of the task and the trial ID (0 to 4) respectively.
Example:

Command

**Compute p-values of ANOVA-based test to select features of the task TB/death_pvals on the first trial.**

```
python main.py select TB/death_pvals --T 0
```

Safety information

Note that these commands will fail without the data and without the types of the features.

## 5.2 Manual selection following experts

Features for the hemorrhagic shock prediction (task named TB/hemo) in the Traumabase database are defined following Jiang et al.:

CITATION

Wei Jiang, Julie Josse, Marc Lavielle, TraumaBase Group (2020). Logistic Regression with Missing Covariates -- Parameter Estimation, Model Selection and Prediction within a Joint-Modeling Framework. Computational Statistics and Data Analysis.

LINK

https://doi.org/10.1016/j.csda.2019.106907

Features for the breast cancer prediction (task named UKBB/breast_25) are defined following Läll et al.:

CITATION

Kristi Läll, Maarja Lepamets, Marili Palover, Tõnu Esko, Andres Metspalu, Neeme Tõnisson, Peeter Padrik, Reedik Mägi, Krista Fischer (2019). Polygenic prediction of breast cancer: comparison of genetic predictors and implications for risk stratification. BMC Cancer.
LINK
https://doi.org/10.1186/s12885-019-5783-1

There is only 1 trial for these tasks.

## Prediction

3,095w 1d 16h

6 **Scale**

To study the influence of the scale on the results, we decided to work on 4 sizes of the training set: 2 500, 10 000, 25 000 and 100 000. For each one of these sizes are run the following operations.

**Nested cross-validations**

Two nested cross-validations are used. The outer one yields 5 training and test sets. The training set has 2 500, 10 000, 25 000 or 100 000 samples depending on the scale. The test set is composed of all the remaining samples. Note that the size of the test set is considerably larger with a train set of 2 500 samples than with 100 000. On each training set, we perform a cross-validated hyper-parameter search –the inner cross-validation– and select the best hyper-parameters. We evaluate the best model on the respective test set. We assess the quality of the prediction with a coefficient of determination for regressions and the area under the ROC curve for classification. We average the scores obtained on the 5 test sets of the outer cross-validation to give the final score.

The test set size is at least 10% the size of the training set. If a prediction task has not enough samples once the feature selection is performed (eg 110 000 samples for the 100 000 scale), it is skipped for the corresponding scale. As a result the biggest scale has fewer available tasks than the smallest one (resp. 4 against 13).

To draw the 5 folds, we used *StratifiedShuffleSplit* (resp. *ShuffleSplit*) from scikit-learn for classifications (resp. regressions). We used *GridSearchCV* from scikit-learn to perform the cross-validated hyper-parameters tuning.

Evaluating a method on a prediction task is done by running:

Command

**Benchmark a method on a prediction task**

```
python main.py predict {task_name} {method_id} --T {T}
```

Be careful to replace placeholders {task_name}, {method_id} and {T} by the name of the task, the ID or name of the method and the trial ID (0 to 4) respectively.
Example:

Command

**Benchmark method with ID 0 on the task TB/death_pvals on the trial 0.**

```
python main.py predict TB/death_pvals 0 --T 0
```

Some methods of the benchmark use bagging. To add bagging to an available method, specify the number of estimators you want in the ensemble with the *nbagging* option. For instance:

Command

**Benchmark method with ID 0 bagged with 100 base estimators, on the task TB/death_pvals on the trial 0.**

```
python main.py predict TB/death_pvals 0 --T 0 --nbagging 100
```

**Safety information**

Note that these commands will fail without the data and without the types of the features.

Results are dumped in the *results/* folder.

To run the full benchmark of the article, we needed 520 000 CPU hours.

520000:00:00 CPU hours to run the full benchmark

### 6.1 Imputation

5 imputation methods are available:

- Imputation with the mean.
- Imputation with the median.
- Iterative imputation.
- Imputation with the nearest neighbors.
- Multiple Imputation using Bagging.

For each of them, new binary features can be added to the data. This binary mask encodes whether a value was originally missing or not.

The imputer is fitted on the train set only and both the train and test sets are then imputed with the fitted imputer. Doing so avoids leaking information from the train set to the test set and then helps to avoid overfitting.

We used *SimpleImputer*, *IterativeImputer, KNNImputer, BaggingClassifier and BaggingRegressor* from scikit-learn.

## Results

7 Once the results of all the methods are obtained, they are gathered in a single CSV file using the following command:

Command

**Merge all results in a single csv file**

```
python main.py aggregate --root results/
```

This creates a *scores.csv* file in the *scores/* folder.

**The aggregated results obtained during our experiment are given in our repository.** This allows to reproduce the figures and tables and to analyze further the results without needing the original data.

## Figures and tables

8 Most of the figures and tables of our article can be easily reproduced without requiring the original data (based on saved results only). Use commands defined in the **Makefile** to easily reproduce figures and tabs. (Note that some commands will fail because they require data or raw results that are not available in the repository).

All figures and tables are saved in the *graphics/* folder of the repository.

The main figure can be reproduced with:

Command

**Reproduce the main figure of the article.**

```
python main.py figs boxplot
```

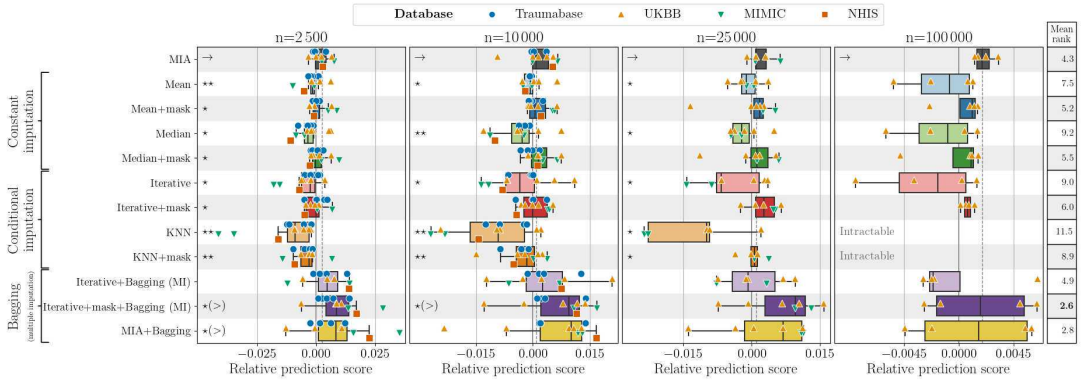
or

## Command

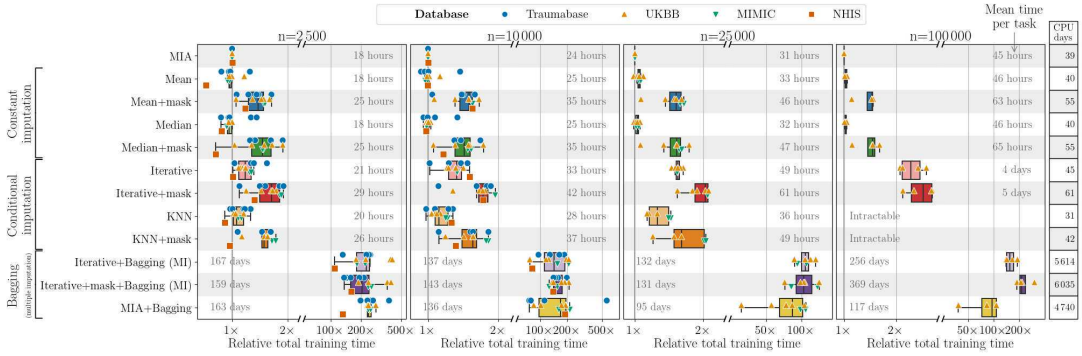
**Reproduce the main figure of the article.**

```
make boxplot
```

## Expected result




Main figure part 1: Comparison of prediction performance across the 12 methods for 13 prediction tasks spread over 4 databases, and for 4 sizes of dataset (2 500, 10 000, 25 000 and 100 000 samples).




Main figure part 2: Comparison of training times across the 12 methods for 13 prediction tasks spread over 4 databases, and for 4 sizes of dataset (2 500, 10 000, 25 000 and 100 000 samples).

## Citations

**Step 5.2**

Wei Jiang, Julie Josse, Marc Lavielle, TraumaBase Group. Logistic Regression with Missing Covariates -- Parameter Estimation, Model Selection and Prediction within a Joint-Modeling Framework
**https://doi.org/10.1016/j.csda.2019.106907**

**Step 5.2**

Kristi Läll, Maarja Lepamets, Marili Palover, Tõnu Esko, Andres Metspalu, Neeme Tõnisson, Peeter Padrik, Reedik Mägi, Krista Fischer. Polygenic prediction of breast cancer: comparison of genetic predictors and implications for risk stratification
**https://doi.org/10.1186/s12885-019-5783-1**