Using the `missForest` Package

Daniel J. Stekhoven
stekhoven@stat.math.ethz.ch

Friday, 13th of May, 2011

Contents

1 Introduction

1.1 What is this document? (And what it isn't!)

This *package vignette* is an application focussed user guide for the R package `missForest`. The functionality is explained using a couple of real data examples. Argument selection with respect to feasibility and accuracy issues are discussed and illustrated using these real data sets. Do not be alarmed by the length of this document which is mainly due to some major R output included for illustrative reasons.

This document is *not* a theoretical primer for the fundamental approach of the `missForest` algorithm. It also does not contain any simulations or comparative studies with other imputation methods. For this information we point the interested reader to Stekhoven and Bühlmann [2011].

1.2 The `missForest` algorithm

`missForest` is a nonparametric imputation method for basically any kind of data. It can cope with mixed-type of variables, nonlinear relations, complex interactions and high dimensionality ($p \gg n$). It only requires the observation (i.e. the rows of the data frame supplied to the function) to be pairwise independent. The algorithm is based on random forest (Breiman [2001]) and is dependent on its R implementation `randomForest` by Andy Liaw and Matthew Wiener. Put simple (for those who have skipped the previous paragraph): for each variable `missForest` fits a

random forest on the observed part and then predicts the missing part. The algorithm continues to repeat these two steps until a stopping criterion is met or the user specified maximum of iterations is reached. For further details see Stekhoven and Bühlmann [2011].

To understand the remainder of this user guide it is important to know that `missForest` is running iteratively, continuously updating the imputed matrix variable-wise, and is assessing its performance between iterations. This assessment is done by considering the difference(s) between the previous imputation result and the new imputation result. As soon as this difference (in case of one type of variable) or differences (in case of mixed-type of variables) increase the algorithm stops.

`missForest` provides the user with an estimate of the imputation error. This estimate is based on the out-of-bag (OOB) error estimate of random forest. Stekhoven and Bühlmann [2011] showed that this estimate produces an appropriate representation of the true imputation error.

1.3 Installation

The R package `missForest` is available from the Comprehensive R Archive Network (CRAN, `http://cran.r-project.org/`) and as such can be installed in the default way using the `install.packages` function:

```
> install.packages(missForest, dependencies = TRUE)
```

Make sure to include the `dependencies = TRUE` argument to install also the `randomForest` package unless it is already installed.

2 Missing value imputation with `missForest`

In this section we describe using the `missForest` function. We will shed light on all arguments which can or have to be supplied to the algorithm. Also, we will discuss how to make `missForest` faster or more accurate. Finally, an interpretation of the OOB imputation error estimates is given.

2.1 Description of the data used

Iris data This complete data set contains five variables of which one is categorical with three levels. It is contained in the R base and can be loaded directly by typing `data(iris)`. The data were collected by Anderson [1935].

Oesophageal cancer data This complete data set comes from a case-control study of oesophageal cancer in Ile-et-Vilaine, France. It is contained in the R base and can be loaded directly by typing `data(esoph)`. The data were collected by Breslow and Day [1980].

Musk data This data set describes the shapes of 92 molecules of which 47 are musks and 45 are non-musks. Since a molecule can have many conformations due to rotating bonds, there are $n = 476$ different conformations in the set. The classification into musk and non-musk molecules is removed. For further details see Frank and Asuncion [2010].

2.2 `missForest` in a nutshell

After you have properly installed `missForest` you can load the package in your R session:

```
> library(missForest)
```

We will load now the famous Iris data set and artificially remove 10% of the entries in the data completely at random using the `prodNA` function from the `missForest` package:

```
> data(iris)
> iris.mis <- prodNA(iris, noNA = 0.1)
> summary(iris.mis)
```

```
  Sepal.Length     Sepal.Width      Petal.Length     Petal.Width
 Min.   : 4.300   Min.   : 2.000   Min.   : 1.000   Min.   : 0.100
 1st Qu.: 5.100   1st Qu.: 2.800   1st Qu.: 1.600   1st Qu.: 0.300
 Median : 5.800   Median : 3.000   Median : 4.400   Median : 1.300
 Mean   : 5.862   Mean   : 3.068   Mean   : 3.856   Mean   : 1.221
 3rd Qu.: 6.400   3rd Qu.: 3.325   3rd Qu.: 5.100   3rd Qu.: 1.800
 Max.   : 7.900   Max.   : 4.400   Max.   : 6.900   Max.   : 2.500
 NA's   :17.000   NA's   :14.000   NA's   :17.000   NA's   :14.000
       Species
 setosa    :46
 versicolor:44
 virginica :47
 NA's      :13
```

We can see that there is an evenly distributed amount of missing values over the variables in the data set. With *completely at random* we mean that the process of deleting entries is not influenced by the data or the data generating process.

The missing data is now imputed by simply handing it over to `missForest` :

```
> iris.imp <- missForest(iris.mis)
```

```
  missForest iteration 1 in progress...done!
  missForest iteration 2 in progress...done!
  missForest iteration 3 in progress...done!
  missForest iteration 4 in progress...done!
```

Except for the iteration numbering no additional print-out is given. The results are stored in the R object `iris.imp` which is a list. We can call upon the imputed data matrix by typing `iris.imp$ximp`. *Note: A common mistake is to use* `iris.imp` *instead of* `iris.imp$ximp` *for subsequent analyses.*

Additionally, `missForest` provides an OOB imputation error estimate which can be extracted using the same `$` notation as with the imputed data matrix:

```
> iris.imp$OOBerror
```

```
     NRMSE        PFC
0.15271479 0.05109489
```

As mentioned before the Iris data set contains two types of variables, continuous and categorical. This is why the OOB imputation error supplies two values for the result of the imputation (default setting). The first value is the normalized root mean squared error (NRMSE, see Oba et al. [2003]) for the continuous part of the imputed data set, e.g., `Sepal.Length`, `Sepal.Width`, `Petal.Length` and `Petal.Width`. The second value is the proportion of falsely classified entries (PFC) in the categorical part of the imputed data set, e.g., `Species`. In both cases good performance of `missForest` leads to a value close to 0 and bad performance to a value around 1.

If you are interested in assessing the reliability of the imputation for single variables, e.g., to decide which variables to use in a subsequent data analysis, `missForest` can return the OOB errors for each variable separately instead of aggregating over the whole data matrix. This can be done using the argument `variablewise = TRUE` when calling the `missForest` function.

```
> iris.imp <- missForest(iris.mis, variablewise = TRUE)

  missForest iteration 1 in progress...done!
  missForest iteration 2 in progress...done!
  missForest iteration 3 in progress...done!

> iris.imp$OOBerror

       MSE        MSE        MSE        MSE        PFC
0.14045330 0.08964998 0.10744467 0.03020781 0.07299270
```

We can see that the output has the same length as there are variables in the data. For each variable the resulting error and the type of error measure, i.e., mean squared error (MSE) or PFC, is returned. Note that we are not using the NRMSE here.

2.3 Additional output using `verbose`

In 2.2 the print-out of `missForest` showed only which iteration is taking place at the moment. Anyhow, if you are imputing a large data set or choose to use ridiculously large `mtry` and/or `ntree` arguments (see 2.5) you might be interested in getting additional information on how `missForest` is performing.

By setting the logical `verbose` argument to `TRUE` the print-out is extended threefold:

`estimated error(s)` The OOB imputation error estimate for the continuous and categorical parts of the imputed data set. *Note: If there is only one type of variable there will be only one value with the corresponding error measure.*

`difference(s)` The difference between the previous and the new imputed continuous and categorical parts of the data set. The difference for the set of continuous variables $\mathbf{N}$ in the data set is computed by

$$\frac{\sum_{j \in \mathbf{N}} (\mathbf{X}_{new}^{imp} - \mathbf{X}_{old}^{imp})^2}{\sum_{j \in \mathbf{N}} (\mathbf{X}_{new}^{imp})^2},$$

and for the set of categorical variables the difference corresponds to the PFC.

`time` The runtime of the iteration in seconds.

If we rerun the previous imputation of the Iris data [1] setting `verbose = TRUE` we get:

```
> set.seed(81)
> iris.imp <- missForest(iris.mis, verbose = TRUE)
```

[1] Since random forest – as its name suggests – is using a random number generator (RNG) the result for two trials on the same missing data set will be different. To avoid this from happening in the given illustrative example we use the `set.seed` function before applying `missForest` on the `iris.mis` data set. This causes the RNG to be reset to the same state as before (where we invisibly called `set.seed(81)` already but did not want to trouble the concerned reader with technical details).

```
  missForest iteration 1 in progress...done!
    estimated error(s): 0.1571594 0.05109489
    difference(s): 0.007836621 0.06666667
    time: 0.108 seconds

  missForest iteration 2 in progress...done!
    estimated error(s): 0.1533133 0.06569343
    difference(s): 2.107119e-05 0
    time: 0.186 seconds

  missForest iteration 3 in progress...done!
    estimated error(s): 0.1527148 0.05109489
    difference(s): 1.311007e-05 0
    time: 0.103 seconds

  missForest iteration 4 in progress...done!
    estimated error(s): 0.1520093 0.05109489
    difference(s): 1.323415e-05 0
    time: 0.107 seconds
```

The above print-out shows that `missForest` needs four iterations to finish. If we check the final OOB imputation error estimate:

```
> iris.imp$OOBerror

     NRMSE        PFC
0.15271479 0.05109489
```

we can see that it used the result from the second last iteration, i.e., the third instead of the last one. This is because the stopping criterion was triggered and the fact that the differences increase indicate that the new imputation is probably a less accurate imputation than the previous one. However, we can also see that the *estimated* error(s) is lower for the last imputation than for the one before. But we will show later on that the true imputation error is lower for iteration 3 (the impatient reader can jump to section 2.6).

2.4 Changing the number of iterations with `maxiter`

Depending on the composition and structure of the data it is possible that `missForest` needs more than the typical four to five iterations (see 2.3) until the stopping criterion kicks in. From an optimality point of view we do want `missForest` to stop due to the stopping criterion and not due to the limit of iterations. However, if the difference between iterations is seriously shrinking towards nought and the estimated error is in a stalemate the only way to keep computation time at a reasonable level is to limit the number of iterations using the argument `maxiter`.

We show this using the `esoph` data. First, we run `missForest` on a data set where we removed 5% of the entries at random:

```
> data(esoph)
> esoph.mis <- prodNA(esoph, 0.05)
> set.seed(96)
> esoph.imp <- missForest(esoph.mis, verbose = TRUE)

  missForest iteration 1 in progress...done!
    estimated error(s): 0.6110967 0.725664
```

```
  difference(s): 0.01264188 0.03030303
  time: 0.123 seconds

missForest iteration 2 in progress...done!
  estimated error(s): 0.5762851 0.7063344
  difference(s): 0.004066078 0
  time: 0.088 seconds

missForest iteration 3 in progress...done!
  estimated error(s): 0.5753828 0.7069133
  difference(s): 0.0001354506 0.003787879
  time: 0.086 seconds

missForest iteration 4 in progress...done!
  estimated error(s): 0.5938148 0.6947065
  difference(s): 0.0002132959 0
  time: 0.091 seconds

missForest iteration 5 in progress...done!
  estimated error(s): 0.6026959 0.7179121
  difference(s): 0.000156741 0.003787879
  time: 0.085 seconds

missForest iteration 6 in progress...done!
  estimated error(s): 0.5322237 0.7070521
  difference(s): 0.0001195036 0.003787879
  time: 0.09 seconds

missForest iteration 7 in progress...done!
  estimated error(s): 0.5579731 0.7106889
  difference(s): 7.824896e-05 0.003787879
  time: 0.089 seconds

missForest iteration 8 in progress...done!
  estimated error(s): 0.5464919 0.7311144
  difference(s): 2.614136e-05 0
  time: 0.124 seconds

missForest iteration 9 in progress...done!
  estimated error(s): 0.592407 0.6937996
  difference(s): 3.632201e-05 0
  time: 0.088 seconds
```

We can see that it takes `missForest` nine iterations to come to a stop. The returned imputation result was reached in iteration 8 having estimated errors of 0.55 and 0.73 and differences of $3 \cdot 10^{-5}$ and 0. In iteration 6 the estimated errors are smaller (i.e. 0.53 and 0.70) and the differences are $1 \cdot 10^{-4}$ and $4 \cdot 10^{-3}$. So why is `missForest` not simply taking the sixth iteration and calls it a day? Because the difference in the continuous part of the data set is still reduced in each iteration up until iteration 9. This stopping strategy is on average (taking all possible data sets into account) quite good but can have its caveats at one specific data sets. In the above case of the `esoph` data we can get the result of the fourth iteration by doing the following:

```
> set.seed(96)
> esoph.imp <- missForest(esoph.mis, verbose = TRUE, maxiter = 6)

  missForest iteration 1 in progress...done!
    estimated error(s): 0.6110967 0.725664
    difference(s): 0.01264188 0.03030303
    time: 0.086 seconds

  missForest iteration 2 in progress...done!
    estimated error(s): 0.5762851 0.7063344
    difference(s): 0.004066078 0
    time: 0.089 seconds

  missForest iteration 3 in progress...done!
    estimated error(s): 0.5753828 0.7069133
    difference(s): 0.0001354506 0.003787879
    time: 0.089 seconds

  missForest iteration 4 in progress...done!
    estimated error(s): 0.5938148 0.6947065
    difference(s): 0.0002132959 0
    time: 0.126 seconds

  missForest iteration 5 in progress...done!
    estimated error(s): 0.6026959 0.7179121
    difference(s): 0.000156741 0.003787879
    time: 0.101 seconds

  missForest iteration 6 in progress...done!
    estimated error(s): 0.5322237 0.7070521
    difference(s): 0.0001195036 0.003787879
    time: 0.091 seconds
```

The returned result is now given by iteration 4. Quintessentially, there are two uses for the `maxiter` argument:

1. Controlling the run time in case of stagnating performance;
2. extract a preferred iteration result not supplied by the stopping criterion.

2.5 Speed and accuracy trade-off manipulating `mtry` and `ntree`

`missForest` grows in each iteration for each variable a random forest to impute the missing values. With a large number of variables p this can lead to computation times beyond today's perception of feasibility. There are two ways to speed up the imputation process of `missForest`:

1. Reducing the number of trees grown in each forest using the argument `ntree`;
2. reducing the number of variables randomly sampled at each split using the argument `mtry`.

It is imperative to know that reducing either of these numbers will probably result in reduced accuracy. This is why we speak of a speed and accuracy *trade-off*.

2.5.1 ntree

The effect of reducing `ntree` on the computation time is linear, e.g., halving `ntree` will half computation time for a single iteration. The default value in `missForest` is set to 100 which is fairly large. Smaller values in the tens can give appropriate results already. We show this using the Musk data:

```
> musk.mis <- prodNA(musk, 0.05)
> musk.imp <- missForest(musk.mis, verbose = TRUE, maxiter = 3)

  missForest iteration 1 in progress...done!
    estimated error(s): 0.1491825
    difference(s): 0.02383702
    time: 280.739 seconds

  missForest iteration 2 in progress...done!
    estimated error(s): 0.1367353
    difference(s): 0.0001208087
    time: 277.011 seconds

  missForest iteration 3 in progress...done!
    estimated error(s): 0.137418
    difference(s): 3.836082e-05
    time: 278.287 seconds
```

The computation time is about 14 minutes and we end up with an estimated NRMSE of 0.14. *Note: The response was removed from the Musk data, that is why there is only the estimated NRMSE and also only the difference for the continuous part of the data set.*

If we repeat the imputation using the `ntree` argument and setting it to 20 we get:

```
> musk.imp <- missForest(musk.mis, verbose = TRUE, maxiter = 3, ntree = 20)

  missForest iteration 1 in progress...done!
    estimated error(s): 0.1724939
    difference(s): 0.02383371
    time: 56.705 seconds

  missForest iteration 2 in progress...done!
    estimated error(s): 0.1576795
    difference(s): 0.0002417658
    time: 55.833 seconds

  missForest iteration 3 in progress...done!
    estimated error(s): 0.1591702
    difference(s): 0.0001966117
    time: 56.053 seconds
```

The computation time is now around 3 minutes which is approximately a fifth of the previous computation time using 100 trees (as a matter of fact, taking the floor values of the iteration times in seconds then the former imputation took *exactly* five times longer than the latter). The estimated NRMSE has increased to 0.16 – an increase of 14% compared to before. In some application this might seem as an unacceptable increase of imputation error. However, if the number of variables is large enough, e.g., in the thousands like in gene expression data, the amount of computation time saved will surpass the amount of imputation error increased.

2.5.2 mtry

The effect on computation time when changing `mtry` is not as straight forward as with `ntree`. It is however more pronounced in settings with high-dimensionality (e.g. $p \gg n$, where n is the number of observations) and complex structures. The default setting in `missForest` is $\sqrt{p}$. This choice qualifies for a quite nice trade-off between imputation error and computation time. Anyhow, certain data might demand different choices either putting a focus on better imputation error or better computation time. We leave this delicate choice to the user of these certain data sets.

2.6 Testing the appropriateness by supplying `xtrue`

Whenever imputing data with real missing values the question arises how good the imputation was. In `missForest` the estimated OOB imputation error gives a nice indication at what you have to expect. A wary user might want to make an additional assessment (or back the OOB estimate up) by performing cross-validation or – in the optimal case – testing `missForest` previously on complete data. For both cases `missForest` offers the `xtrue` argument which simply takes in the same data matrix as `xmis` but with no missing values present. The strategy for testing the performance is the same as shown in the previous examples using `prodNA`:

1. Generate a data matrix with missing values;
2. impute this artificially generated data matrix;
3. compare the complete and imputed data matrices.

The functions to use for this strategy are `prodNA`, `missForest` and `mixError`. Using again the Iris data this would look like:

```
> iris.mis <- prodNA(iris, noNA = 0.1)
> iris.imp <- missForest(iris.mis)

> iris.err <- mixError(iris.imp$ximp, iris.mis, iris)
> print(iris.err)

    NRMSE       PFC
0.1169748 0.0000000
```

Note: We want to point out once more that the user has to extract the imputed matrix from the `missForest` output using the `$` list notation. Not doing so will generate the following error:

```
> iris.err <- mixError(iris.imp, iris.mis, iris)

Error in mixError(iris.imp, iris.mis, iris) :
  Wrong input for 'xmis' - you probably forgot to point at the
  list element $ximp from the missForest output object.
```

We can simplify the above strategy by using `xtrue`. If combined with `verbose = TRUE` the user even gets additional information on the performance of `missForest` between iterations:

```
> iris.imp <- missForest(iris.mis, xtrue = iris, verbose = TRUE)
```

```
  missForest iteration 1 in progress...done!
    error(s): 0.1218268 0
    estimated error(s): 0.1571594 0.05109489
    difference(s): 0.007836621 0.06666667
    time: 0.156 seconds

  missForest iteration 2 in progress...done!
    error(s): 0.1212555 0
    estimated error(s): 0.1533133 0.06569343
    difference(s): 2.107119e-05 0
    time: 0.149 seconds

  missForest iteration 3 in progress...done!
    error(s): 0.1169748 0
    estimated error(s): 0.1527148 0.05109489
    difference(s): 1.311007e-05 0
    time: 0.154 seconds

  missForest iteration 4 in progress...done!
    error(s): 0.1213011 0
    estimated error(s): 0.1520093 0.05109489
    difference(s): 1.323415e-05 0
    time: 0.145 seconds
```

Supplying `xtrue` adds the line `error(s)` to the `missForest` output. We can observe that the true imputation error really is lower for the second last iteration as mentioned in section 2.2. Additionally, the output object (in the above example `iris.imp`) contains now a list element `error` which can be called directly:

```
> iris.imp$error

    NRMSE       PFC
0.1169748 0.0000000
```

3 Concluding remarks

Imputation using `missForest` can be done very easily. The OOB imputation error estimate facilitates the interpretation of such imputation results. However, it should always be kept in mind that imputing data with missing values does not increase the information contained within this data. It is only a way to have completeness for further data analysis. Many methods of data analysis require complete observations. In such complete case analyses observations missing only a single entry will be completely removed from the data and therefore the information content is reduced. Imputing the data beforehand prevents this reduction. For further details on the effect of imputation on the subsequent data analysis we suggest the books of Schafer [1997] and Little and Rubin [1987].

References

E. Anderson. The irises of the gaspe peninsula. *Bulletin of the American Iris Society*, 59:2–5, 1935.

L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. ISSN 0885-6125.

N. E. Breslow and N. E. Day. Statistical methods in cancer research. 1: The analysis of case-control studies. *IARC Lyon / Oxford University Press*, 1980.

A. Frank and A. Asuncion. UCI machine learning repository, 2010. URL `http://archive.ics.uci.edu/ml`.

R.J.A. Little and D.B. Rubin. *Statistical Analysis with Missing Data*. Wiley New York, 1987. ISBN 0-471-80254-9.

S. Oba, M. Sato, I. Takemasa, M. Monden, K. Matsubara, and S. Ishii. A Bayesian missing value estimation method for gene expression profile data. *Bioinformatics*, 19(16):2088–2096, 2003. ISSN 1367-4803.

J.L. Schafer. *Analysis of Incomplete Multivariate Data*. Chapman & Hall, 1997. ISBN 0-412-04061-1.

D.J. Stekhoven and P. Bühlmann. MissForest - nonparametric missing value imputation for mixed-type data. *Bioinformatics*, 2011. doi: 10.1093/bioinformatics/btr597.