How to impute missing values?

Genevieve Robin, Aude Sportisse

09 July 2019

Contents

If you have a dataset which contains missing values, it is relevant to impute the missing values, mainly for two reasons: (i) these values may be particularly interesting in themselves or (ii) the fully completed data is required to perform some estimation method that does not handle the missing data.

In this section we provide, for some of the main packages (the list is of course not thorough) to impute missing values, links to vignettes and tutorials, as well as a description of their main functionalities and reusable code. The goal is not to describe all the methods precisely, as many resources are already available, but rather to provide an overview of several imputation options. The methods we focus on are gathered in the table below.

Package	Data Types	Underlying Method	Imputation	Computational Time	Comments
Amelia	quantitative and binary	multivariate gaussian model	multiple	+	Binary variables modeled as Gaussians
mice	mixed	multivariate imputation by chained equations	multiple	-	Very flexible to data types, no parameter to tune
missForest	mixed	random forests	single	-	Requires large sample sizes, no parameter to tune
missMDA	mixed	component methods	single/multiple	+	Rank parameter to tune
softImpute	quantitative	low-rank matrix completion	single	+	Very fast, strong theoretical guarantees, regularization parameter to tune

```r
library(Amelia)
library(mice)
library(missForest)
library(missMDA)
```

```r
library(softImpute)
```

Consider the Los Angeles ozone pollution data in 1976, available in R using the function data.

This dataset already contains some missing values.

```
##      maxO3              T9             T12             T15
##  Min.   : 42.00   Min.   :11.30   Min.   :14.00   Min.   :14.90
##  1st Qu.: 70.00   1st Qu.:16.20   1st Qu.:18.48   1st Qu.:19.10
##  Median : 81.00   Median :18.10   Median :20.30   Median :22.10
##  Mean   : 89.89   Mean   :18.45   Mean   :21.38   Mean   :22.75
##  3rd Qu.:107.00   3rd Qu.:20.05   3rd Qu.:23.73   3rd Qu.:26.00
##  Max.   :166.00   Max.   :27.00   Max.   :32.70   Max.   :33.70
##  NA's   :13       NA's   :9       NA's   :16      NA's   :19
##       Ne9             Ne12            Ne15            Vx9
##  Min.   :0.000   Min.   :0.000   Min.   :0.000   Min.   :-6.5778
##  1st Qu.:3.000   1st Qu.:4.000   1st Qu.:3.000   1st Qu.:-3.4015
##  Median :6.000   Median :5.000   Median :5.000   Median :-0.9040
##  Mean   :4.887   Mean   :5.119   Mean   :4.889   Mean   :-1.2315
##  3rd Qu.:7.000   3rd Qu.:7.000   3rd Qu.:7.000   3rd Qu.: 0.6919
##  Max.   :8.000   Max.   :8.000   Max.   :8.000   Max.   : 5.1962
##  NA's   :6       NA's   :11      NA's   :13      NA's   :6
##       Vx12              Vx15             maxO3v          vent      pluie
##  Min.   :-7.8785   Min.   :-9.000   Min.   : 42.00   Est   : 9   Pluie:41
##  1st Qu.:-3.6941   1st Qu.:-3.939   1st Qu.: 70.50   Nord :29   Sec  :63
##  Median :-1.9039   Median :-1.830   Median : 81.00   Ouest:44   NA's : 8
##  Mean   :-1.6552   Mean   :-1.745   Mean   : 89.91   Sud  :19
##  3rd Qu.:-0.5209   3rd Qu.: 0.000   3rd Qu.:103.50   NA's :11
##  Max.   : 6.5778   Max.   : 5.000   Max.   :166.00
##  NA's   :14        NA's   :14       NA's   :5
```

softImpute

The `softImpute` package can be used to impute quantitative data. It fits a low-rank matrix approximation to a matrix with missing values via nuclear-norm regularization. A vignette is available online, as well as the original article (Hastie et al. 2015).

The softImpute function computes, based on an incomplete data set, a low-dimensional factorization which can be used to impute the missing values. The function is used as follows:

```r
# keep only quantitative variables
dat_miss_quanti <- as.matrix(ozone[, 1:11])
# perform softImpute
sft <- softImpute(x=dat_miss_quanti, rank.max=2, lambda=0, type=c("als", "svd"))
```

The main arguments are the following (more details can be found on the help page).

- `x`: the data set with missing values (matrix).
- `rank.max`: the restricted rank of the solution, which should not be bigger than min(dim(x))-1.
- `lambda`: the nuclear-norm regularization parameter.
- `type`: indicates the algorithm which should be used, among “svd” and “als”. “svd” returns an exact solution, while “als” returns an approximate solution (in exchange for a faster computation time).

To compute the imputed data set based on the softImpute results, one may use the following code:

```r
# compute the factorization
dat_imp_sft <- sft$u%*%diag(sft$d)%*%t(sft$v)
# replace missing values by computed values
dat_imp_sft[which(!is.na(dat_miss_quanti))] <- dat_miss_quanti[which(!is.na(dat_miss_quanti))]
```

To calibrate the parameter lambda, one may perform cross-validation, the code is given below. One uses the function produce_NA detailed in "amputation.R" available in the related R source code of "How to generate missing values?".

```r
source('amputation.R')

cv_sft <- function(y,
                   N = 10,
                   len = 20) {
  y <- as.matrix(y)
  Y2 <- y
  Y2[is.na(Y2)] <- 0
  d <- dim(y)
  n <- d[1]
  p <- d[2]
  m <- sum(!is.na(y))
  lambda1.max <- max(svd(Y2)$d)
  lambda1.min <- 1e-3*lambda1.max
  grid.lambda1 <-
    exp(seq(log(lambda1.min), log(lambda1.max), length.out = len))
  ylist <-
    lapply(1:N, function(k)
      produce_NA(as.matrix(y),perc.missing = 0.2))$data.incomp
  res.cv <- lapply(1:N, function(k) {
    sapply(1:len,
           function(i) {
             yy <-produce_NA(as.matrix(y),perc.missing = 0.2)$data.incomp
             res <-
               softImpute(as.matrix(yy),
                          lambda = grid.lambda1[i])
             u <- res$u
             d <- res$d
             v <- res$v
             if (is.null(dim(u))) {
               res <- d * u %*% t(v)
             } else {
               res <- u %*% diag(d) %*% t(v)
             }
             imp <- as.matrix(yy)
             imp[is.na(yy)] <- res[is.na(yy)]
             return(sqrt(sum((res - y) ^ 2, na.rm = T)))
           })

  })
  res.cv <- colMeans(do.call(rbind, res.cv))
  l <- which.min(res.cv)
  lambda <- grid.lambda1[l]
  return(lambda)
}
```

```
lambda_sft <- cv_sft(dat_miss_quanti)
```

Then, the imputation procedure can be performed using the value of lambda computed with cross-validation (the other parameters are set to their default value):

```
sft <- softImpute(x=dat_miss_quanti, lambda=lambda_sft)
dat_imp_sft <- sft$u%*%diag(sft$d)%*%t(sft$v)
dat_imp_sft[which(!is.na(dat_miss_quanti))] <- dat_miss_quanti[which(!is.na(dat_miss_quanti))]
head(dat_imp_sft)
```

```
##           [,1] [,2] [,3] [,4] [,5]     [,6] [,7]    [,8]       [,9]
## [1,]  87.00000 15.6 18.5 18.4    4 4.000000    8  0.6946 -1.7101000
## [2,]  63.87634 17.0 18.4 17.7    5 5.000000    7 -4.3301 -4.0000000
## [3,]  92.00000 15.3 17.6 19.5    2 5.000000    4  2.9544  1.8794000
## [4,] 114.00000 16.2 19.7 22.5    1 4.135278    0  0.9848 -0.8513703
## [5,]  94.00000 17.4 20.5 20.4    8 8.000000    7 -0.5000 -2.9544000
## [6,]  80.00000 17.7 19.8 18.3    6 6.000000    7 -5.6382 -5.0000000
##           [,10]     [,11]
## [1,] -0.6946000  84.00000
## [2,] -3.0000000  87.00000
## [3,]  0.5209000  82.00000
## [4,] -0.8676625  92.00000
## [5,] -4.3301000 114.00000
## [6,] -6.0000000  78.48951
```

mice

The `mice` package implements a multiple imputation methods for multivariate missing data. It can impute mixes of continuous, binary, unordered categorical and ordered categorical data, as well as two-level data. The original article describing the software, as well as the source package and example code are available online (Buuren and Groothuis-Oudshoorn 2011).

The mice function computes, based on an incomplete data set, multiple imputations by chained equations and thus returns m imputed data sets.

The main arguments are the following (more details can be found on the help page).

- `data`: the data set with missing values (matrix).
- `m`: number of multiple imputations.
- `method`: the imputation method to use.

In this case the predictive mean matching method is performed. Other imputation methods can be used, type `methods(mice)` for a list of the available imputation methods.

```
mice_mice <- mice(ozone,m=5,method="pmm") #contains m=5 completed datasets.
#get back the first completed dataset of the five available in mice_res
mice::complete(mice_mice,1)
```

The pool function combines all the results together based on Rubin's rules.

```
mice_with <- with(mice_mice, exp = lm(maxO3 ~ T9 + Ne9))
pool(mice_with)
```

```
## Class: mipo    m = 5
##               estimate        ubar             b            t dfcom        df
```

```
## (Intercept) 26.818791 205.6756516 22.50411027 232.6805839   109 71.76134
## T9           4.534808   0.4243114  0.05127139   0.4858371   109 68.00446
## Ne9         -4.150855   0.6015571  0.16142481   0.7952669   109 36.78947
##                   riv    lambda       fmi
## (Intercept) 0.1312986 0.1160601 0.1397071
## T9          0.1450012 0.1266385 0.1512387
## Ne9         0.3220139 0.2435783 0.2815995
```

References

Buuren, Stef van, and Karin Groothuis-Oudshoorn. 2011. "mice: Multivariate Imputation by Chained Equations in R." *Journal of Statistical Software* 45 (3): 1–67. http://www.jstatsoft.org/v45/i03/.

Hastie, Trevor, Rahul Mazumder, Jason D Lee, and Reza Zadeh. 2015. "Matrix Completion and Low-Rank Svd via Fast Alternating Least Squares." *The Journal of Machine Learning Research* 16 (1). JMLR. org: 3367–3402.