Deductive imputation with the deducorrect package

Mark van der Loo and Edwin de Jonge

Package version 1.1.2

February 14, 2012

Abstract

Numerical and categorical data used for statistical analyses is often plagued with missing values and inconsistencies. In many cases, a number of missing values may be derived, based on the consistency rules imposed on the data and the observed values in a record. The methods used for such derivations are called *deductive imputation*. In this paper, we describe the newly developed deductive imputation functionality of R package deducorrect. The package gained methods to deductively impute numerical as well as categorical data. Methods for setting up a partial data editing system are discussed as well.

This vignette (at version 1.1-1) is a literal transcript of Van der Loo and de Jonge (2011a). Please use that paper when referencing imputation functionality of the package. This vignette may be updated if the package is developed further.

Contents

1 Introduction

The quality of raw survey data is only rarely sufficient to allow for immediate statistical analysis. The presence of missing values (item nonresponse) and inconsistencies impedes straightforward application of standard statistical estimation methods, and statisticians often have to spend considerable effort to counterbalance the effect of such errors.

There are basically two ways to take the effect of data quality issues into account. The first is to adapt the statistical analysis such that the effects of these issues are taken into account. One well-documented example is to use weighting methods which take the effect of (selective) item nonresponse into account (Kalton and Kasprzyk, 1986; Bethlehem et al., 2011). The second way is to clean up the dataset so that missing values are completed and inconsistencies have been repaired. The latter method has the advantage that statistical analyses of the data become to a degree independent of the models used in data cleaning. Whichever way is chosen, in most cases additional assumptions are necessary to clean data or interpret the results of data analyses.

Recently, a number of near assumption-free data-cleaning methods have been reported which rely almost purely on record consistency rules imposed *a priori* on the data. Examples of such rules include account balances, positivity demands on variables or forbidden value combinations in categorical data. In a previous paper (Van der Loo et al., 2011) we reported on methods which use data consistency rules and information in inconsistent records to track down and repair typing errors, rounding errors and sign errors. The theory behind these methods was first published by Scholtus (2008, 2009) and the methods were implemented by us in R package deducorrect. Since these so-called deductive correction methods are based on adapting values, they are not suited for completing missing values.

In this paper, we report on an extension of the deducorrect package which allows for deductive imputation of missing values in either numerical or categorical data. The implemented methods were proposed by Pannekoek (2006) and De Waal et al. (2011). By deductive imputation we mean methods which use the observed values in a record together with consistency rules imposed on the record to uniquely derive values where possible. The values may be missing because of nonresponse, or they may be deemed missing by an error localization algorithm such as implemented in the editrules package (De Jonge and Van der Loo, 2011; Van der Loo and de Jonge, 2011b).

In section 2, we further introduce the concept of deductive imputation and show the easiest way of imputing values with the deducorrect package. In sections 3 and 4 we expand a bit on the theory and demonstrate the use of lower-level functionality of the package. Examples in R code are given throughout to help new users getting started.

2 Deductive imputation

2.1 Overview

Deductive imputation relies on in-record consistency rules to derive the value of variables which have not been completed from variables that have. These methods therefore rely on the assumption that the values used in the derivation have been completed correctly. For example, suppose we have a numerical record $\mathbf{x} = (x_1, x_2, x_3)$, subject to the rules

$$\begin{aligned} x_1 + x_2 &= x_3 \qquad (1)\\ \mathbf{x} &\geq \mathbf{0}. \qquad (2)\end{aligned}$$

Suppose we are given two values of $\mathbf{x}$, for example (NA, x_2, x_3), where NA stands for Not Available. In principle, the unknown value is easily derived from rule (1), but one must take care not to violate any other rules. For example, if $x_2 < 0$, the derived value for x_1 is most likely not the true value, since at least one of the values used to derive x_1 is invalid. Moreover, if $x_2 > x_3$, the derived value for x_1 will be negative, and therefore violate rule (2). For categorical data, analogous situations may arise.

The deductive imputation routines of the deducorrect package offer two mechanisms to avoid inconsistencies. The first is to explicitly check if consistent deductive imputation is possible based on the observed values. This is switched on by default for the functions deduImpute, deductiveZeros, the editmatrix method of solSpace and deductiveLevels. These functions will be discussed below. The second mechanism is the ability to point out variables besides the missing ones, which should be considered as if they were missing. A typical example would be to use the result of an error localization algorithm which points out erroneous fields in a record.

In the context of a complete automated data editing system, there are several places where deductive imputation or correction can be applied. Typically, one will apply such methods before the data is treated with more complicated imputation models. Figure 1 shows a workflow for automatic deductive data cleaning. It contains all (near) assumption-free corrections and imputations of the deducorrect package. If after these steps, missing values or errors remain, one has to resort to other methods and accept extra model assumptions. It should be noted that a common step such as detecting and repairing unit measure errors is not included here. However, such methods are easily implemented in R, and we refer to De Waal et al. (2011) for an overview.

Deductive imputation appears after the error localization step in the process flow chart of Figure 1. At that point one is certain that the missing variables together with the variables pointed out by the error localization algorithm can be imputed consistent with the edit rules. Error localization is not strictly necessary to perform deductive imputation with the deducor-

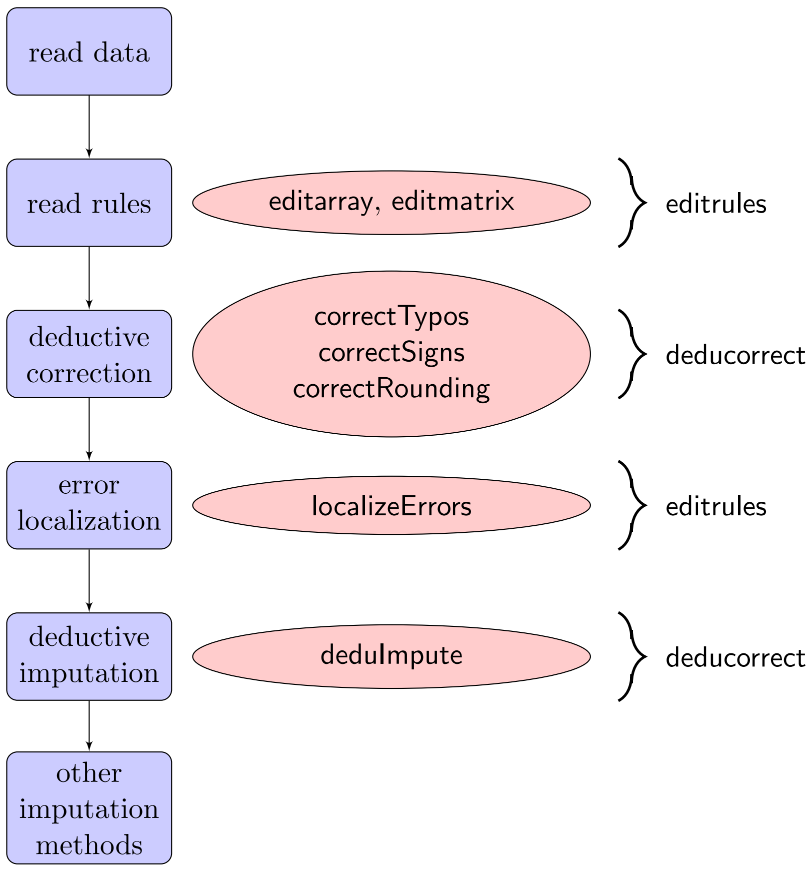

Figure 1: Flow diagram showing how functionality of the deducorrect and editrules can be combined to perform the deductive corrections, deductive imputations and error localization. All steps except deductive correction are available for numerical as well as categorical data. The ellipses indicate some of the R functions from the packages noted on the right. Error localization is not strictly necessary to perform deductive imputation, but in this way the maximum number deductively imputable values will be derived.

rect package since by default, the imputation routines check if consistent imputation is possible. However, the workflow in Figure 1 guarantees that as many deductive imputations take place as possible. For performance reasons, the user can choose to skip these checks when the workflow of Figure 1 is followed.

2.2 Imputation with deduImpute

The simplest way to do deductive imputations with the deducorrect package is to use the deduImpute function. It can be used for both numerical and categorical data. The function accepts an editmatrix or editarray containing the editrules and a data.frame containing the records. The return value is an object of class deducorrect, similar to the values returned by the correct-functions of deducorrect [see Van der Loo et al. (2011)].

For numerical data deduImpute uses two methods (described in sections 3.1 and 3.2) to impute as many empty values as possible. It uses the functions solSpace and deductiveZeros iteratively for each record until no deductive improvements can be made. Here, we will use the example from De Waal et al. (2011), Chapter 9.2. This example uses the following edits, based on a part of the Dutch Structural Business Survey balance account.

$$\begin{aligned}
x_1 + x_2 &= x_3 \\
x_2 &= x_4 \\
x_5 + x_6 + x_7 &= x_8 \\
x_3 + x_8 &= x_9 \\
x_9 - x_{10} &= x_{11} \\
x_6 &\geq 0 \\
x_7 &\geq 0
\end{aligned} \tag{3}$$

The rule $x_2 = x_4$ may seem odd for readers not familiar with survey statistics. However, these rules correspond to cases where respondents have to copy a figure from one page on a paper form to another[1]. In Figure 2 we give an example where the following record subject to the edits in Eq. (3) is treated.

```
   x1 x2  x3 x4 x5 x6 x7 x8 x9 x10 x11
1 145 NA 155 NA NA NA NA 86 NA 217  NA
```

The record contains missing values. However, by assuming that all non-missing values are correct, values can be derived for x_2, x_4, x_9 and x_{11} just by considering the equality- and nonnegativity rules in the edit set.

The assumption that all missing values can be imputed consistently may not alway be valid: the nonmissing values may have been filled in erroneously, yielding faulty derived values to impute. The reason is that deduImpute does not take into account all edit rules: only nonnegativity rules and equality rules are used to derive imputed values.

The deduImpute function has two mechanisms to get around this. The first is to set the option checkFeasibility=TRUE. This causes solutions causing new inconsistencies to be rejected. The second mechanism is to provide a user-specified adapt array to increase the number of variables which may be imputed, missing or not. The adapt array is a boolean array, stating which variable may be changed in which record. A convenient example is to use the adapt array as generated by the localizeErrors function from the editrules package. By specifying an adapt array, deduImpute will try to fix records by

[1]In spite of the availability of web-based forms, many respondents still prefer paper forms.

```
> E <- editmatrix(c(
+           "x1 + x2      == x3",
+           "x2           == x4",
+           "x5 + x6 + x7 == x8",
+           "x3 + x8      == x9",
+           "x9 - x10     == x11",
+           "x6 >= 0",
+           "x7 >= 0"
+ ))
> dat <- data.frame(
+     x1=c(145,145),
+     x2=c(NA,NA),
+     x3=c(155,155),
+     x4=c(NA,NA),
+     x5=c(NA, 86),
+     x6=c(NA,NA),
+     x7=c(NA,NA),
+     x8=c(86,86),
+     x9=c(NA,NA),
+     x10=c(217,217),
+     x11=c(NA,NA)
+ )
> dat

   x1 x2  x3 x4 x5 x6 x7 x8 x9 x10 x11
1 145 NA 155 NA NA NA NA 86 NA 217  NA
2 145 NA 155 NA 86 NA NA 86 NA 217  NA

> d <- deduImpute(E,dat)
> d$corrected

   x1 x2  x3 x4 x5 x6 x7 x8  x9 x10 x11
1 145 10 155 10 NA NA NA 86 241 217  24
2 145 10 155 10 86  0  0 86 241 217  24
```

Figure 2: A simple example with deduImpute. The return value is an object of class deducorrect.

imputing values which are either missing or may be adapted according to adapt.

For categorical data, deduImpute uses the deductiveLevels function, discussed in section 4. The function accepts an editarray holding the categorical edits and a data.frame holding records to be imputed.

Before introducing our example, we note that a categorical record is a member of a discrete set, written as the cartesian product.

$$D = D_1 \times D_2 \times \ldots \times D_n, \tag{4}$$

where each D_k is the set of categories for a single variable. An edit e can be written as a subset of D:

$$e = A_1 \times A_2 \times \cdots \times A_n, \tag{5}$$

where each $A_k \subset D_k$. The interpretation is that if a record $\mathbf{v} \in e$, then that record is invalid.

Here, we reproduce example 9.3 of De Waal et al. (2011) [first published by (Kartika, 2001)]. Consider four categorical variables with domains $D_1 = \{\mathrm{a,b,c,d}\}$, $D_2 = D_3 = \{\mathrm{a,b,c}\}$ and $D_4 = \{\mathrm{a,b}\}$. We define the edit rules

$$\begin{aligned}
e_1 &= D_1 \times \{\mathrm{c}\} \times \{\mathrm{a,b}\} \times \{\mathrm{a}\} && (6)\\
e_2 &= D_1 \times \{\mathrm{b,c}\} \times D_3 \times \{\mathrm{b}\} && (7)\\
e_3 &= \{\mathrm{a,b,d}\} \times \{\mathrm{a,c}\} \times \{\mathrm{b,c}\} \times D_4 && (8)\\
e_4 &= \{\mathrm{c}\} \times D_2 \times \{\mathrm{b,c}\} \times \{\mathrm{a}\}. && (9)
\end{aligned}$$

Out of 72 possible records, only the following 20 are valid:

$$\begin{array}{cccc}
(\mathrm{a,a,a,a}) & (\mathrm{b,a,a,a}) & (\mathrm{c,a,a,a}) & (\mathrm{d,a,a,a})\\
(\mathrm{a,a,a,b}) & (\mathrm{b,a,a,b}) & (\mathrm{c,a,a,b}) & (\mathrm{d,a,a,b})\\
(\mathrm{a,b,a,a}) & (\mathrm{b,b,a,a}) & (\mathrm{c,a,b,b}) & (\mathrm{d,b,a,a})\\
(\mathrm{a,b,b,a}) & (\mathrm{b,b,b,a}) & (\mathrm{c,a,c,b}) & (\mathrm{d,b,b,a})\\
(\mathrm{a,b,c,a}) & (\mathrm{b,b,c,a}) & (\mathrm{c,b,a,a}) & (\mathrm{d,b,c,a}).
\end{array}$$

Figure 3 shows how these rules can be defined in R using the editarray function of the editrules package. Consider the record (c,b,NA,NA). By simply considering the list of valid records above it is clear that if v_1 and v_2 are assumed correct, the only possible valid imputation is $v_3 = v_4 = \mathrm{a}$. Indeed this is returned by deduImpute in Figure 3. The record $(\mathsf{NA}, \mathsf{NA}, \mathsf{NA}, \mathrm{b})$ cannot be imputed completely, since there are six possible records with $v_4 = \mathrm{b}$. However, all of them have $v_2 = \mathrm{a}$, so this may be imputed with certainly. Finally, the record $(\mathrm{b,c,a},\mathsf{NA})$ cannot be imputed since there is no valid record with these values for v_1, v_2 and v_3.

```
> M <- editarray(c(
+ "v1 %in% letters[1:4]",
+ "v2 %in% letters[1:3]",
+ "v3 %in% letters[1:3]",
+ "v4 %in% letters[1:2]",
+ "if (v2 == 'c'  & v3 != 'c' & v4 == 'a' ) FALSE",
+ "if (v2 != 'a'  & v4 == 'b') FALSE",
+ "if (v1 != 'c'  & v2 != 'b' & v3 != 'a') FALSE",
+ "if (v1 == 'c'  & v3 != 'a' & v4 == 'a' ) FALSE"
+ ))
> Mdat <- data.frame(
+     v1 = c('c', NA,'b'),
+     v2 = c('b', NA,'c'),
+     v3 = c( NA, NA,'a'),
+     v4 = c( NA,'b', NA),
+     stringsAsFactors=FALSE
+ )
> s <- deduImpute(M, Mdat)
> s$corrected

    v1 v2   v3   v4
1    c  b    a    a
2 <NA>  a <NA>    b
3    b  c    a <NA>

> s$status

     status imputations
1 corrected           2
2   partial           1
3   invalid           0

> s$corrections

  row variable old new
1   1       v3  NA   a
2   1       v4  NA   a
3   2       v2  NA   a
```

Figure 3: Deductive imputations for categorical data using deduImpute.

3 Deductive imputation of numerical data

The valid value combinations of numerical data records with n variables are usually limited to some subset of $\mathbb{R}^n$. Common cases include balance accounts (linear restrictions) combined with linear inequality rules (positivity rules for example). In such cases the set of valid records is a convex polytope. In certain cases, when the values for a number of variables have been fixed, the set of possible values for a number of the remaining variables reduces to a point. In such cases deductive imputation is possible.

3.1 Imputation with solSpace and imputess

3.1.1 Area of application

The combination of functions solSpace and imputess can be used to impute numerical data under linear equality restrictions:

$$\mathbf{Ax} = \mathbf{b}, \text{ with } \mathbf{A} \in \mathbb{R}^{m\times n}, \quad \mathbf{x} \in \mathbb{R}^n \text{ and } \mathbf{b} \in \mathbb{R}^m. \tag{10}$$

If $\mathbf{x}$ has missing values, then solSpace returns a representation of the linear space of imputations valid under Eqn. (10). The function imputess performs the actual imputation. It is important to note that these functions do not take into account the presence of any inequality restrictions.

3.1.2 How it works

Consider a numerical record $\mathbf{x}$ with n_{miss} values missing. The values may be missing because of nonresponse, or they may be deemed missing by an error localization procedure (see the next subsection). We will write $\mathbf{x} = (\mathbf{x}_{\text{obs}}, \mathbf{x}_{\text{miss}})$, with $\mathbf{x}_{\text{obs}}$ the observed values and $\mathbf{x}_{\text{miss}}$ the missing ones. Supposing further that $\mathbf{x}$ must obey a set of equality restrictions as in Eqn. (10), we may write $\mathbf{A} = [\mathbf{A}_{\text{obs}}, \mathbf{A}_{\text{miss}}]$. Consequently we have (De Waal et al., 2011)

$$\mathbf{A}_{\text{miss}}\mathbf{x}_{\text{miss}} = \mathbf{b} - \mathbf{A}_{\text{obs}}\mathbf{x}_{\text{obs}}. \tag{11}$$

This gives

$$\mathbf{x}_{\text{miss}} = \mathbf{x}_0 + \mathbf{Cz}, \tag{12}$$

with $\mathbf{z}$ an arbitrary real vector of dimension n_{miss} and $\mathbf{x}_0$ and $\mathbf{C}$ constant.

The purpose of solSpace is to compute $\mathbf{x}_0$ and $\mathbf{C}$. Together they determine the vector space of values available for $\mathbf{x}_{\mathsf{miss}}$. Deductive imputation can be realized by observing that if any rows of $\mathbf{C}$ are filled with zeros, then the sole value for the corresponding values of $\mathbf{x}_{\mathsf{miss}}$ are given the corresponding values in $\mathbf{x}_0$. The values of $\mathbf{x}_0$ and $\mathbf{C}$ are given by

$$\begin{aligned} \mathbf{x}_0 &= \mathbf{A}^+_{\text{miss}}(\mathbf{b} - \mathbf{A}_{\text{obs}}\mathbf{x}_{\text{obs}}) && (13)\\ \mathbf{C} &= \mathbf{A}^+_{\text{miss}}\mathbf{A}_{\text{miss}} - \mathbf{1}. && (14) \end{aligned}$$

Here, $\mathbf{1}$ is the identity matrix and $\mathbf{A}^{+}_{\text{miss}}$ is the pseudoinverse of $\mathbf{A}$, obeying

$$\mathbf{A}_{\text{miss}}\mathbf{A}^{+}_{\text{miss}}\mathbf{A}_{\text{miss}} = \mathbf{A}_{\text{miss}}. \tag{15}$$

See De Waal et al. (2011) for details on the imputation method or Greville (1959) for an excellent discussion on the pseudoinverse.

3.1.3 An example

The solSpace function returns the $\mathbf{x}_0$ and $\mathbf{C}$ as a list. For example consider the first record from Figure 2:

```
> (x <- dat[1,])

   x1 x2  x3 x4 x5 x6 x7 x8 x9 x10 x11
1 145 NA 155 NA NA NA NA 86 NA 217  NA
```

Using the editmatrix defined in the same figure, we get:

```
> (s <- solSpace(E,x))

$x0
         [,1]
x2   10.00000
x4   10.00000
x5   28.66667
x6   28.66667
x7   28.66667
x9  241.00000
x11  24.00000

$C
    x2 x4         x5         x6         x7 x9 x11
x2   0  0  0.0000000  0.0000000  0.0000000  0   0
x4   0  0  0.0000000  0.0000000  0.0000000  0   0
x5   0  0 -0.6666667  0.3333333  0.3333333  0   0
x6   0  0  0.3333333 -0.6666667  0.3333333  0   0
x7   0  0  0.3333333  0.3333333 -0.6666667  0   0
x9   0  0  0.0000000  0.0000000  0.0000000  0   0
x11  0  0  0.0000000  0.0000000  0.0000000  0   0
```

solSpace has an extra argument adapt which allows extra fields of $\mathbf{x}$ to be considered missing. An example of its use would be to determine erroneous fields with errorLocalizer (of the editrules package) and to determine the imputation space with solSpace.

The top two and bottom two rows of $\mathbf{C}$ in the example have zero coefficients, yielding a unique solution for x_2, x_4, x_9 and x_{11}. The unique values may be imputed with imputess:

```
> imputess(x, s$x0, s$C)

   x1 x2  x3 x4 x5 x6 x7 x8  x9 x10 x11
1 145 10 155 10 NA NA NA 86 241 217  24
```

If a **z**-vector is provided as well [See Eq. (12)], all values may be imputed. Here, we choose $\mathbf{z} = \mathbf{0}$ (arbitrarily).

```
> ( y <- imputess(x, s$x0, s$C, z=rep(0,ncol(s$C))) )

   x1 x2  x3 x4       x5       x6       x7 x8  x9 x10 x11
1 145 10 155 10 28.66667 28.66667 28.66667 86 241 217  24
```

Using violatedEdits from the editrules package, we may verify that this record satisfies every inequality rule as well (E as in figure 2).

```
> any(violatedEdits(E,y,tol=1e-8))

[1] FALSE
```

To demonstrate the use of the adapt argument, consider the following case.

```
> Ey <- editmatrix(c(
+     "yt == y1 + y2 + y3",
+     "y4 == 0"))
> y <- c(yt=10, y1=NA, y2=3, y3=7,y4=12)

> (s <- solSpace(Ey,y))

NULL
```

However, using the adapt argument, which is a logical indicator stating which entries may be adapted, we get the following.

```
> (s <- solSpace(Ey, y, adapt=c(FALSE,FALSE,FALSE,FALSE,TRUE)))

$x0
   [,1]
y1    0
y4    0

$C
   y1 y4
y1  0  0
y4  0  0

> imputess(y,x0=s$x0,C=s$C)

yt y1 y2 y3 y4
10  0  3  7  0
```

3.2 Imputation with deductiveZeros

3.2.1 Area of application

This method can be used to impute missing values in numerical records subject to

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \text{ with } \mathbf{A} \in \mathbb{R}^{m\times n}, \quad \mathbf{x} \in \mathbb{R}^n \text{ and } \mathbf{b} \in \mathbb{R}^m \tag{16}$$

$$x_j \geq 0 \text{ for at least one } j \in \{1, 2, \ldots, n\}. \tag{17}$$

Economic survey data are often subject to account balances of the form $x_t = x_1 + x_2 + \cdots x_k$. For example, x_t might be the total personnel cost and the x_i are costs related to permanent staff, temporary staff, externals, *etc.* It is not uncommon for respondents to leave fields open which are not relevant to them. For example, if a company has not hired any temporary staff, the corresponding field might be left empty while a 0 would have been appropriate.

In such cases, missing values are bounded from above by the sum rules while they are bounded from below by the nonnegativity constraint. If the missing values are ignored, and the completed values add up to the required totals, then missing values may be uniquely imputed with 0. The function deductiveZeros detects such cases.

3.2.2 How it works

Consider again the notation of Section 3.1.2. We write (following notation of De Waal et al. (2011)).

$$\mathbf{b}^* = \mathbf{b} - \mathbf{A}_{\text{obs}}\mathbf{x}_{\text{obs}}. \tag{18}$$

If any $b^*_l = 0$, this means that the sum rule $\mathbf{a}_l \cdot \mathbf{x} = b_l$ is obeyed if missing values are ignored. For those cases, the following properties are checked.

- Each $a_{\text{miss},lj}$ has the same sign.
- Each $a_{\text{miss},lj} \neq 0$ corresponds to a variable x_j that is constrained to be nonnegative.

If these demands are obeyed, the corresponding value $x_{\text{miss},j}$ may be imputed with 0.

3.2.3 An example

The function deductiveZeros does not perform imputation itself but computes an indicator stating which values may be imputed. As a first example consider the following.

```
> Ey <- editmatrix(c(
+     "yt == y1 + y2 + y3",
+     "y1 >= 0", "y2 >= 0 ","y3 >= 0"))
> y <- c(yt=10, y1=NA, y2=3, y3=7)
> (I<-deductiveZeros(Ey,y))

   yt    y1    y2    y3
FALSE  TRUE FALSE FALSE
```

The record y can be imputed in one statement.

```
> y[I] <- 0
> y

yt y1 y2 y3
10  0  3  7
```

4 Deductive Imputation of categorical data

As shown in Eq. (4), a categorical data record is a member of a discrete set of value combinations D (the domain). In practice, not every record in D may be acceptable. For example if

$$D = \{\text{child}, \text{adult}\} \times \{\text{married}, \text{unmarried}\}, \tag{19}$$

then the record (child, married) may be excluded from the set of valid records. Therefore, if we have a record with (NA, married), and assume that the marital status is correct, there is only one possible value for the age class, namely “adult”. So just like for numerical data, if the known values limit the number of options for the unknowns to a unique value, deductive imputation is possible.

4.1 Imputation with deductiveLevels

4.1.1 Area of application

The function deductiveLevels works on purely categorical data where the number of categories for each variable is known and fixed, as in Eq. (4). It determines which missing values in a record are determined uniquely by the known values, and these unique values are returned.

4.1.2 How it works

The algorithm behind deductiveLevels has been described by De Waal et al. (2011) and is reproduced here in Algorithm 1. The Algorithm is described in terms of the functions eliminate and substValue, both of which are implemented in the editrules package and have been described extensively by

Algorithm 1 DEDUCTIVELEVELS(E,$\mathbf{v}$)

```
Input: An editarray $E$, a partially complete record $\mathbf{v}$
  Determine the index $I \subset \{1, 2, \ldots n\}$ in $\mathbf{v}$ of observed values.
  $E \leftarrow$ SUBSTVALUE($E, I, \mathbf{v}_I$)
  if $\neg$ISFEASIBLE($E$) then
      return $\varnothing$
  end if
  $M \leftarrow \{1, 2, \ldots, n\}\backslash I$                ▷ Index of missing values in $\mathbf{v}$
  $T \leftarrow \varnothing$
  $S \leftarrow \varnothing$
  while $M\backslash T \neq \varnothing$ do
      $m \leftarrow M_1$
      $F \leftarrow E$
      for $k \in M\backslash m$ do                         ▷ Eliminate all but $k$ from $F$
          $F \leftarrow$ ELIMINATE($F, k$)
      end for
      if There is one possible value $\tilde{v}$ for variable $m$ in $F$ then
          $E \leftarrow$ SUBSTVALUE($E, m, \tilde{v}$)
          $M \leftarrow M\backslash m$
          $S \leftarrow S \cup (m, \tilde{v})$
      else
          $T \leftarrow T \cup m$
      end if
  end while
Output: Unique imputations $S$.
```

Van der Loo and de Jonge (2011a). In short, deductiveLevels derives deductive imputations by first substituting all observed values in the edit rules. Subsequently, all variables but one are eliminated from the remaining edits. If only one possible value remains for the remaining variable, it may be used as a deductive imputation and substituted in the set of edits. This process is repeated until all missing values are treated.

4.1.3 An example

Consider the variables v_1 =*gender*, v_2 =*pregnant* and v_3 =*chromosome*. The value domain and edit rules are given by

$$D_1 = \{\text{male, female}\} \tag{20}$$
$$D_2 = \{\text{TRUE}, \text{FALSE}\} \tag{21}$$
$$D_3 = \{\text{xx, xy}\} \tag{22}$$
$$e_1 = \{\text{male}\} \times \{\text{TRUE}\} \times D_3 \tag{23}$$
$$e_2 = \{\text{male}\} \times D_2 \times \{\text{xx}\}. \tag{24}$$

The corresponding editarray can be defined as follows.

```
> E <- editarray(c(
+     "gender     %in% c('male','female')",
+     "pregnant   %in% c(TRUE,FALSE)",
+     "chromosome %in% c('XX','XY')",
+     "if (gender == 'male') !pregnant",
+     "if (gender == 'male') chromosome == 'XY'"))
```

Now, consider the record (male, FALSE, NA). Using deductiveLevels we find:

```
> v <- c(gender='male',pregnant=FALSE,chromosome=NA)
> (s <- deductiveLevels(E,v))

chromosome
      "XY"
```

And imputation can be performed as follows:

```
> v[names(s)] <- s
> v

    gender   pregnant chromosome
    "male"    "FALSE"       "XY"
```

The deductiveLevels function has an optional argument, allowing to switch off the feasibility check. To illustrate this, consider the record (male, TRUE, NA). Clearly, there is no way to impute this record consistently by just imputing the *chromosome* variable. If we choose v_3 = XX, this conflicts with the gender (male) if we choose XY this conflicts with the gender implied by v_2 (pregnant). In this case deductiveLevels returns NULL.

```
> v <- c(gender='male',pregnant=TRUE,chromosome=NA)
> deductiveLevels(E,v)

NULL
```

The reason is that deductiveLevels checks if feasible imputations are possible after substituting all observed values into the edits. This check can be time-consuming since it potentially involves many variable elimination steps. It may be turned off by passing checkFeasibility=FALSE:

```
> deductiveLevels(E,v,checkFeasibility=FALSE)

chromosome
      "XY"
```

However, one must be careful since, as shown above, the result may be an inconsistent imputation. The reason to include this option is that users may provide an additional parameter, called adapt, allowing deductiveLevels to impute more variables. If the adapt parameter is chosen such that missing values plus adaptable values can lead to consistent imputation, the consistency check may be turned off. For example, we may choose to adapt the pregnancy status.

```
> adapt <- c(gender=FALSE,pregnant=TRUE,chromosome=TRUE)
> (s <- deductiveLevels(E,v,adapt=adapt,checkFeasibility=FALSE))

  pregnant chromosome
   "FALSE"       "XY"
```

So that the imputed value becomes

```
> v[names(s)] <- s
> v

    gender   pregnant chromosome
    "male"    "FALSE"       "XY"
```

which is indeed a valid record. In general, the adapt parameter should be derived via a consistent error localization mechanism, such as implemented in the editrules package. Only those cases it is safe to gain some performance by switching the feasibility check off.

5 Conclusions

Missing values and inconsistencies in raw data often hinder statistical analyses. However, in many cases, correct values can be derived using only the available values and consistency rules imposed on the data (deductive imputation). As of version 1.1, the deducorrect package includes functionality for deductive imputation of both numerical and categorical data. The functionality of the deducorrect package can be used as-is, or may be integrated with the error localization functionality of the editrules package.

Future work on the package will include several performance enhancements and visualisation options.

References

Bethlehem, J., F. Cobben, and B. Schouten (2011). *Handbook of Nonresponse in Household Surveys*, Volume 562 of *Wiley handbooks in survey methodology.* John Wiley & Sons.

De Jonge, E. and M. Van der Loo (2011). Manipulation of linear edits and error localization with the editrules package. Technical Report 2011020, Statistics Netherlands, The Hague/Heerlen.

De Waal, T., J. Pannekoek, and S. Scholtus (2011). *Handbook of statistical data editing and imputation.* Wiley handbooks in survey methodology. Hoboken, New Jersey: John Wiley & Sons.

Greville, T. N. E. (1959). The pseudoinverse of a rectangular or singular matrix and its application to the solution of systems of linear equations. *SIAM Review 1*, 38–43.

Kalton, G. and D. Kasprzyk (1986). The treatment of missing survey data. *Survey methodology 12*, 1–16.

Kartika, W. (2001). Consistent imputation of categorical and numerical data. Technical report, Statistics Netherlands, Den Haag.

Pannekoek, J. (2006). Regression imputation with linear equality constraints on the variables. In *UNECE Work session on statistical data editing in Bonn.* http://www.unece.org/stats/documents/2006.09.sde.html.

Scholtus, S. (2008). Algorithms for correcting some obvious inconsistencies and rounding errors in business survey data. Technical Report 08015, Statistics Netherlands, Den Haag. The papers are available in the inst/doc directory of the R package or via the website of Statistics Netherlands.

Scholtus, S. (2009). Automatic correction of simple typing error in numerical data with balance edits. Technical Report 09046, Statistics Netherlands, Den Haag. The papers are available in the inst/doc directory of the R package or via the website of Statistics Netherlands.

Van der Loo, M. and E. de Jonge (2011a). Deductive correction with the deducorrect package. Technical Report 2011xx, Statistics Netherlands, Den Haag. In press.

Van der Loo, M. and E. de Jonge (2011b). Manipulation of categorical edits and error localization with the editrules package. Technical Report 201129, Statistics Netherlands, The Hague/Heerlen.

Van der Loo, M., E. de Jonge, and S. Scholtus (2011). Correction of rounding, typing and sign errors with the `deducorrect` package. Technical

Report 201119, Statistics Netherlands, Den Haag. This paper is included with the package.