Manipulation of categorical and mixed data edits and error localization with the `editrules` package

package version 1.0.0

Mark van der Loo and Edwin de Jonge

August 1, 2011

Abstract

This vignette is far from finished. Version 2.0 fo the package will have the full vignette. At the moment, functionality for treating categorical data has alpha status and is not public yet.

Contents

1 Introduction

2 Defining and checking categorical constraints

We spend some time to define the boolean representation, since it eventually allow for an elegant elimination method.

2.1 The `editarray` object

A categorical record $\mathbf{v}$ is a combination of values from the cartesian product space

$$D = D_1 \times D_2 \times \ldots \times D_n, \tag{1}$$

where each D_k is a finite set of d_k categories. The total number of categories is $d = \sum_{k=1}^{n} d_k$. Categorical records can be represented as a boolean vector. A boolean vector of dimension d is an element of the boolean algebra

$$\mathbb{B}^d = \left(\{0,1\}^d, \wedge, \vee, \neg\right), \tag{2}$$

where 0 and 1 have the usual interpretations FALSE and TRUE and the logical operators work elementwise on their operands. To represent a record $\mathbf{v}$, assign to every category i in D_k a unique standard basisvector $\vec{\delta}_k(i)$ of $\mathbb{B}^{d_k}$. The boolean representation $\rho(\mathbf{v})$ of the full record reads

$$\mathbf{v} \xrightarrow{\rho} \vec{\delta}_1(v_1) \oplus \vec{\delta}_2(v_2) \oplus \ldots \oplus \vec{\delta}_n(v_n). \tag{3}$$

An edit e is a subset of D, which can be written as the cartesian product

$$e = \mathbf{e}_1 \times \mathbf{e}_2 \ldots \times \mathbf{e}_n, \tag{4}$$

where each $\mathbf{e}_k \subseteq D_k$. The interpretation of an edit is that if a record $\mathbf{v} \in e$, then $\mathbf{v}$ is considered invalid. We say that a record *fails* an edit when it satisfies Eq. (8). A variable v_i is *involved* in an edit if $\mathbf{e}_i \subset D_j$ is a proper subset. Conversely, we say that e contains v_k if v_k is involved in e. A variable for which $\mathbf{e}_k = D_k$ is not contained in e.

Since every category i of D_k is mapped to a unique basis vector $\vec{\delta}_k(i)$, edits have a boolean representation $\rho(e)$, given by

$$e \xrightarrow{\rho} \bigvee_{i \in \mathbf{e}_1} \vec{\delta}_1(i) \oplus \bigvee_{i \in \mathbf{e}_2} \vec{\delta}_2(i) \oplus \ldots \oplus \bigvee_{i \in \mathbf{e}_k} \vec{\delta}_n(i), \tag{5}$$

which may simply be written as

$$\rho(e) = \mathbf{a} = \mathbf{a}_1 \oplus \mathbf{a}_2 \oplus \ldots \oplus \mathbf{a}_n. \tag{6}$$

One may think of the $\mathbf{a}_k$ as boolean vectors indicating which elements of D_k occur in $\mathbf{e}_k$.

```
> #editarray(c(
> #    "gender %in% c('male','female')",
> #    "pregnant %in% c('yes','no')",
> #    "if (gender == 'male') pregnant == 'no'"
> #    )
> #)
```

Figure 1: Defining a simple `editarray` with the `editarray` function. Column heads of the array are abbreviated versions of variable names and categories seperated by a colon (by default). When printed to screen, a `character` version of the edits is shown as well, for readability.

In the `editrules` package, a set of categorical edits is represented as an `editarray` object. Formally, we denote an edit array E for n categorical variables and m edits as

$$E = \langle \mathbf{A} \rangle, \text{ with } \mathbf{A} \in \{0,1\}^{m \times d} \text{ and } d = \sum_{k=1}^{n} d_k. \tag{7}$$

Here, each row of $\mathbf{A}$ contains the boolean representation of one edit, and the d_k denote the number of categories of each variable. The brackets are used to indicate a combination of objects. The `editarray` is the central object for computing with catgegorical edits, just like the `editmatrix` is the central object for computations with linear edits.

It is both tedious and error prone to define and maintain an editarray by hand. In practice, categorical edits are usually stated verbosely, such as: "if the gender is male, then pregnant must be false", or "if you are under-aged, you cannot be married". To facilitate the definition of edit arrays, `editrules` is equipped with a parser, which takes R-statements in `character` format, and translates them to an `editarray`.

Figure 1 shows a simple example of defining an editmatrix with the editrules package. The first two edits in Figure 1 define the data model. The `editarray` function derives the datamodel based on the variable names and categories it finds in the edits, whether they are univariate (defining domains) of multivariate. This means that if all possible variables and categories are mentioned in the multivariate edits, the correct datamodel will be derived as well. It is important to note that most functions working with categorical edits, assume that the full datamodel is represented in the columns of an `editarray`. The function `datamodel` accepts an edit array as input and returns an overview of variables and their categories.

When printed to the screen, the boolean array is shown with column heads of the form

`<variable><separator><category>`,

where both variable names and categories are abbreviated for readability, and the standard separator is a colon (`:`). The separator may not occur as a

Table 1: Functions and operators that may be used to define edits with `editarray`

Operator	Description
`%in%`	Set membership*
`==`	Equality*
`if( <condition> ) <expression>`	conditional statement
`c('<cat1>','<cat2>',...)`	categories, `character` or `logical`
`&&`	logical AND
`||`	logical OR
`!`	logical NOT

*Right-hand side is evaluated.

symbol in either variable or category name, and its value can be determined by passing a custom `sep` argument to `editarray`.

Internally, `editarray` uses the R internal `parse` function to transform the `character` expressions to a parse tree, which is subsequently traversed recursively to derive the entries of the editmatrix. The opposite is also possible. The R internal function `as.character` has been overloaded to derive a `character` representation from a boolean representation. When printed to the screen, both the boolean and textual representation are shown.

The character expressions that can be read by `editarray`, such as

```
"if ( gender == 'male' ) pregnant = 'no'"
```

follows standard R syntax, which should be already familliar to the reader. Note that double quotes are used to enclose the whole string, while single quotes are used for category names. Table 1 shows which operators and functions can be used to specify categorical edit rules. Categories may be litaral characters, or booleans. It is worth noting that expressions on the right hand side of the `%in%` and `==` operators are evaluated. One useful application of this is that the categories, or data model can be defined outside of the edits:

```
> #xval <- letters[1:4]
> #yval <- c(TRUE,FALSE)
> #editarray(c( "x %in% xval","y %in% yval","if ( x %in% c('a','b') ) !y "))
```

The above example also illustrates the use of boolean categories.

2.2 Basic manipulations and edit checking

Table 2 shows lists basic functions of `editmatarray` objects. The `datamodel` function retrieves the variables and categories in an edit array, and returns them as a two-column `data.frame`.

Table 2: Basic functions for objects of class `editarray`. Only mandatory arguments are shown, refer to the built-in documentation for optional arguments.

Function	description
`datamodel(E)`	get datamodel
`getArr`	get array $\mathbf{A}$
`as.data.frame(E)`	coerce to `data.frame`
`as.character(E)`	coerce to `character` vector
`violatedEdits(E,x)`	check which edits are violated by `x`
`duplicated(E)`	detect duplicate edits
`isObviouslyRedundant(E)`	detect simple redundancies
`isSubset(E)`	detect edits contained in another edit
`isObviouslyInfeasible`	detect if `E` contains a contradiction
`substValue`	substitute a value

With `getArr` the array part of an `editarray` can be retrieved. The other contents of an `editarrray` are of no use for users and are `editrules` internal.

The function `violatedEdits` takes a `data.frame` or named `character` vector as input and returns a logical array where each row indicates which edits are violated by the input data. The relation

$$\mathbf{v} \in e \Leftrightarrow \sum_{j=1}^{d} (\rho(\mathbf{v}) \wedge \rho(e))_j = n, \tag{8}$$

is used to test the validity of records. Here, n is the number of variables and the sum is over the coefficients of $\rho(\mathbf{v}) \wedge \rho(e)$, interpreted as numbers. The relation holds since a record can have at most one coefficient per variable equal to 1 in the boolean representation.

2.3 Obvious redundancy and infeasibility

When manipulating edit sets, edits may arise which have $\mathbf{e}_k = \varnothing$ for one of the variables. A record can never be an element of such an edit, and such edits are therefore redundant. Such redundancies are easily detected since all coefficients of the corresponding $\mathbf{a}_k$ (Eq. (6)) are equal to zero for these edits. The function `isObviouslyRedundant` returns a logical vector indicating which edits in an editarray have become obviously redundant. If an edit is a subset of another edit, it is also redundant. The function `isSubset` returns a logical, indicating if an edit is a subset of any other edit in the `editarray`.

Manipulations may also lead to edits of the form $e = D$, in which case every possible record is invalid, and the editarray has become impossible

to satisfy. The function `isObviouslyInfeasible` detects whether any such edits are present.They can be detected by checking if $\sum_{j=1}^{d} \rho(e)_j = d$.

3 Manipulation of categorical restrictions

3.1 Value substitution

Substituting a value in an editarray consists of two steps. First, all edits that exclude the value which is to be subsituted are removed. They have become redundant, since no record can be contained in such edits. Second, if variable k is substituted by category $j \in D_k$, for every remaining edit, all coefficients of $\mathbf{a}_k$ except the jth are set to 0. The function `substValue` can be used to substitute one or more values in an `editarray`.

3.2 Variable elimination by category resolution

Suppose e_1 and e_2 are edits, with boolean representations $\mathbf{a}$ and $\mathbf{b}$ respectively. We define the *resolution operator* $\mathfrak{R}_j$ as follows:

$$\mathbf{a}\,\mathfrak{R}_j\,\mathbf{b} = (a_1 \wedge b_1, \ldots, a_{j-1} \wedge b_{j-1}, a_j \vee b_j, a_{j+1} \wedge b_{j+1}, \ldots, a_d \wedge b_d). \quad (9)$$

Observe that if a record $\mathbf{v} \in \mathbf{a}\,\mathfrak{R}_j\,\mathbf{b}$, it must hold that $\mathbf{v} \in e_1$ and/or $\mathbf{v} \in e_2$. The converse is also true: If $\mathbf{v} \in e_1$ or e_2, then it must be in $e_1\,\mathfrak{R}_j\,e_2$. The operator is called resolution operator since its action strongly resembles a resolution operation from formal logic. If $a_j = \neg b_j$, the operator "resolves" or eliminates one of the categories $j \in D$. If $a_j = b_j$, the resulting edit is the intersection of the original edits.

Given the boolean vectors $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ in $\mathbb{B}^d$. It is not difficult to show that the resolution operator has the following properties

$$\begin{array}{ll} \text{symmetry:} & \mathbf{a}\,\mathfrak{R}_j\,\mathbf{b} = \mathbf{b}\,\mathfrak{R}_j\,\mathbf{a} \\ \text{associativity:} & (\mathbf{a}\,\mathfrak{R}_j\,\mathbf{b})\,\mathfrak{R}_j\,\mathbf{c} = \mathbf{a}\,\mathfrak{R}_j(\mathbf{b}\,\mathfrak{R}_j\,\mathbf{c}) \\ \text{idempotency:} & \mathbf{a}\,\mathfrak{R}_j\,\mathbf{a} = \mathbf{a}. \end{array} \quad (10)$$

The following distributive property also holds

$$\text{distributivity:} \quad (\mathbf{a}\,\mathfrak{R}_i\,\mathbf{b})\,\mathfrak{R}_j\,\mathbf{c} = (\mathbf{a}\,\mathfrak{R}_j\,\mathbf{c})\,\mathfrak{R}_i(\mathbf{b}\,\mathfrak{R}_j\,\mathbf{c}). \quad (11)$$

References