msmtools

Francesco Grossetti

2016-04-20

Contents

0.1 Introduction

Package **msmtools** is an `R` package whose main goal is to facilitate the workflow with longitudinal datasets which need to be analyzed in the context of multi-state models. In particular, `msmtools` acts as the `msm` package companion.

0.2 Longitudinal Dataset

Everytime we observe a given subject multiple times, we come up with a longitudinal dataset. This means that measures are repeated n times in a sequence which, in general, may not be equal for all the subjects. Moreover, a longitudinal dataset could be viewed as a multilevel dataset: a first level is given by the subject, and a second level is given by the single observation carried out on that subject. A very common case of longitudinal dataset deals with hospital admissions. A patient, our subject, can have a series of entries which correspond to hospital admissions. Each hospital admission is recorded in a single row of the dataset. Let's consider a simplified version of the `hosp` dataset which comes with `msmtools` package and represents synthetic hospital admissions for 10 patients. For a detailed description of the dataset, please run `?hosp`. For demonstration purposes, we extract only the first 2 patients, reducing the `hosp` dataset to a test sample of 17 rows per 8 variables as you can see below.

```
##     subj adm_number gender age label_2     dateIN    dateOUT   dateCENS
##  1:    1          1      F  83    dead 2008-11-30 2008-12-12 2011-04-28
##  2:    1          2      F  83    dead 2009-01-26 2009-02-16 2011-04-28
##  3:    1          3      F  83    dead 2009-05-13 2009-05-15 2011-04-28
```

```
##  4:    1          4      F  83    dead 2009-05-20 2009-05-25 2011-04-28
##  5:    1          5      F  83    dead 2009-06-12 2009-06-16 2011-04-28
##  6:    1          6      F  83    dead 2009-06-20 2009-06-25 2011-04-28
##  7:    1          7      F  83    dead 2009-07-17 2009-07-22 2011-04-28
##  8:    1          8      F  84    dead 2010-04-15 2010-04-20 2011-04-28
##  9:    1          9      F  84    dead 2010-10-11 2010-10-14 2011-04-28
## 10:    1         10      F  85    dead 2011-01-14 2011-01-17 2011-04-28
## 11:    1         11      F  85    dead 2011-04-27 2011-04-28 2011-04-28
## 12:    2          1      F  99   alive 2007-09-17 2007-09-27 2012-12-31
## 13:    2          2      F 100   alive 2009-04-09 2009-04-17 2012-12-31
## 14:    2          3      F 103   alive 2012-04-16 2012-04-20 2012-12-31
## 15:    2          4      F 103   alive 2012-04-24 2012-05-19 2012-12-31
## 16:    2          5      F 103   alive 2012-05-20 2012-05-25 2012-12-31
## 17:    2          6      F 103   alive 2012-08-19 2012-08-21 2012-12-31
```

So, these two patients are 'observed' 11 and 6 times through years, respectively.

These data format are very common when dealing with observational studies, or with chronic disease monitoring and with hospital admissions recording. In general, they are a well stabilized system to collect information.

0.3 Enhancing the Longitudinal Structure with `augment()`

Why the standard longitudinal structure is not enough if a multi-state model has to be run? A first observation could be that we are not able to infer anything about the state in which a given subject (i.e. patient) is at a particular point in time (i.e. hospital admission). The function `augment()` comes into play for this reason: to take advantage of the longitudinal structure in order to extract usable information to fuel a multi-state model. `augment()` takes a longitudinal dataset with exact starting and ending times and reshape it to produce an *augmented* version. For instance, if you apply `augment()` to the dataset above, you get what follows:

```
## Warning in augment(data = hosp, data_key = subj, n_events = adm_number, :
## no t_death has been passed. Assuming that dateCENS contains both censoring
## and death time
```

```
##     subj adm_number gender age label_2  augmented status n_status
##  1:    1          1      F  83    dead 2008-11-30     IN     1 IN
##  2:    1          1      F  83    dead 2008-12-12    OUT    1 OUT
##  3:    1          2      F  83    dead 2009-01-26     IN     2 IN
##  4:    1          2      F  83    dead 2009-02-16    OUT    2 OUT
##  5:    1          3      F  83    dead 2009-05-13     IN     3 IN
##  6:    1          3      F  83    dead 2009-05-15    OUT    3 OUT
##  7:    1          4      F  83    dead 2009-05-20     IN     4 IN
```

```
##  8:    1          4      F  83    dead 2009-05-25    OUT    4 OUT
##  9:    1          5      F  83    dead 2009-06-12     IN     5 IN
## 10:    1          5      F  83    dead 2009-06-16    OUT    5 OUT
## 11:    1          6      F  83    dead 2009-06-20     IN     6 IN
## 12:    1          6      F  83    dead 2009-06-25    OUT    6 OUT
## 13:    1          7      F  83    dead 2009-07-17     IN     7 IN
## 14:    1          7      F  83    dead 2009-07-22    OUT    7 OUT
## 15:    1          8      F  84    dead 2010-04-15     IN     8 IN
## 16:    1          8      F  84    dead 2010-04-20    OUT    8 OUT
## 17:    1          9      F  84    dead 2010-10-11     IN     9 IN
## 18:    1          9      F  84    dead 2010-10-14    OUT    9 OUT
## 19:    1         10      F  85    dead 2011-01-14     IN    10 IN
## 20:    1         10      F  85    dead 2011-01-17    OUT   10 OUT
## 21:    1         11      F  85    dead 2011-04-27     IN    11 IN
## 22:    1         11      F  85    dead 2011-04-28   DEAD      DEAD
## 23:    2          1      F  99   alive 2007-09-17     IN     1 IN
## 24:    2          1      F  99   alive 2007-09-27    OUT    1 OUT
## 25:    2          2      F 100   alive 2009-04-09     IN     2 IN
## 26:    2          2      F 100   alive 2009-04-17    OUT    2 OUT
## 27:    2          3      F 103   alive 2012-04-16     IN     3 IN
## 28:    2          3      F 103   alive 2012-04-20    OUT    3 OUT
## 29:    2          4      F 103   alive 2012-04-24     IN     4 IN
## 30:    2          4      F 103   alive 2012-05-19    OUT    4 OUT
## 31:    2          5      F 103   alive 2012-05-20     IN     5 IN
## 32:    2          5      F 103   alive 2012-05-25    OUT    5 OUT
## 33:    2          6      F 103   alive 2012-08-19     IN     6 IN
## 34:    2          6      F 103   alive 2012-08-21    OUT    6 OUT
## 35:    2          6      F 103   alive 2012-08-21    OUT    6 OUT
##     subj adm_number gender age label_2  augmented status n_status
```

Despite the fact that not the same variables have been reported because of layout concerns, two things come up at first sight. In the first place, the number of rows is more than doubled. We now have 35 observations against the initial 17. In the second place, new variables have been created. We will describe them in a minute.

Given the complexity of the data, which can be very high, building a subject specific status flag which marks a its condition at given time steps, could be tricky and computationally intensive. At the end of the study, so at the censoring time, a subject, in general, can be alive, dead inside a given transition if death occurs within `t_start` and `t_end`, or outside a given transition if death occurs otherwise. After n events, the corresponding flag sequence is given by $2n + 1$ for subjects alive and dead outside the transition, while it is just $2n$ for subjects who died inside of it. Let us consider an individual with 3 events. His/her status combinations will be as follows:

- **ALIVE**: IN-OUT | IN-OUT | IN-OUT | OUT

- **DEAD OUT**: IN-OUT | IN-OUT | IN-OUT | DEAD
- **DEAD IN**: IN-OUT | IN-OUT | IN-DEAD.

This operation produces a dataset in the augmented long format which allows to neatly model transitions between the given states.

From now on, we refer to each row as a transition for which we define a state in which the subject lies. `augment()` automatically creates 4 new variables (if argument `more_status` is missing):

- *augmented*: the new timing variable for the process when looking at transitions. If `t_augmented` is missing, then `augment()` creates *augmented* by default. The function looks directly to `t_start` and `t_end` to build it and thus it inherits their class;
- *status*: a status flag which looks at `state`. `augment()` automatically checks whether argument `pattern` has 2 or 3 unique values and computes the correct structure of a given subject. The variable is cast as character;
- *status_num*: the corresponding integer version of *status*;
- *n_status*: a mix of *status* and *status_num* cast as character. *status_num* comes into play when a model on the progression of the process is intended.

0.4 What if a more complex status structure is needed?

`augment()` by default takes a very simple status structure given by 3 different values. In general, this is enough to define a multi-state model. But what if we need a more complex structure. Let's consider again the dataset `hosp` for the 3rd, 4th, 5th, and 6th patient with the following variables:

```
data( hosp )
hosp[ 18:28, .( subj, adm_number, rehab, it, rehab_it,
                dateIN, dateOUT, dateCENS ) ]
```

```
##     subj adm_number rehab it rehab_it     dateIN    dateOUT   dateCENS
##  1:    3          1     0  0       df 2012-09-18 2012-09-27 2012-12-31
##  2:    3          2     0  1       it 2012-11-28 2012-12-15 2012-12-31
##  3:    3          3     1  0    rehab 2012-12-18 2012-12-28 2012-12-31
##  4:    4          1     0  0       df 2008-08-13 2008-09-20 2012-12-31
##  5:    4          2     0  0       df 2012-03-18 2012-03-19 2012-12-31
##  6:    4          3     0  1       it 2012-07-02 2012-07-20 2012-12-31
##  7:    5          1     0  0       df 2006-02-09 2006-02-25 2008-04-16
##  8:    6          1     0  0       df 2009-03-05 2009-03-16 2010-12-19
##  9:    6          2     0  0       df 2009-07-06 2009-07-20 2010-12-19
## 10:    6          3     0  0       df 2010-11-17 2010-11-23 2010-12-19
## 11:    6          4     0  0       df 2010-12-05 2010-12-19 2010-12-19
```

As you can see, we have two variables which take into account the type of hospital admission. *rehab* marks a rehabilitation admission while *it* marks an intensive therapy one. They are both binary and integer variables, so one can compose them to get something which is informative and, at the same time, usable in the context of 'making a status'. We then created the variable *rehab_it* which marks all the information in one place and it is a character. You can pass *rehab_it* to the argument `more_status` to tell `augment()` to add these information into a new structure. Now, it is important to remember that `augment()` introduces some rules when you require to compute a more complex status structure. As you can see from the dataset, many values of *rehab_it* are set to `df`. This stands for 'default' and when `augment()` finds it, it just compute the default status you already passed to argument state (i.e. in this case, it can be 'IN', 'OUT', or 'DEAD'). The argument `more_status` always looks for the value `df`, hence whenever you need to specify a default transition make sure to label it with this value. So, if we run `augment()` on this sample, we obtain the following:

```
hosp_augmented = augment( data = hosp, data_key = subj,
                          n_events = adm_number, pattern = label_2,
                          t_start = dateIN, t_end = dateOUT,
                          t_cens = dateCENS, more_status = rehab_it,
                          verbose = FALSE )
```

```
## Warning in augment(data = hosp, data_key = subj, n_events = adm_number, :
## no t_death has been passed. Assuming that dateCENS contains both censoring
## and death time
```

```
hosp_augmented[ 36:60, .( subj, adm_number, rehab_it,
                          augmented, status, status_exp, n_status_exp ) ]
```

```
##     subj adm_number rehab_it  augmented status status_exp n_status_exp
##  1:    3          1       df 2012-09-18     IN      df_IN       1 df_IN
##  2:    3          1       df 2012-09-27    OUT     df_OUT      1 df_OUT
##  3:    3          2       it 2012-11-28     IN      it_IN       2 it_IN
##  4:    3          2       it 2012-12-15    OUT     it_OUT      2 it_OUT
##  5:    3          3    rehab 2012-12-18     IN   rehab_IN   3 rehab_IN
##  6:    3          3    rehab 2012-12-28    OUT  rehab_OUT  3 rehab_OUT
##  7:    3          3    rehab 2012-12-28    OUT  rehab_OUT  3 rehab_OUT
##  8:    4          1       df 2008-08-13     IN      df_IN       1 df_IN
##  9:    4          1       df 2008-09-20    OUT     df_OUT      1 df_OUT
## 10:    4          2       df 2012-03-18     IN      df_IN       2 df_IN
## 11:    4          2       df 2012-03-19    OUT     df_OUT      2 df_OUT
## 12:    4          3       it 2012-07-02     IN      it_IN       3 it_IN
## 13:    4          3       it 2012-07-20    OUT     it_OUT      3 it_OUT
## 14:    4          3       it 2012-07-20    OUT     it_OUT      3 it_OUT
## 15:    5          1       df 2006-02-09     IN      df_IN       1 df_IN
```

```
## 16:    5          1       df 2006-02-25    OUT     df_OUT     1 df_OUT
## 17:    5          1       df 2008-04-16   DEAD       DEAD         DEAD
## 18:    6          1       df 2009-03-05     IN      df_IN      1 df_IN
## 19:    6          1       df 2009-03-16    OUT     df_OUT     1 df_OUT
## 20:    6          2       df 2009-07-06     IN      df_IN      2 df_IN
## 21:    6          2       df 2009-07-20    OUT     df_OUT     2 df_OUT
## 22:    6          3       df 2010-11-17     IN      df_IN      3 df_IN
## 23:    6          3       df 2010-11-23    OUT     df_OUT     3 df_OUT
## 24:    6          4       df 2010-12-05     IN      df_IN      4 df_IN
## 25:    6          4       df 2010-12-19   DEAD       DEAD         DEAD
##     subj adm_number rehab_it  augmented status status_exp n_status_exp
```

Beside the usual status variables, of which we reported only status, `augment()` computed two more:

- *status_exp*: is the direct expansion of status and the variable you passed to `more_status`, which in this case is *rehab_it*. The function composes them by pasting a '_' in between. This is the main reason why it is worth to build a character variable if you know you need to fuel it in as an indicator of a more complex status structure;
- *n_status_exp*: similar to what has been done before, `augment()` mixes information coming from the current expandend status and the number of admission to give you the time evolution of the process.

1 Graphical Assessment of a Multi-state Model

msmtools has been mainly developed to easily manage and work with longitudinal datasets which need to be restructured in order to get msm to work properly.

However, **msmtools** comes with two more functions which try to address graphically and in a very efficient way the problem of the Goodness-of-Fit (Gof) for a multi-state model. When dealing with this type of models, GoF is always a tough quest. Furthermore, up to now, no formal statistical tests are defined when a multi-state model is computed within an exact time framework.

1.1 Comparing fitted and empirical survival with `survplot()`

One of the most common graphical method to assess whether a multi-state model is behaving the way we expect, is to compare the empirical survival with the fitted one. `survplot()` helps out doing this and few more things. The function is a wrapper of the already known `plot.survfit.msm()` from the package **msm**.

Suppose we ran a multi-state model on dataset hosp with the following code:

```r
hosp_augmented = augment( data = hosp, data_key = subj,
                          n_events = adm_number, pattern = label_2,
                          t_start = dateIN, t_end = dateOUT,
                          t_cens = dateCENS, verbose = FALSE )
```

```
## Warning in augment(data = hosp, data_key = subj, n_events = adm_number, :
## no t_death has been passed. Assuming that dateCENS contains both censoring
## and death time
```

```r
# let's define the initial transition matrix for our model
Qmat = matrix( data = 0, nrow = 3, ncol = 3, byrow = TRUE )
Qmat[ 1, 1:3 ] = 1
Qmat[ 2, 1:3 ] = 1
colnames( Qmat ) = c( 'IN', 'OUT', 'D' )
rownames( Qmat ) = c( 'IN', 'OUT', 'D' )
Qmat
```

```
##     IN OUT D
## IN   1   1 1
## OUT  1   1 1
## D    0   0 0
```

```r
# attaching the msm package and running the model using gender, age, # rehab and it as
library( msm )
msm_model = msm( status_num ~ augmented_int,
                 subject = subj, data = hosp_augmented,
                 covariates = ~ gender + age + rehab + it,
                 exacttimes = TRUE, gen.inits = TRUE,
                 qmatrix = Qmat, method = 'BFGS',
                 control = list( fnscale = 6e+05, trace = 0,
                                 REPORT = 1, maxit = 10000 ) )
```