Merging Files of Time-varying Covariates

Mark Lunt
Arthritis Research UK Epidemiology Unit, University of Manchester,
Stopford Building, Oxford Road, Manchester, M13 9PT, U.K.

Abstract.

This insert describes a new command, `tvc_merge`, which can be used merge two datasets containing variables whose values, for a given individual, change over time. If the time points at which the variables change differ between the two datasets, a simple merge will not be sufficient: each record in the first dataset will need to be split into two records at each timepoint that the variables change in the second dataset, and the appropriate values from the second dataset merged to each of these records. Equally, records from the second dataset may also need to be split at times that the variables in the first dataset change. The use of the `tvc_merge` command is illustrated using an example from a drug surveillance system, in which adverse events need to be matched to the particular drug being taken at the time the event occured.

Keywords: time-varying covariates, merging

1 Introduction

The ado-file `tvc_merge` is used to merge two datasets of time-varying covariates, when the end-points of the time intervals in the two datasets are different (if the time intervals were the same, `merge` could be used). Intervals in the first dataset will be split whenever a change in covariates occurs in the second dataset, and vice versa. This is best understood by means of an example.

2 Example

The example below is based on analyses of the British Society for Rheumatology Biologics Register (BSRBR). This is a register of patients taking biologic therapies for arthritic conditions, and an important purpose of the register is to monitor the therapies for safety. To this end, consultants are sent a questionnaire every six months ascertain any adverse events the patient may have suffered. There may be (and often are) multiple records for each subject, since a subject may have several adverse events. There are a number of different types of adverse events (infections, cancers, death etc.) in which we have an interest.

Since we are interested in whether the rate of adverse events differs between the different biologic therapies in use, the dates on which the subject starts, stops and changes treatment are also recorded. Therefore, the total followup time for an individual can be divided into separate intervals in which they were taking a particular drug. There

may be gaps in this history, in which subjects were not taking any medication.

As an example, consider the data in Figure 1. This shows the experience of 2 subjects in this study. Subject 1 entered the study at time 0, and their last followup was at time 240. They took treatment 1 from time 0 until 120, and treatment 2 from time 180 to 230. They suffered two adverse events, one of type 1 at time 100 and one of type 2 at time 210. Subject 2 entered at time 20 and their last followup was at time 260. They started taking treatment 1 at time 150 and were still taking it at the time of their last followup. They suffered a single adverse event, of type 2, at time 120.

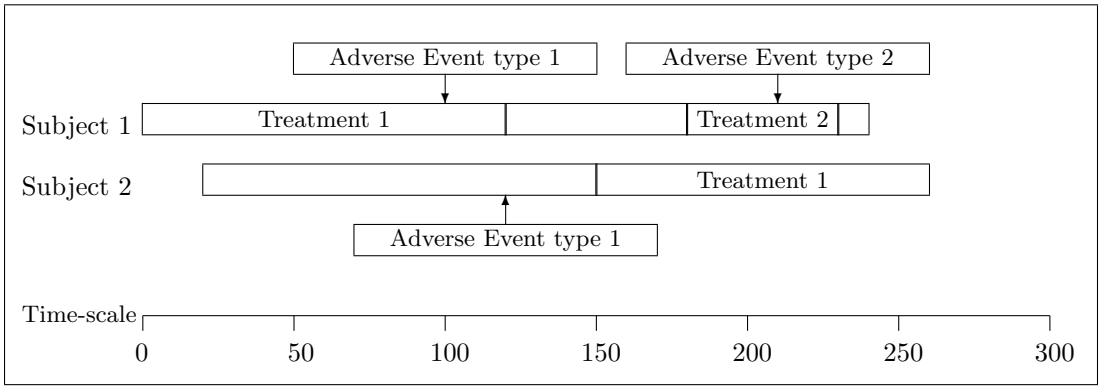

Figure 1: Example data

Given these histories, we wish to create a number of records for each subject, such that each record ends with either an adverse event or a change in treatment. The data should then look like Table 1. This data can then be `stset` and used for survival analysis.

ID	start	stop	treatment	event_type
1	0	100	1	1
1	100	120	1	0
1	120	180	0	0
1	180	210	2	2
1	210	230	2	0
1	230	240	0	0
2	20	120	0	2
2	120	150	0	0
2	150	260	1	0

Table 1: Data in final form for analysis

3 Preparing the data for the `tvc_merge` command

It is fairly straightforward to produce a complete history of treatments, given the start and stop dates of each treatment. For the two subjects in Figure 1, we would have the data in Table 2, stored in a dataset called `treatment.dta`. Note that there is no stop data for subject 2, since they were still on treatment 1 at the end of their followup.

ID	start	stop	treatment
1	0	120	1
1	180	230	2
2	150	.	1

Table 2: Treatment Data

In order to divide the followup period into intervals according to which drug was being taken, we need to know know the date that each subject entered the study, and the date of their last followup. This data is stored in a table called `dates.dta` and would be as listed in Table 3

ID	entry_date	last_fup_date
1	0	240
2	20	260

Table 3: Start and End of Followup

In order to divide the complete followup time into separate intervals in which one treatment was given, we need to

1. Merge `treatment.dta` and `dates.dta`
2. Replace missing stop dates with the date of the last followup (we know that on this date, they had not stopped taking the treatment, but we do not know what happened after this date)
3. If the stop date of one treatment interval is not equal to the start date of the next, add a new interval to cover the time between them, with the treatment set to 0 (For example, for subject 1, treatment 1 stops at time 120, but treatment 2 only began at time 180, so we need an addtional interval between them).
4. If the stop date of the last treatment interval is less than the last followup date, add a new interval from the end of treatment to the end of followup. (E.g. subject 1, time 230–240).
5. If the start date of the first treatment interval is greater than the date of entry to the study, add a new interval. (E.g. subject 2, time 20–150).

The stata code for steps one and two is given below. Note that `tvc_merge` (just like `merge`) cannot work if there is already a variable called `_merge` in either dataset, so this variable is dropped before proceeding.

```
use treatment.dta
sort ID
merge ID using dates.dta
replace stop = last_fup_date if stop == .
drop _merge
```

For step 3, we can use the `expand` function to create a new record if necessary, then replace the variables `start`, `stop` and `treat` in the new record.

```
sort ID start

// if stop for this record is less than start of the next, we need a
// new record to fill the gap between them
by ID: gen to_expand = 1 + (stop < start[_n+1])*(start[_n+1] ~= .)
expand to_expand
sort ID start

// set new_record to 1 for the records we've just added
by ID start: gen new_record = _n == 2

// change the values in the new record to those appropriate for a gap
// in treatment
replace treat = 0 if new_record == 1
replace stop  = start[_n + 1] if new_record == 1
replace start = stop[_n - 1]  if new_record == 1
drop to_expand new_record
```

The code for steps 4 and 5 is almost the same, only the conditions for creating a new record and the replacement values differ.

```
sort ID start

// If the last record ends before the end of followup, add a new record
by ID: gen to_expand = 1 + (stop < last_fup_date)*(_n == _N)
expand to_expand
sort ID start

// set new_record to 1 for the records we've just added
by ID start: gen new_record = _n == 2

// change the values in the new record to those appropriate for a
// period of no treatment at the end of the followup
replace treat = 0 if new_record == 1
replace stop  = last_fup_date if new_record == 1
replace start = stop[_n - 1]  if new_record == 1
drop to_expand new_record



sort ID start

// If the first record begins after the start of followup, add a new
// record to fill the gap at the beginning of followup
by ID: gen to_expand = 1 + (start > entry_date)*(_n == 1)
expand to_expand
sort ID start

// set new_record to 1 for the records we've just added
by ID start: gen new_record = _n == 2

// change the values in the new record to those appropriate for a
// period of no treatment at the end of the followup
replace treat = 0 if new_record == 1
replace start = entry_date if new_record == 1
sort ID start
replace stop  = start[_n + 1] if new_record == 1
drop to_expand new_record

// save the modified file
save treat_expand, replace
```

The treatment data should now look like Table 4.

Next, we prepare the adverse events data. The adverse events data for these two

ID	start	stop	treatment
1	0	120	1
1	120	180	0
1	180	230	2
1	230	240	0
2	20	150	0
2	150	260	1

Table 4: Expanded Treatment Data

subjects are given in Table 5. We record the date of the event and the type of event.

ID	Date	event_type
1	100	1
1	210	2
2	120	2

Table 5: Adverse Events Data

In order to use `tvc_merge`, the adverse events data also needs to be divided into into intervals, with an adverse event at the end of each interval. However, we do not need to ensure that these intervals cover the entire followup period, since we are only concerned with the time the events occur. So we need to:

1. Merge `adverse_events.dta` and `dates.dta`
2. Rename `event_date` as `stop`
3. Create a `start` variable equal to the value of `stop` for the previous interval.
4. Set `start` to `entry_date` for the first interval for each subject.

The stata code to do this is:

```
use adverse_events.dta
sort ID
merge ID using dates
drop _merge
rename date stop

sort ID
by ID: gen start = stop[_n - 1]
by ID: replace start = entry_date if _n == 1

save adverse_expand, replace
```

The adverse events data should now look like Table 6. Both datasets are now in a suitable form to be merged by `tvc_merge`.

ID	start	stop	event_type
1	0	100	1
1	100	210	2
2	20	120	2

Table 6: Adjusted Adverse Events Data

4 Use of the `tvc_merge` command

The syntax for the `tvc_merge` command is

`tvc_merge` *startvar stopvar* `using` *filename*`, id(`*idvar*`)` [`failure(`*failure_vars*`)` `merge(`*merge_var*`)`]

The program needs to know the names of the variables which define the start and end of each interval, and these names need to be the same in both files being merged. Since there will be several records for each subject, it needs to know the variable that identifies each subject, which is passed in the option `id`.

The program also needs to be able to distinguish between variables that take their value througout the interval (in our case the treatment variable) and those representing the events in which we are interested, which only take their value at the end of the interval (in our case `event_type`). A complete list of all such failure-type variables needs to be given to `tvc_merge`, using the option `failure`.

The option `merge` produces a variable which says, in the same way as the variable `_merge` with the `merge` command, whether there was data in the master file, the using file or both for this particular record. However, in this instance, it will not be very informative: any time periods after the last event will be treated as having missing event data. Also, any subjects without events will be treated as missing from the event data file. Since it is far less useful in this case than in the standard `merge` case, this variable has been made optional.

So, for the data we have looked at above, the stata code to merge the two files would be:

```
use treat_expand
tvc_merge start stop using adverse_expand, id(ID) failure(event_type)
sort ID start stop
```

After running this command, the dataset should look like Table 1. The treatment intervals have been split each time an adverse event occured, and the adverse event intervals have been split each time a treatment changed. The data is now in a form

that can be `stset` for analysis, using the command:

```
stset stop, id(ID) fail(event_type) exit(time .) origin(time start)
```

if you are interested in the rate of any adverse events, or

```
stset stop, id(ID) fail(event_type==1) exit(time .) origin(time start)
```

if you are only interested in the rate of adverse events of type 1.

5 Discussion

When dividing the treatment up into intervals, we made the implicit assumption that only one treatment could be in use at a time. This may not be the case: sometimes a new drug will be added to the original drug, rather than replacing it. This situation can be handled by `tvc_merge` in the following way:

1. Create n files of treatment data, each one containing records for treatment with only one of the n treatments.
2. In the ith file, rename `treatment` `treatment`i.
3. Expand each of the n files so that they each span the entire followup period as above.
4. Use `tvc_merge` $n-1$ times to merge each of these n files.

This will result in a dataset of separate intervals spanning the entire followup period, with n variables indicating which treatments, if any, were taken during that interval. Note that this process does not involve the adverse event data, only covariate data: you are not obliged to use the `failure` option.

It is convenient if all of the events occur within the time period for which the treatments are known, but this is not essential. If an event occurs outside the followup time, and extra interval will will be added on, and all of the treatment covariates set to missing in this interval. It will therefore not be possible to include this event in any analysis involving the covariates, but this does mean that no events are thrown away in the merging process, and it is easy to see from the final dataset which events are not included in the analysis and why they are excluded.

6 How the `tvc_merge` command works

The ado file first preserves the current data set, then drops all variables except `ID`, `start` and `stop`. It then reshapes the data so that there are only two variables: `ID` and the date of an interval endpoint. It then repeats this with the using file, and appends

the two lists of dates. This gives a single file containing all of the interval endpoints from both datasets.

By renaming the date variable as `_X_start` and then generating a new variable `_X_stop` as `_X_start[_n+1]`, we now have a file consisting of intervals small enough to lie completely within any interval in either the master file or the using file. The master and using files are then merged with this interval file using `joinby`[1], so that all of the intervals in the original file match up to every interval in the new file. Any observations with `start` > `_X_start` or `stop` < `_X_stop` are dropped, so that only one observation is retained for every interval in the sub-divided intervals file. This ensures that appropriate covariates for the time period from `_X_start` to `_X_stop` are used.

Next, we need to set the variables in the `failure` option to 0 if the end of the interval is not the time that the event took place: i.e if `stop` is not equal to `_X_stop`. Finally, the variables `start` and `stop` are dropped, and `_X_start` and `_X_stop` are renamed `start` and `stop`. This gives a dataset divided as finely as necessary, with the covariates and failure variables taking the correct values for each interval.

7 Acknowledgements

The funding for this work was provided by the Arthritis Research Campaign.

[1] In fact, this is done in two stages: `joinby` is used to merge the interval file with a file consisting only of ID, start and stop date, and the superfluous records dropped from this file. It is then merged with the master file by ID, start and stop, to include all of the information in the master file. Using `joinby` with the entire master file may have been quicker, but it could require many times as much memory as doing it in this two-stage way.