OVERVIEW OF DURMOD

SIMEN GAURE

Abstract. This is a walkthrough of an estimation of a generated dataset with the **durmod** package. Also, various tunable parameters and details are provided.

1. A dataset

The **durmod** package fits a mixed proportional hazard model with competing risks to duration data. The model is the one from [1].

Let's have a look at a generated dataset which simulates an unemployment register with two competing risks.

```
library(durmod)
data(durdata)
head(durdata,15)
```

```
##     id          x1          x2 alpha       d    duration     state
##  1:  1 -0.03751376 -0.29495984     0     job  5.98661231     unemp
##  2:  2 -1.57460441 -0.16972008     0     job 30.41322751     unemp
##  3:  3 -0.48596752  0.29442951     0     job  1.97320112     unemp
##  4:  4  0.46518623  0.74250053     0 program  0.06019839     unemp
##  5:  4  0.46518623  0.74250053     1     job 12.98877267 onprogram
##  6:  5 -0.90409807 -1.70517209     0     job  2.98180289     unemp
##  7:  6 -0.27743280 -0.15416450     0     job  8.31021131     unemp
##  8:  7  0.38643441  0.26033651     0 program  1.12433425     unemp
##  9:  7  0.38643441  0.26033651     1     job  0.27791520 onprogram
## 10:  8 -0.06040412  1.01461940     0     job 11.35695783     unemp
## 11:  9 -0.68617976 -0.37062699     0 program  5.58821616     unemp
## 12:  9 -1.20316486  0.58482910     1    none  4.86126097 onprogram
## 13:  9 -1.59044730  1.23948887     1    none 69.55052288 onprogram
## 14: 10 -1.90613679  0.03444083     0     job  1.84738970     unemp
## 15: 11  1.80375975  0.13039843     0     job  0.49224085     unemp
```

There is an `id` which identifies an individual. The individuals have been through a process. At the outset they are all unemployed, this is recorded by the factor `state`. As unemployed they face two hazards, i.e. probabilites per time unit. Either they can get a job, or they can enter a labour market programme, like a subsidized wage job or similar.

These transitions are recorded in the `d` factor. In our simulation, individuals who transition to `"job"`, exit the dataset. If a transition to labour market programme occurs, the state variable changes to `"onprogram"`, and the dummy `alpha` changes

Date: June 26, 2019.

to 1. It is also possible to do a `"none"` transition, this is typically necessary if a covariate changes, since the model has piecewise constant explanatory covariates. Also, when on a programme, one of the hazards disappear, it is no longer possible to make a transition to a programme, we're already on it.

Each row of the dataset has a `duration`, this is the time until the transition marked in `d` occurs.

In our dataset, we limit the total duration for each individudal to 80. That means that some individuals do not exit the dataset by doing a transition, but with a `d=="none"`.

2. The mixed proportional hazard competing risk model

There are two covariates, `x1` and `x2`. These are assumed to influence the two hazards. We also assume the `alpha` enters the hazard.

We model the baseline hazard for transition to job as,

$$h^j(\mu^j) = \exp(x_1\beta_1^j + x_2\beta_2^j + \alpha\beta_3^j + \mu^j) \tag{1}$$

The hazard for transition to programme is,

$$h^p(\mu^p) = \exp(x_1\beta_1^p + x_2\beta_2^p + \mu^p) \tag{2}$$

Here we have included an "intercept", a μ, it could equally well have been written as a multiplicative factor $\exp(\mu)$ instead. This $\exp(\mu)$-term is the "proportional hazard".

The likelihood for a single observation k consists of two parts. Let $H(\mu) = h^j(\mu^j) + h^p(\mu^p)$ be the sum of the hazards, where μ is the vector (μ^j, μ^p).

For an observation k there is a survival probability/density up until the transition:

$$s_k(\mu) = \exp(-t_k H(\mu)), \tag{3}$$

where t_k is the duration of the period.

If there is a transition, $s(\mu)$ is multiplied by the transition hazard, $h^d(\mu)$, where d is either p or j. If there is no transition, $h^d(\mu)$ is taken to be 1. Taken together, all the observations for an individual i yields an individual likelihood. We call it $\ell_i(\mu)$.

$$\ell_i(\mu) = \prod_{k\in K_i} h^{d_k}(\mu)s_k(\mu), \tag{4}$$

where K_i is the set of observations for individual i.

However, there is also a mixture part, designed to account for unobserved individual heterogeneity. The μ-vector is stochastic with a discrete distribution. That is, there is an n, a set of probabilites p_j, and vectors μ_j, for $j = 1..n$. Of course, we have $\sum_{j=1}^n p_j = 1$.

The mixture likelihood for an individual i is $L_i = \sum_j p_j \ell_i(\mu_j)$.

The log-likelihood for the dataset is thus, $L = \sum_i \log(L_i)$.

The L must be maximized with respect to the five βs, the n, the probabilites p_j, and the vectors μ_j for $j = 1..n$.

3. Estimation

The estimation proceeds as follows. We start with $n = 1$, estimate the βs and the two μs. Then we increase n to 2, let p_2 be a small probability, and find a vector μ_2 which increases the likelihood. This is used as starting point for a new likelihood maximization. Then n is increased to 3, and we proceed in this fashion, adding masspoints to the distribution until we are no longer able to increase the likelihood.

In **durmod** we use the `mphcrm` function for this purpose. Here is an example. First we create a "riskset", a specification of which hazards are experienced in various states:

```
risksets <- list(unemp=c('job','program'), onprogram='job')
```

Note that the names of the list `risksets` are the same as the levels in the factor `state`. And that the entries in the list are levels of the factor `d`, i.e. possible transitions.

Then we create a set of control parameters. Since this vignette is to be created by the busy CRAN repository, we limit ourselves to 6 iterations, i.e. no more than 6 masspoints in the distribution. For the same reason we also limit to 2 cpus, or threads, in the computation. The default is to use all the available cpus/cores.

```
ctrl <- mphcrm.control(iters=6, threads=2)
```

Then we are ready to estimate. There are a couple of special terms in the formula we use:

```
set.seed(42) # for reproducibility
opt <- mphcrm(d ~ x1 + x2 +
                C(job, alpha) + ID(id) + D(duration) + S(state),
              data=durdata, risksets=risksets, control=ctrl)
## mphcrm 11:59:37 i:1 pts:1 L:-24065.4040 g:0.0007 mp:1 rcond:0.0038 e:-0.0000 t:0.3s
## mphcrm 11:59:38 i:2 pts:2 L:-23106.8269 g:3.37e-05 mp:0.29108 rcond:0.009 e:0.6031 t:0.6s
## mphcrm 11:59:39 i:3 pts:3 L:-22989.6996 g:2.12e-05 mp:0.056365 rcond:0.0074 e:0.8709 t:1.9
## mphcrm 11:59:40 i:4 pts:4 L:-22965.1207 g:0.000331 mp:0.034185 rcond:0.0015 e:1.0192 t:1.0
## mphcrm 11:59:42 i:5 pts:5 L:-22950.4433 g:3.9e-05 mp:0.025213 rcond:0.0011 e:1.1776 t:1.6s
## mphcrm 11:59:45 i:6 pts:6 L:-22942.5330 g:5.16e-05 mp:0.020932 rcond:0.00061 e:1.5231 t:3.
```

The left hand side of the formula, `d`, is the outcome, the transition that is taken. The `C(job, alpha)` term is a list of conditional covariates, the `alpha` should only explain the `"job"` transition. The `ID(id)` specifies that the covariate `id` identifies individuals. The `D(duration)` specifies that the covariate `duration` contains the durations of the observations. Finally, the `S(state)` term specifies that the covariate `state` is a factor which indexes into the `risksets` argument.

`mphcrm` writes diagnostic output, one line per iteration. It contains potentially useful information. There is a time stamp, the iteration number, the number of masspoints, the resulting log likelihood, the 2-norm of the gradient, the smallest probability in the masspoint distribution, the reciprocal condition number of the Fisher matrix, the entropy of the masspoint distribution, and the time used in the iteration.

`mphcrm` returns a list with one entry for each iteration, it has a special print method which sums up the estimation in reverse order:

```
print(opt)
## iter6: estimate with 6 points, log-likelihood: -22942.5330
##
##     job.x1     job.x2  job.alpha program.x1 program.x2
##  0.8872396 -0.8954421  0.1546196  0.8964912  0.4371326
##
## Proportional hazard distribution
##                prob        job    program
## point  1 0.32694721 0.09257757 0.10895815
## point  2 0.25533364 0.02815197 0.02132034
## point  3 0.22461763 0.13030036 0.01780745
## point  4 0.13502572 0.42282266 0.19405495
## point  5 0.03714412 0.00785169 0.07728948
## point  6 0.02093168 0.00336387 0.00165849
##
## iter5: estimate with 5 points, log-likelihood: -22950.4433
## iter4: estimate with 4 points, log-likelihood: -22965.1207
## iter3: estimate with 3 points, log-likelihood: -22989.6996
## iter2: estimate with 2 points, log-likelihood: -23106.8269
## iter1: estimate with 1 points, log-likelihood: -24065.4040
## nullmodel: estimate with 1 points, log-likelihood: -27087.0860
```

Unless something has gone wrong, you will normally be interested in the first entry, the one with the largest likelihood. We can look at a summary:

```
best <- opt[[1]]
summary(best)
## $loglik
## [1] -22942.53
##
## $coefs
##                 value         se          t      Pr(>|t|)
## job.x1      0.8872396 0.02038280  43.528828  0.000000e+00
## job.x2     -0.8954421 0.02404830 -37.235148 3.562522e-281
## job.alpha   0.1546196 0.06653970   2.323719  2.016409e-02
## program.x1  0.8964912 0.02868375  31.254321 4.834826e-203
## program.x2  0.4371326 0.03219168  13.579056  1.449701e-41
##
## $moments
##               mean    variance         sd
## job     0.12417787 0.015540243 0.12466051
## program 0.07417521 0.003873459 0.06223712
```

It has three entries, `"loglik"`, which is simply the log likelihood, `"coefs"` which is the values and standard errors of the estimated coefficients. And `"moments"`, which is the first and second moments of the proportional hazard distribution.

If we had limited to 5 masspoints, the `alpha` estimate would be negative, i.e. the labour market programme has a negative effect on the hazard for getting a job. The

reader may run the example with a maximum of 15 iterations to obtain a positive effect.

We can see how the `alpha` changes with more points:

```
t(sapply(opt, function(o) summary(o)$coefs["job.alpha",]))
##                 value         se          t     Pr(>|t|)
## iter6      0.15461959 0.06653970  2.3237194 0.0201640929
## iter5     -0.02101247 0.04948837 -0.4245940 0.6711434319
## iter4     -0.02428244 0.04688692 -0.5178936 0.6045460542
## iter3     -0.13523891 0.04077778 -3.3164856 0.0009154114
## iter2     -0.11946379 0.03658254 -3.2655963 0.0010966822
## iter1     -0.07797898 0.02361968 -3.3014414 0.0009658768
## nullmodel  0.00000000         NA         NA           NA
```

4. More options

4.1. **Interval timing.** The example above had exactly recorded time. For some applications we do have that, while in other applications we only have a time interval when the transition is known to have taken place. The data above is actually a prime example, perhaps we only have labour data on a monthly basis. When a transition takes place, it is only registered at the end of the month, and there is no record of the day. In this case, the `duration` would be 1 for every observation, and one should use the `timing="interval"` argument in `mphcrm`. The observation likelihood is replaced by,

$$h^{d_k}(\mu)\exp(-t_k H(\mu))\frac{1-\exp(-t_k H(\mu))}{H(\mu)}. \tag{5}$$

It is the fractional part which distinguishes it from the exact model.

If the hazards are small and we use unit intervals, the difference between the interval and exact model is quite small, so one may opt for using the exact model instead.

4.2. **No timing.** In some applications there isn't any time. A transition occurs, or not. In this case the specification `timing="none"` can be used. It will use a logit model for the transition probabilities.

4.3. **Factors.** `mphcrm` treats factors specially. There is, I think, nothing special to see, but internally `mphcrm` does not create a large model matrix filled with dummy variables. This means that factors with many levels is quite fast to estimate.

5. Control parameters

There are many control parameters. Here are some you may want to tinker with.

- `threads`. An integer. The number of parallel threads used by `mphcrm`. The default is taken from `getOption("durmod.threads")`, which is initialized from the environment variable `DURMOD_THREADS`, `OMP_NUM_THREADS`, `OMP_THREAD_LIMIT`, `NUMBER_OF_PROCESSORS`, or else from `parallel::detectCores()`.

  It is not always true that the estimation runs twice as fast on twice as many cpus. This depends on the cpu- and memory architecture of your computer, as well as on the implementation of OpenMP in the compiler used to compile the C++ parts of **durmod**. Besides, not all parts of **durmod** run

in parallel, so by Amdahl's law you may not expect linear speedup when the number of cpus tends to infinity.

Also, if you intend to use your computer for something else while `mphcrm` runs, you should not give it all your cpus, but save one or two for your other work. If one of the 16 threads in `mphcrm` shares a cpu with your mail program trying to sort your inbox, the speed may be halved.

- `iters.` An integer. The number of iterations to perform. The estimation may stop earlier, if neither the log likelihood *nor* the entropy improves. The default is 15.
- `ll.improve.` A numeric. The amount the log-likelihood must increase with to be considered an improvement. The default is 0.001.
- `e.improve` A numeric. The amount the entropy of the hazard distribution must increase with to be considered an improvement. The default is 0.001.
- `callback.` A function. If the one-line diagnostic from `mphcrm` is insufficient, it is possible to write your own. It will replace the default callback (which you can call from your function). In this way you can e.g. get diagnostics on particular coefficients, save intermediate results to file, or other partakings. See `mphcrm.callback`.
- `jobname.` A character string. The initial portion of the one-line diagnostic. Useful if you e.g. use `parallel::mclapply` to run several estimations in parallel. They can have individual names so you can see the progress. The default is `"mphcrm"`.
- `method.` A character string. Either `"BFGS"` (the default), or `"L-BFGS-B"`. The latter saves some memory, if that is a problem with estimations with a very large number of coefficients.
- `trap.interrupt.` A logical. If you decide to interrupt an interactive estimation before it has terminated, either because you don't want to wait, or because it seems to have run astray, the default behaviour for `mphcrm` is to catch the interrupt, and return gracefully with the result of the estimation so far. This behaviour can be switched off with `trap.interrupt=FALSE`.
- `cluster.` Cluster specification from package **parallel** or **snow**. In addition to utilizing all the cores/cpus on a computer, `mphcrm` may also spread across several computers. It supports running on a cluster from package `parallel` or `snow`. The dataset will be split among the cluster nodes, with approximately equally many observations on each. The nodes will then do their share of the likelihood computations. If using a cluster, the `threads` parameter will be the number of cpus used on each cluster node. In general, when using parallelization, one should make sure that the cpus are not overbooked and that the nodes you are running on are approximately equally fast.

References

1. Simen Gaure, Knut Røed, and Tao Zhang, *Time and causality: A monte carlo assessment of the timing-of-events approach*, Journal of Econometrics **141** (2007), no. 2, 1159 – 1195.

Ragnar Frisch Centre for Economic Research, Oslo, Norway