Distributed Statistical Modeling

Balasubramanian Narasimhan, Daniel L. Rubin, Sam M. Gross, Marina Bendersky, Philip W. Lavori

2020-01-21

Introduction

We demonstrate the possibility of fitting statistical models stratified by sites in a manner that brings computation to the data that may be distributed across sites or more generally, partitioned in some manner. (For simplicity, we will call these partitions, sites.) The infrastructure consists of a single master process that issues queries to worker processes running at each of the sites. A query is merely a function call, more specifically a request to each site to evaluate a pre-defined function $f(\beta)$ on the data at that site, for a given value of parameters β. The master process uses these queries to aggregate and execute an optimization algorithm resulting in a model fit, the results of which should be *indistinguishable from* those that might be obtained if all the data had been in a single place. Of course, this assumes a lossless serialization format, like Google Protocol Buffers for example, but we make do with JSON for now. Also, in comparisons below, we don't use exactly the same iterations as the survival package and so minor differences will be seen.

The advantages are many, chief among them the fact that no raw data needs to be shared between sites. The modeling entity, however, can make an unlimited number of queries of the sites, where each query is a request to compute a model-specific function for a specified value of parameters. This may pose a security concern that we ignore for now. However, it leads to some further interesting questions regarding what may be learned such computation.

Setup

It must be noted that for users to be able to `knit` this document, or to run these examples in an R session, an `opencpu` server must be running with appropriate settings. On MacOS and Unix, this is done by designating an empty directory as workspace and adding the following lines to the `${HOME}/.Rprofile`.

```r
library(distcomp)
distcompSetup(workspace = "full_path_to_workspace_directory",
              ssl_verifyhost = 0L, ssl_verifypeer = 0L)
```

On windows, the same should be done in the `RHOME\etc\Rprofile.site` file.

Note: On Yosemite (MacOS 10.10.4 and below), we have found that references to `localhost` sometimes fail in the opencpu URL; rather the explicit IP address `127.0.0.1` is needed.

In what follows, we assume that such initialization profile has been done. Furthermore, we assume that the `opencpu` server has been started in the *same session* as the one where this markdown document is being knitted via `library(opencpu)`. This is merely a convenience that allows us to refer to the `opencpu` server URL programmatically via `opencpu$url()`; otherwise, alternative means would have to be found to refer to the `opencpu` URL in a permanent manner. For example, if `opencpu` is started in another terminal thus:

```r
library(opencpu)
ocpu_server_start()
```

one could note down the URL that it displays after starting, and use it in the following function definition, as we have done.

```
opencpu <- list(url = function() "http://localhost:5656/ocpu")
```

to ensure that the expression `opencpu$url()` returns the appropriate URL.

To summarize: assuming that an `opencpu` server has been started via `library(opencpu)` with a proper R profile and the expression `opencpu$url()` has been set up appropriately, one can proceed to knit this document in that R session.

A simple example

We take a simple example from the `survival` package, the `ovarian` dataset.

```
library(knitr)
library(survival)
data(ovarian)
str(ovarian)
```

```
## 'data.frame':    26 obs. of  6 variables:
##  $ futime  : num  59 115 156 421 431 448 464 475 477 563 ...
##  $ fustat  : num  1 1 1 0 1 0 1 1 0 1 ...
##  $ age     : num  72.3 74.5 66.5 53.4 50.3 ...
##  $ resid.ds: num  2 2 2 2 2 1 2 2 2 1 ...
##  $ rx      : num  1 1 1 2 1 1 2 2 1 2 ...
##  $ ecog.ps : num  1 1 2 1 1 2 2 2 1 2 ...
```

```
kable(ovarian)
```

futime	fustat	age	resid.ds	rx	ecog.ps
59	1	72.3315	2	1	1
115	1	74.4932	2	1	1
156	1	66.4658	2	1	2
421	0	53.3644	2	2	1
431	1	50.3397	2	1	1
448	0	56.4301	1	1	2
464	1	56.9370	2	2	2
475	1	59.8548	2	2	2
477	0	64.1753	2	1	1
563	1	55.1781	1	2	2
638	1	56.7562	1	1	2
744	0	50.1096	1	2	1
769	0	59.6301	2	2	2
770	0	57.0521	2	2	1
803	0	39.2712	1	1	1
855	0	43.1233	1	1	2
1040	0	38.8932	2	1	2
1106	0	44.6000	1	1	1
1129	0	53.9068	1	2	1
1206	0	44.2055	2	2	1
1227	0	59.5890	1	2	2
268	1	74.5041	2	1	2
329	1	43.1370	2	1	1
353	1	63.2192	1	2	2
365	1	64.4247	2	2	1
377	0	58.3096	1	2	1

A Cox fit on the aggregated data

A simple Cox model fit estimates the effect of age on survival, stratified by drug.

```r
cp <- coxph(Surv(futime, fustat) ~ age + strata(rx), data = ovarian, ties = "breslow")
print(cp)
```

```
## Call:
## coxph(formula = Surv(futime, fustat) ~ age + strata(rx), data = ovarian,
##     ties = "breslow")
##
##        coef exp(coef) se(coef)     z       p
## age 0.13735   1.14723  0.04741 2.897 0.00376
##
## Likelihood ratio test=12.69  on 1 df, p=0.0003678
## n= 26, number of events= 12
```

The above shows the intial and final log likelihood values at 0 and the estimated coefficient respectively and the actual estimated coefficient in the last line.

For our setting, we can pretend that this data set is actually from two sites, one containing the control or placebo group (`rx = 1`)and the other containing the drug group (`rx = 2`).

The model definition

We first need to define the computation. The available computations can be listed:

```r
print(availableComputations())
```

```
## $QueryCount
## $QueryCount$desc
## [1] "Distributed Query Count"
##
## $QueryCount$definitionApp
## [1] "defineNewQueryCountModel"
##
## $QueryCount$setupWorkerApp
## [1] "setupQueryCountWorker"
##
## $QueryCount$setupMasterApp
## [1] "setupQueryCountMaster"
##
## $QueryCount$makeDefinition
## function ()
## {
##     data.frame(id = getComputationInfo("id"), compType = getComputationInfo("compType"),
##         projectName = getComputationInfo("projectName"), projectDesc = getComputationInfo("projectDe
##         he = getComputationInfo("he"), filterCondition = getComputationInfo("filterCondition"),
##         stringsAsFactors = FALSE)
## }
## <bytecode: 0x7ff55bbbcf58>
## <environment: 0x7ff558a38628>
##
## $QueryCount$makeMaster
## function (defn, partyNumber, debug = FALSE)
## {
##     if (!is.null(defn$he) && defn$he) {
```

```
##         HEQueryCountMaster$new(defn = defn, partyNumber = partyNumber,
##             debug = debug)
##     }
##     else {
##         QueryCountMaster$new(defn = defn, debug = debug)
##     }
## }
## <bytecode: 0x7ff55bbbc6d0>
## <environment: 0x7ff558a38628>
##
## $QueryCount$makeWorker
## function (defn, data, pubkey_bits = NULL, pubkey_n = NULL, den_bits = NULL)
## {
##     if (defn$he) {
##         HEQueryCountWorker$new(defn = defn, data = data, pubkey_bits = pubkey_bits,
##             pubkey_n = pubkey_n, den_bits = den_bits)
##     }
##     else {
##         QueryCountWorker$new(defn = defn, data = data)
##     }
## }
## <bytecode: 0x7ff55bbbfa48>
## <environment: 0x7ff558a38628>
##
##
## $StratifiedCoxModel
## $StratifiedCoxModel$desc
## [1] "Stratified Cox Model"
##
## $StratifiedCoxModel$definitionApp
## [1] "defineNewCoxModel"
##
## $StratifiedCoxModel$setupWorkerApp
## [1] "setupCoxWorker"
##
## $StratifiedCoxModel$setupMasterApp
## [1] "setupCoxMaster"
##
## $StratifiedCoxModel$makeDefinition
## function ()
## {
##     data.frame(id = getComputationInfo("id"), compType = getComputationInfo("compType"),
##         projectName = getComputationInfo("projectName"), projectDesc = getComputationInfo("projectDe
##         he = getComputationInfo("he"), formula = getComputationInfo("formula"),
##         stringsAsFactors = FALSE)
## }
## <bytecode: 0x7ff55bbbf0a8>
## <environment: 0x7ff558a38628>
##
## $StratifiedCoxModel$makeMaster
## function (defn, partyNumber, debug = FALSE)
## {
##     if (!is.null(defn$he) && defn$he) {
##         stop("Not implemented")
##     }
##     else {
##         CoxMaster$new(defn = defn, debug = debug)
##     }
## }
## <bytecode: 0x7ff55bbbe820>
## <environment: 0x7ff558a38628>
##
## $StratifiedCoxModel$makeWorker
## function (defn, data, pubkey_bits = NULL, pubkey_n = NULL, den_bits = NULL)
## {
##     if (!is.null(defn$he) && defn$he) {
##         stop("Not implemented")
##     }
##     else {
##         CoxWorker$new(defn = defn, data = data)
##     }
## }
## <bytecode: 0x7ff55bbc3e00>
## <environment: 0x7ff558a38628>
##
##
## $RankKSVD
## $RankKSVD$desc
## [1] "Rank K SVD"
##
## $RankKSVD$definitionApp
## [1] "defineNewSVDModel"
##
## $RankKSVD$setupWorkerApp
## [1] "setupSVDWorker"
##
## $RankKSVD$setupMasterApp
## [1] "setupSVDMaster"
##
## $RankKSVD$makeDefinition
## function ()
## {
##     data.frame(id = getComputationInfo("id"), compType = getComputationInfo("compType"),
##         projectName = getComputationInfo("projectName"), projectDesc = getComputationInfo("projectDe
##         he = getComputationInfo("he"), rank = getComputationInfo("rank"),
##         ncol = getComputationInfo("ncol"), stringsAsFactors = FALSE)
## }
## <bytecode: 0x7ff55bbc35e8>
## <environment: 0x7ff558a38628>
##
## $RankKSVD$makeMaster
## function (defn, partyNumber, debug = FALSE)
## {
##     if (!is.null(defn$he) && defn$he) {
##         stop("Not implemented")
##     }
##     else {
##         SVDMaster$new(defn = defn, debug = debug)
##     }
## }
## <bytecode: 0x7ff55bbc2c48>
## <environment: 0x7ff558a38628>
##
## $RankKSVD$makeWorker
## function (defn, data, pubkey_bits = NULL, pubkey_n = NULL, den_bits = NULL)
## {
##     if (!is.null(defn$he) && defn$he) {
##         stop("Not implemented")
##     }
##     else {
##         SVDWorker$new(defn = defn, data = data)
##     }
## }
## <bytecode: 0x7ff55bbc2318>
## <environment: 0x7ff558a38628>
```

So, we can define the ovarian data computation as follows.

```
ovarianDef <- data.frame(compType = names(availableComputations())[2],
                         formula = "Surv(futime, fustat) ~ age",
                         he = FALSE,
                         id = "Ovarian", stringsAsFactors=FALSE)
```

The data for each site

We split the `ovarian` data into two sites as indicated earlier.

```
siteData <- with(ovarian, split(x = ovarian, f = rx))
```

Setting up the sites

We can now set up each site with its own data. A site is merely a list of two items, a (unique) `name` and an `opencpu` URL.

```
nSites <- length(siteData)
sites <- lapply(seq.int(nSites), function(i) list(name = paste0("site", i),
                                                   url = opencpu$url()))
```

By default, on each site, data for a computation is stored under the name `data.rds` and the definition itself is stored under the name `defn.rds`. If the sites are physically separate, then everything proceeds smoothly. However, here, in our case, we are using the same `opencpu` server for simulating both sites. We therefore have to save the files under different names, just for this experiment, say `site1.rds` and `site2.rds` for this example. This is all taken care of by the code which checks to see if the `opencpu` URLs refer to local hosts or not. (In fact, as this code is executing, one can examine the contents of the workspace to see what is happening) We now `Map` the upload function to each site so that the computation becomes well-defined.

```
ok <- Map(uploadNewComputation, sites,
          lapply(seq.int(nSites), function(i) ovarianDef),
          siteData)

stopifnot(all(as.logical(ok)))
```

Reproducing original aggregated analysis in a distributed fashion

We are now ready to reproduce the original aggregated analysis. We first create a master object using the same definition.

```r
master <- CoxMaster$new(defn = ovarianDef)
```

We then add the worker sites specifying a name and a URL for each. names.

```r
for (i in seq.int(nSites)) {
    master$addSite(name = sites[[i]]$name, url = sites[[i]]$url)
}
```

And we now maximize the partial likelihood, by calling the `run` method of the master.

```r
result <- master$run()
```

We then print the summary.

```r
master$summary()
```

```
##        coef exp(coef)  se(coef)       z           p
## 1 0.1373399  1.147218 0.0473947 2.89779 0.003758017
```

As we can see, the results we get from the distributed analysis are the same as we got for the original aggregated analysis. We print them separately here for comparison.

A larger example

We turn to a larger the example from Therneau and Grambsch using the `pbc` data where the stratifying variable is `ascites`.

The aggregated fit

```r
data(pbc)
pbcCox <- coxph(Surv(time, status==2) ~ age + edema + log(bili) +
                  log(protime) + log(albumin) + strata(ascites), data = pbc,
                ties = "breslow")
print(pbcCox)
```

```
## Call:
## coxph(formula = Surv(time, status == 2) ~ age + edema + log(bili) +
##     log(protime) + log(albumin) + strata(ascites), data = pbc,
##     ties = "breslow")
##
##                   coef exp(coef)  se(coef)      z        p
## age           0.031351  1.031848  0.009075  3.455 0.000551
## edema         0.599345  1.820926  0.321269  1.866 0.062103
## log(bili)     0.866262  2.378005  0.100658  8.606  < 2e-16
## log(protime)  3.034061 20.781462  1.038838  2.921 0.003493
## log(albumin) -2.966183  0.051499  0.781774 -3.794 0.000148
##
## Likelihood ratio test=145.9  on 5 df, p=< 2.2e-16
## n= 312, number of events= 125
##    (106 observations deleted due to missingness)
```

The distributed fit

We split the data using `ascites` and proceed the usual way as shown above

```
pbcDef <- data.frame(compType = names(availableComputations())[2],
                     he = FALSE,
                     formula = paste("Surv(time, status==2) ~ age + edema +",
                       "log(bili) + log(protime) + log(albumin)"),
                     id = "pbc", stringsAsFactors = FALSE)
siteData <- with(pbc, split(x = pbc, f = ascites))
nSites <- length(siteData)
sites <- lapply(seq.int(nSites), function(i) list(name = paste0("site", i),
                                                     url = opencpu$url()))
ok <- Map(uploadNewComputation, sites,
          lapply(seq.int(nSites), function(i) pbcDef),
          siteData)
stopifnot(all(as.logical(ok)))
master <- CoxMaster$new(defn = pbcDef)
for (site in sites) {
    master$addSite(site$name, site$url)
}
```

```
result <- master$run()
```

We then print the summary.

```
kable(master$summary())
```

coef	exp(coef)	se(coef)	z	p
0.0310247	1.0315110	0.0090692	3.420887	0.0006242
0.6019922	1.8257525	0.3205949	1.877735	0.0604174
0.8682667	2.3827773	0.1006068	8.630302	0.0000000
3.0276949	20.6495786	1.0393738	2.912999	0.0035798
-2.9765945	0.0509661	0.7809580	-3.811465	0.0001381

The results should be comparable to the aggregated fit above.

Bone Marrow Transplant Example

This uses the `bmt` data from Klein and Moschberger. Some variable renaming, first.

```
if (!require("KMsurv")) {
  stop("Please install the KMsurv package before proceeding")
}
```

```
##
## BMT data
##

library(KMsurv)
data(bmt)
bmt$tnodis <- bmt$t2 ## time to disease relapse/death
bmt$inodis <- bmt$d3 ## disease relapse/death indicator
bmt$tplate <- bmt$tp ## time to platelet recovery
bmt$iplate <- bmt$dp ## platelet recovery
bmt$agep <- bmt$z1 ## age of patient in years
bmt$aged <- bmt$z2 ## age of donor in years
bmt$fab <- bmt$z8 ## fab grade 4 or 5 + AML
```

```r
bmt$imtx <- bmt$z10 ## MTX used
bmt <- bmt[order(bmt$tnodis), ] ## order by time to disease relapse/death
bmt <- cbind(1:nrow(bmt)[1], bmt)
names(bmt)[1] <- "id"

##
#####
##
bmt$agep.c <- bmt$agep - 28
bmt$aged.c <- bmt$aged - 28

bmt$imtx <- factor(bmt$imtx)
```

The aggregated fit

```r
bmt.cph <- coxph(formula = Surv(tnodis, inodis) ~ fab + agep.c * aged.c +
                 factor(group) + strata(imtx), data = bmt, ties = "breslow")

print(bmt.cph)
```

```
## Call:
## coxph(formula = Surv(tnodis, inodis) ~ fab + agep.c * aged.c +
##     factor(group) + strata(imtx), data = bmt, ties = "breslow")
##
##                       coef  exp(coef)   se(coef)      z       p
## fab              0.9078000  2.4788630  0.2789888  3.254 0.00114
## agep.c           0.0055094  1.0055246  0.0199670  0.276 0.78261
## aged.c          -0.0016373  0.9983640  0.0181633 -0.090 0.92817
## factor(group)2 -1.0338861  0.3556223  0.3647168 -2.835 0.00459
## factor(group)3 -0.3390880  0.7124198  0.3678433 -0.922 0.35662
## agep.c:aged.c    0.0028450  1.0028491  0.0009499  2.995 0.00274
##
## Likelihood ratio test=31.03  on 6 df, p=2.501e-05
## n= 137, number of events= 83
```

The distributed fit

We'll use `imtx` for splitting data into sites.

```r
bmtDef <- data.frame(compType = names(availableComputations())[2],
                     he = FALSE,
                     formula = paste("Surv(tnodis, inodis) ~ fab +",
                       "agep.c * aged.c + factor(group)"),
                     id = "bmt", stringsAsFactors = FALSE)
siteData <- with(bmt, split(x = bmt, f = imtx))
nSites <- length(siteData)
sites <- lapply(seq.int(nSites), function(i) list(name = paste0("site", i),
                                                  url = opencpu$url()))
ok <- Map(uploadNewComputation, sites,
          lapply(seq.int(nSites), function(i) bmtDef),
          siteData)

stopifnot(all(as.logical(ok)))
master <- CoxMaster$new(defn = bmtDef)
```

```r
for (site in sites) {
    master$addSite(site$name, site$url)
}
```

```r
result <- master$run()
```

We then print the summary.

```r
kable(master$summary())
```

coef	exp(coef)	se(coef)	z	p
0.9083860	2.4803160	0.2789473	3.2564784	0.0011280
0.0054343	1.0054490	0.0199716	0.2720993	0.7855457
-0.0017046	0.9982969	0.0181680	-0.0938216	0.9252508
-1.0338625	0.3556307	0.3648730	-2.8334862	0.0046043
-0.3375983	0.7134818	0.3680206	-0.9173355	0.3589669
0.0028547	1.0028588	0.0009480	3.0111928	0.0026022

Byar and Greene Prostate Cancer Data Example

This example is the largest of them all and also has four strata rather than 2.

```r
prostate <- readRDS("prostate.RDS")
```

The aggregated fit

```r
pcph <- coxph(Surv(dtime, status) ~ stage + strata(rx) + age + wt + pf + hx +
                  sbp + dbp + ekg + hg + sz + sg + ap + bm, data = prostate)
print(pcph)
```

```
## Call:
## coxph(formula = Surv(dtime, status) ~ stage + strata(rx) + age +
##     wt + pf + hx + sbp + dbp + ekg + hg + sz + sg + ap + bm,
##     data = prostate)
##
##                           coef  exp(coef)   se(coef)      z        p
## stageIV             -0.1775897  0.8372859  0.1759230 -1.009 0.312747
## age                  0.0242084  1.0245038  0.0091574  2.644 0.008203
## wt                  -0.0102559  0.9897965  0.0047551 -2.157 0.031020
## pfBedridden(<50%)   -1.3727528  0.2534084  0.8511503 -1.613 0.106783
## pfBedridden(>50%)   -1.1749372  0.3088384  0.8503538 -1.382 0.167063
## pfnormal            -1.5886734  0.2041963  0.8255417 -1.924 0.054304
## hx                   0.5104190  1.6659890  0.1203711  4.240 2.23e-05
## sbp                 -0.0330859  0.9674555  0.0292787 -1.130 0.258463
## dbp                  0.0494337  1.0506759  0.0477904  1.034 0.300956
## ekgblock/conduction -0.1459025  0.8642420  0.3823871 -0.382 0.702790
## ekgheart strain      0.3931730  1.4816747  0.2825381  1.392 0.164051
## ekgnormal           -0.0579591  0.9436886  0.2818278 -0.206 0.837061
## ekgold MI            0.0074423  1.0074700  0.3010725  0.025 0.980279
## ekgrecent MI         0.8189465  2.2681091  1.0555450  0.776 0.437836
## ekgrhythmic disturb  0.2803126  1.3235435  0.3077139  0.911 0.362321
## hg                  -0.0689290  0.9333930  0.0319957 -2.154 0.031215
## sz                   0.0178037  1.0179631  0.0046075  3.864 0.000112
## sg                   0.1002007  1.1053927  0.0416067  2.408 0.016028
## ap                   -0.0013841  0.9986169  0.0009971 -1.388 0.165098
## bm                    0.3196404  1.3766326  0.1813332  1.763 0.077947
##
## Likelihood ratio test=99.43  on 20 df, p=1.592e-12
## n= 475, number of events= 338
##    (27 observations deleted due to missingness)
```

The distributed fit

The distributed fit for this particular example doesn't work in the current implementation. This is because the X matrix for each site is singular. The math holds, obviously, but the current implementation is based on re-using as much of the `survival` package as possible. We have to work harder to implement the distributed computation in the situation where X is singular at at least one site. This affects the computation of the variance (or equivalently, the information matrix). Some work needs to be done to work around this and figure out how best to reuse what's already in the `survival` package.

However, in order to demonstrate that the distributed fit really works, we show below an alternative implementation that yields the same result as the aggregated one.

An object representing the sites

Here's a reference object for each site. It has several fields: a `data` field containing the data, a `formula` field (as used in the well-known `survival` R package) describing the model being fit. These two are the only ones needed for initializing a site. Other fields that are generated based on these two fields are `modelDataFrame` containing the actual model data used for fitting the model and a `modelMatrix`.

```r
site <- setRefClass("siteObject",
                    fields = list(
                        data = "data.frame",
                        formula = "formula",
                        modelDataFrame = "data.frame",
                        coxControl = "list",
                        modelMatrix = "matrix"),
                    methods = list(
                        initialize = function(formula, data) {
                            'Initialize the object with a formula and dataset'
                            formula <<- formula
                            data <<- data
                            temp <- coxph.control()
                            temp$iter.max <- 0
                            coxControl <<- temp
                            stopifnot(kosher())
                        },
                        kosher = function() {
                            'Check that the class data passes sanity checks'
                            modelDataFrame <<- model.frame(formula, data = data)
                            lhs <- modelDataFrame[, 1]
                            ordering <- order(lhs[, 1])
                            modelDataFrame <<- modelDataFrame[ordering, ]
                            data <<- data[ordering, ]
                            modelMatrix <<- model.matrix(formula, data = modelDataFrame)
                            TRUE
                        },
                        dimP = function() {
                            'Return the number of covariates'
                            ncol(modelMatrix) - 1
```

```
                          })
                      )
site$accessors(c("data", "formula", "modelDataFrame", "modelMatrix"))
```

The `initialize` method above mostly sets the fields, generates values for other fields, and does a mild sanity check. It will not proceed further if the function `kosher` returns false. For now the `kosher` function merely orders the data frame by follow-up time, but in a production system a number of other checks might be performed, such as ensuring all named variables are available at the site.

The method `dimP` merely returns the number of columns of the model matrix.

The (partial) log likelihood function for each site

For our example, the (partial) log likelihood (named `localLogLik`) is simple and can be computed directly. Assuming failure times t_i and event indicators δ_i, it is precisely:

$$l(\beta) = \sum_{i=1}^{n} \delta_i \left[z_i\beta - \log\Big(\sum_{j \in R(t_i)} \exp(z_j\beta) \Big) \right]$$

where β is the vector of parameters, z_i is row i of the model matrix (covariates for subject i) and $R(t_i)$ is the risk set at time t_i.

The first derivative with respect to β is:

$$l'(\beta) = Z^T\delta - \sum_{i=1}^{n} \delta_i \frac{\sum_{j\in R(t_i)} \exp(z_j\beta) z_j^T}{\sum_{j\in R(t_i)} \exp(z_j\beta)}.$$

```
localLogLik <- function(beta) {
    beta <- c(0, beta) ## model matrix has intercept in model
    z <- modelMatrix
    delta <- modelDataFrame[, 1][, 2] ## event indicators
    zBeta <- z %*% beta
    sum.exp.zBeta <- rev(cumsum(rev(exp(zBeta))))
    ld <- delta %*% (z  - apply(diag(as.numeric(zBeta)) %*% z, 2, function(x) rev(cumsum(rev(x)))) / su
    result <- sum(delta * (zBeta - log(sum.exp.zBeta))) # assuming Breslow
    attr(result, "gradient") <- ld
    result
}
site$methods(logLik = localLogLik)
```

The alternative distributed fit.

We are now ready to do the alternative distributed fit. We split the `prostate` data into two sites as indicated earlier.

```
siteData <- with(prostate, split(x = prostate, f = rx))
sites <- lapply(siteData,
                function(x) {
                    site$new(data = x,
                             formula = Surv(dtime, status) ~ stage + age + wt + pf + hx + sbp + dbp + e
                })
```

Ok, now we can reproduce the original aggregated analysis by writing a full likelihood routine.

```
logLik <- function(beta, sites) {
    sum(sapply(sites, function(x) x$logLik(beta)))
}
```

All that remains is to maximize this log likelihood.

```r
mleResults <- nlm(f=function(x) -logLik(x, sites),
                  p = rep(0, sites[[1]]$dimP()),
                  gradtol = 1e-10, iterlim = 1000)
```

```
## Warning in nlm(f = function(x) -logLik(x, sites), p = rep(0, sites[[1]]
## $dimP()), : NA/Inf replaced by maximum positive value

## Warning in nlm(f = function(x) -logLik(x, sites), p = rep(0, sites[[1]]
## $dimP()), : NA/Inf replaced by maximum positive value

## Warning in nlm(f = function(x) -logLik(x, sites), p = rep(0, sites[[1]]
## $dimP()), : NA/Inf replaced by maximum positive value
```

We print the coefficient estimates side-by-side for comparison.

```r
d <- data.frame(distCoef = mleResults$estimate, aggCoef = pcph$coefficients)
rownames(d) <- names(pcph$coefficients)
kable(d)
```

	distCoef	aggCoef
stageIV	-0.1740491	-0.1775897
age	0.0245191	0.0242084
wt	-0.0105003	-0.0102559
pfBedridden(<50%)	-1.3893175	-1.3727528
pfBedridden(>50%)	-1.1506815	-1.1749372
pfnormal	-1.5842329	-1.5886734
hx	0.5206160	0.5104190
sbp	-0.0329601	-0.0330859
dbp	0.0526530	0.0494337
ekgblock/conduction	-0.1505854	-0.1459025
ekgheart strain	0.3980164	0.3931730
ekgnormal	-0.0557321	-0.0579591
ekgold MI	0.0064296	0.0074423
ekgrecent MI	0.8560623	0.8189465
ekgrhythmic disturb	0.2816922	0.2803126
hg	-0.0707047	-0.0689290
sz	0.0180493	0.0178037
sg	0.1016070	0.1002007
ap	-0.0013375	-0.0013841
bm	0.3180923	0.3196404

Session Information

```r
sessionInfo()
```

```
## R version 3.6.2 (2019-12-12)
## Platform: x86_64-apple-darwin19.2.0 (64-bit)
## Running under: macOS Catalina 10.15.2
##
## Matrix products: default
## BLAS/LAPACK: /usr/local/Cellar/openblas/0.3.7/lib/libopenblasp-r0.3.7.dylib
##
## locale:
```

```
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] graphics  grDevices datasets  stats     utils     methods   base
##
## other attached packages:
## [1] KMsurv_0.1-5   distcomp_1.3   survival_3.1-8 rmarkdown_2.0  knitr_1.26
## [6] pkgdown_1.4.1  devtools_2.2.1 usethis_1.5.1
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_0.2.5  xfun_0.11         remotes_2.1.0     purrr_0.3.3
##  [5] splines_3.6.2     lattice_0.20-38   sodium_1.1        testthat_2.3.1
##  [9] htmltools_0.4.0   yaml_2.2.0        gmp_0.5-13.5      rlang_0.4.2
## [13] pkgbuild_1.0.6    later_1.0.0       pillar_1.4.3      glue_1.3.1
## [17] withr_2.1.2       sessioninfo_1.1.1 stringr_1.4.0     homomorpheR_0.2-4
## [21] codetools_0.2-16  evaluate_0.14     memoise_1.1.0     callr_3.4.0
## [25] fastmap_1.0.1     httpuv_1.5.2      ps_1.3.0          curl_4.3
## [29] fansi_0.4.1       highr_0.8         Rcpp_1.0.3        xtable_1.8-4
## [33] backports_1.1.5   promises_1.1.0    desc_1.2.0        pkgload_1.0.2
## [37] jsonlite_1.6      mime_0.8          fs_1.3.1          digest_0.6.23
## [41] stringi_1.4.3     processx_3.4.1    dplyr_0.8.3       shiny_1.4.0
## [45] grid_3.6.2        rprojroot_1.3-2   cli_2.0.1         tools_3.6.2
## [49] magrittr_1.5      tibble_2.1.3      crayon_1.3.4      pkgconfig_2.0.3
## [53] MASS_7.3-51.5     ellipsis_0.3.0    Matrix_1.2-18     prettyunits_1.1.0
## [57] assertthat_0.2.1  httr_1.4.1        R6_2.4.1          compiler_3.6.2
```