parglms: fitting generalized linear and related models with parallel evaluation of contributions to sufficient statistics

Vincent J. Carey, stvjc at channing.harvard.edu

December 2014

Contents

1 Introduction

The main concern of the *parglms* package is supporting efficient computations for modeling with dispersed data. The motivating use case is an eQTL analysis as exemplified in *geuvStore2*. A *BatchJobs* Registry mediates access to collections of millions of statistics on SNP-transcript association. We want to use statistical modeling to perform inference on features of SNPs contributing to various phenotypic conditions through effects on gene expression.

Formally, let y denote an N vector with mean function $\mu(\beta)$ and variance function $V(\mu)$. The parameter vector of interest, β, is of dimension p, satisfying $g(\mu) = X\beta$, where X is $N \times p$. In conventional use of generalized linear models (GLMs), functions μ and V have simple forms and are programmed in the `family` elements for `stats::glm`. The models are fit by solving equations of the form

$$D^t W[y - \mu(\beta)] = 0$$

where W is diagonal with elements prescribed by the reciprocal variance function, and $D = \partial\mu/\partial\beta$.

The key motivation for this package is the recognition that steps towards the solution of the equation can often be arithmetically decomposed in various ways, and neither holistic nor serial computation is required for most of the calculations. We can therefore accomplish the model fitting task in a scalable way, decomposing the data into parts that fit comfortably in memory, and performing operations in parallel among the parts whenever possible.

2 Illustration with a data.frame: dispersal and analysis

We use BatchJobs to disperse data into small chunks.

```
library(MASS)
library(BatchJobs)
```

```
## Loading required package: BBmisc
```

```
##
## Attaching package: 'BBmisc'
```

```
## The following object is masked from 'package:base':
##
##     isFALSE
```

```
## The development of BatchJobs and BatchExperiments is discontinued.
```

```
## Consider switching to 'batchtools' for new features and improved stability
```

```
## Sourced 1 configuration files:
```

```
##   1: /Library/Frameworks/R.framework/Versions/3.6/Resources/library/BatchJobs/etc/BatchJobs_global_c
```

```
## BatchJobs configuration:
##   cluster functions: Interactive
##   mail.from:
##   mail.to:
##   mail.start: none
##   mail.done: none
##   mail.error: none
##   default.resources:
##   debug: FALSE
##   raise.warnings: FALSE
##   staged.queries: TRUE
##   max.concurrent.jobs: Inf
##   fs.timeout: NA
##   measure.mem: TRUE
```

```
data(anorexia)  # N = 72
myr = makeRegistry("abc", file.dir=tempfile())
```

```
## Creating dir: /tmp/RtmpbMpBP8/file754f5bd1d076
```

```
## Saving registry: /tmp/RtmpbMpBP8/file754f5bd1d076/registry.RData
```

```
chs = chunk(1:nrow(anorexia), n.chunks=18) # 4 recs/chunk
f = function(x) anorexia[x,]
options(BBmisc.ProgressBar.style="off")
batchMap(myr, f, chs)
```

```
## Adding 18 jobs to DB.
```

```
showStatus(myr)
```

```
## Status for 18 jobs at 2019-10-29 21:35:55
## Submitted:  0 (  0.00%)
## Started:    0 (  0.00%)
## Running:    0 (  0.00%)
## Done:       0 (  0.00%)
## Errors:     0 (  0.00%)
## Expired:    0 (  0.00%)
## Time: min=NAs avg=NAs max=NAs
```

We are now poised to rewrite the data.frame contents into chunks.

```
## Syncing registry ...
```

```
submitJobs(myr)
waitForJobs(myr)
```

```
loadResult(myr,1)
```

```
##   Treat Prewt Postwt
## 1  Cont  80.7   80.2
## 2  Cont  89.4   80.1
## 3  Cont  91.8   86.4
## 4  Cont  74.0   86.3
```

The `parGLM` method will fit the model specified in the formula. The task of iterating over chunks is left to the *BiocParallel* `bplapply`, and this will implicitly use whatever concurrent computing approach has been registered.

```
library(parglms)
pp = parGLM( Postwt ~ Treat + Prewt, myr,
   family=gaussian, binit = c(0,0,0,0), maxit=10, tol=.001 )
```

```
## Warning: executing %dopar% sequentially: no parallel backend registered
```

```
names(pp)
```

```
## [1] "coefficients"    "eff.variance"    "robust.variance" "s2"
## [5] "niter"           "converged"       "formula"         "N"
## [9] "theCall"
```

```
pp$coef
```

```
##                    [,1]
## (Intercept) 49.7711090
## TreatCont   -4.0970655
## TreatFT      4.5630627
## Prewt        0.4344612
```

3 Illustration with geuvStore2

In this application we model the probability that a SNP has been identified as a GWAS hit, as a function of aspects of its genomic context and its association with expression as measured using RNA-seq in GEUVADIS.

In the `decorate` function, we emend the outputs of *gQTLstats* `cisAssoc` after applying storeToFDR and enumerateByFDR, as serialized in a GRanges (demoEnum) with information on GWAS hit status and enclosing chromatin state. This is intensive; the `litdec` function simply computes GWAS hit status and chromatin state.

```
litdec = function(grWithFDR) {
 tmp = grWithFDR
 library(gQTLstats)
 if (!exists("hmm878")) data(hmm878)
 seqlevelsStyle(hmm878) = "NCBI"
 library(GenomicRanges)
 ov = findOverlaps(tmp, hmm878)
 states = hmm878$name
 states[ which(states %in% c("13_Heterochrom/lo", "14_Repetitive/CNV",
      "15_Repetitive/CNV")) ] = "hetrep_1315"
 levs = unique(states)
 tmp$hmmState = factor(rep("hetrep_1315", length(tmp)),levels=levs)
 tmp$hmmState = relevel(tmp$hmmState, "hetrep_1315")
 tmp$hmmState[ queryHits(ov) ] = factor(states[ subjectHits(ov) ],
   levels=levs)
 if (!exists("gwrngs19")) data(gwrngs19)
```

```
 library(GenomeInfoDb)
 seqlevelsStyle(gwrngs19) = "NCBI"
 tmp$isGwasHit = 1*(tmp %in% gwrngs19)
 tmp
}

decorate = function(grWithFDR) {
#
# the data need a distance/MAF filter
#
 library(gQTLstats)
 data(filtFDR)
 if (!exists("hmm878")) data(hmm878)
 library(gwascat)
 if (!exists("gwrngs19")) data(gwrngs19)
 if (!exists("gwastagger")) data(gwastagger) # will use locations here
 library(GenomeInfoDb)
 seqlevelsStyle(hmm878) = "NCBI"
 seqlevelsStyle(gwrngs19) = "NCBI"
 seqlevelsStyle(gwastagger) = "NCBI"
 tmp = grWithFDR
 tmp$isGwasHit = 1*(tmp %in% gwrngs19)
 tmp$isGwasTagger = 1*(tmp %in% gwastagger)
 #levs = unique(hmm878$name)
 library(GenomicRanges)
 ov = findOverlaps(tmp, hmm878)
 states = hmm878$name
 states[ which(states %in% c("13_Heterochrom/lo", "14_Repetitive/CNV",
      "15_Repetitive/CNV")) ] = "hetrep_1315"
 levs = unique(states)
 tmp$hmmState = factor(rep("hetrep_1315", length(tmp)),levels=levs)
 tmp$hmmState = relevel(tmp$hmmState, "hetrep_1315")
 tmp$hmmState[ queryHits(ov) ] = factor(states[ subjectHits(ov) ],
   levels=levs)
 tmp$estFDR = getFDRfunc(filtFDR)( tmp$chisq )
 tmp$fdrcat = cut(tmp$estFDR, c(-.01, .01, .05, .1, .25, .5, 1.01))
 tmp$fdrcat = relevel(tmp$fdrcat, "(0.5,1.01]")
 #tmp$distcat = cut(tmp$mindist, c(-1,0,1000,5000,10000,50000,100000,250000,500001))
 tmp$distcat = cut(tmp$mindist, c(-1,0,1000,5000,10000,25000,50001))
 #tmp$distcat = relevel(tmp$distcat, "(2.5e+05,5e+05]")
 tmp$distcat = relevel(tmp$distcat, "(2.5e+04,5e+04]")
 tmp$MAFcat = cut(tmp$MAF, c(.049, .075, .1, .25, .51))
 tmp$MAFcat = relevel(tmp$MAFcat, "(0.25,0.51]")
 kp = c("seqnames", "start", "probeid", "snp", "estFDR", "fdrcat", "hmmState",
  "distcat", "MAFcat", "isGwasHit", "isGwasTagger")
 names(tmp) = NULL
 as(tmp, "data.frame")[,kp]
}
```

We'll try this out here:

```
suppressPackageStartupMessages({
library(geuvStore2)
library(gQTLBase)
```

```
library(gQTLstats)
})
prst = g17transRegistry()
```

Now we can fit a very simple model for SNP phenorelevance. We set the extractor component of the registry to the litdec function defined above.

```
prst$extractor = function(store,i) litdec(loadResult(store,i)[[1]])
p1 = parGLM( isGwasHit ~ hmmState, prst,
   family=binomial, binit=rep(0,13), tol=.001,
   maxit = 10 )
summaryPG(p1)
#ans= list(coef=p1$coef, s.e.=sqrt(diag(p1$eff.var)))
#ans$z = ans[[1]]/ans[[2]]
#do.call(cbind, ans)
```