RISKmisTESTING

2023-12-14

Package Installation

```
library(devtools)
```

```
## Loading required package: usethis
```

```
library(survival)
library(doRNG)
```

```
## Loading required package: foreach
```

```
## Loading required package: rngtools
```

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-8
```

```
library(ggplot2)
library(data.table)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:data.table':
##
##     between, first, last
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(MASS)
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```
devtools::install_github("MR236/BIOS615Risk_Pred")
```

```
## Skipping install of 'RISKmis' from a github remote, the SHA1 (75a22b17) has not changed since last i
##   Use `force = TRUE` to force installation
```

```
library(RISKmis)
```

Testing of Initial Estimator Function

First we generate some data using the authors' original specifications:

```
n <- 10000
m <- 500
beta0 <- c(0.7,0.7,0.7,-0.5,-0.5,-0.5,0.3,0.3,0.3,0)
p <- length(beta0)
sigmaZ <- 1
misT <- noise.loglinear.S
sim_data <- sim.gen.rev(n, beta0, sigmaZ,m,misT)
labelled_data <- sim_data$data[1:m,]
```

Then we can get estimates using the original method and our method:

```
betadelta <- init.beta.new(delta = labelled_data$delta,C=labelled_data$C,Z = labelled_data$Z)$beta
betadelta
```

```
##  [1]  0.60153956  0.51078204  0.62134251  0.00000000 -0.70952477 -0.36573788
##  [7]  0.43241595  0.07590303  0.08716286  0.02950849
```

```
# original method requires a bit of extra preparation
h = sd(labelled_data$C)/(sum(labelled_data$delta))^0.26
KC = dnorm(as.matrix(dist(labelled_data$C/h,diag=T,upper=T)))/h
betadelta_old <- init.beta(delta = labelled_data$delta,Z = labelled_data$Z,KC=KC, link=expit, dlink=dexp
betadelta_old
```

```
##  [1]  0.62308943  0.51571278  0.60024481 -0.01861804 -0.70639698 -0.39716474
##  [7]  0.43397244  0.07776657  0.09895481  0.03573930
```

Note that the results are not exactly the same but very similar. Asymptotic bias and MSE are near-identical as shown in simulations (see paper). Note the superior speed of our method is clear even with this relatively small sample size.

Testing of Score Function

First some data pre-processing:

```
data <- sim_data$data
dataS <- sim_data$dataS
beta.std = betadelta/sqrt(sum(betadelta^2))
lp = drop(data$Z %*% beta.std)
sdlp = sd(lp)

h1 = sdlp/(sum(dataS$delta1))^0.3
h2 = sdlp/(sum(dataS$delta2))^0.3
```

Then we calculate the score using our function and the original:

```
# Our method
Sk = rep(0,p*2)
Sk[1:p] = Sk_sym_new(lp,data$Z,
                         dataS$X1,dataS$delta1,
                         data$C,h1)
Sk[p+1:p] = Sk_sym_new(lp,data$Z,
                           dataS$X2,dataS$delta2,
```

```
                          data$C,h2)

# old_method
Sk_old = rep(0,p*2)
Sk_old[1:p] = Sk_sym(lp,data$Z,
                     dataS$X1,dataS$delta1,
                     data$C,dnorm,h1)
Sk_old[p+1:p] = Sk_sym(lp,data$Z,
                       dataS$X2,dataS$delta2,
                       data$C,dnorm,h2)

Sk-Sk_old
```

```
##  [1] -1.131823e-09 -1.202343e-09 -1.175475e-09  3.527064e-10  9.353335e-11
##  [6]  1.545220e-10 -6.607603e-10 -7.471728e-10 -6.828384e-10 -1.709256e-10
## [11] -1.045355e-09 -1.057068e-09 -1.023796e-09  3.495394e-10  2.232026e-11
## [16]  1.755364e-10 -5.477685e-10 -6.922502e-10 -6.067165e-10 -2.095954e-10
```

There are two separate function calls for each method, one for each of the two surrogate outcomes in the simulation scenario. Note the extremely small approximation errors displayed and large computational efficiency advantage for our method.

Testing of the Full Method

To demonstrate the full method we will use a much smaller dimension of data, as the full method involves calling the above functions 1000 times each.

```
n <- 1000
m <- 250
beta0 <- c(0.7,0.7,0.7,-0.5,-0.5,-0.5,0.3,0.3,0.3,0)
p <- length(beta0)
sigmaZ <- 1
misT <- noise.loglinear.S
sim_data <- sim.gen.rev(n, beta0, sigmaZ,m,misT)
data_labelled <- sim_data$data
data_unlabelled <- sim_data$dataS
```

Our method, taking around 45 seconds:

```
t1 <- Sys.time()
result <- Full_Estimator_New(data_labelled, data_unlabelled, n=n, m=m, Nperturb=500)
t2 <- Sys.time()
time_new <- t2-t1
# time to run
time_new
```

```
## Time difference of 32.62628 secs
```

```
# estimated coefficients
result[[1]]
```

```
##  [1]  0.7450003  0.5416760  0.6391603 -0.4597936 -0.4399046 -0.3449316
##  [7]  0.1256726  0.2341338  0.1853699  0.2569925
```

Authors' method, taking around 3 minutes:

```
t1 <- Sys.time()
result_old <- Full_Estimator_Original(data_labelled, data_unlabelled, n=n,m=m, Nperturb=500)
```

```r
t2 <- Sys.time()
time_old <- t2-t1
# time to run
time_old
```

```
## Time difference of 2.605844 mins
```

```r
# estimated coefficients
result_old[[1]]
```

```
##  [1]  0.8275438  0.6674822  0.7482868 -0.4999623 -0.5645110 -0.3841026
##  [7]  0.1761126  0.2452208  0.2545596  0.2682133
```