IntegrativeCox Example

Aaron J. Molstad and Rohit K. Patra

Updated on July 20th, 2022

In this document, we provide a short example of how to (1) format data for integrative survival analysis with our method and (2) how to implement our method. First, let us generate $J = 12$ cancer datasets in the manner described in Section 5 of the manuscript.

```
library(Matrix)
library(glmnet)
```

```
## Loaded glmnet 4.1-2
```

```
library(MASS)
library(survival)
set.seed(1)

J <- 12
Ns <- rep(c(1250, 1350, 1450), 4)
nreps <- 100
p <- 50
r <- 6
quan.cens <- .35
kappa <- seq(2000, 2110, by = 10)

genCoxDat <- function(uu, p, Ns, quan.cens){

  set.seed(1)
  SigmaX <- matrix(0, p, p)
  for(j in 1:p){
    for(k in 1:p){
      SigmaX[j,k] <- .7^(abs(j-k))
    }
  }
  X <- list(NA)
  for(j in 1:J){
    X[[j]] <- mvrnorm(n = Ns[j], mu = rep(0, p),
                      SigmaX, tol = 1e-06, empirical = FALSE)
  }

  simulGomp <- function(datIndex, lambda, alpha, beta, rateC){

    N <- dim(X[[datIndex]])[1]
    x <- as.matrix(X[[datIndex]])
    v <- runif(n=N)
    Tlat <- (1/alpha)*log(1 - (alpha*log(v))/(lambda*exp(x%*%beta)))
```

```
    # censoring times
    if(datIndex%%3 == 0){
      temp <- quantile(Tlat, quan.cens+.2)
    } else {
      temp <- quantile(Tlat, quan.cens)
    }

    C <- rexp(n=N, rate=1/temp)

    # follow-up times and event indicators
    time <- pmin(Tlat, C)
    status <- 1*(Tlat <= C)

    # data set
    list("id"=1:N, "time"=time, "status"=status, "X"=x, "Tlat" = Tlat, "C" = C, "linPred" = x%*%beta)
  }


  get_beta <- function(sigma.temp){
    beta <- matrix(0, nrow=p, ncol=J)
    nonzeroes <- sample(1:p, 20, replace=FALSE)
    temp <- svd(matrix(rnorm(r*J), nrow=r, ncol=J))$v*(sqrt(2)/sqrt(r))
    beta[nonzeroes, ] <- matrix(runif(20*r, 1, 2)*
                                sample(c(-1, 1), 20*r, replace=TRUE), nrow=20)%*%t(temp)
    return(beta)
  }

  sigma.temp <- NULL
  beta <- get_beta(sigma.temp)

  dat <- list(NA)
  for(kk in 1:J){
    alpha <- pi/(600*sqrt(6))
    lambda <- alpha*exp(- 0.5772 - alpha*kappa[kk])
    dat[[kk]] <- simulGomp(datIndex = kk, lambda=lambda, alpha=alpha, beta=beta[,kk], rateC=0.1)
  }

  return(list("dat" = dat, "beta" = beta, "SigmaX" = SigmaX))

}

simDat <- genCoxDat(uu = uu, p = p, Ns = Ns, quan.cens = quan.cens)
beta <- simDat$beta
dat <- simDat$dat
SigmaX <- simDat$SigmaX
simDat <- NULL


# ------------------------------------------------------
# Split into training, validation, and testing sets
# ------------------------------------------------------
datVal <- list(NA)
datTest <- list(NA)
```

```
for(j in 1:J){

  ValInds <- (100*((j-1)%%3 + 1) + 1):(100*((j-1)%%3 + 1) + 150)
  TestInds <- (100*((j-1)%%3 + 1) + 151):Ns[j]
  datVal[[j]] <- list(
    "X" = dat[[j]]$X[ValInds,],
    "time" = dat[[j]]$time[ValInds],
    "status" = dat[[j]]$status[ValInds],
    "Tlat" = dat[[j]]$Tlat[ValInds],
    "linPred" = dat[[j]]$linPred[ValInds])

  datTest[[j]] <- list(
    "X" = dat[[j]]$X[TestInds,],
    "time" = dat[[j]]$time[TestInds],
    "status" = dat[[j]]$status[TestInds],
    "Tlat" = dat[[j]]$Tlat[TestInds],
    "linPred" = dat[[j]]$linPred[TestInds])

  dat[[j]]$X <- dat[[j]]$X[-c(TestInds, ValInds),]
  dat[[j]]$time <- dat[[j]]$time[-c(TestInds, ValInds)]
  dat[[j]]$status <- dat[[j]]$status[-c(TestInds, ValInds)]

}
```

Now, we have three types of datasets: training, testing and validation (`dat`, `datVal`, and $ `datTest`$). Each is a list of length $J = 12$, and has subject id, event time (time), status (alive or deceased at event time), the predictor matrix X, and some other information which is not used for model fitting.

```
length(dat)
```

```
## [1] 12
```

```
str(dat[[1]])
```

```
## List of 7
##  $ id     : int [1:1250] 1 2 3 4 5 6 7 8 9 10 ...
##  $ time   : num [1:100] 775 696 969 3288 2424 ...
##  $ status : num [1:100] 0 0 0 1 0 0 0 1 1 0 ...
##  $ X      : num [1:100, 1:50] 2.09 0.549 -0.963 1.446 1.823 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : NULL
##  $ Tlat   : num [1:1250, 1] 2170 1941 1912 3288 2784 ...
##  $ C      : num [1:1250] 775 696 969 6911 2424 ...
##  $ linPred: num [1:1250, 1] 0.2464 0.0676 0.9203 -1.0646 -0.3114 ...
```

Next, we load the functions and fit the model using `IntCox` from the script `LRCox.R`. Note that one must first set the appropriate path `functionspath`. To shorten computing time, we will only consider $s \in \{5, 10, 15\}$ and $r \in \{4, 6\}$. This will take a few minutes. To track progress, you may set `quiet` or `silent` equal to `FALSE`.

```
# ------- load functions
source(paste(functionspath, "LRCox.R", sep=""))
sourceCpp(paste(functionspath, "updateBeta.cpp", sep=""))

# ------- fit model for multiple (s,r) combinations
fit <- IntCox(svec = c(5, 10, 15), rvec = c(4,6), dat = dat, mu = 0.1,
```

```
                quiet = TRUE, silent = TRUE, rho0 = 100)
```

Examining the output, we see that `fit` has three elements: `beta, s, r`. The array `beta` is $p \times J \times s^\star \times r^\star$ where $s^\star$ is the number of candidate s and $r^\star$ is the number of candidate r. Now, let us check the validation partial log-likelihood to determine which pair of tuning parameters is best.

```
valerrsOurs <- array(0, dim=c(length(fit$s), length(fit$r),J))
for(kk in 1:(length(fit$s))){
  for(jj in 1:length(fit$r)){
    for(ll in 1:J){
      valerrsOurs[kk,jj,ll] <- coxnet.deviance(y=Surv(datVal[[ll]]$time, datVal[[ll]]$status),
                                               pred = datVal[[ll]]$X%*%fit$beta[,ll,kk,jj])
    }
  }
}

out <- which(apply(valerrsOurs, c(1,2),sum) == min(apply(valerrsOurs, c(1,2),sum)), arr.ind=TRUE)
betaLR <- fit$beta[,,out[1,1], out[1,2]]
cat("Tuning parameters selected by validation set:" , "\n")
```

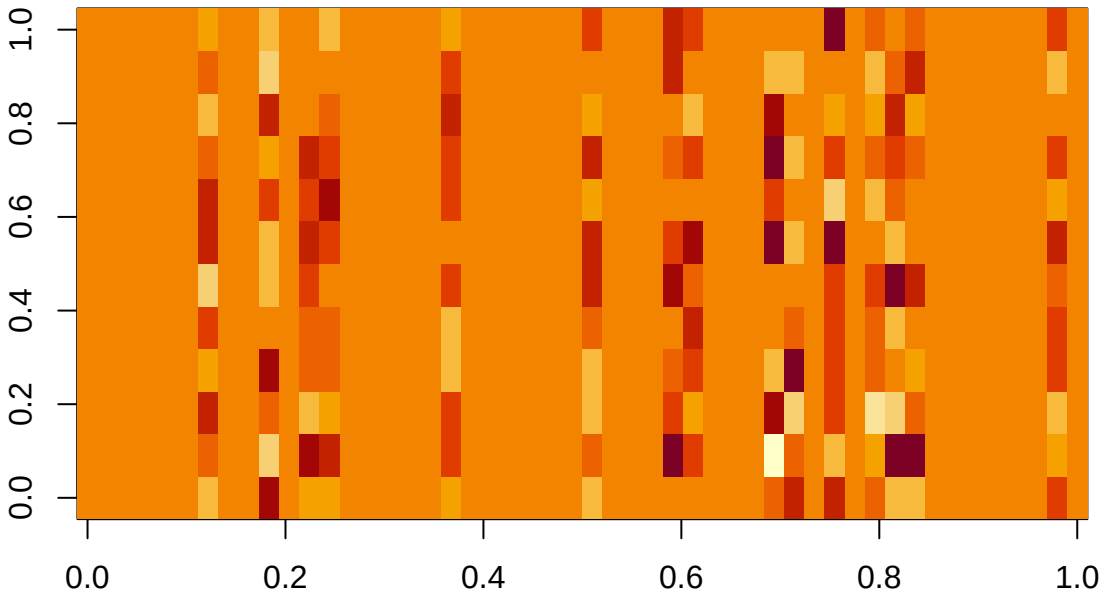
```
## Tuning parameters selected by validation set:
```

```
cat("s = ", fit$s[out[1,1]], ", r = ", fit$r[out[1,2]], "\n")
```

```
## s =  15 , r =  6
```

We can examine the regression coefficient estimate:

```
image(betaLR)
```



Similarly, let us extra the left singular vectors of the coefficient matrix estimate so that we may construct the factors for, say, the first population.

```
U <- svd(betaLR)$u
u.fit <- U[,1:fit$r[out[1,2]]]
X1factors <- datTest[[1]]$X%*%u.fit
dim(X1factors)
```

```
## [1] 1000    6
```

Now, let us fit a model to the testing data using the estimated factors from the training data.

```
library(survival)
summary(coxph(Surv(datTest[[1]]$time, datTest[[1]]$status)~X1factors))
```

```
## Call:
## coxph(formula = Surv(datTest[[1]]$time, datTest[[1]]$status) ~
##     X1factors)
##
##   n= 1000, number of events= 345
##
##                coef exp(coef) se(coef)       z Pr(>|z|)
## X1factors1 -0.84358   0.43017  0.06617 -12.749  < 2e-16 ***
## X1factors2 -0.21427   0.80713  0.06871  -3.119  0.00182 **
## X1factors3  0.04485   1.04587  0.06606   0.679  0.49719
## X1factors4  0.11888   1.12623  0.05390   2.206  0.02740 *
## X1factors5  0.34032   1.40539  0.05363   6.346 2.22e-10 ***
## X1factors6 -0.04474   0.95625  0.06725  -0.665  0.50592
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##            exp(coef) exp(-coef) lower .95 upper .95
## X1factors1    0.4302     2.3247    0.3778    0.4897
## X1factors2    0.8071     1.2390    0.7054    0.9235
## X1factors3    1.0459     0.9561    0.9189    1.1904
## X1factors4    1.1262     0.8879    1.0133    1.2517
## X1factors5    1.4054     0.7115    1.2652    1.5612
## X1factors6    0.9562     1.0458    0.8382    1.0910
##
## Concordance= 0.722  (se = 0.017 )
## Likelihood ratio test= 185.9  on 6 df,   p=<2e-16
## Wald test            = 179.7  on 6 df,   p=<2e-16
## Score (logrank) test = 181.5  on 6 df,   p=<2e-16
```

We can see that the factors lead to relatively high concordance, and all tests suggest they are significantly associated with survival.