Bayesian hierarchical Cox Survival Models

Nengjun Yi

This vignette describes Bayesian hierarchical Cox survival models and the EM Newtone-Raphson algorithm, and then the methods for assessing the fitted model and its predictive values. Finally, it explains how to set up and fit Bayesian Cox survival models using the `bcoxph` function in the **`BhGLM`** package.

Cox proportional hazards models

For censored survival outcomes, we observe a pair of response variables $y_i = (t_i, d_i)$ for each individual, where the censoring indicator d_i takes 1 if the observed survival time t_i for individual i is uncensored and 0 if it is censored. Denote the true survival time by T_i for individual i. Thus, when $d_i = 1$, $T_i = t_i$, whereas when $d_i = 0$, $T_i > t_i$. Cox proportional hazards model is the most widely used method for studying the relationship between the censored survival response and some explanatory variables X, and assumes that the hazard function of survival time T takes the form:

$$h(t \mid X) = h_0(t)\exp(X\beta) \tag{1}$$

where the baseline hazard function $h_0(t)$ is unspecified , X and β are the vectors of explanatory variables and coefficients, respectively, and $X\beta$ is the linear predictor or called the prognostic index.

Fitting classical Cox models is to estimate β by maximizing the partial log-likelihood:

$$pl(\beta) = \sum_{i=1}^{n} d_i \log\left(\exp(X_i\beta) \Big/ \sum_{i' \in R(t_i)} \exp(X_{i'}\beta) \right) \tag{2}$$

where $R(t_i)$ is the risk set at time t_i. In the presence of ties, the partial log-likelihood can be approximated by the Breslow or the Efron methods. The standard algorithm for maximizing the partial log-likelihood is the Newton-Raphson algorithm, which has been implemented by the function `coxph` in the package **`survival`**. However, the classical Cox survival models cannot deal with numerous and correlated predictors.

Prior distributions

Bayesian Cox survival models include priors on the parameters, which can constraint the coefficients in reasonable ranges and thus allows the model to be reliably fitted and to identify important predictors. For specifying appropriate priors for all the coefficients, it is important to transform all variables to have a common scale. The function `covariates` in **`BhGLM`** can be used to transform both continuous and categorical predictors. The function `bcoxph` employs four types of informative priors on the coefficients β_j: double-exponential and Student-t distributions, and spike-and-slab mixture double-exponential and Student-t distributions. For the detailed description, refer to the vignette "**Bayesian hierarchical GLMs**".

EM Newton-Raphson algorithm

The `bcoxph` function employs the EM Newton-Raphson algorithm to fit the above Bayesian Cox survival models by finding the posterior modes of the parameters, i.e. estimating the parameters by maximizing the posterior density. The EM Newton-Raphson algorithm incorporates an EM procedure into the usual Newton-Raphson algorithm for fitting Cox survival models. The EM procedure is based on the hierarchical formulations of the double-exponential and Student-t distributions, i.e. which can be expressed as normal distributions with unknown variances following exponential or inverse-gamma distribution. The algorithm treats the variances and the indicator variables (for the spike-and-slab priors) as missing data.

The algorithm is fast for fitting a Cox survival model with tens or a few hundreds of predictors, however, can be slow for larger models due to the need to calculate the inverse of a large matrix. The `bcoxph` function implements two methods to update the coefficients, i.e., jointly updating all coefficients, or updating a group of coefficients at a time and proceeding by cycling through all the groups at each iteration. For models with thousands of predictors, the group update method can be much faster.

Summarizing the fitted model and inferences

The EM Newton-Raphson algorithm returns the estimates of the coefficients and their standard deviations. The *p*-values for testing the hypotheses H_0: $\beta_j = 0$ can be calculated similarly as the classical framework. The `summary.bh` function in **`BhGLM`** provides the estimates of the coefficients, the standard deviations, and the *p*-values. The `plot.bh` function graphically displays these estimates.

Evaluating the fitted model and the predictive performance

After building a hierarchical Cox model, we obtain the estimates of the prognostic indices, $\hat{\eta}_i = X_i\hat{\beta}$, and we then can predict the survival probability $S(t \mid X)$ for any observation X. To assess the prognostic utility of the fitted model, we need to evaluate the fitted model and its predictive values. The package **`BhGLM`** provides several ways to evaluate a fitted Cox survival model, including the *partial log-likelihood*, the *concordance index* (*C-index*), the *survival curves*, and the *survival prediction error*. The function `predict.bh` returns the partial log-likelihood and C-index for the fitted model. The function `surv.curves` plots survival curves of groups of individuals and test the difference between curves using the log-rank method. The `peCox` function plots prediction error curves.

The package `BhGLM` provides two ways to evaluate the predictive performance of the model. If we have a training data set and a validation data set, we can use the training data to fit a Bayesian Cox survival model and then evaluate the predictive performance on the validation data (treated as new data) using the function `predict.bh`. If we have only one data set, we can perform K-fold cross-validation using the function `cv.bh`, which returning cross-validated partial log-likelihood, Houwelinggen's cross-validated partial log-likelihood, and C-index. The function `cv.bh` also returns the cross-validated prognostic index, which can be used to calculate the survival curves and prediction error curves with the functions `surv.curves` and `peCox`.

Examples

We first use the functions `sim.x` and `sim.y` to simulate data, and then explain how to use `bcoxph` and other functions for analyzing Bayesian Cox survival models. We simulate 1000 individuals, 200 predictors, six non-zero coefficients, and censored survival response.

```
library(BhGLM)

set.seed(1234)
N = 1000
K = 200
# simulate K correlated variables
x = sim.x (n=N, m=K, corr=0.6)
# assign 6 non-zero coefficients, and all others are zero
h = rep(0.1, 6)
nz = as.integer(seq(5, K, by=K/length(h)))
# simulate responses
yy = sim.y (x=x[, nz], mu=0, herit=h, p.neg=0.5)

# show the non-zero coefficients
yy$coefs

#          x5         x38         x71        x105        x138        x171
# -0.5064184 -0.5148792  0.4967354  0.5126193 -0.4946622  0.4959975

# simulated censored survival outcome
y = yy$y.surv
y[1:20]
```

Bayesian Cox survival models with double-exponential and t priors

We first analyze the simulated survival response using double-exponential and t priors with scale 0.05:

```
f1 = bcoxph(y ~ ., data = x, prior = "de", prior.scale = 0.05)
# EM Newton-Raphson iterations: 16
# Computational time: 0.149 minutes

f2 = bcoxph(y ~ ., data = x, prior = "t", prior.scale = 0.05)
# EM Newton-Raphson iterations: 11
# Computational time: 0.101 minutes
```

We numerically summarize the fitted model using `summary.bh`:

```
out1 = summary.bh(f1, digits = 4)
head(out1)
#       coef se(coef) exp(coef) lower.95 upper.95    pvalue
# x1  0.0000   0.0006    1.0000   0.9987   1.0013 9.944e-01
# x2  0.0003   0.0046    1.0003   0.9913   1.0095 9.457e-01
# x3  0.0059   0.0175    1.0059   0.9719   1.0411 7.366e-01
# x4  0.0000   0.0001    1.0000   0.9998   1.0002 9.997e-01
```

```
# x5 -0.3117   0.0611    0.7322   0.6496   0.8253 3.326e-07
# x6 -0.0002   0.0038    0.9998   0.9924   1.0072 9.542e-01
```

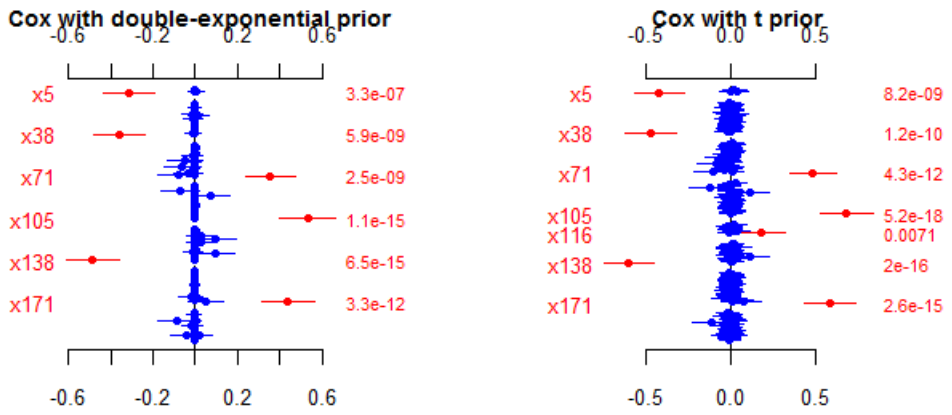
```
out2 = summary.bh(f2, digits = 4)
head(out2)
```

```
#       coef se(coef) exp(coef) lower.95 upper.95    pvalue
# x1  0.0147   0.0334    1.0149   0.9505   1.0835 6.590e-01
# x2  0.0225   0.0343    1.0228   0.9562   1.0939 5.120e-01
# x3  0.0332   0.0372    1.0338   0.9610   1.1121 3.726e-01
# x4  0.0001   0.0318    1.0001   0.9397   1.0644 9.976e-01
# x5 -0.4197   0.0728    0.6572   0.5698   0.7580 8.154e-09
# x6 -0.0139   0.0332    0.9862   0.9240   1.0525 6.748e-01
```

We graphically summarize the fitted model using `plot.bh`:

```
par (mfrow = c(1, 2), cex.axis = 1, mar = c(3, 4, 4, 4))

plot.bh (f1, threshold = 0.01, gap = 10, main = "Cox with double-exponential prior",
         col.pts=c("red", "blue") )
plot.bh (f2, threshold = 0.01, gap = 10, main = "Cox with t prior",
         col.pts=c("red", "blue"))
```



In the above analyses, we jointly update all the 200 coefficients. However, we can update a group of coefficients (say 50) at each time as follows. The group update algorithm can be much faster than the joint update if the model includes thousands of predictors.

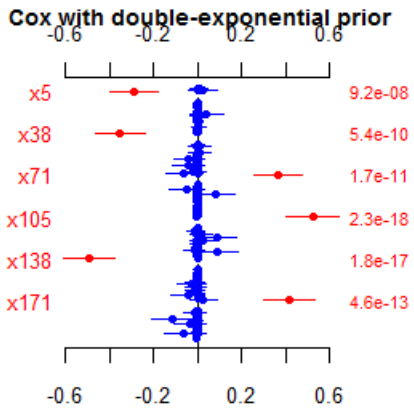
```
f3 = bcoxph(y ~ ., data = x, prior = "de", prior.scale = 0.05, method.coef=50)
# EM Newton-Raphson iterations: 15
# Computational time: 0.034 minutes
```

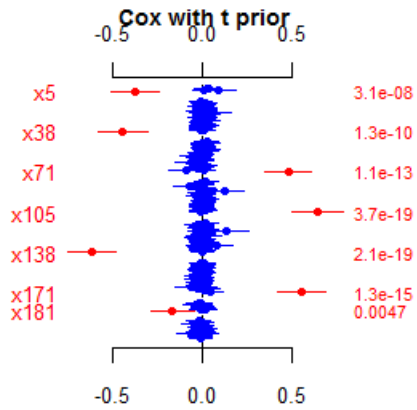
```
f4 = bcoxph(y ~ ., data = x, prior = "t", prior.scale = 0.05, method.coef=50)
# EM Newton-Raphson iterations: 10
# Computational time: 0.038 minutes
```

```
par (mfrow = c(1, 2), cex.axis = 1, mar = c(3, 4, 4, 4))

plot.bh (f3, threshold = 0.01, gap = 10, main = "Cox with double-exponential prior",
         col.pts=c("red", "blue") )
```

```
plot.bh (f4, threshold = 0.01, gap = 10, main = "Cox with t prior",
        col.pts=c("red", "blue"))
```

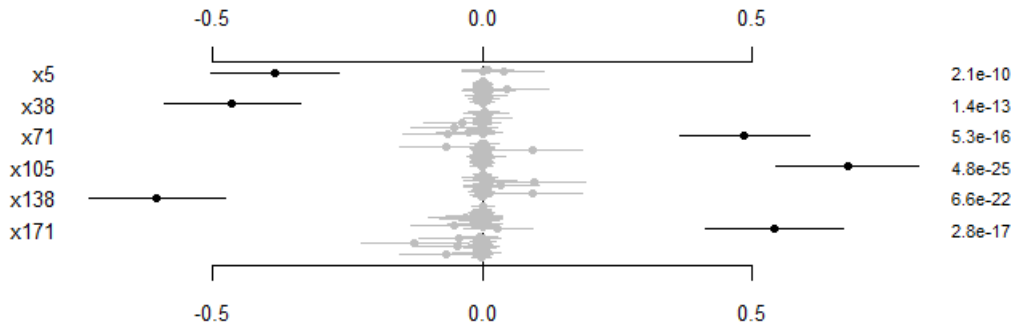




Bayesian Cox survival models with spike-and-slab double-exponential prior

We can analyze the data using the spike-and-slab double-exponential prior with s0=0.05 and s1=0.5 by running:

```
f5 = bcoxph(y ~ ., data = x, prior = "mde", ss=c(0.05, 0.5), method.coef=50)
plot.bh (f5, threshold = 0.01, gap = 10)
```



With the spike-and-slab prior, we can incorporate any known group structure into the model. Although this simulated data have no meaningful group structure, we explain how to incorporate any group structure into the analysis. Assume that the predictors form groups: x1-x50, x51-x100, x101-x150, x151-x200. We can incorporate this group structure by running:

```
f6 = bcoxph (y ~ ., data=x, prior="mde", ss=c(0.05, 0.5), group=c(0, 50, 100, 150, 200) )
plot.bh (f6, threshold = 0.01, gap = 10)
```

To check whether the group structure has been correctly set, we can look at:

```
f6$group.vars
f6$ungroup.vars
```

Assessing the fitted models and the predictive values

We can use the function `predict.bh` to calculate measures for assessing the fitted model:

```
pre = predict.bh(f3)
pre$measures
```

```
#    loglik    Cindex
# -2755.429     0.729
```

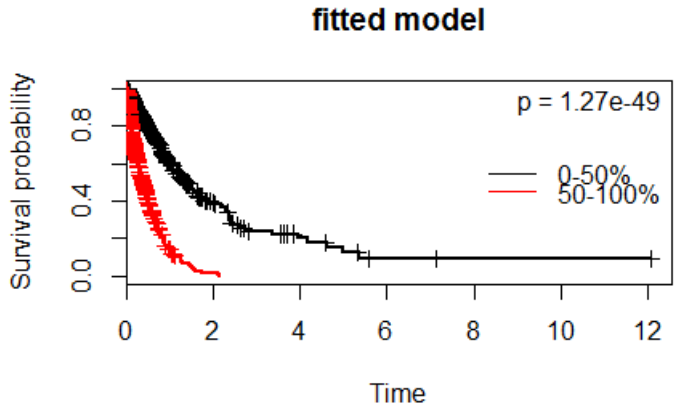
If we have new data new.x and new.y, we can use predict.bh(f3, new.x, new.y)$measures to evaluate the predictive performance of the model f3. We also can use cross-validation to assess the predictive performance:

```
cv = cv.bh(f3, nfolds=10, ncv=1)
cv$measures
```

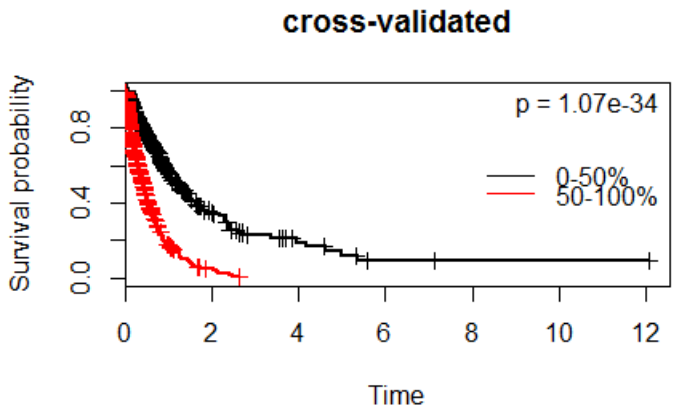
```
#       CVPL         pl     Cindex
# -3256.168 -2788.198     0.702
```

We can display survival curves for the fitted model and the cross-validation model using the function `surv.curves`:

```
cols = c("black", "red")
# for the fitted model
surv.curves(y, f3$linear.predictors, probs=0.50, col=cols, main="fitted model")
legend("right", legend=c("0-50%", "50-100%"), col=cols, lwd=1, bty="n")
```




```
# for cross-validated model
surv.curves(y, cv$lp, probs=0.50, col=cols, main="cross-validated")
legend("right", legend=c("0-50%", "50-100%"), col=cols, lwd=1, bty="n")
```



We can display prediction error curves for the fitted model and the cross-validation model using the function `peCox`:

```
cols = c("black", "red")
pecv = peCox(y, cv$lp, col=cols[1])
pe = peCox(y, f3$linear.predictors, col=cols[2], add=T)
legend("topright", legend=c("cross-validated", "fitted model"), col=cols, lwd=1, bty="n")
```