Package 'survival.svb'

January 17, 2022

Type Package

Title Fit High-Dimensional Proportional Hazards Models

Version 0.0-2

Date 2022-01-17

Author Michael Komodromos

Maintainer Michael Komodromos <mk1019@ic.ac.uk>

Description Implementation of methodology designed to perform: (i) variable selection, (ii) effect estimation, and (iii) uncertainty quantification, for high-dimensional survival data. Our method uses a spike-and-slab prior with Laplace slab and Dirac spike and approximates the corresponding posterior using variational inference, a popular method in machine learning for scalable conditional inference. Although approximate, the variational posterior provides excellent point estimates and good control of the false discovery rate. For more information see Komodromos et al. (2021) <arXiv:2112.10270>.

License GPL-3

Depends R (>= 4.0.0)

Imports Rcpp (>= 1.0.6), glmnet, survival

LinkingTo Rcpp, RcppEigen

NeedsCompilation yes

URL https://github.com/mkomod/survival.svb

BugReports https://github.com/mkomod/survival.svb/issues

RoxygenNote 7.1.1

Encoding UTF-8

Repository CRAN

Date/Publication 2022-01-17 10:10:02 UTC

R topics documented:

`elbo` *Compute the evidence lower bound (ELBO)*

Description

Compute the evidence lower bound (ELBO)

Usage

```
elbo(Y, delta, X, fit, nrep = 10000, center = TRUE)
```

Arguments

`Y`	Failure times.
`delta`	Censoring indicator, 0: censored, 1: uncensored.
`X`	Design matrix.
`fit`	Fit model.
`nrep`	Number of Monte Carlo samples.
`center`	Should the design matrix be centered.

Value

Returns a list containing:

`mean`	The mean of the ELBO.
`sd`	The standard-deviation of the ELBO.
`expected.likelihood`	The expectation of the likelihood under the variational posterior.
`kl`	The KL between the variational posterior and prior.

Details

The evidence lower bound (ELBO) is a popular goodness of fit measure used in variational inference. Under the variational posterior the ELBO is given as

$$ELBO = E_{\tilde{\Pi}}[\log L_p(\beta; Y, X, \delta)] - KL(\tilde{\Pi} \| \Pi)$$

where $\tilde{\Pi}$ is the variational posterior, Π is the prior, $L_p(\beta; Y, X, \delta)$ is Cox's partial likelihood. Intuitively, within the ELBO we incur a trade-off between how well we fit to the data (through the expectation of the log-partial-likelihood) and how close we are to our prior (in terms of KL divergence). Ideally we want the ELBO to be as small as possible.

svb.fit *Fit sparse variational Bayesian proportional hazards models.*

Description

Fit sparse variational Bayesian proportional hazards models.

Usage

```
svb.fit(
  Y,
  delta,
  X,
  lambda = 1,
  a0 = 1,
  b0 = ncol(X),
  mu.init = NULL,
  s.init = rep(0.05, ncol(X)),
  g.init = rep(0.5, ncol(X)),
  maxiter = 1000,
  tol = 0.001,
  alpha = 1,
  center = TRUE,
  verbose = TRUE
)
```

Arguments

`Y`	Failure times.
`delta`	Censoring indicator, 0: censored, 1: uncensored.
`X`	Design matrix.
`lambda`	Penalisation parameter, default: `lambda=1`.
`a0`	Beta distribution parameter, default: `a0=1`.
`b0`	Beta distribution parameter, default: `b0=ncol(X)`.
`mu.init`	Initial value for the mean of the Gaussian component of the variational family (μ), default taken from LASSO fit.
`s.init`	Initial value for the standard deviations of the Gaussian component of the variational family (s), default: `rep(0.05,ncol(X))`.
`g.init`	Initial value for the inclusion probability (γ), default: `rep(0.5,ncol(X))`.
`maxiter`	Maximum number of iterations, default: `1000`.
`tol`	Convergence tolerance, default: `0.001`.
`alpha`	The elastic-net mixing parameter used for initialising `mu.init`. When `alpha=1` the lasso penalty is used and `alpha=0` the ridge penalty, values between 0 and 1 give a mixture of the two penalties, default: `1`.

center	Center X prior to fitting, increases numerical stability, default: TRUE
verbose	Print additional information: default: TRUE.

Value

Returns a list containing:

beta_hat	Point estimate for the coefficients β, taken as the mean under the variational approximation. $\hat{\beta}_j = E_{\tilde{\Pi}}[\beta_j] = \gamma_j \mu_j$.
inclusion_prob	Posterior inclusion probabilities. Used to describe the posterior probability a coefficient is non-zero.
m	Final value for the means of the Gaussian component of the variational family μ.
s	Final value for the standard deviation of the Gaussian component of the variational family s.
g	Final value for the inclusion probability (γ).
lambda	Value of lambda used.
a0	Value of α_0 used.
b0	Value of β_0 used.
converged	Describes whether the algorithm converged.

Details

Rather than compute the posterior using MCMC, we turn to approximating it using variational inference. Within variational inference we re-cast Bayesian inference as an optimisation problem, where we minimize the Kullback-Leibler (KL) divergence between a family of tractable distributions and the posterior, Π.

In our case we use a mean-field variational family,

$$Q = \{\prod_{j=1}^{p} \gamma_j N(\mu_j, s_j^2) + (1 - \gamma_j)\delta_0\}$$

where μ_j is the mean and s_j the std. dev for the Gaussian component, γ_j the inclusion probabilities, δ_0 a Dirac mass at zero and p the number of coefficients.

The components of the variational family (μ, s, γ) are then optimised by minimizing the Kullback-Leibler divergence between the variational family and the posterior,

$$\tilde{\Pi} = \arg\min KL(Q \| \Pi).$$

We use co-ordinate ascent variational inference (CAVI) to solve the above optimisation problem.

Examples

```
n <- 125                        # number of sample
p <- 250                        # number of features
s <- 5                          # number of non-zero coefficients
censoring_lvl <- 0.25           # degree of censoring


# generate some test data
set.seed(1)
b <- sample(c(runif(s, -2, 2), rep(0, p-s)))
X <- matrix(rnorm(n * p), nrow=n)
Y <- log(1 - runif(n)) / -exp(X %*% b)
delta  <- runif(n) > censoring_lvl    # 0: censored, 1: uncensored
Y[!delta] <- Y[!delta] * runif(sum(!delta)) # rescale censored data


# fit the model
f <- survival.svb::svb.fit(Y, delta, X, mu.init=rep(0, p))


## Larger Example
n <- 250                        # number of sample
p <- 1000                       # number of features
s <- 10                         # number of non-zero coefficients
censoring_lvl <- 0.4            # degree of censoring


# generate some test data
set.seed(1)
b <- sample(c(runif(s, -2, 2), rep(0, p-s)))
X <- matrix(rnorm(n * p), nrow=n)
Y <- log(1 - runif(n)) / -exp(X %*% b)
delta  <- runif(n) > censoring_lvl    # 0: censored, 1: uncensored
Y[!delta] <- Y[!delta] * runif(sum(!delta)) # rescale censored data


# fit the model
f <- survival.svb::svb.fit(Y, delta, X)


# plot the results
plot(b, xlab=expression(beta), main="Coefficient value", pch=8, ylim=c(-2,2))
points(f$beta_hat, pch=20, col=2)
legend("topleft", legend=c(expression(beta), expression(hat(beta))),
       pch=c(8, 20), col=c(1, 2))
plot(f$inclusion_prob, main="Inclusion Probabilities", ylab=expression(gamma))
```

Index