Package ‘bayesCL’

April 14, 2017

Version 0.0.1

Date 2017-04-10

Title Bayesian Inference on a GPU using OpenCL

Author Rok Cesnovar, Erik Strumbelj

Maintainer Rok Cesnovar <rok.cesnovar@fri.uni-lj.si>

Description Bayesian Inference on a GPU. The package currently supports sampling from PolyaGamma, Multinomial logit and Bayesian lasso.

License GPL (>= 3)

Depends R (>= 2.14.0)

NeedsCompilation yes

SystemRequirements OpenCL library; single-precision AMD or Nvidia GPU;

Repository CRAN

Date/Publication 2017-04-14 21:38:23 UTC

RoxygenNote 6.0.1

R topics documented:

lasso	*Bayesian Lasso*

Description

Inference for Bayesian lasso regression models by Gibbs sampling from the Bayesian posterior distribution.

Usage

```
lasso(X, y, T=1000, lambda2=1, beta = NULL, s2=var(y-mean(y)),
          rd=NULL, ab=NULL, icept=TRUE,
          normalize=TRUE, device=0, parameters=NULL)
```

Arguments

`X`	`data.frame`, `matrix`, or vector of inputs `X`
`y`	vector of output responses `y` of length equal to the leading dimension (rows) of `X`, i.e., `length(y) == nrow(X)`
`T`	total number of MCMC samples to be collected
`beta`	initial setting of the regression coefficients.
`lambda2`	square of the initial lasso penalty parameter.
`s2`	initial variance parameter.
`rd`	`=c(r, delta)`, the alpha (shape) parameter and β (rate) parameter to the gamma distribution prior `G(r,delta)` for the λ^2 parameter under the lasso model. A default of `NULL` generates appropriate non-informative values depending on the nature of the regression.
`ab`	`=c(a, b)`, the α (shape) parameter and the β (scale) parameter for the inverse-gamma distribution prior `IG(a,b)` for the variance parameter `s2`. A default of `NULL` generates appropriate non-informative values depending on the nature of the regression.
`icept`	if `TRUE`, an implicit intercept term is fit in the model, otherwise the the intercept is zero; default is `TRUE`.
`normalize`	if `TRUE`, each variable is standardized to have unit L2-norm, otherwise it is left alone; default is `TRUE`.
`device`	If no external pointer is provided to function, we can provide the ID of the device to use.
`parameters`	a 9 dimensional vector of parameters to tune the GPU implementation.

Details

The Bayesian lasso model, hyperprior for the lasso parameter, and Gibbs Sampling algorithm implemented by this function are identical to that is described in detail in Park & Casella (2008). The GPU implementation is derived from the CPU implementation blasso from package monomvn.

Value

`lasso` returns an object of class `"lasso"`, which is a `list` containing a copy of all of the input arguments as well as of the components listed below.

`mu`	a vector of `T` samples of the (un-penalized) “intercept” parameter.
`beta`	a `T*ncol(X)` `matrix` of `T` samples from the (penalized) regression coefficients.
`s2`	a vector of `T` samples of the variance parameter
`lambda2`	a vector of `T` samples of the penalty parameter.
`tau2i`	a `T*ncol(X)` `matrix` of `T` samples from the (latent) inverse diagonal of the prior covariance matrix for `beta`, obtained for Lasso regressions.

See Also

`rpg`,`mlr`

Examples

```
set.seed(0)
n_samples  <- 500
n_features <- 40
X <- matrix(rnorm(n_features * n_samples), nrow = n_samples)
y <- 2 * X[,1] - 3 * X[,2] + rnorm(n_samples) # only features 1 & 2 are relevant

X_train <- X[1:400,]
y_train <- y[1:400]
X_test  <- X[401:500,]
y_test  <- y[401:500]


# START ------------------------------------------------------------------------

# first, standardize data !!!
X_train <- scale(X_train)

tmp00 <- bayesCL::lasso(X = X_train,
                          y = y_train,
                          T = 500,  # number of Gibbs sampling iterations
                          icept = T,
                          device=0  ) # use constant term (intercept), we do


#scale test data based on train data means and scales!!
X_test <- scale(X_test,
                center = attr(X_train, "scaled:center"),
                scale = attr(X_train, "scaled:scale"))


p_train1 <- colMeans(tmp00$beta %*% t(X_train))
p_test1 <- colMeans(tmp00$beta %*% t(X_test))

plot(y_train, p_train1, col = "red", xlab = "actual", ylab = "predicted")
points(y_test, p_test1, col = "green")
```

`mlr` *Bayesian Multinomial Logistic Regression*

Description

Inference for Bayesian multinomial logistic regression models by Gibbs sampling from the Bayesian posterior distribution.

Usage

```
mlr(y, X, n=rep(1,nrow(as.matrix(y))),
                    m.0=array(0, dim=c(ncol(X), ncol(y))),
                    P.0=array(diag(0, ncol(X)), dim=c(ncol(X),ncol(X),ncol(y))),
                    samp=1000, burn=500, float=0, device=0, parameters=NULL)
```

Arguments

`y`	an N x J-1 dimensional matrix; y_{ij} is the average response for category j at x_i.
`X`	an N x P dimensional design matrix; x_i is the ith row.
`n`	an N dimensional vector; n_i is the total number of observations at each x_i.
`m.0`	a P x J-1 matrix with the β_j's prior means.
`P.0`	a P x P x J-1 array of matrices with the β_j's prior precisions.
`samp`	the number of MCMC iterations saved.
`burn`	the number of MCMC iterations discarded.
`float`	a number representing the degree of precision to use: for single-precision floating point use 0, for or double-precision floating point use 1.
`device`	if no external pointer is provided to function, we can provide the ID of the device to use.
`parameters`	a 9 dimensional vector of parameters to tune the GPU implementation.

Details

Classic multinomial logistic regression for classifiction.

We assume that $\beta_J = 0$ for purposes of identification.

Value

mlr returns a list.

beta	a samp x P x J-1 array; the posterior sample of the regression coefficients.
w	a samp x N' x J-1 array; the posterior sample of the latent variable. WARNING: N' may be less than N if data is combined.
y	the response matrix–different than input if data is combined.
X	the design matrix–different than input if data is combined.
n	the number of samples at each observation–different than input if data is combined.

See Also

rpg,lasso

Examples

```
## Use the iris dataset.
data(iris)
N <- nrow(iris)
P <- ncol(iris)
J <- nlevels(iris$Species)

X     <- model.matrix(Species ~ ., data=iris);
y.all <- model.matrix(~ Species - 1, data=iris);
y     <- y.all[,-J];

out <- mlr(y, X, samp=1000, burn=100, device=0);
```

---

prepare *GPU preparation for PolyaGamma sampling and/or Bayesian Inference*

Description

Generates the external pointer to the GPU. This function compiles the OpenCL code, creates the command queue, etc. It can be used in order to avoid compilation/creation in each call of the rpg, mlr, and lasso.

Usage

```
prepare(precision=0,device=-1, parameters=NULL )
```

Arguments

`precision`	the number of random variates to simulate.
`device`	the ID of the device for which to generate the helper variables.
`parameters`	a 9 dimensional vector of parameters to tune the GPU implementation.

Details

This is used in order to avoid unnecesarry recompilation of OpenCL kernel and creation of contexts, command queues, etc.. The output of this function is a pointer that can be passed to the mlr, lasso and rpg functions. If the pointer is not passed to these functions, the prepare function is called from inside the mlr/lasso/rpg functions in each call. If no device number is specified, a list of devices with their respective IDs will be shown and you will be prompted to enter a number. In order to tune the implementation you can specify your own values for implementation parameters, which is a 9 dimensional vector.

Value

This function returns an external pointer to a C structure for the GPU.

See Also

`rpg`,`lasso`,`mlr`

Examples

```
  gpu_pointer <- prepare(precision=0, device=0)
```

`rpg` *Polya-Gamma Random Variates using a GPU*

Description

Generate random variates from the Polya-Gamma distribution on a GPU.

Usage

```
rpg(num=1, n=1, z=0.0, batch=32, local=128, staticseed=FALSE,
seed=0, float=0, ptr=NULL, device=0)
```

Arguments

`num`	the number of random variates to simulate.
`n`	shape parameter, a positive integer.
`z`	parameter associated with tilting.
`batch`	the number of samples created by each GPU thread
`local`	the number of threads in a thread-block on a GPU
`staticseed`	parameter to determine whether to use a static seed or not.
`seed`	the value of the static seed, if used.
`float`	parameter to determine whether to use single-precision floating point or double-precision floating point.
`ptr`	an external pointer to the C structure with the GPU helper variables.
`device`	if no external pointer is provided to function, we can provide the ID of the device to use.

Details

A random variable X with distribution PG(n,z) is distributed like

$$X \sim \sum_{k=1}^{\infty} G(n,1)/(2\pi^2(k-1/2)^2 + z^2/2).$$

The density for X may be derived by exponentially tilting the PG(n,0) density:

$$p(x|n,z) \propto \exp(-xz^2/2)p(x|n,0).$$

The GPU implementation is derived from the CPU implementation rpg.devroye from package BayesLogit.

Value

This function returns `num` Polya-Gamma samples.

See Also

`prepare`,`mlr`

Examples

```
random_variates <- rpg(num=100, n=1, z=0.0, device=0)
```

Index