Package 'BayesValidate'

February 19, 2015

Version 0.0

Date 2005-06-25

Title BayesValidate Package

Author Samantha Cook <cook@stat.columbia.edu>.

Maintainer Samantha Cook <cook@stat.columbia.edu>

Depends R (>= 2.0.1)

Description BayesValidate implements the software validation method
described in the paper ``Validation of Software for Bayesian
Models using Posterior Quantiles'' (Cook, Gelman, and Rubin,
2005). It inputs a function to perform Bayesian inference as
well as functions to generate data from the Bayesian model
being fit, and repeatedly generates and analyzes data to check
that the Bayesian inference program works properly.

License GPL (>= 2)

Repository CRAN

Date/Publication 2006-03-30 09:22:53

NeedsCompilation no

R topics documented:

---

quant   *Calculate empirical quantile of the first entry in a vector*

Description

quant inputs a vector and returns the empirical quantile of the first argument in the vector with respect to all entries in the vector. Used as part of the function validate for Bayesian software validation, this function is used to calculate the empirical quantile of a "true" parameter value with respect to a collection of posterior draws of that parameter.

Usage

```
quant(draws)
```

Arguments

draws   Vector of parameter draws, with entry of interest, i.e., the value whose quantile is being calculated, at the beginning.

Details

Calculates the rank of the first entry of the vector with respect to all other entries, subtracts .5, and divides by the length of the vector.

Value

The empirical quantile of the first entry of the vector, a scalar between 0 and 1.

Author(s)

Samantha Cook <cook@stat.columbia.edu>

Examples

```
set.seed(314)
x<-rnorm(1000)
quant(x)
```

---

validate   *Tests correctness of Bayesian Model-Fitting Software*

Description

Inputs functions to generate and analyze data. Compares output from these functions to test that the model-fitting software works correctly.

Usage

```
validate(generate.param, generate.param.inputs = NULL, generate.data,
generate.data.inputs = NULL, analyze.data, analyze.data.inputs = NULL,
n.rep = 20, n.batch = NULL, params.batch = NULL, print.reps = FALSE)
```

Arguments

generate.param   Function for generating parameters from prior distribution Should output a vector of parameters. Function should look like generate.param <- function() {...} or generate.param <- function(generate.param.inputs) {...}

generate.param.inputs

    Inputs to the function generate.param

generate.data   Function for generating data given parameters. Should take as input the output from generate.param. Should output data matrix to be analyzed. Function should look like generate.data <- function(theta.true) {...} or generate.data <- function(theta.true, generate.data.inputs) {...}

generate.data.inputs

    Inputs to the function generate.data (in addition to theta.true)

analyze.data   Function for generating sample from posterior distribution. Should take as input the output from generate.data and generate.param. Should output a matrix of parameters, each row is one parameter draw. Function should look like analyze.data <- function(data.rep,theta.true) {...} or analyze.data <- function(data.rep,theta.true, analyze.data.inputs) {...}. ! This is the software being tested !!

analyze.data.inputs

    Inputs to the function analyze.data (in addition to data.rep and theta.true)

n.rep   Number of replications to be performed, default is 20.

n.batch   Lengths of parameter batches. A parameter batch might consist of, for example, all person-level means in a hierarchical model or any group of parameters that is sampled in a loop. Must sum to n.param (length of parameter vector) and correspond to the order of parameters as they are output from analyze.data. For example, if there are 5 total parameters with the first two in one batch and the next three in another batch, use n.batch=c(2,3)

params.batch   Names of parameter batches, used as the y axis in the output plot. Must have length equal to the number of batches. Can consist of text (e.g., params.batch=c("alpha","beta")) or an expression (e.g., params.batch=expression(alpha,beta)). Not used if n.batch not provided.

print.reps   Indcator of whether or not to print the replication number, default is FALSE

Details

Validate tests whether software developed to fit a specific Bayesian model works properly, capitalizing on properties of Bayesian posterior distributions. The validation method involves repeatedly generating parameters and data from the model to be fit and then fitting the same model to these simulated data (i.e., generating a sample from the posterior distribution). For all scalar parameters, the quantile of the "true" parameter value with respect to its posterior distribution should follow a uniform distribution if the software is written correctly. Testing that the software works amounts to testing that these quantiles are uniformly distributed. For each scalar parameter, the function gives a p-value for a test that its quantiles are uniformly distributed.

Value

p.vals	p-values for "unbatched" parameters. Null if all batches consist of only one parameter.
p.batch	p-values for the means of batched parameters. Null if n.batch not provided
adj.min.p	The smallest of p.batch (or, if p.batch=NULL, the smallest of p.vals), with the Bonferroni correction for multiple comparisons applied.

Author(s)

Samantha Cook <cook@stat.columbia.edu>

References

http://www.stat.columbia.edu/~cook/Cook_Software_Validation.pdf

See Also

quant

Examples

```
set.seed(314)

##functions for generating parameters mu, sigma^2 from their prior distribution
rinvchisq <- function(n,v,s){
        alpha <- v/2
        beta <- alpha*s;
        draws <- 1/rgamma(n,alpha,beta)
        return(draws)}

generate.param <- function(hyper){
mu.0 <- hyper[1]
kappa.0 <- hyper[2]
nu.0 <- hyper[3]
sigsq.0 <- hyper[4]
sigsq <- rinvchisq(1, nu.0, sigsq.0)
mu <- rnorm(1, mu.0, sqrt(sigsq/kappa.0))
return(c(sigsq,mu))}

##generate normal data with mean mu, variance sigma^2, sample size n
generate.data <- function(params,n){
y <- rnorm(n,params[2],sqrt(params[1]))
return(y)}

##generate from the posterior distribution of mu, sigma^2
analyze.data <- function(y,params.true,inputs){
n <- length(y)
mu.0 <- inputs[1]
kappa.0 <- inputs[2]
nu.0 <- inputs[3]
sigsq.0 <- inputs[4]
```

```
n.draws <- inputs[5]
kappa.n <- kappa.0 + n
mu.n <- (mu.0*kappa.0 + sum(y))/kappa.n
nu.n <- nu.0 + n
sigsq.n <- ( nu.0*sigsq.0 + (n-1)*var(y) +
kappa.0*n*(mean(y) - mu.0)^2/kappa.n ) / nu.n


sigsq.post <- rinvchisq(n.draws, nu.n, sigsq.n)

var.mu.post <- sigsq.post/(kappa.n)
mu.post <- rnorm(n.draws, mu.n, sqrt(var.mu.post))

return(cbind(sigsq.post,mu.post))}

##generate from the posterior distribution of mu, sigma^2
##error sampling sigma^2
analyze.data.error1 <- function(y,params.true,inputs){
n <- length(y)
mu.0 <- inputs[1]
kappa.0 <- inputs[2]
nu.0 <- inputs[3]
sigsq.0 <- inputs[4]
n.draws <- inputs[5]
kappa.n <- kappa.0 + n
mu.n <- (mu.0*kappa.0 + sum(y))/kappa.n
nu.n <- nu.0 + n
sigsq.n <- ( nu.0*sigsq.0 + (n-1)*var(y) +
kappa.0*n*(mean(y) - mu.0)^2/kappa.n ) / nu.n


sigsq.post <- rinvchisq(n.draws, nu.n, sigsq.0)

var.mu.post <- sigsq.post/(kappa.n)
mu.post <- rnorm(n.draws, mu.n, sqrt(var.mu.post))

return(cbind(sigsq.post,mu.post))}

##generate from the posterior distribution of mu, sigma^2
##error sampling mu
analyze.data.error2 <- function(y,params.true,inputs){
n <- length(y)
mu.0 <- inputs[1]
kappa.0 <- inputs[2]
nu.0 <- inputs[3]
sigsq.0 <- inputs[4]
n.draws <- inputs[5]
kappa.n <- kappa.0 + n
mu.n <- (mu.0*kappa.0 + sum(y))/kappa.n
nu.n <- nu.0 + n
sigsq.n <- ( nu.0*sigsq.0 + (n-1)*var(y) +
kappa.0*n*(mean(y) - mu.0)^2/kappa.n ) / nu.n
```

```
sigsq.post <- rinvchisq(n.draws, nu.n, sigsq.n)

var.mu.post <- sigsq.post/(kappa.n)
mu.post <- rnorm(n.draws, mu.n, var.mu.post)

return(cbind(sigsq.post,mu.post))}

##function inputs
hyper<-c(6,5,5,7)
n<-20
n.draws<-5000

generate.param.inputs<-hyper
generate.data.inputs<-n
analyze.data.inputs<-c(hyper,n.draws)

##run validation function for the three model-fitting functions
tst.0 <- validate(generate.param = generate.param, generate.param.inputs =
generate.param.inputs, generate.data = generate.data,
generate.data.inputs = generate.data.inputs, analyze.data =
analyze.data, analyze.data.inputs = analyze.data.inputs,
n.rep = 20, params.batch = expression(sigma^2,mu), n.batch = c(1,1))


tst.1 <- validate(generate.param = generate.param, generate.param.inputs =
generate.param.inputs, generate.data = generate.data,
generate.data.inputs = generate.data.inputs, analyze.data =
analyze.data.error1, analyze.data.inputs = analyze.data.inputs,
n.rep = 20, params.batch = expression(sigma^2,mu), n.batch = c(1,1))


tst.2 <- validate(generate.param = generate.param, generate.param.inputs =
generate.param.inputs, generate.data = generate.data,
generate.data.inputs = generate.data.inputs, analyze.data =
analyze.data.error2, analyze.data.inputs = analyze.data.inputs,
n.rep = 20, params.batch = expression(sigma^2,mu), n.batch = c(1,1))
```

Index