Package 'optismixture'

August 25, 2015

Title Optimal Mixture Weights in Multiple Importance Sampling

Description Code for optimal mixture weights in importance sampling. Workhorse functions penoptpersp() and penoptpersp.alpha.only() minimize estimated variances with and without control variates respectively. It can be used in adaptive mixture importance sampling, for example, function batch.estimation() does two stages, a pilot estimate of mixing alpha and a following importance sampling.

Version 0.1

Author Hera Y. He, Art B. Owen

Maintainer Hera Y. He <yhe1@stanford.edu>

Depends R (>= 3.0.2)

Imports mvtnorm, Matrix

License GPL-2

LazyData true

NeedsCompilation no

Repository CRAN

Date/Publication 2015-08-25 19:20:56

R topics documented:

`alpha2N` *Internal function. convert mixture proportions to mixture sample size with a fixed total sample size*

Description

Internal function. convert mixture proportions to mixture sample size with a fixed total sample size

Usage

```
alpha2N(n, alpha)
```

Arguments

`n`	total number of samples
`alpha`	the vector of mixture proportions

Value

the vector of sample sizes for each mixture component

`batch.estimation` *Two stage estimation, a pilot estimate of mixing alpha and a following importance sampling, with or without control variates*

Description

Two stage estimation, a pilot estimate of mixing alpha and a following importance sampling, with or without control variates

Usage

```
batch.estimation(seed, batch.size, mixture.param,
  eps = rep(0.1/mixture.param$J, mixture.param$J), fname = "f",
  rpname = "rp", rqname = "rq", dpname = "dp", dqname = "dq",
  cv = TRUE, opt.info = FALSE, opt.param = list(reltol = 0.001, relerr =
  0.001, rho0 = 1, maxin = 20, maxout = 30))
```

Arguments

seed	seed for sampling
batch.size	length two vector of batch sizes
mixture.param	mixture.param = list(p, J, ...), where p is the dimension of the sample, and J is the number of mixture components, including the defensive one. mixture.param should be compatible with user defined functions *f(n, j, mixture.param), rp(n, mixture.param), rq(n, j, mixture.param), dp(xmat, mixture.param), dq(xmat, j, mixture.param)*
eps	the lower bound for optimizing *alpha*. if *eps* is of length 1, it is expanded to *rep(eps, J)*, default to be *rep(0.1/J, J)*
fname	name of user defined function *fname(xmat, j, mixture.param)*. *xmat* is an $n \times p$ matrix of n samples with p dimensions. *fname* returns a vector of function values for each row in *xmat*. *fname* is defined for $j = 1, \cdots, J$. $j = 1, \cdots, J-1$ corresponds to different proposal mixture components, and $j = J$ corresponds to the defensive mixture component.
rpname	name of user definded function *rpname(n, mixture.param)*. It generates n random samples from target distribution *pname*. Parameters can be specified in *mixture.param*. *rpname* returns an $n \times p$ matrix.
rqname	name of user defined function *rqname(n, j, mixture.param)*. It generate n random samples from the j^{th} mixture component of proposal mixture distribution. *rqname* returns an $n \times p$ matrix. *rqname* is defined for $j = 1, \cdots, J-1$.
dpname	name of user defined function *dpname(xmat, mixture.param)*. It returns the density of *xmat* from the target distribution $pname$ as a vector of length *nrow(xmat)*. Note that only the ratio between *dpname* and *dqname* matters. So *dpname* and *dqname* can return values of $C \times$*dpname* and $C \times$*dqname* respectively.
dqname	name of user defined function *dqname(xmat, j, mixture.param)*. It returns the density of *xmat* from the proposal distribution q_j as a vector of length *nrow(xmat)*. *dqname* is defined for $j = 1, \cdots, J-1$. Note that only the ratio between *dpname* and *dqname* matters. So *dpname* and *dqname* can return values of $C \times$*dpname* and $C \times$*dqname* respectively.
cv	TRUE indicates optimizing *alpha* and *beta* at the same time, and estimate with the formula $$\hat{\mu}_{\alpha_{**},\beta} = \frac{1}{n_2} \sum_{i=1}^{n_2} \frac{f(x_i)p(x_i) - \beta^{\mathsf{T}}\big(\boldsymbol{q}(x_i) - p(x_i)\mathbf{1}\big)}{q_{\alpha_{**}}(x_i)}.$$ FALSE indicates optimizing *alpha* only, and estimate with the formula $$\hat{\mu}_{\alpha_*} = \frac{1}{n_2} \sum_{i=1}^{n_2} \frac{f(x_i)p(x_i)}{q_{\alpha_*}(x_i)}.$$
opt.info	logical value indicating whether to save the returned value of the optimization procedure. See penoptpersp and penoptpersp.alpha.only for the returned value.
opt.param	a list of paramters for the damped Newton method with backtracking line search

reltol relative tolerence in dampednewton step, default to be 10^{-2}

relerr Newton step stop when within (1+*relerr*) of minimum variance, default to be 10^{-3}

rho0 initial value for *rho*, default to be 1

Only need to supply part of the list to change the default value.

Details

Estimate $E_p f$ with a two step importance sample procedure. See He & Owen(2014) for details.

Value

a list of

mu.hat the estimate for mu

sd.hat estimated sd of mu.hat

alpha.opt the estimated optimal alpha

beta.opt if cv = TRUE, the estimated optimal beta

opt.info if opt.info = TRUE, also return the list(x=x, y=y, z=z, alpha=alpha,beta=beta,rho=rho,f=f rhopen=rhopen, outer=log(rho0/rho,mu), relerr = relerr, alphasum = sum(alpha)) from the optimization after batch 1.

References

Boyd, S. and Vandeberghe, L. (2004). *Convex optimization.* Cambridge University Press, Cambridge

Examples

```
library(optismixture)
seed <- 1
p <- 5
rho <- 1/2
gamma <- 2.4
sigma.dvar <- function(rho, p){
   sigma <- matrix(0, p, p)
   for(i in 1:(p-1)){
       for(j in (i+1):p){
           sigma[i,j] <- rho^(abs(i-j))
       }
   }
   sigma <- sigma + t(sigma)
   diag(sigma) <- 1
   return(sigma)
}
sigma <- sigma.dvar(rho, p)
batch.size <- c(10^4, 1002)
j.vec <- 2^-(seq(1,5,1))
eps.status <- 1
```

```
eps.safe <- 0.1
## initialization and construct derived parameters
mu <- rep(0, p)
x0 <- matrix(1, 1, p)
x0.mat <- rbind(rep(1,p), rep(-1, p))
j.mat <- data.frame(centerid = rep(1:dim(x0.mat)[1], each = length(j.vec)),
                    variance = rep(j.vec, 2))
J <- dim(j.mat)[1] + 1
eps <- rep(0.1/J, J)
mixture.param <- list(x0 = x0, x0.mat = x0.mat, p = p,
sigma = sigma, gamma = gamma, j.mat = j.mat, J = J)
f <- function(x, j, mixture.param){
    f1 <- function(x, mixture.param){
        x0 <- mixture.param$x0
        gamma <- mixture.param$gamma
        return(sum((x - x0)^2)^(-gamma/2))
    }
    return(apply(x, 1, f1, mixture.param))
}
dq <- function(x, j, mixture.param){
    centerid <- mixture.param$j.mat[j, 1]
    j.param <- mixture.param$j.mat[j, 2]
   return(mvtnorm::dmvnorm(x, mixture.param$x0.mat[centerid,], j.param*diag(mixture.param$p)))
}
dp <- function(x, mixture.param){
    return(mvtnorm::dmvnorm(x, rep(0, mixture.param$p), mixture.param$sigma))
}
rq <- function(n, j, mixture.param){
    centerid <- mixture.param$j.mat[j, 1]
    j.param <- mixture.param$j.mat[j,2]
   return(mvtnorm::rmvnorm(n, mixture.param$x0.mat[centerid, ], j.param*diag(mixture.param$p)))
}
rp <- function(n, mixture.param){
    mu <- rep(0, mixture.param$p)
    sigma <- mixture.param$sigma
    return(mvtnorm::rmvnorm(n, mu, sigma))
}
a <- batch.estimation(seed, batch.size, mixture.param, eps, cv = FALSE,
fname = "f", rpname = "rp", rqname = "rq", dpname = "dp", dqname = "dq")
```

`compatible.test` *Test the compatibility of user defined functions* fname, rpname, rqname, dpname, dqname *with* mixture.param

---

Description

Test the compatibility of user defined functions *fname, rpname, rqname, dpname, dqname* with *mixture.param*

Usage

```
compatible.test(fname, rpname, dpname, rqname, dqname, mixture.param)
```

Arguments

fname	name of user defined function *fname(xmat, j, mixture.param)*. *xmat* is an $n \times p$ matrix of n samples with p dimensions. *fname* returns a vector of function values for each row in *xmat*. *fname* is defined for $j = 1, \cdots, J$. $j = 1, \cdots, J-1$ corresponds to different proposal mixture components, and $j = J$ corresponds to the defensive mixture component.
rpname	name of user definded function *rpname(n, mixture.param)*. It generates n random samples from target distribution *pname*. Parameters can be specified in *mixture.param*. *rpname* returns an $n \times p$ matrix.
dpname	name of user defined function *dpname(xmat, mixture.param)*. It returns the density of *xmat* from the target distribution $pname$ as a vector of length *nrow(xmat)*. Note that only the ratio between *dpname* and *dqname* matters. So *dpname* and *dqname* can return values of $C \times$*dpname* and $C \times$*dqname* respectively.
rqname	name of user defined function *rqname(n, j, mixture.param)*. It generate n random samples from the j^{th} mixture component of proposal mixture distribution. *rqname* returns an $n \times p$ matrix. *rqname* is defined for $j = 1, \cdots, J-1$.
dqname	name of user defined function *dqname(xmat, j, mixture.param)*. It returns the density of *xmat* from the proposal distribution q_j as a vector of length *nrow(xmat)*. *dqname* is defined for $j = 1, \cdots, J-1$. Note that only the ratio between *dpname* and *dqname* matters. So *dpname* and *dqname* can return values of $C \times$*dpname* and $C \times$*dqname* respectively.
mixture.param	mixture.param = list(p, J, ...), where p is the dimension of the sample, and J is the number of mixture components, including the defensive one. mixture.param should be compatible with user defined functions *f(n, j, mixture.param), rp(n, mixture.param), rq(n, j, mixture.param), dp(xmat, mixture.param), dq(xmat, j, mixture.param)*

Value

Stop with error or print the success message.

---

do.mixture.sample *Internal function. sample from the mixture distribution q_α*

Description

Internal function. sample from the mixture distribution q_α

Usage

```
do.mixture.sample(seed, b, n, J, mixture.param, alpha, fname, rpname, rqname,
  dpname, dqname)
```

Arguments

seed	seed for sampling
b	batch index for the samples
n	total number of samples
J	number of mixture components, including the defensive one
mixture.param	mixture.param = list(p, J, ...), where p is the dimension of the sample, and J is the number of mixture components, including the defensive one. mixture.param should be compatible with user defined functions *f(n, j, mixture.param), rp(n, mixture.param), rq(n, j, mixture.param), dp(xmat, mixture.param), dq(xmat, j, mixture.param)*
alpha	vector of mixture proportions
fname	name of user defined function *fname(xmat, j, mixture.param)*. *xmat* is an $n \times p$ matrix of n samples with p dimensions. *fname* returns a vector of function values for each row in *xmat*. *fname* is defined for $j = 1, \cdots, J$. $j = 1, \cdots, J-1$ corresponds to different proposal mixture components, and $j = J$ corresponds to the defensive mixture component.
rpname	name of user definded function *rpname(n, mixture.param)*. It generates n random samples from target distribution *pname*. Parameters can be specified in *mixture.param*. *rpname* returns an $n \times p$ matrix.
rqname	name of user defined function *rqname(n, j, mixture.param)*. It generate n random samples from the j^{th} mixture component of proposal mixture distribution. *rqname* returns an $n \times p$ matrix. *rqname* is defined for $j = 1, \cdots, J-1$.
dpname	name of user defined function *dpname(xmat, mixture.param)*. It returns the density of *xmat* from the target distribution $pname$ as a vector of length *nrow(xmat)*. Note that only the ratio between *dpname* and *dqname* matters. So *dpname* and *dqname* can return values of $C \times$*dpname* and $C \times$*dqname* respectively.
dqname	name of user defined function *dqname(xmat, j, mixture.param)*. It returns the density of *xmat* from the proposal distribution q_j as a vector of length *nrow(xmat)*. *dqname* is defined for $j = 1, \cdots, J-1$. Note that only the ratio between *dpname* and *dqname* matters. So *dpname* and *dqname* can return values of $C \times$*dpname* and $C \times$*dqname* respectively.

Value

a list of

x the matrix(size $n \times p$) of samples from the mixture distribution q_α

fx the vector of *fname* evaluated at sample matrix *x*

qx the matrix of densities under each mixture component, i.e. *qx[,j]* is the density under q_j. The defensive component is the J^{th} column

px the vector of densities under proposal distribution *pname*

alpha the rounded *alpha* used for sampling, i.e. $alpha2N(n, alpha)/n$

`do.plain.mc` *Do plain monte carlo with target density*

Description

Do plain monte carlo with target density

Usage

```
do.plain.mc(plainmc.N, mixture.param, fname = "f", rpname = "rp")
```

Arguments

`plainmc.N`	number of samples
`mixture.param`	mixture.param = list(p, J, ...), where p is the dimension of the sample, and J is the number of mixture components, including the defensive one. mixture.param should be compatible with user defined functions *f(n, j, mixture.param), rp(n, mixture.param), rq(n, j, mixture.param), dp(xmat, mixture.param), dq(xmat, j, mixture.param)*
`fname`	name of user defined function *fname(xmat, j, mixture.param)*. *xmat* is an $n \times p$ matrix of n samples with p dimensions. *fname* returns a vector of function values for each row in *xmat*. *fname* is defined for $j = 1, \cdots, J$. $j = 1, \cdots, J-1$ corresponds to different proposal mixture components, and $j = J$ corresponds to the defensive mixture component.
`rpname`	name of user definded function *rpname(n, mixture.param)*. It generates n random samples from target distribution *pname*. Parameters can be specified in *mixture.param*. *rpname* returns an $n \times p$ matrix.

Value

a list of

plainmc.N number of samples for the plain monte carlo

mu.hat estimated $E_p f$ from plain monte carlos samples

sd.hat estimated sd for *mu.hat*

`get.index.b` *Internal function. Get the row index in the stacked sample matrices for the $b\hat{}th$ batch*

Description

Internal function. Get the row index in the stacked sample matrices for the b^{th} batch

Usage

```
get.index.b(batch.size, b)
```

Arguments

`batch.size`	vector of batch sizes
`b`	the index of the batch of interest

Details

In the computation, samples from each batches are stacked be rows to form x.mat, as well as fx.mat, px.mat, qx.mat. To extract the samples from batch b and their corresponding fx, px, qx values, the indices for batch b will be needed. They can be obtained by calling get.index.b(batch.size, b)

Value

the row index or sample matrices of batch b for all b's in b.vec

`get.initial.alpha` *Internal function. Calculate the initial alpha vector for the optimization of* alpha *with a lower bound constraint*

Description

Internal function. Calculate the initial alpha vector for the optimization of *alpha* with a lower bound constraint

Usage

```
get.initial.alpha(eps, J)
```

Arguments

`eps`	a vector of lower bound values for vector *alpha*
`J`	number of mixture components

Value

The initial alpha vector for optimization

`get.var` *Internal function. With stratified samples, calculate the variance of the estimate from importance sampling without control variates*

Description

Internal function. With stratified samples, calculate the variance of the estimate from importance sampling without control variates

Usage

```
get.var(Y, nvec)
```

Arguments

`Y`	vector of stratified samples of length n. i.e. $Y_1 = Y[1 : nvec[1]]$ are sampled from q_1, $Y_i = Y[(nvec[i-1]+1) : nvec[i]]$ are sample from q_i.
`nvec`	the vector of number of samples from each mixture component. It sums up to n.

Details

Suppose we sample Y from a mixture $q_\alpha = \alpha_1 * q_1 + ... + \alpha_J * q_J$. To estimate $\mathrm{mean}(Y)$, fixing the number of samples from each mixture component and getting a stratified sample would reduce the variance of the estimate. The formula for $\mathrm{Var}(\hat{\mu})$ with stratified samples is

$$\mathrm{Var}(\hat{\mu}) = 1/n \times \sum_{j=1}^{J} \alpha_j \mathrm{Var}(Y_j)$$

Value

the variance estimate of $\hat{\mu} = 1/n \sum_{i=1}^{n} Y[i]$

`mixture.is.estimation` *For a given mixture weight alpha, use importance sample with or without control variates for estimation*

Description

For a given mixture weight alpha, use importance sample with or withour control variates for estimation

Usage

```
mixture.is.estimation(seed, N, mixture.param, alpha, fname = "f",
  rpname = "rp", rqname = "rq", dpname = "dp", dqname = "dq",
  cv = TRUE)
```

Arguments

seed	seed for sampling
N	sample size
mixture.param	mixture.param = list(p, J, ...), where p is the dimension of the sample, and J is the number of mixture components, including the defensive one. mixture.param should be compatible with user defined functions *f(n, j, mixture.param), rp(n, mixture.param), rq(n, j, mixture.param), dp(xmat, mixture.param), dq(xmat, j, mixture.param)*
alpha	the mixture weight, sum up to 1
fname	name of user defined function *fname(xmat, j, mixture.param)*. *xmat* is an $n \times p$ matrix of n samples with p dimensions. *fname* returns a vector of function values for each row in *xmat*. *fname* is defined for $j = 1, \cdots, J$. $j = 1, \cdots, J-1$ corresponds to different proposal mixture components, and $j = J$ corresponds to the defensive mixture component.
rpname	name of user definded function *rpname(n, mixture.param)*. It generates n random samples from target distribution *pname*. Parameters can be specified in *mixture.param*. *rpname* returns an $n \times p$ matrix.
rqname	name of user defined function *rqname(n, j, mixture.param)*. It generate n random samples from the j^{th} mixture component of proposal mixture distribution. *rqname* returns an $n \times p$ matrix. *rqname* is defined for $j = 1, \cdots, J-1$.
dpname	name of user defined function *dpname(xmat, mixture.param)*. It returns the density of *xmat* from the target distribution $pname$ as a vector of length *nrow(xmat)*. Note that only the ratio between *dpname* and *dqname* matters. So *dpname* and *dqname* can return values of $C \times$*dpname* and $C \times$*dqname* respectively.
dqname	name of user defined function *dqname(xmat, j, mixture.param)*. It returns the density of *xmat* from the proposal distribution q_j as a vector of length *nrow(xmat)*. *dqname* is defined for $j = 1, \cdots, J-1$. Note that only the ratio between *dpname* and *dqname* matters. So *dpname* and *dqname* can return values of $C \times$*dpname* and $C \times$*dqname* respectively.
cv	TRUE indicates optimizing *alpha* and *beta* at the same time, and estimate with the formula $$\hat{\mu}_{\alpha_{**},\beta} = \frac{1}{n_2} \sum_{i=1}^{n_2} \frac{f(x_i)p(x_i) - \beta^{\mathsf{T}}\big(\boldsymbol{q}(x_i) - p(x_i)\mathbf{1}\big)}{q_{\alpha_{**}}(x_i)}.$$ FALSE indicates optimizing *alpha* only, and estimate with the formula $$\hat{\mu}_{\alpha_*} = \frac{1}{n_2} \sum_{i=1}^{n_2} \frac{f(x_i)p(x_i)}{q_{\alpha_*}(x_i)}.$$

`optismixture` *Optimal Mixture Weights in Multiple Importance Sampling*

Description

Workhorse functions `penoptpersp` and `penoptpersp.alpha.only` minimize estimated variances with and without control variates respectively. It can be used in adaptive mixture importance sampling, for example, function `batch.estimation` does a two-stage estimation, a pilot estimate of mixing α and a following importance sampling estimation.

`penoptpersp` *penalized optimization of the constrained linearized perspective function*

Description

penalized optimization of the constrained linearized perspective function

Usage

```
penoptpersp(x, y, z, a0 = NULL, b0 = NULL, eps = NULL, reltol = NULL,
  relerr = NULL, rho0 = NULL, maxin = NULL, maxout = NULL)
```

Arguments

`x`	$n \times K$ matrix
`y`	length n vector
`z`	$n \times J$ matrix
`a0`	length J vector
`b0`	length K vector
`eps`	length J vector, default to be rep(0.1/J, J)
`reltol`	relative tolerence for Newton step, between 0 to 1, default to be 10^{-3}. For each inner loop, we optimize $f_0 + \rho \times \text{pen}$ for a fixed ρ, we stop when the Newton decrement $f(x) - inf_y \hat{f}(y) \leq f(x) * \text{reltol}$, where $\hat{f}$ is the second-order approximation of f at x
`relerr`	stop when within (1+*relerr*) of minimum variance, default to be 10^{-3}, between 0 to 1.
`rho0`	initial value for ρ, default to be 1
`maxin`	maximum number of inner iterations
`maxout`	maximum number of outer iterations

Details

To minimize $\sum_i \frac{(y_i - x_i^T\beta)^2}{z_i^T\alpha}$ over α and β, subject to $\alpha_j > \epsilon_j$ for $j = 1, \cdots, J$ and $\sum_{j=1}^J \alpha_j < 1$,

Instead we minimize $\sum_i \frac{(y_i - x_i^T\beta)^2}{z_i^T\alpha} + \rho \times \text{pen}$ for a decreasing sequence of ρ

where $\text{pen} = -(\sum_{j=1}^J(\log(\alpha_j - \epsilon_j)) + \log(1 - \sum_{j=1}^J \alpha_j))$

starting values are $\alpha = a0$ and $\beta = b0$. They can be missing.

The optimization stops when within (1+*relerr*) of minimum variance.

Value

a list of

x input x

y input y

z input z

alpha optimized alpha

beta optimized beta

rho value of rho

f value of the objective function

rhopen value of rho*pen when returned

outer number of outer loops

relerr relative error

alphasum sum of optimized alpha

`penoptpersp.alpha.only` *penalized optimization of the constrained linearized perspective function*

Description

penalized optimization of the constrained linearized perspective function

Usage

```
penoptpersp.alpha.only(y, z, a0, eps = NULL, reltol = NULL, relerr = NULL,
  rho0 = NULL, maxin = NULL, maxout = NULL)
```

Arguments

y	length n vector
z	$n \times J$ matrix
a0	length J vector
eps	length J vector, default to be rep(0.1/J, J)
reltol	relative tolerence for Newton step, between 0 to 1, default to be 10^{-3}. For each inner loop, we optimize $f_0 + \rho \times \text{pen}$ for a fixed ρ, we stop when the Newton decrement $f(x) - inf_y \hat{f}(y) \leq f(x) * \text{reltol}$, where $\hat{f}$ is the second-order approximation of f at x
relerr	relerr stop when within (1+*relerr*) of minimum variance, default to be 10^{-3}, between 0 to 1.
rho0	initial value for ρ, default to be 1
maxin	maximum number of inner iterations
maxout	maximum number of outer iterations

Details

To minimize $\sum_i \frac{y_i^2}{z_i^T \alpha}$ over α subject to $\alpha_j > \epsilon_j$ for $j = 1, \cdots, J$ and $\sum_{j=1}^J \alpha_j < 1$,

Instead we minimize $\sum_i \frac{y_i^2}{z_i^T \alpha} + \rho \times \text{pen}$ for a decreasing sequence of ρ

where $\text{pen} = -(\sum_{j=1}^J (\log(\alpha_j - \epsilon_j)) + \log(1 - \sum_{j=1}^J \alpha_j))$

starting values are $\alpha = a0$ and can be missing.

The optimization stops when within (1+*relerr*) of minimum variance.

Value

a list of

y input y

z input z

alpha optimized alpha

rho value of rho

f value of the objective function

rhopen value of rho*pen when returned

outer number of outer loops

relerr relative error

alphasum sum of optimized alpha

Index