Package ‘fasta’

October 13, 2022

Type Package

Title Fast Adaptive Shrinkage/Thresholding Algorithm

Version 0.1.0

Description A collection of acceleration schemes for proximal gradient methods for estimating penalized regression parameters described in Goldstein, Studer, and Baraniuk (2016) <arXiv:1411.3406>. Schemes such as Fast Iterative Shrinkage and Thresholding Algorithm (FISTA) by Beck and Teboulle (2009) <doi:10.1137/080716542> and the adaptive stepsize rule introduced in Wright, Nowak, and Figueiredo (2009) <doi:10.1109/TSP.2009.2016892> are included. You provide the objective function and proximal mappings, and it takes care of the issues like stepsize selection, acceleration, and stopping conditions for you.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1

NeedsCompilation no

Author Eric C. Chi [aut, cre, trl, cph],
Tom Goldstein [aut] (MATLAB original,
https://www.cs.umd.edu/~tomg/projects/fasta/),
Christoph Studer [aut],
Richard G. Baraniuk [aut]

Maintainer Eric C. Chi <`ecchi1105@gmail.com`>

Repository CRAN

Date/Publication 2018-04-10 10:01:37 UTC

R topics documented:

`fasta` *Fast Adaptive Shrinkage/Thresholding Algorithm*

Description

`fasta` implements back-tracking with Barzelai-Borwein step size selection

Usage

```
fasta(f, gradf, g, proxg, x0, tau1, max_iters = 100, w = 10,
  backtrack = TRUE, recordIterates = FALSE, stepsizeShrink = 0.5,
  eps_n = 1e-15)
```

Arguments

`f`	function handle for computing the smooth part of the objective
`gradf`	function handle for computing the gradient of objective
`g`	function handle for computing the nonsmooth part of the objective
`proxg`	function handle for computing proximal mapping
`x0`	initial guess
`tau1`	initial stepsize
`max_iters`	maximum iterations before automatic termination
`w`	lookback window for non-montone line search
`backtrack`	boolean to perform backtracking line search
`recordIterates`	boolean to record iterate sequence
`stepsizeShrink`	multplier to decrease step size
`eps_n`	epsilon to prevent normalized residual from dividing by zero

Examples

```
#------------------------------------------------------------------------
# LEAST SQUARES: EXAMPLE 1 (SIMULATED DATA)
#------------------------------------------------------------------------

set.seed(12345)
n <- 100
p <- 25
X <- matrix(rnorm(n*p),n,p)
beta <- matrix(rnorm(p),p,1)
y <- X%*%beta + rnorm(n)
beta0 <- matrix(0,p,1) # initial starting vector

f <- function(beta){ 0.5*norm(X%*%beta - y, "F")^2 }
gradf <- function(beta){ t(X)%*%(X%*%beta - y) }
```

```
g <- function(beta) { 0 }
proxg <- function(beta, tau) { beta }
x0 <- double(p) # initial starting iterate
tau1 <- 10

sol <- fasta(f,gradf,g,proxg,x0,tau1)
# Check KKT conditions
gradf(sol$x)

#------------------------------------------------------------------------
# LASSO LEAST SQUARES: EXAMPLE 2 (SIMULATED DATA)
#------------------------------------------------------------------------

set.seed(12345)
n <- 100
p <- 25
X <- matrix(rnorm(n*p),n,p)
beta <- matrix(rnorm(p),p,1)
y <- X%*%beta + rnorm(n)
beta0 <- matrix(0,p,1) # initial starting vector
lambda <- 10

f <- function(beta){ 0.5*norm(X%*%beta - y, "F")^2 }
gradf <- function(beta){ t(X)%*%(X%*%beta - y) }
g <- function(beta) { lambda*norm(as.matrix(beta),'1') }
proxg <- function(beta, tau) { sign(beta)*(sapply(abs(beta) - tau*lambda,
  FUN=function(x) {max(x,0)})) }
x0 <- double(p) # initial starting iterate
tau1 <- 10

sol <- fasta(f,gradf,g,proxg,x0,tau1)
# Check KKT conditions
cbind(sol$x,t(X)%*%(y-X%*%sol$x)/lambda)

#------------------------------------------------------------------------
# LOGISTIC REGRESSION: EXAMPLE 3 (SIMLUATED DATA)
#------------------------------------------------------------------------

set.seed(12345)
n <- 100
p <- 25
X <- matrix(rnorm(n*p),n,p)
y <- sample(c(0,1),nrow(X),replace=TRUE)
beta <- matrix(rnorm(p),p,1)
beta0 <- matrix(0,p,1) # initial starting vector
f <- function(beta) { -t(y)%*%(X%*%beta) + sum(log(1+exp(X%*%beta))) } # objective function
gradf <- function(beta) { -t(X)%*%(y-plogis(X%*%beta)) } # gradient
g <- function(beta) { 0 }
proxg <- function(beta, tau) { beta }
x0 <- double(p) # initial starting iterate
tau1 <- 10

sol <- fasta(f,gradf,g,proxg,x0,tau1)
```

```
# Check KKT conditions
gradf(sol$x)

#------------------------------------------------------------------------
# LASSO LOGISTIC REGRESSION: EXAMPLE 4 (SIMLUATED DATA)
#------------------------------------------------------------------------

set.seed(12345)
n <- 100
p <- 25
X <- matrix(rnorm(n*p),n,p)
y <- sample(c(0,1),nrow(X),replace=TRUE)
beta <- matrix(rnorm(p),p,1)
beta0 <- matrix(0,p,1) # initial starting vector
lambda <- 5

f <- function(beta) { -t(y)%*%(X%*%beta) + sum(log(1+exp(X%*%beta))) } # objective function
gradf <- function(beta) { -t(X)%*%(y-plogis(X%*%beta)) } # gradient
g <- function(beta) { lambda*norm(as.matrix(beta),'1') }
proxg <- function(beta, tau) { sign(beta)*(sapply(abs(beta) - tau*lambda,
  FUN=function(x) {max(x,0)})) }
x0 <- double(p) # initial starting iterate
tau1 <- 10

sol <- fasta(f,gradf,g,proxg,x0,tau1)
# Check KKT conditions
cbind(sol$x,-gradf(sol$x)/lambda)
```

Index