Package 'lbfgs'

August 29, 2016

Type Package

Title Limited-memory BFGS Optimization

Version 1.2.1

Date 2014-07-08

Maintainer Antonio Coppola <acoppola@college.harvard.edu>

Description

A wrapper built around the libLBFGS optimization library by Naoaki Okazaki. The lbfgs package implements both the Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) and the Orthant-Wise Quasi-Newton Limited-Memory (OWL-QN) optimization algorithms. The L-BFGS algorithm solves the problem of minimizing an objective, given its gradient, by iteratively computing approximations of the inverse Hessian matrix. The OWL-QN algorithm finds the optimum of an objective plus the L1-norm of the problem's parameters. The package offers a fast and memory-efficient implementation of these optimization routines, which is particularly suited for high-dimensional problems.

License GPL (>= 2)

Imports Rcpp (>= 0.11.2)

LinkingTo Rcpp

Author Antonio Coppola [aut, cre, cph],
Brandon Stewart [aut, cph],
Naoaki Okazaki [aut, cph],
David Ardia [ctb, cph],
Dirk Eddelbuettel [ctb, cph],
Katharine Mullen [ctb, cph],
Jorge Nocedal [ctb, cph]

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-08-31 11:23:32

R topics documented:

`lbfgs` *Optimize function using libLBFGS library*

Description

Performs function optimization using the Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) and Orthant-Wise Limited-memory Quasi-Newton optimization (OWL-QN) algorithms. A wrapper to the libLBFGS library by Naoaki Okazaki, based on an implementation of the L-BFGS method written by Jorge Nocedal. Please note that significant portions of this help file are taken from Okazaki's original documentation. For further information, please refer to the libLBFGS page.

Usage

```
lbfgs(call_eval, call_grad, vars, environment=NULL,
      ..., invisible = 0, m = 6, epsilon = 1e-5, past = 0,
      delta = 0, max_iterations = 0,
      linesearch_algorithm = "LBFGS_LINESEARCH_DEFAULT",
      max_linesearch = 20, min_step = 1e-20,
      max_step = 1e+20, ftol = 1e-4, wolfe = 0.9,
      gtol = 0.9, orthantwise_c = 0,
      orthantwise_start = 0,
      orthantwise_end = length(vars))
```

Arguments

`call_eval`	The function to be optimized. This should be either an R object taking in a numeric vector as its first parameter, and returning a scalar output, or an external pointer to a C++ function compiled using the `inline` interface. C++ implementations should yield considerable speed improvements. For more info about implementing the objective function and the gradient as compiled C++ functions, see the accompanying vignette.
`call_grad`	A function returning the gradient vector of the objective. This should be either an R object taking in a numeric vector as its first parameter, and returning a scalar output, or an external pointer to a C++ function compiled using the `inline` interface. C++ implementations should yield considerable speed improvements. For more info about implementing the objective function and the gradient as compiled C++ functions, see the accompanying vignette.
`vars`	A numeric vector containing the initial values for all variables.
`environment`	An R environment containing all extra arguments to be passed to the objective function and to the gradient, which must be matched exactly. If the objective function and the gradient are implemented in C++, extra arguments must be passed using this option, rather than the `...` construct. If the functions are implemented in R, extra arguments should be passed to them using the `...` construct instead.

`...` Other arguments to be passed to `call_eval` and `call_grad`. Note that these must be matched exactly. Use this construct for extra arguments if `call_eval` and `call_grad` are implemented as R functions. If`call_eval` and `call_grad` are C++ functions, use the `environment` construct instead.

`invisible` Defaults to `0`. Set to `1` to suppress console output.

`m` The number of corrections to approximate the inverse Hessian matrix. The L-BFGS routine stores the computation results of previous `m` iterations to approximate the inverse hessian matrix of the current iteration. This parameter controls the size of the limited memories (corrections). The default value is `6`. Values less than `3` are not recommended. Large values will result in excessive computing time.

`epsilon` Epsilon for convergence test. This parameter determines the accuracy with which the solution is to be found. A minimization terminates when `||g|| < epsilon * max(1, ||x||)`, where `||.||` denotes the Euclidean `(L2)` norm. The default value is `1e-5`.

`past` Distance for delta-based convergence test. This parameter determines the distance, in iterations, to compute the rate of decrease of the objective function. If the value of this parameter is zero, the library does not perform the delta-based convergence test. The default value is `0`.

`delta` Delta for convergence test. This parameter determines the minimum rate of decrease of the objective function. The library stops iterations when the following condition is met: `(f' - f)} \code{/ f < delta}, where \code{f'` is the objective value of past iterations ago, and `f` is the objective value of the current iteration. The default value is `0`.

`max_iterations` The maximum number of iterations. The `lbfgs()` function terminates an optimization process with maximum iterations status code when the iteration count exceedes this parameter. Setting this parameter to zero continues an optimization process until a convergence or error. The default value is `0`.

`linesearch_algorithm`
The line search algorithm. This parameter specifies a line search algorithm to be used by the L-BFGS routine. Valid arguments are the following:

`LBFGS_LINESEARCH_MORETHUENTE`: More-Thuente method.

`LBFGS_LINESEARCH_BACKTRACKING_ARMIJO`: Backtracking method with the Armijo condition. The backtracking method finds the step length such that it satisfies the sufficient decrease (Armijo) condition, `- f(x + a * d) <= f(x) + ftol * a * g(x)^T d`, where `x` is the current point, `d` is the current search direction, and `a` is the step length.

`LBFGS_LINESEARCH_BACKTRACKING`: The backtracking method with the default (regular Wolfe) condition.

`LBFGS_LINESEARCH_BACKTRACKING_WOLFE`: Backtracking method with regular Wolfe condition. The backtracking method finds the step length such that it satisfies both the Armijo condition and the curvature condition, `- g(x + a * d)^T`

d >= wolfe * g(x)^T d.

LBFGS_LINESEARCH_BACKTRACKING_STRONG_WOLFE: Backtracking method with strong Wolfe condition. The backtracking method finds the step length such that it satisfies both the Armijo condition and the following condition, - |g(x + a * d)^T d| <= wolfe * |g(x)^T d|.

If OWL-QN is invoked (orthantwise_c != 0), BACKTRACKING is used by default. Otherwise, the default option is MORETHUENTE. Note that the More-Thuente method cannot be used with OWL-QN, and the function will halt if such a combination of parameters is specified.

max_linesearch The maximum number of trials for the line search.This parameter controls the number of function and gradients evaluations per iteration for the line search routine. The default value is 20.

min_step The minimum step of the line search routine. The default value is 1e-20. This value need not be modified unless the exponents are too large for the machine being used, or unless the problem is extremely badly scaled (in which case the exponents should be increased).

max_step The maximum step of the line search. The default value is 1e+20. This value need not be modified unless the exponents are too large for the machine being used, or unless the problem is extremely badly scaled (in which case the exponents should be increased).

ftol A parameter to control the accuracy of the line search routine. The default value is 1e-4. This parameter should be greater than zero and smaller than 0.5.

wolfe A coefficient for the Wolfe condition. This parameter is valid only when the backtracking line-search algorithm is used with the Wolfe condition. The default value is 0.9. This parameter should be greater the ftol parameter and smaller than 1.0.

gtol A parameter to control the accuracy of the line search routine. The default value is 0.9. If the function and gradient evaluations are inexpensive with respect to the cost of the iteration (which is sometimes the case when solving very large problems) it may be advantageous to set this parameter to a small value. A typical small value is 0.1. This parameter should be greater than the ftol parameter (default 1e-4) and smaller than 1.0.

orthantwise_c Coefficient for the L1 norm of variables. This parameter should be set to zero for standard minimization problems. Setting this parameter to a positive value activates Orthant-Wise Limited-memory Quasi-Newton (OWL-QN) method, which minimizes the objective function F(x) combined with the L1 norm |x| of the variables, {F(x) + C |x|}. This parameter is the coefficient for the |x|, i.e., C. As the L1 norm |x| is not differentiable at zero, the library modifies function and gradient evaluations from a client program suitably. The default value is zero. Note that the objective function minimized by alternative packages (e.g., glmnet) is of the form : F(x)/N + C |x|, where N is the number of parameters. lbfgs does not divide the likelihood function by N. To achieve equivalence with glmnet result, take this difference of implementation into account.

orthantwise_start
: Start index for computing L1 norm of the variables. This parameter is valid only for OWL-QN method (i.e., `orthantwise_c != 0`). This parameter `b` (`0 <= b < N`) specifies the index number from which the library computes the L1 norm of the variables x, `|x| := |x_{b}| + |x_{b+1}| + ... + |x_{N}|`. In other words, variables `x_1, ..., x_{b-1}` are not used for computing the L1 norm. Setting `b (0 < b < N)`, one can protect variables, `x_1, ..., x_{b-1}` (e.g., a bias term of logistic regression) from being regularized. The default value is zero. Note that the parameters are indexed starting from zero, and not one.

orthantwise_end
: End index for computing `L1` norm of the variables. This parameter is valid only for OWL-QN method (i.e., `orthantwise_c != 0`). This parameter `e (0 < e <= N)` specifies the index number at which the library stops computing the L1 norm of the variables `x`. Note that the parameters are indexed starting from zero, and not one.

Value

A list with the following components:

`value`	The minimized value of the objective function.
`par`	A numerical array. The best set of parameters found.
`convergence`	An integer code. Zero indicates that convergence was reached without issues. Negative values indicate errors in the execution of the L-BFGS routine.
`message`	A character object detailing execution errors. This component is only returned if the convergence code is different form zero.

Examples

```
# Rosenbrock Banana function

objective <- function(x) {
    x1 <- x[1]
    x2 <- x[2]
    100 * (x2 - x1 * x1)^2 + (1 - x1)^2
}

gradient <- function(x) { ## Gradient of 'fr'
    x1 <- x[1]
    x2 <- x[2]
    c(-400 * x1 * (x2 - x1 * x1) - 2 * (1 - x1),
       200 *      (x2 - x1 * x1))
}

output <- lbfgs(objective, gradient, c(-1.2,1))

# An example using OWL-QN to perform a Poisson regression using data from
# Golub, Todd R., et al. "Molecular classification of cancer: class discovery
```

```
# and class prediction by gene expression monitoring." Science 286.5439 (1999):
# 531-537. A workspace with the dataset ("Leukemia.RData") is included
# in the package distribution.

# data(Leukemia)

# X <- Leukemia$x
# y <- Leukemia$y
# X1 <- cbind(1, X)

# pois.likelihood <- function(par, X, y, prec=0) {
#   Xbeta <- X%*%par
#   -(sum(y*Xbeta - exp(Xbeta)) -.5*sum(par^2*prec))
# }

# pois.gradient <- function(par, X, y, prec=0) {
#   Xbeta <- X%*%par
#   expXbeta <- exp(Xbeta)
#   -(crossprod(X,(y-expXbeta)) -par*prec)
# }

# output <- lbfgs(pois.likelihood,pois.gradient, X=X1, y=y, prec=0,
#           rep(0, ncol(X1)), invisible=1, orthantwise_c=10,
#           linesearch_algorithm="LBFGS_LINESEARCH_BACKTRACKING",
#           orthantwise_start = 1, orthantwise_end = ncol(X1))

# Trivial Example

objective <- function(x){
  a <- x[1]
  b <- x[2]
  return(a^2 + b^2)
}

gradient <- function(x){
  return(2*x)
}

output <- lbfgs(objective, gradient, c(100,13))
```

Leukemia *Data from Golub et al. 1999*

Description

Data from Golub, Todd R., et al. "Molecular classification of cancer: class discovery and class prediction by gene expression monitoring." Science 286.5439 (1999): 531-537. The study uses microarray data to perform cancer classification based on gene expression monitoring.

Usage

```
data(Leukemia)
```

Value

y	A vector of binary values specifying the cancer class for 72 leukemia patients. A value of 0 corresponds to patients with acute lymphoblastic leukemia (ALL), and 1 corresponds to patients with acute myeloid leukemia (AML).
x	A 72-by-3571 matrix specifying the levels of expressions of 3571 genes for the 72 different patients.

Index