Package: stochQN (via r-universe)

November 2, 2026

Type Package

Title Stochastic Limited Memory Quasi-Newton Optimizers

Version 0.1.2-1

Author David Cortes

Maintainer David Cortes <david.cortes.rivera@gmail.com>

URL https://github.com/david-cortes/stochQN

BugReports https://github.com/david-cortes/stochQN/issues

Description Implementations of stochastic, limited-memory quasi-Newton optimizers, similar in spirit to the LBFGS (Limited-memory Broyden-Fletcher-Goldfarb-Shanno) algorithm, for smooth stochastic optimization. Implements the following methods: oLBFGS (online LBFGS) (Schraudolph, N.N., Yu, J. and Guenter, S., 2007 <http://proceedings.mlr.press/v2/schraudolph07a.html>), SQN (stochastic quasi-Newton) (Byrd, R.H., Hansen, S.L., Nocedal, J. and Singer, Y., 2016 <doi:10.48550/arXiv.1401.7020>), adaQN (adaptive quasi-Newton) (Keskar, N.S., Berahas, A.S., 2016, <doi:10.48550/arXiv.1511.01169>). Provides functions for easily creating R objects with partial_fit/predict methods from some given objective/gradient/predict functions. Includes an example stochastic logistic regression using these optimizers. Provides header files and registered C routines for using it directly from C/C++.

License BSD_2_clause + file LICENSE

NeedsCompilation yes

RoxygenNote 6.1.1

Repository https://cran.r-universe.dev

Date/Publication 2021-09-26 03:10:02 UTC

RemoteUrl https://github.com/cran/stochQN

RemoteRef HEAD

RemoteSha c595f81ca70ab26bf44263d8664b7ef4bf0084fa

Contents

adaQN . . . 2
adaQN_free . . . 6
coef.stoch_logistic . . . 8
get_curr_x . . . 9
get_iteration_number . . . 10
oLBFGS . . . 10
oLBFGS_free . . . 13
partial_fit . . . 15
partial_fit_logistic . . . 16
predict.stoch_logistic . . . 17
predict.stochQN_guided . . . 17
print.adaQN . . . 18
print.adaQN_free . . . 18
print.oLBFGS . . . 19
print.oLBFGS_free . . . 19
print.SQN . . . 20
print.SQN_free . . . 20
print.stoch_logistic . . . 21
run_adaQN_free . . . 21
run_oLBFGS_free . . . 22
run_SQN_free . . . 23
SQN . . . 24
SQN_free . . . 26
stochastic.logistic.regression . . . 29
summary.stoch_logistic . . . 31
update_fun . . . 32
update_gradient . . . 33
update_hess_vec . . . 33

Index **34**

---

adaQN *adaQN guided optimizer*

Description

Optimizes an empirical (possibly non-convex) loss function over batches of sample data.

Usage

```
adaQN(x0, grad_fun, obj_fun = NULL, pred_fun = NULL,
  initial_step = 0.01, step_fun = function(iter) 1/sqrt((iter/100) +
  1), callback_iter = NULL, args_cb = NULL, verbose = TRUE,
  mem_size = 10, fisher_size = 100, bfgs_upd_freq = 20,
  max_incr = 1.01, min_curvature = 1e-04, y_reg = NULL,
  scal_reg = 1e-04, rmsprop_weight = 0.9, use_grad_diff = FALSE,
```

```
  check_nan = TRUE, nthreads = -1, X_val = NULL, y_val = NULL,
  w_val = NULL)
```

Arguments

x0	Initial values for the variables to optimize.
grad_fun	Function taking as unnamed arguments 'x_curr' (variable values), 'X' (covariates), 'y' (target variable), and 'w' (weights), plus additional arguments ('...'), and producing the expected value of the gradient when evalauted on that data.
obj_fun	Function taking as unnamed arguments 'x_curr' (variable values), 'X' (covariates), 'y' (target variable), and 'w' (weights), plus additional arguments ('...'), and producing the expected value of the objective function when evalauted on that data. Only required when using 'max_incr'.
pred_fun	Function taking an unnamed argument as data, another unnamed argument as the variable values, and optional extra arguments ('...'). Will be called when using 'predict' on the object returned by this function.
initial_step	Initial step size.
step_fun	Function accepting the iteration number as an unnamed parameter, which will output the number by which 'initial_step' will be multiplied at each iteration to get the step size for that iteration.
callback_iter	Callback function which will be called at the end of each iteration. Will pass three unnamed arguments: the current variable values, the current iteration number, and 'args_cb'. Pass 'NULL' if there is no need to call a callback function.
args_cb	Extra argument to pass to the callback function.
verbose	Whether to print information about iteration statuses when something goes wrong.
mem_size	Number of correction pairs to store for approximation of Hessian-vector products.
fisher_size	Number of gradients to store for calculation of the empirical Fisher product with gradients. If passing 'NULL', will force 'use_grad_diff' to 'TRUE'.
bfgs_upd_freq	Number of iterations (batches) after which to generate a BFGS correction pair.
max_incr	Maximum ratio of function values in the validation set under the average values of 'x' during current epoch vs. previous epoch. If the ratio is above this threshold, the BFGS and Fisher memories will be reset, and 'x' values reverted to their previous average. If not using a validation set, will take a longer batch for function evaluations (same as used for gradients when using 'use_grad_diff' = 'TRUE'). Pass 'NULL' for no function-increase checking.
min_curvature	Minimum value of (s * y) / (s * s) in order to accept a correction pair. Pass 'NULL' for no minimum.
y_reg	Regularizer for 'y' vector (gets added y_reg * s). Pass 'NULL' for no regularization.
scal_reg	Regularization parameter to use in the denominator for AdaGrad and RMSProp scaling.
rmsprop_weight	If not 'NULL', will use RMSProp formula instead of AdaGrad for approximated inverse-Hessian initialization.

use_grad_diff	Whether to create the correction pairs using differences between gradients instead of empirical Fisher matrix. These gradients are calculated on a larger batch than the regular ones (given by batch_size * bfgs_upd_freq).
check_nan	Whether to check for variables becoming NaN after each iteration, and reverting the step if they do (will also reset BFGS memory).
nthreads	Number of parallel threads to use. If set to -1, will determine the number of available threads and use all of them. Note however that not all the computations can be parallelized, and the BLAS backend might use a different number of threads.
X_val	Covariates to use as validation set (only used when passing 'max_incr'). If not passed, will use a larger batch of stored data, in the same way as for Hessian-vector products in SQN.
y_val	Target variable for the covariates to use as validation set (only used when passing 'max_incr'). If not passed, will use a larger batch of stored data, in the same way as for Hessian-vector products in SQN.
w_val	Sample weights for the covariates to use as validation set (only used when passing 'max_incr'). If not passed, will use a larger batch of stored data, in the same way as for Hessian-vector products in SQN.

Value

an 'adaQN' object with the user-supplied functions, which can be fit to batches of data through function 'partial_fit', and can produce predictions on new data through function 'predict'.

References

- Keskar, N.S. and Berahas, A.S., 2016, September. "adaQN: An Adaptive Quasi-Newton Algorithm for Training RNNs." In Joint European Conference on Machine Learning and Knowledge Discovery in Databases (pp. 1-16). Springer, Cham.
- Wright, S. and Nocedal, J., 1999. "Numerical optimization." (ch 7) Springer Science, 35(67-68), p.7.

See Also

partial_fit , predict.stochQN_guided , adaQN_free

Examples

```
### Example regression with randomly-generated data
library(stochQN)

### Will sample data y ~ Ax + epsilon
true_coefs <- c(1.12, 5.34, -6.123)

generate_data_batch <- function(true_coefs, n = 100) {
  X <- matrix(
    rnorm(length(true_coefs) * n),
    nrow=n, ncol=length(true_coefs))
```

```
  y <- X %*% true_coefs + rnorm(n)
  return(list(X = X, y = y))
}

### Regular regression function that minimizes RMSE
eval_fun <- function(coefs, X, y, weights=NULL, lambda=1e-5) {
  pred <- as.numeric(X %*% coefs)
  RMSE <- sqrt(mean((pred - y)^2))
  reg  <- 2 * lambda * as.numeric(coefs %*% coefs)
  return(RMSE + reg)
}

eval_grad <- function(coefs, X, y, weights=NULL, lambda=1e-5) {
  pred <- X %*% coefs
  grad <- colMeans(X * as.numeric(pred - y))
  grad <- grad + 2 * lambda * as.numeric(coefs^2)
  return(grad)
}

pred_fun <- function(X, coefs, ...) {
  return(as.numeric(X %*% coefs))
}

### Initialize optimizer form arbitrary values
x0 <- c(1, 1, 1)
optimizer <- adaQN(x0, grad_fun=eval_grad,
  pred_fun=pred_fun, obj_fun=eval_fun, initial_step=1e-0)
val_data <- generate_data_batch(true_coefs, n=100)

### Fit to 50 batches of data, 100 observations each
for (i in 1:50) {
  set.seed(i)
  new_batch <- generate_data_batch(true_coefs, n=100)
  partial_fit(
    optimizer,
    new_batch$X, new_batch$y,
    lambda=1e-5)
  x_curr <- get_curr_x(optimizer)
  i_curr <- get_iteration_number(optimizer)
  if ((i_curr %% 10)  == 0) {
    cat(sprintf(
      "Iteration %d - E[f(x)]: %f - values of x: [%f, %f, %f]\n",
      i_curr,
      eval_fun(x_curr, val_data$X, val_data$y, lambda=1e-5),
      x_curr[1], x_curr[2], x_curr[3]))
  }
}

### Predict for new data
new_batch <- generate_data_batch(true_coefs, n=10)
yhat <- predict(optimizer, new_batch$X)
```

adaQN_free *adaQN Free-Mode Optimizer*

Description

Optimizes an empirical (perhaps non-convex) loss function over batches of sample data. Compared to function/class 'adaQN', this version lets the user do all the calculations from the outside, only interacting with the object by means of a function that returns a request type and is fed the required calculation through methods 'update_gradient' and 'update_function'.

Order in which requests are made:

========== loop ==========

* calc_grad
    ... (repeat calc_grad)

if max_incr > 0:
    * calc_fun_val_batch

if 'use_grad_diff':
    * calc_grad_big_batch (skipped if below max_incr)

==========

After running this function, apply 'run_adaQN_free' to it to get the first requested piece of information.

Usage

```
adaQN_free(mem_size = 10, fisher_size = 100, bfgs_upd_freq = 20,
  max_incr = 1.01, min_curvature = 1e-04, scal_reg = 1e-04,
  rmsprop_weight = 0.9, y_reg = NULL, use_grad_diff = FALSE,
  check_nan = TRUE, nthreads = -1)
```

Arguments

mem_size	Number of correction pairs to store for approximation of Hessian-vector products.
fisher_size	Number of gradients to store for calculation of the empirical Fisher product with gradients. If passing 'NULL', will force 'use_grad_diff' to 'TRUE'.
bfgs_upd_freq	Number of iterations (batches) after which to generate a BFGS correction pair.
max_incr	Maximum ratio of function values in the validation set under the average values of 'x' during current epoch vs. previous epoch. If the ratio is above this threshold, the BFGS and Fisher memories will be reset, and 'x' values reverted to their previous average. Pass 'NULL' for no function-increase checking.
min_curvature	Minimum value of (s * y) / (s * s) in order to accept a correction pair. Pass 'NULL' for no minimum.
scal_reg	Regularization parameter to use in the denominator for AdaGrad and RMSProp scaling.

rmsprop_weight	If not 'NULL', will use RMSProp formula instead of AdaGrad for approximated inverse-Hessian initialization.
y_reg	Regularizer for 'y' vector (gets added y_reg * s). Pass 'NULL' for no regularization.
use_grad_diff	Whether to create the correction pairs using differences between gradients instead of Fisher matrix. These gradients are calculated on a larger batch than the regular ones (given by batch_size * bfgs_upd_freq). If 'TRUE', empirical Fisher matrix will not be used.
check_nan	Whether to check for variables becoming NaN after each iteration, and reverting the step if they do (will also reset BFGS and Fisher memory).
nthreads	Number of parallel threads to use. If set to -1, will determine the number of available threads and use all of them. Note however that not all the computations can be parallelized, and the BLAS backend might use a different number of threads.

Value

An 'adaQN_free' object, which can be used through functions 'update_gradient', 'update_fun', and 'run_adaQN_free'

References

- Keskar, N.S. and Berahas, A.S., 2016, September. "adaQN: An Adaptive Quasi-Newton Algorithm for Training RNNs." In Joint European Conference on Machine Learning and Knowledge Discovery in Databases (pp. 1-16). Springer, Cham.
- Wright, S. and Nocedal, J., 1999. "Numerical optimization." (ch 7) Springer Science, 35(67-68), p.7.

See Also

update_gradient , update_fun , run_adaQN_free

Examples

```
### Example optimizing Rosenbrock 2D function
### Note that this example is not stochastic, as the
### function is not evaluated in expectation based on
### batches of data, but rather it has a given absolute
### form that never varies.
library(stochQN)


fr <- function(x) { ## Rosenbrock Banana function
 x1 <- x[1]
 x2 <- x[2]
 100 * (x2 - x1 * x1)^2 + (1 - x1)^2
}
grr <- function(x) { ## Gradient of 'fr'
 x1 <- x[1]
```

```
 x2 <- x[2]
 c(-400 * x1 * (x2 - x1 * x1) - 2 * (1 - x1),
   200 * (x2 - x1 * x1))
}


### Initial values of x
x_opt = as.numeric(c(0, 2))
cat(sprintf("Initial values of x: [%.3f, %.3f]\n",
   x_opt[1], x_opt[2]))

### Will use constant step size throughout
### (not recommended)
step_size = 1e-2

### Initialize the optimizer
optimizer = adaQN_free()

### Keep track of the iteration number
curr_iter <- 0

### Run a loop for many iterations
### (Note that some iterations might require more
###  than 1 calculation request)
for (i in 1:200) {
 req <- run_adaQN_free(optimizer, x_opt, step_size)
 if (req$task == "calc_grad") {
   update_gradient(optimizer, grr(req$requested_on))
 } else if (req$task == "calc_fun_val_batch") {
   update_fun(optimizer, fr(req$requested_on))
 }

 ### Track progress every 10 iterations
 if (req$info$iteration_number > curr_iter) {
  curr_iter <- req$info$iteration_number
 }
 if ((curr_iter %% 10) == 0) {
  cat(sprintf(
    "Iteration %3d - Current function value: %.3f\n",
    req$info$iteration_number, fr(x_opt)))
 }
}
cat(sprintf("Current values of x: [%.3f, %.3f]\n",
   x_opt[1], x_opt[2]))
```

`coef.stoch_logistic` *Retrieve fitted coefficients from stochastic logistic regression object*

Description

Retrieve fitted coefficients from stochastic logistic regression object

Usage

```
## S3 method for class 'stoch_logistic'
coef(object, ...)
```

Arguments

object	A ‘stoch_logistic‘ object as output by function ‘stochastic.logistic.regression‘. Must have already been fit to at least 1 batch of data.
...	Not used.

Value

An (n x 1) matrix with the coefficients, in the same format as those from ‘glm‘.

See Also

stochastic.logistic.regression

---

get_curr_x *Get current values of the optimization variables*

Description

Get current values of the optimization variables

Usage

```
get_curr_x(optimizer)
```

Arguments

optimizer	An optimizer (guided-mode) from this module, as output by functions ‘oLBFGS‘, ‘SQN‘, ‘adaQN‘.

Value

A numeric vector with the current values of the variables being optimized.

See Also

oLBFGS , SQN , adaQN

get_iteration_number *Get current iteration number from the optimizer object*

Description

Get current iteration number from the optimizer object

Usage

```
get_iteration_number(optimizer)
```

Arguments

optimizer	An optimizer (guided-mode) from this module, as output by functions 'oLBFGS', 'SQN', 'adaQN'.

Value

The current iteration number.

See Also

oLBFGS , SQN , adaQN

oLBFGS *oLBFGS guided optimizer*

Description

Optimizes an empirical (convex) loss function over batches of sample data.

Usage

```
oLBFGS(x0, grad_fun, pred_fun = NULL, initial_step = 0.01,
  step_fun = function(iter) 1/sqrt((iter/10) + 1),
  callback_iter = NULL, args_cb = NULL, verbose = TRUE,
  mem_size = 10, hess_init = NULL, min_curvature = 1e-04,
  y_reg = NULL, check_nan = TRUE, nthreads = -1)
```

Arguments

x0	Initial values for the variables to optimize.
grad_fun	Function taking as unnamed arguments ‘x_curr‘ (variable values), ‘X‘ (covariates), ‘y‘ (target variable), and ‘w‘ (weights), plus additional arguments (‘...‘), and producing the expected value of the gradient when evalauted on that data.
pred_fun	Function taking an unnamed argument as data, another unnamed argument as the variable values, and optional extra arguments (‘...‘). Will be called when using ‘predict‘ on the object returned by this function.
initial_step	Initial step size.
step_fun	Function accepting the iteration number as an unnamed parameter, which will output the number by which ‘initial_step‘ will be multiplied at each iteration to get the step size for that iteration.
callback_iter	Callback function which will be called at the end of each iteration. Will pass three unnamed arguments: the current variable values, the current iteration number, and ‘args_cb‘. Pass ‘NULL‘ if there is no need to call a callback function.
args_cb	Extra argument to pass to the callback function.
verbose	Whether to print information about iteration statuses when something goes wrong.
mem_size	Number of correction pairs to store for approximation of Hessian-vector products.
hess_init	Value to which to initialize the diagonal of H0. If passing ‘NULL‘, will use the same initializion as for SQN ((s_last * y_last) / (y_last * y_last)).
min_curvature	Minimum value of (s * y) / (s * s) in order to accept a correction pair. Pass ‘NULL‘ for no minimum.
y_reg	Regularizer for ’y’ vector (gets added y_reg * s). Pass ‘NULL‘ for no regularization.
check_nan	Whether to check for variables becoming NA after each iteration, and reverting the step if they do (will also reset BFGS memory).
nthreads	Number of parallel threads to use. If set to -1, will determine the number of available threads and use all of them. Note however that not all the computations can be parallelized, and the BLAS backend might use a different number of threads.

Value

an ‘oLBFGS‘ object with the user-supplied functions, which can be fit to batches of data through function ‘partial_fit‘, and can produce predictions on new data through function ‘predict‘.

References

- Schraudolph, N.N., Yu, J. and Guenter, S., 2007, March. "A stochastic quasi-Newton method for online convex optimization." In Artificial Intelligence and Statistics (pp. 436-443).
- Wright, S. and Nocedal, J., 1999. "Numerical optimization." (ch 7) Springer Science, 35(67-68), p.7.

See Also

partial_fit , predict.stochQN_guided , oLBFGS_free

Examples

```
### Example regression with randomly-generated data
library(stochQN)

### Will sample data y ~ Ax + epsilon
true_coefs <- c(1.12, 5.34, -6.123)

generate_data_batch <- function(true_coefs, n = 100) {
  X <- matrix(
    rnorm(length(true_coefs) * n),
    nrow=n, ncol=length(true_coefs))
  y <- X %*% true_coefs + rnorm(n)
  return(list(X = X, y = y))
}

### Regular regression function that minimizes RMSE
eval_fun <- function(coefs, X, y, weights=NULL, lambda=1e-5) {
  pred <- as.numeric(X %*% coefs)
  RMSE <- sqrt(mean((pred - y)^2))
  reg  <- lambda * as.numeric(coefs %*% coefs)
  return(RMSE + reg)
}

eval_grad <- function(coefs, X, y, weights=NULL, lambda=1e-5) {
  pred <- X %*% coefs
  grad <- colMeans(X * as.numeric(pred - y))
  grad <- grad + 2 * lambda * as.numeric(coefs^2)
  return(grad)
}

pred_fun <- function(X, coefs, ...) {
  return(as.numeric(X %*% coefs))
}

### Initialize optimizer form arbitrary values
x0 <- c(1, 1, 1)
optimizer <- oLBFGS(x0, grad_fun=eval_grad,
  pred_fun=pred_fun, initial_step=1e-1)
val_data <- generate_data_batch(true_coefs, n=100)

### Fit to 50 batches of data, 100 observations each
set.seed(1)
for (i in 1:50) {
  new_batch <- generate_data_batch(true_coefs, n=100)
  partial_fit(
    optimizer,
    new_batch$X, new_batch$y,
    lambda=1e-5)
```

```
  x_curr <- get_curr_x(optimizer)
  i_curr <- get_iteration_number(optimizer)
  if ((i_curr %% 10)  == 0) {
    cat(sprintf(
      "Iteration %d - E[f(x)]: %f - values of x: [%f, %f, %f]\n",
      i_curr,
      eval_fun(x_curr, val_data$X, val_data$y, lambda=1e-5),
      x_curr[1], x_curr[2], x_curr[3]))
  }
}

### Predict for new data
new_batch <- generate_data_batch(true_coefs, n=10)
yhat <- predict(optimizer, new_batch$X)
```

`oLBFGS_free` *oLBFGS Free-Mode Optimizer*

Description

Optimizes an empirical (convex) loss function over batches of sample data. Compared to function/class 'oLBFGS', this version lets the user do all the calculations from the outside, only interacting with the object by means of a function that returns a request type and is fed the required calculation through a method 'update_gradient'.

Order in which requests are made:

========== loop ==========

* calc_grad

* calc_grad_same_batch (might skip if using check_nan)

==========

After running this function, apply 'run_oLBFGS_free' to it to get the first requested piece of information.

Usage

```
oLBFGS_free(mem_size = 10, hess_init = NULL, min_curvature = 1e-04,
  y_reg = NULL, check_nan = TRUE, nthreads = -1)
```

Arguments

`mem_size`	Number of correction pairs to store for approximation of Hessian-vector products.
`hess_init`	Value to which to initialize the diagonal of H0. If passing 'NULL', will use the same initializion as for SQN ((s_last * y_last) / (y_last * y_last)).
`min_curvature`	Minimum value of (s * y) / (s * s) in order to accept a correction pair. Pass 'NULL' for no minimum.

y_reg	Regularizer for 'y' vector (gets added y_reg * s). Pass 'NULL' for no regularization.
check_nan	Whether to check for variables becoming NA after each iteration, and reverting the step if they do (will also reset BFGS memory).
nthreads	Number of parallel threads to use. If set to -1, will determine the number of available threads and use all of them. Note however that not all the computations can be parallelized, and the BLAS backend might use a different number of threads.

Value

An 'oLBFGS_free' object, which can be used through functions 'update_gradient' and 'run_oLBFGS_free'

References

- Schraudolph, N.N., Yu, J. and Guenter, S., 2007, March. "A stochastic quasi-Newton method for online convex optimization." In Artificial Intelligence and Statistics (pp. 436-443).
- Wright, S. and Nocedal, J., 1999. "Numerical optimization." (ch 7) Springer Science, 35(67-68), p.7.

See Also

update_gradient , run_oLBFGS_free

Examples

```
### Example optimizing Rosenbrock 2D function
### Note that this example is not stochastic, as the
### function is not evaluated in expectation based on
### batches of data, but rather it has a given absolute
### form that never varies.
### Warning: this optimizer is meant for convex functions
### (Rosenbrock's is not convex)
library(stochQN)


fr <- function(x) { ## Rosenbrock Banana function
 x1 <- x[1]
 x2 <- x[2]
 100 * (x2 - x1 * x1)^2 + (1 - x1)^2
}
grr <- function(x) { ## Gradient of 'fr'
 x1 <- x[1]
 x2 <- x[2]
 c(-400 * x1 * (x2 - x1 * x1) - 2 * (1 - x1),
 200 * (x2 - x1 * x1))
}

### Initial values of x
x_opt = as.numeric(c(0, 2))
cat(sprintf("Initial values of x: [%.3f, %.3f]\n",
```

```
 x_opt[1], x_opt[2]))

### Will use a constant step size throughout
### (not recommended)
step_size <- 1e-1

### Initialize the optimizer
optimizer <- oLBFGS_free()

### Keep track of the iteration number
curr_iter <- 0

### Run a loop for 100 iterations
### (Note that each iteration requires 2 calculations,
###  hence the 200)
for (i in 1:200) {
 req <- run_oLBFGS_free(optimizer, x_opt, step_size)
 if (req$task == "calc_grad") {
   update_gradient(optimizer, grr(req$requested_on))
 } else if (req$task == "calc_grad_same_batch") {
   update_gradient(optimizer, grr(req$requested_on))
 }

 ### Track progress every 10 iterations
 if (req$info$iteration_number > curr_iter) {
   curr_iter <- req$info$iteration_number
   if ((curr_iter %% 10) == 0) {
   cat(sprintf(
    "Iteration %3d - Current function value: %.3f\n",
   req$info$iteration_number, fr(x_opt)
   ))
   }
 }
}
cat(sprintf("Current values of x: [%.3f, %.3f]\n",
 x_opt[1], x_opt[2]))
```

`partial_fit` *Partial fit stochastic model to new data*

Description

Runs one iteration of the stochastic optimizer on the new data passed here.

Usage

```
partial_fit(optimizer, X, y = NULL, weights = NULL, ...)
```

Arguments

optimizer	A stochastic optimizer from this package as output by functions ‘oLBFGS‘, ‘SQN‘, ‘adaQN‘. Will be modified in-place.
X	Covariates to pass to the user-defined gradient / objective / Hessian-vector functions.
y	Target variable to pass to the user-defined gradient / objective / Hessian-vector functions.
weights	Target variable to pass to the user-defined gradient / objective / Hessian-vector functions.
...	Additional arguments to pass to the user-defined gradient / objective / Hessian-vector functions.

Value

No return value (object is modified in-place).

See Also

oLBFGS , SQN , adaQN

---

partial_fit_logistic *Update stochastic logistic regression model with new batch of data*

Description

Perform a quasi-Newton iteration to update the model with new data.

Usage

```
partial_fit_logistic(logistic_model, X, y = NULL, w = NULL)
```

Arguments

logistic_model	A ‘stoch_logistic‘ object as output by function ‘stochastic.logistic.regression‘. Will be modified in-place.
X	Data with covariates. If passing a ‘data.frame‘, the model object must have been initialized with a formula, and ‘X‘ must also contain the target variable (‘y‘). If passing a matrix, must also pass ‘y‘. Note that whatever factor levels are present in the first batch of data, will be taken as the whole factor levels.
y	The target variable, when using matrices. Ignored when using formula.
w	Sample weights (optional). If required, must pass them at every partial fit iteration.

Value

No return value. Model object is updated in-place.

See Also

stochastic.logistic.regression

`predict.stoch_logistic`
Prediction function for stochastic logistic regression

Description

Makes predictions for new data from the fitted model. Model have already been fit to at least 1 batch of data.

Usage

```
## S3 method for class 'stoch_logistic'
predict(object, newdata, type = "prob", ...)
```

Arguments

`object`	A ‘stoch_logistic‘ object as output by function ‘stochastic.logistic.regression‘.
`newdata`	New data on which to make predictions.
`type`	Type of prediction to make. Can pass ‘prob‘ to get probabilities for the positive class, or ‘class‘ to get the predicted class.
`...`	Not used.

Value

A vector with the predicted classes or probabilities for ‘newdata‘.

See Also

stochastic.logistic.regression

`predict.stochQN_guided`
Predict function for stochastic optimizer object

Description

Calls the user-defined predict function for an object optimized through this package’s functions.

Usage

```
## S3 method for class 'stochQN_guided'
predict(object, newdata, ...)
```

Arguments

object	Optimizer from this module as output by functions ‘oLBFGS‘, ‘SQN‘, ‘adaQN‘. Must have been constructed with a predict function.
newdata	Data on which to make predictions (will be passed to the user-provided function).
...	Additional arguments to pass to the user-provided predict function.

See Also

oLBFGS , SQN , adaQN

---

print.adaQN *Print summary info about adaQN guided-mode object*

Description

Print summary info about adaQN guided-mode object

Usage

```
## S3 method for class 'adaQN'
print(x, ...)
```

Arguments

x	An ‘adaQN‘ object as output by function of the same name.
...	Not used.

---

print.adaQN_free *Print summary info about adaQN free-mode object*

Description

Print summary info about adaQN free-mode object

Usage

```
## S3 method for class 'adaQN_free'
print(x, ...)
```

Arguments

x	An ‘adaQN_free‘ object as output by function of the same name.
...	Not used.

print.oLBFGS *Print summary info about oLBFGS guided-mode object*

Description

Print summary info about oLBFGS guided-mode object

Usage

```
## S3 method for class 'oLBFGS'
print(x, ...)
```

Arguments

x	An ‘oLBFGS‘ object as output by function of the same name.
...	Not used.

print.oLBFGS_free *Print summary info about oLBFGS free-mode object*

Description

Print summary info about oLBFGS free-mode object

Usage

```
## S3 method for class 'oLBFGS_free'
print(x, ...)
```

Arguments

x	An ‘oLBFGS_free‘ object as output by function of the same name.
...	Not used.

print.SQN *Print summary info about SQN guided-mode object*

Description

Print summary info about SQN guided-mode object

Usage

```
## S3 method for class 'SQN'
print(x, ...)
```

Arguments

x	An ‘SQN‘ object as output by function of the same name.
...	Not used.

print.SQN_free *Print summary info about SQN free-mode object*

Description

Print summary info about SQN free-mode object

Usage

```
## S3 method for class 'SQN_free'
print(x, ...)
```

Arguments

x	An ‘SQN_free‘ object as output by function of the same name.
...	Not used.

print.stoch_logistic *Print general info about stochastic logistic regression object*

Description

Print general info about stochastic logistic regression object

Usage

```
## S3 method for class 'stoch_logistic'
print(x, ...)
```

Arguments

x	A ‘stoch_logistic‘ object as output by function ‘stochastic.logistic.regression‘.
...	Not used.

See Also

stochastic.logistic.regression

run_adaQN_free *Run adaQN optimizer in free-mode*

Description

Run the next step of an adaQN optimization procedure, after the last requested calculation has been fed to the optimizer. When run for the first time, there is no request, so the function just needs to be run on the object as it is returned from function ‘adaQN_free‘.

Usage

```
run_adaQN_free(optimizer, x, step_size)
```

Arguments

optimizer	An ‘adaQN_free‘ optimizer, for which its last request must have been served. Will be updated in-place.
x	Current values of the variables being optimized. Must be a numeric vector. Will be updated in-place.
step_size	Step size for the quasi-Newton update.

Value

A request with the next piece of required information. The output will be a list with the following levels:

- task Requested task (one of "calc_grad", "calc_fun_val_batch", "calc_grad_big_batch").
- requested_on Values of ‘x‘ at which the requested information must be calculated.
- info
  - x_changed_in_run Whether the ‘x‘ vector was updated.
  - iteration_number Current iteration number (in terms of quasi-Newton updates).
  - iteration_info Information about potential problems encountered during the iteration.

See Also

adaQN_free

`run_oLBFGS_free` *Run oLBFGS optimizer in free-mode*

Description

Run the next step of an oLBFGS optimization procedure, after the last requested calculation has been fed to the optimizer. When run for the first time, there is no request, so the function just needs to be run on the object as it is returned from function ‘oLBFGS_free‘.

Usage

```
run_oLBFGS_free(optimizer, x, step_size)
```

Arguments

`optimizer`	An ‘oLBFGS_free‘ optimizer, for which its last request must have been served. Will be updated in-place.
`x`	Current values of the variables being optimized. Must be a numeric vector. Will be updated in-place.
`step_size`	Step size for the quasi-Newton update.

Value

A request with the next piece of required information. The output will be a list with the following levels:

- task Requested task (one of "calc_grad" or "calc_grad_same_batch").
- requested_on Values of ‘x‘ at which the requested information must be calculated.
- info
  - x_changed_in_run Whether the ‘x‘ vector was updated.
  - iteration_number Current iteration number (in terms of quasi-Newton updates).
  - iteration_info Information about potential problems encountered during the iteration.

See Also

oLBFGS_free

`run_SQN_free` *Run SQN optimizer in free-mode*

Description

Run the next step of an SQN optimization procedure, after the last requested calculation has been fed to the optimizer. When run for the first time, there is no request, so the function just needs to be run on the object as it is returned from function 'SQN_free'.

Usage

```
run_SQN_free(optimizer, x, step_size)
```

Arguments

`optimizer`	An 'SQN_free' optimizer, for which its last request must have been served. Will be updated in-place.
`x`	Current values of the variables being optimized. Must be a numeric vector. Will be updated in-place.
`step_size`	Step size for the quasi-Newton update.

Value

A request with the next piece of required information. The output will be a list with the following levels:

- task Requested task (one of "calc_grad", "calc_grad_big_batch", "calc_hess_vec").
- requested_on
  - req_x Values of 'x' at which the requested information (gradient/Hessian) must be calculated.
  - req_vec Vector by which the Hessian must be multiplied. Will output 'NULL' when this calculation is not needed.
- info
  - x_changed_in_run Whether the 'x' vector was updated.
  - iteration_number Current iteration number (in terms of quasi-Newton updates).
  - iteration_info Information about potential problems encountered during the iteration.

See Also

SQN_free

SQN *SQN guided optimizer*

Description

Optimizes an empirical (convex) loss function over batches of sample data.

Usage

```
SQN(x0, grad_fun, hess_vec_fun = NULL, pred_fun = NULL,
  initial_step = 0.001, step_fun = function(iter) 1/sqrt((iter/10) +
  1), callback_iter = NULL, args_cb = NULL, verbose = TRUE,
  mem_size = 10, bfgs_upd_freq = 20, min_curvature = 1e-04,
  y_reg = NULL, use_grad_diff = FALSE, check_nan = TRUE,
  nthreads = -1)
```

Arguments

x0	Initial values for the variables to optimize.
grad_fun	Function taking as unnamed arguments ‘x_curr‘ (variable values), ‘X‘ (covariates), ‘y‘ (target variable), and ‘w‘ (weights), plus additional arguments (‘...‘), and producing the expected value of the gradient when evalauted on that data.
hess_vec_fun	Function taking as unnamed arguments ‘x_curr‘ (variable values), ‘vec‘ (numeric vector), ‘X‘ (covariates), ‘y‘ (target variable), and ‘w‘ (weights), plus additional arguments (‘...‘), and producing the expected value of the Hessian (with variable values at ‘x_curr‘) when evalauted on that data, multiplied by the vector ‘vec‘. Not required when using ‘use_grad_diff‘ = ‘TRUE‘.
pred_fun	Function taking an unnamed argument as data, another unnamed argument as the variable values, and optional extra arguments (‘...‘). Will be called when using ‘predict‘ on the object returned by this function.
initial_step	Initial step size.
step_fun	Function accepting the iteration number as an unnamed parameter, which will output the number by which ‘initial_step‘ will be multiplied at each iteration to get the step size for that iteration.
callback_iter	Callback function which will be called at the end of each iteration. Will pass three unnamed arguments: the current variable values, the current iteration number, and ‘args_cb‘. Pass ‘NULL‘ if there is no need to call a callback function.
args_cb	Extra argument to pass to the callback function.
verbose	Whether to print information about iteration statuses when something goes wrong.
mem_size	Number of correction pairs to store for approximation of Hessian-vector products.
bfgs_upd_freq	Number of iterations (batches) after which to generate a BFGS correction pair.
min_curvature	Minimum value of (s * y) / (s * s) in order to accept a correction pair. Pass ‘NULL‘ for no minimum.

y_reg	Regularizer for 'y' vector (gets added y_reg * s). Pass 'NULL' for no regularization.
use_grad_diff	Whether to create the correction pairs using differences between gradients instead of Hessian-vector products. These gradients are calculated on a larger batch than the regular ones (given by batch_size * bfgs_upd_freq).
check_nan	Whether to check for variables becoming NaN after each iteration, and reverting the step if they do (will also reset BFGS memory).
nthreads	Number of parallel threads to use. If set to -1, will determine the number of available threads and use all of them. Note however that not all the computations can be parallelized, and the BLAS backend might use a different number of threads.

Value

an 'SQN' object with the user-supplied functions, which can be fit to batches of data through function 'partial_fit', and can produce predictions on new data through function 'predict'.

References

- Byrd, R.H., Hansen, S.L., Nocedal, J. and Singer, Y., 2016. "A stochastic quasi-Newton method for large-scale optimization." SIAM Journal on Optimization, 26(2), pp.1008-1031.
- Wright, S. and Nocedal, J., 1999. "Numerical optimization." (ch 7) Springer Science, 35(67-68), p.7.

See Also

partial_fit , predict.stochQN_guided , SQN_free

Examples

```
### Example logistic regression with randomly-generated data
library(stochQN)

### Will sample data y ~ Bernoulli(sigm(Ax))
true_coefs <- c(1.12, 5.34, -6.123)

generate_data_batch <- function(true_coefs, n = 100) {
  X <- matrix(rnorm(length(true_coefs) * n), nrow=n, ncol=length(true_coefs))
  y <- 1 / (1 + exp(-as.numeric(X %*% true_coefs)))
  y <- as.numeric(y >= runif(n))
  return(list(X = X, y = y))
}

### Logistic regression likelihood/loss
eval_fun <- function(coefs, X, y, weights=NULL, lambda=1e-5) {
  pred    <- 1 / (1 + exp(-as.numeric(X %*% coefs)))
  logloss <- mean(-(y * log(pred) + (1 - y) * log(1 - pred)))
  reg     <- lambda * as.numeric(coefs %*% coefs)
  return(logloss + reg)
}
```

```
eval_grad <- function(coefs, X, y, weights=NULL, lambda=1e-5) {
  pred <- 1 / (1 + exp(-(X %*% coefs)))
  grad <- colMeans(X * as.numeric(pred - y))
  grad <- grad + 2 * lambda * as.numeric(coefs^2)
  return(as.numeric(grad))
}

eval_Hess_vec <- function(coefs, vec, X, y, weights=NULL, lambda=1e-5) {
  pred <- 1 / (1 + exp(-as.numeric(X %*% coefs)))
  diag <- pred * (1 - pred)
  Hp   <- (t(X) * diag) %*% (X %*% vec)
  Hp   <- Hp / NROW(X) + 2 * lambda * vec
  return(as.numeric(Hp))
}

pred_fun <- function(X, coefs, ...) {
  return(1 / (1 + exp(-as.numeric(X %*% coefs))))
}


### Initialize optimizer form arbitrary values
x0 <- c(1, 1, 1)
optimizer <- SQN(x0, grad_fun=eval_grad, pred_fun=pred_fun,
  hess_vec_fun=eval_Hess_vec, initial_step=1e-0)
val_data <- generate_data_batch(true_coefs, n=100)

### Fit to 250 batches of data, 100 observations each
set.seed(1)
for (i in 1:250) {
  new_batch <- generate_data_batch(true_coefs, n=100)
  partial_fit(optimizer, new_batch$X, new_batch$y, lambda=1e-5)
  x_curr <- get_curr_x(optimizer)
  i_curr <- get_iteration_number(optimizer)
  if ((i_curr %% 10)  == 0) {
    cat(sprintf("Iteration %3d - E[f(x)]: %f - values of x: [%f, %f, %f]\n",
      i_curr, eval_fun(x_curr, val_data$X, val_data$y, lambda=1e-5),
      x_curr[1], x_curr[2], x_curr[3]))
  }
}

### Predict for new data
new_batch <- generate_data_batch(true_coefs, n=10)
yhat <- predict(optimizer, new_batch$X)
```

SQN_free *SQN Free-Mode Optimizer*

Description

Optimizes an empirical (convex) loss function over batches of sample data. Compared to function/class 'SQN', this version lets the user do all the calculations from the outside, only interacting with the object by means of a function that returns a request type and is fed the required calculation through methods 'update_gradient' and 'update_hess_vec'.

Order in which requests are made:

========== loop ==========

* calc_grad
    ... (repeat calc_grad)

if 'use_grad_diff':
    * calc_grad_big_batch

else:
    * calc_hess_vec

==========

After running this function, apply 'run_SQN_free' to it to get the first requested piece of information.

Usage

```
SQN_free(mem_size = 10, bfgs_upd_freq = 20, min_curvature = 1e-04,
  y_reg = NULL, use_grad_diff = FALSE, check_nan = TRUE,
  nthreads = -1)
```

Arguments

mem_size	Number of correction pairs to store for approximation of Hessian-vector products.
bfgs_upd_freq	Number of iterations (batches) after which to generate a BFGS correction pair.
min_curvature	Minimum value of (s * y) / (s * s) in order to accept a correction pair. Pass 'NULL' for no minimum.
y_reg	Regularizer for 'y' vector (gets added y_reg * s). Pass 'NULL' for no regularization.
use_grad_diff	Whether to create the correction pairs using differences between gradients instead of Hessian-vector products. These gradients are calculated on a larger batch than the regular ones (given by batch_size * bfgs_upd_freq).
check_nan	Whether to check for variables becoming NaN after each iteration, and reverting the step if they do (will also reset BFGS memory).
nthreads	Number of parallel threads to use. If set to -1, will determine the number of available threads and use all of them. Note however that not all the computations can be parallelized, and the BLAS backend might use a different number of threads.

Value

An 'SQN_free' object, which can be used through functions 'update_gradient', 'update_hess_vec', and 'run_SQN_free'

References

- Byrd, R.H., Hansen, S.L., Nocedal, J. and Singer, Y., 2016. "A stochastic quasi-Newton method for large-scale optimization." SIAM Journal on Optimization, 26(2), pp.1008-1031.
- Wright, S. and Nocedal, J., 1999. "Numerical optimization." (ch 7) Springer Science, 35(67-68), p.7.

See Also

update_gradient , update_hess_vec , run_oLBFGS_free

Examples

```
### Example optimizing Rosenbrock 2D function
### Note that this example is not stochastic, as the
### function is not evaluated in expectation based on
### batches of data, but rather it has a given absolute
### form that never varies.
### Warning: this optimizer is meant for convex functions
### (Rosenbrock's is not convex)
library(stochQN)


fr <- function(x) { ## Rosenbrock Banana function
 x1 <- x[1]
 x2 <- x[2]
 100 * (x2 - x1 * x1)^2 + (1 - x1)^2
}
grr <- function(x) { ## Gradient of 'fr'
 x1 <- x[1]
 x2 <- x[2]
 c(-400 * x1 * (x2 - x1 * x1) - 2 * (1 - x1),
   200 * (x2 - x1 * x1))
}
Hvr <- function(x, v) { ## Hessian of 'fr' by vector 'v'
 x1 <- x[1]
 x2 <- x[2]
 H <- matrix(c(1200 * x1^2 - 400*x2 + 2,
     -400 * x1, -400 * x1, 200),
   nrow = 2)
 as.vector(H %*% v)
}

### Initial values of x
x_opt = as.numeric(c(0, 2))
cat(sprintf("Initial values of x: [%.3f, %.3f]\n",
   x_opt[1], x_opt[2]))

### Will use constant step size throughout
### (not recommended)
step_size = 1e-3
```

```
### Initialize the optimizer
optimizer = SQN_free()

### Keep track of the iteration number
curr_iter <- 0

### Run a loop for severa, iterations
### (Note that some iterations might require more
###  than 1 calculation request)
for (i in 1:200) {
  req <- run_SQN_free(optimizer, x_opt, step_size)
  if (req$task == "calc_grad") {
    update_gradient(optimizer, grr(req$requested_on$req_x))
  } else if (req$task == "calc_hess_vec") {
    update_hess_vec(optimizer,
      Hvr(req$requested_on$req_x, req$requested_on$req_vec))
  }

  ### Track progress every 10 iterations
  if (req$info$iteration_number > curr_iter) {
   curr_iter <- req$info$iteration_number
  }
  if ((curr_iter %% 10) == 0) {
   cat(sprintf(
    "Iteration %3d - Current function value: %.3f\n",
    req$info$iteration_number, fr(x_opt)))
  }
}
cat(sprintf("Current values of x: [%.3f, %.3f]\n",
   x_opt[1], x_opt[2]))
```

`stochastic.logistic.regression` *Stochastic Logistic Regression*

Description

Stochastic Logistic Regression

Usage

```
stochastic.logistic.regression(formula = NULL, pos_class = NULL,
  dim = NULL, intercept = TRUE, x0 = NULL, optimizer = "adaQN",
  optimizer_args = list(initial_step = 0.1, verbose = FALSE),
  lambda = 0.001, random_seed = 1, val_data = NULL)
```

Arguments

`formula`	Formula for the model, if it is fit to data.frames instead of matrices/vectors.

pos_class	If fit to data in the form of data.frames, a string indicating which of the classes is the positive one. If fit to data in the form of matrices/vector, pass 'NULL'.
dim	Dimensionality of the model (number of features). Ignored when passing 'formula' or when passing 'x0'. If the intercept is added from the option 'intercept' here, it should not be counted towards 'dim'.
intercept	Whether to add an intercept to the covariates. Only ussed when fitting to matrices/vectors. Ignored when passing formula (for formulas without intercept, put '-1' in the RHS to get rid of the intercept).
x0	Initial values of the variables. If passed, will ignore 'dim' and 'random_seed'. If not passed, will generate random starting values ~ Norm(0, 0.1).
optimizer	The optimizer to use - one of 'adaQN' (recommended), 'SQN', 'oLBFGS'.
optimizer_args	Arguments to pass to the optimizer (same ones as the functions of the same name). Must be a list. See the documentation of each optimizer for the parameters they take.
lambda	Regularization parameter. Be aware that the functions assume the log-likelihood (a.k.a. loss) is divided by the number of observations, so this number should be small.
random_seed	Random seed to use for the initialization of the variables. Ignored when passing 'x0'.
val_data	Validation data (only used for 'adaQN'). If passed, must be a list with entries 'X', 'y' (if passing data.frames for fitting), and optionally 'w' (sample weights).

Details

Binary logistic regression, fit in batches using this package's own optimizers.

Value

An object of class 'stoch_logistic', which can be fit to batches of data through functon 'partial_fit_logistic'.

See Also

partial_fit_logistic, coef.stoch_logistic , predict.stoch_logistic , adaQN , SQN, oLBFGS

Examples

```
library(stochQN)

### Load Iris dataset
data("iris")

### Example with X + y interface
X <- as.matrix(iris[, c("Sepal.Length", "Sepal.Width",
  "Petal.Length", "Petal.Width")])
y <- as.numeric(iris$Species == "setosa")

### Initialize model with default parameters
```

```
model <- stochastic.logistic.regression(dim = 4)

### Fit to 10 randomly-subsampled batches
batch_size <- as.integer(nrow(X) / 3)
for (i in 1:10) {
  set.seed(i)
  batch <- sample(nrow(X),
      size = batch_size, replace=TRUE)
  partial_fit_logistic(model, X, y)
}

### Check classification accuracy
cat(sprintf(
  "Accuracy after 10 iterations: %.2f%%\n",
  100 * mean(
    predict(model, X, type = "class") == y)
  ))


### Example with formula interface
iris_df <- iris
levels(iris_df$Species) <- c("setosa", "other", "other")

### Initialize model with default parameters
model <- stochastic.logistic.regression(Species ~ .,
  pos_class="setosa")

### Fit to 10 randomly-subsampled batches
batch_size <- as.integer(nrow(iris_df) / 3)
for (i in 1:10) {
  set.seed(i)
  batch <- sample(nrow(iris_df),
      size=batch_size, replace=TRUE)
  partial_fit_logistic(model, iris_df)
}
cat(sprintf(
  "Accuracy after 10 iterations: %.2f%%\n",
  100 * mean(
    predict(
      model, iris_df, type = "class") == iris_df$Species
      )
  ))
```

---

summary.stoch_logistic *Print general info about stochastic logistic regression object*

Description

Same as 'print' function. To check the fitted coefficients use function 'coef'.

Usage

```
## S3 method for class 'stoch_logistic'
summary(object, ...)
```

Arguments

object	A ‘stoch_logistic‘ object as output by function ‘stochastic.logistic.regression‘.
...	Not used.

See Also

coef.stoch_logistic , stochastic.logistic.regression

---

update_fun *Update objective function value (adaQN)*

Description

Update the (expected) value of the objective function in an ‘adaQN_free‘ object, after it has been requested by the optimizer (do NOT update it otherwise).

Usage

```
update_fun(optimizer, fun)
```

Arguments

optimizer	An ‘adaQN_free‘ object which after the last run had requested a new function evaluation.
fun	Function as evaluated (in expectation) on the values of ‘x‘ that were returned in the request.

Value

No return value (object is updated in-place).

update_gradient *Update gradient (oLBFGS, SQN, adaQN)*

Description

Update the (expected) gradient in an optimizer from this package, after it has been requested by the optimizer (do NOT update it otherwise).

Usage

```
update_gradient(optimizer, gradient)
```

Arguments

optimizer	A free-mode optimizer from this package (‘oLBFGS_free‘, ‘SQN_free‘, ‘adaQN_free‘) which after the last run had requested a new gradient evaluation..
gradient	The (expected value of the) gradient as evaluated on the values of ‘x‘ that were returned in the request. Must be a numeric vector.

Value

No return value (object is updated in-place).

update_hess_vec *Update Hessian-vector product (SQN)*

Description

Update the (expected) values of the Hessian-vector product in an ‘SQN_free‘ object, after it has been requested by the optimizer (do NOT update it otherwise).

Usage

```
update_hess_vec(optimizer, hess_vec)
```

Arguments

optimizer	An ‘SQN_free‘ optimizer which after the last run had requested a new Hessian-vector evaluation.
hess_vec	The (expected) value of the Hessian evaluated at the values of ‘x‘ that were returned in the request, multiplied by the vector that was returned in the same request. Must be a numeric vector.

Value

No return value (object is updated in-place).

Index

adaQN, 2, *9, 10*, *16*, *18*, *30*
adaQN_free, *4*, 6, *22*

coef.stoch_logistic, 8, *30*, *32*

get_curr_x, 9
get_iteration_number, 10

oLBFGS, *9*, 10, *10*, *16*, *18*, *30*
oLBFGS_free, *12*, 13, *23*

partial_fit, *4*, *12*, 15, *25*
partial_fit_logistic, 16, *30*
predict.stoch_logistic, 17, *30*
predict.stochQN_guided, *4*, *12*, 17, *25*
print.adaQN, 18
print.adaQN_free, 18
print.oLBFGS, 19
print.oLBFGS_free, 19
print.SQN, 20
print.SQN_free, 20
print.stoch_logistic, 21

run_adaQN_free, *7*, 21
run_oLBFGS_free, *14*, 22, *28*
run_SQN_free, 23

SQN, *9, 10*, *16*, *18*, 24, *30*
SQN_free, *23*, *25*, 26
stochastic.logistic.regression, *9*, *17*, *21*, 29, *32*
summary.stoch_logistic, 31

update_fun, *7*, 32
update_gradient, *7*, *14*, *28*, 33
update_hess_vec, *28*, 33