Package 'fastglm'

May 8, 2026

Type Package

Title Fast and Stable Fitting of Generalized Linear Models using 'RcppEigen'

Version 0.1.0

Description Fits generalized linear models efficiently using 'RcppEigen'. The iteratively reweighted least squares implementation utilizes the step-halving approach of Marschner (2011) <doi:10.32614/RJ-2011-012> to help safeguard against convergence issues.

BugReports https://github.com/jaredhuling/fastglm/issues

URL https://github.com/jaredhuling/fastglm, https://jaredhuling.org/fastglm/

License GPL (>= 2)

Encoding UTF-8

Depends R (>= 4.0.0)

Imports Rcpp (>= 0.12.13), bigmemory, methods, Matrix, Formula

LinkingTo Rcpp, RcppEigen, BH, bigmemory

Suggests glm2, knitr, rmarkdown, testthat (>= 3.0.0), sandwich, microbenchmark, MASS, statmod, logistf, pscl, speedglm, tweedie

Config/testthat/edition 3

RoxygenNote 7.3.3

VignetteBuilder knitr

NeedsCompilation yes

Author Jared Huling [aut, cre], Douglas Bates [cph], Dirk Eddelbuettel [cph], Romain Francois [cph], Yixuan Qiu [cph], Noah Greifer [ctb] (ORCID: <https://orcid.org/0000-0003-3067-7154>)

Maintainer Jared Huling <jaredhuling@gmail.com>

Repository CRAN

Date/Publication 2026-05-05 13:10:02 UTC

Contents

`fastglm` *Fast generalized linear model fitting*

Description

Fast generalized linear model fitting

bigLm default

Usage

```
fastglm(x, ...)

## Default S3 method:
fastglm(
  x,
  y,
  family = gaussian(),
  weights = NULL,
  offset = NULL,
  start = NULL,
  etastart = NULL,
  mustart = NULL,
  method = 0L,
  tol = 1e-08,
```

```
  maxit = 100L,
  firth = FALSE,
  ...
)
```

Arguments

x	input model matrix. Must be a matrix object
...	not used
y	numeric response vector of length nobs.
family	a description of the error distribution and link function to be used in the model. For fastglm this can be a character string naming a family function, a family function or the result of a call to a family function. For fastglmPure only the third option is supported. (See family for details of family functions.)
weights	an optional vector of 'prior weights' to be used in the fitting process. Should be a numeric vector.
offset	this can be used to specify an a priori known component to be included in the linear predictor during fitting. This should be a numeric vector of length equal to the number of cases
start	starting values for the parameters in the linear predictor.
etastart	starting values for the linear predictor.
mustart	values for the vector of means.
method	an integer scalar with value 0 for the column-pivoted QR decomposition, 1 for the unpivoted QR decomposition, 2 for the LLT Cholesky, or 3 for the LDLT Cholesky
tol	threshold tolerance for convergence. Should be a positive real number
maxit	maximum number of IRLS iterations. Should be an integer
firth	logical; if 'TRUE' apply Firth's (1993) bias-reducing penalty to the score function. Currently supported only for 'family = binomial(link = "logit")' on dense 'x'. See 'logistf::logistf()' for the canonical reference implementation.

Value

A list with the elements

coefficients	a vector of coefficients
se	a vector of the standard errors of the coefficient estimates
rank	a scalar denoting the computed rank of the model matrix
df.residual	a scalar denoting the degrees of freedom in the model
residuals	the vector of residuals
s	a numeric scalar - the root mean square for residuals
fitted.values	the vector of fitted values

See Also

[fastglm_fit()]

Examples

```
x <- matrix(rnorm(10000 * 100), ncol = 100)
y <- 1 * (0.25 * x[,1] - 0.25 * x[,3] > rnorm(10000))

system.time(gl1 <- glm.fit(x, y, family = binomial()))

system.time(gf1 <- fastglm(x, y, family = binomial()))

system.time(gf2 <- fastglm(x, y, family = binomial(), method = 1))

system.time(gf3 <- fastglm(x, y, family = binomial(), method = 2))

system.time(gf4 <- fastglm(x, y, family = binomial(), method = 3))

max(abs(coef(gl1) - gf1$coef))
max(abs(coef(gl1) - gf2$coef))
max(abs(coef(gl1) - gf3$coef))
max(abs(coef(gl1) - gf4$coef))


## Not run:
nrows <- 50000
ncols <- 50
bkFile <- "bigmat2.bk"
descFile <- "bigmatk2.desc"
bigmat <- filebacked.big.matrix(nrow=nrows, ncol=ncols, type="double",
                                backingfile=bkFile, backingpath=".",
                                descriptorfile=descFile,
                                dimnames=c(NULL,NULL))
for (i in 1:ncols) bigmat[,i] = rnorm(nrows)*i
y <- 1*(rnorm(nrows) + bigmat[,1] > 0)

system.time(gfb1 <- fastglm(bigmat, y, family = binomial(), method = 3))

## End(Not run)
```

`fastglm-sandwich` *Empirical estimating-function and bread methods for 'fastglm' objects*

Description

Methods for 'sandwich::estfun()' and 'sandwich::bread()', which let 'sandwich::vcovCL()', 'sandwich::vcovBS()', and the rest of the *sandwich* machinery work directly on 'fastglm' and 'fastglmFit' objects. Load 'sandwich' ('library(sandwich)') before calling them.

Usage

```
estfun.fastglm(x, ...)

estfun.fastglmFit(x, ...)

bread.fastglm(x, ...)

bread.fastglmFit(x, ...)
```

Arguments

x	a fitted object of class ‘"fastglm"‘ or ‘"fastglmFit"‘.
...	not used.

Details

‘estfun(x)‘ returns the per-observation empirical estimating-function contributions, an ‘n x p‘ matrix whose ‘i‘-th row is ‘(y_i - mu_i) * mu.eta_i / variance(mu_i) * x_i / dispersion‘. ‘bread(x)‘ returns ‘(X’ W X)^(-1) * n * dispersion‘ (the *sandwich* convention). For poisson, binomial, and negative-binomial families the dispersion is fixed at 1, exactly matching ‘sandwich::estfun.glm()‘ / ‘sandwich::bread.glm()‘.

Value

‘estfun()‘ returns an ‘n x p‘ matrix; ‘bread()‘ returns a ‘p x p‘ matrix.

---

fastglmPure *Fast generalized linear model fitting*

Description

Fast generalized linear model fitting

Usage

```
fastglmPure(
  x,
  y,
  family = gaussian(),
  weights = rep(1, NROW(y)),
  offset = rep(0, NROW(y)),
  start = NULL,
  etastart = NULL,
  mustart = NULL,
  method = 0L,
  tol = 1e-07,
  maxit = 100L,
```

```
  firth = FALSE
)
```

Arguments

x	input model matrix. Must be a matrix object
y	numeric response vector of length nobs.
family	a description of the error distribution and link function to be used in the model. For fastglmPure this can only be the result of a call to a family function. (See family for details of family functions.)
weights	an optional vector of 'prior weights' to be used in the fitting process. Should be a numeric vector.
offset	this can be used to specify an a priori known component to be included in the linear predictor during fitting. This should be a numeric vector of length equal to the number of cases
start	starting values for the parameters in the linear predictor.
etastart	starting values for the linear predictor.
mustart	values for the vector of means.
method	an integer scalar with value 0 for the column-pivoted QR decomposition, 1 for the unpivoted QR decomposition, 2 for the LLT Cholesky, 3 for the LDLT Cholesky, 4 for the full pivoted QR decomposition, 5 for the Bidiagonal Divide and Conquer SVD
tol	threshold tolerance for convergence. Should be a positive real number
maxit	maximum number of IRLS iterations. Should be an integer
firth	logical; if 'TRUE' apply Firth's (1993) bias-reducing penalty to the score function. Currently supported only for 'family = binomial(link = "logit")' on dense 'x'. The penalty modifies the working response by 'h_i (0.5 - mu_i) / (mu_i (1 - mu_i))' where 'h_i' is the leverage; convergence is checked on the sup-norm of the coefficient update. See 'logistf::logistf()' for the canonical reference implementation.

Value

A list with the elements

coefficients	a vector of coefficients
se	a vector of the standard errors of the coefficient estimates
rank	a scalar denoting the computed rank of the model matrix
df.residual	a scalar denoting the degrees of freedom in the model
residuals	the vector of residuals
s	a numeric scalar - the root mean square for residuals
fitted.values	the vector of fitted values

See Also

[fastglm_fit()]

Examples

```
set.seed(1)
x <- matrix(rnorm(1000 * 25), ncol = 25)
eta <- 0.1 + 0.25 * x[,1] - 0.25 * x[,3] + 0.75 * x[,5] -0.35 * x[,6] #0.25 * x[,1] - 0.25 * x[,3]
y <- 1 * (eta > rnorm(1000))

yp <- rpois(1000, eta ^ 2)
yg <- rgamma(1000, exp(eta) * 1.75, 1.75)

# binomial
system.time(gl1 <- glm.fit(x, y, family = binomial()))

system.time(gf1 <- fastglmPure(x, y, family = binomial(), tol = 1e-8))

system.time(gf2 <- fastglmPure(x, y, family = binomial(), method = 1, tol = 1e-8))

system.time(gf3 <- fastglmPure(x, y, family = binomial(), method = 2, tol = 1e-8))

system.time(gf4 <- fastglmPure(x, y, family = binomial(), method = 3, tol = 1e-8))

max(abs(coef(gl1) - gf1$coef))
max(abs(coef(gl1) - gf2$coef))
max(abs(coef(gl1) - gf3$coef))
max(abs(coef(gl1) - gf4$coef))

# poisson
system.time(gl1 <- glm.fit(x, yp, family = poisson(link = "log")))

system.time(gf1 <- fastglmPure(x, yp, family = poisson(link = "log"), tol = 1e-8))

system.time(gf2 <- fastglmPure(x, yp, family = poisson(link = "log"), method = 1, tol = 1e-8))

system.time(gf3 <- fastglmPure(x, yp, family = poisson(link = "log"), method = 2, tol = 1e-8))

system.time(gf4 <- fastglmPure(x, yp, family = poisson(link = "log"), method = 3, tol = 1e-8))

max(abs(coef(gl1) - gf1$coef))
max(abs(coef(gl1) - gf2$coef))
max(abs(coef(gl1) - gf3$coef))
max(abs(coef(gl1) - gf4$coef))

# gamma
system.time(gl1 <- glm.fit(x, yg, family = Gamma(link = "log")))

system.time(gf1 <- fastglmPure(x, yg, family = Gamma(link = "log"), tol = 1e-8))

system.time(gf2 <- fastglmPure(x, yg, family = Gamma(link = "log"), method = 1, tol = 1e-8))
```

```
system.time(gf3 <- fastglmPure(x, yg, family = Gamma(link = "log"), method = 2, tol = 1e-8))

system.time(gf4 <- fastglmPure(x, yg, family = Gamma(link = "log"), method = 3, tol = 1e-8))

max(abs(coef(gl1) - gf1$coef))
max(abs(coef(gl1) - gf2$coef))
max(abs(coef(gl1) - gf3$coef))
max(abs(coef(gl1) - gf4$coef))
```

`fastglm_fit` *Fast generalized linear model fitting*

Description

'fastglm_fit()' is a fitting method for [glm()]. It works like 'glm.fit()', i.e., by being supplied to the 'method' argument of 'glm()'.

Usage

```
fastglm_fit(
  x,
  y,
  weights = rep(1, NROW(y)),
  start = NULL,
  etastart = NULL,
  mustart = NULL,
  offset = rep(0, NROW(y)),
  family = gaussian(),
  control = list(),
  intercept = TRUE,
  singular.ok = TRUE,
  firth = FALSE
)

fastglm_control(fastmethod = 0L, tol = 1e-07, maxit = 100L, ...)

## S3 method for class 'fastglmFit'
vcov(object, ...)

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

Arguments

x	a design matrix of dimension 'n * p'. Can also be a 'big.matrix' object from **bigmemory**.
y	a vector of observations of length 'n'.

`weights`	an optional vector of 'prior weights' to be used in the fitting process. Should be 'NULL' or a numeric vector.
`start`	optional starting values for the parameters in the linear predictor.
`etastart`	optional starting values for the linear predictor.
`mustart`	optional starting values for the vector of means.
`offset`	this can be used to specify an *a priori* known component to be included in the linear predictor during fitting. This should be 'NULL' or a numeric vector of length equal to the number of cases.
`family`	a description of the error distribution and link function to be used in the model. This must be a family function or the result of a call to a family function. (See ['family'] for details of family functions.)
`control`	a list of parameters for controlling the fitting process. This is passed to 'fastglm_control()'.
`singular.ok, intercept`	See [glm.fit()].
`firth`	'logical'; if 'TRUE' apply Firth's (1993) bias-reducing penalty to the score function. Currently supported only for 'family = binomial(link = "logit")' on dense 'x'. See 'logistf::logistf()' for the canonical reference implementation.
`fastmethod`	'integer'; the method used for fitting. Allowable values include 0 for the column-pivoted QR decomposition, 1 for the unpivoted QR decomposition, 2 for the LLT Cholesky, 3 for the LDLT Cholesky, 4 for the full pivoted QR decomposition, and 5 for the Bidiagonal Divide and Conquer SVD. Default is 0. Can also be supplied as 'method' when not supplied directly as an argument from 'glm()' (see Examples).
`tol`	'numeric'; threshold tolerance for convergence.
`maxit`	'integer'; the maximum number of IRLS iterations.
`...`	for 'vcov()' and 'summary()', other arguments passed downstream.
`object`	a 'fastglmFit' object; the output of a call to 'glm()' with 'method = fastglm_fit'.

Details

The purpose of the functions documented on this page is to facilitate integration with existing [glm()] utilities in base R. 'fastglm_fit()' is just a wrapper for [fastglmPure()] with some additional quality-of-life features. The 'vcov()' and 'summary()' methods use the unscaled coefficient covariance matrix returned directly from the C++ solver, so no refit is required.

Examples

```
set.seed(1234)
n <- 1e4
x <- matrix(rnorm(n * 25), ncol = 25)
eta <- 0.1 + 0.25 * x[,1] - 0.25 * x[,3] + 0.75 * x[,5] -0.35 * x[,6]
dat <- as.data.frame(x)

# binomial
dat$y <- rbinom(n, 1, pnorm(eta))
```

```
system.time({
    gl <- glm(y ~ ., data = dat,
              family = binomial)
})

system.time({
    gf0 <- glm(y ~ ., data = dat,
               family = binomial,
               method = fastglm_fit)
})

system.time({
    gf1 <- glm(y ~ ., data = dat,
               family = binomial,
               method = fastglm_fit,
               fastmethod = 1)
})

# poisson
dat$y <- rpois(n, eta^2)

system.time({
    gl <- glm(y ~ ., data = dat,
              family = poisson)
})

system.time({
    gf0 <- glm(y ~ ., data = dat,
               family = poisson,
               method = fastglm_fit)
})

system.time({
    gf1 <- glm(y ~ ., data = dat,
               family = poisson,
               method = fastglm_fit,
               fastmethod = 1)
})

# gamma
dat$y <- rgamma(n, exp(eta) * 1.75, 1.75)

system.time({
    gl <- glm(y ~ ., data = dat,
              family = Gamma(link = "log"))
})

system.time({
    gf0 <- glm(y ~ ., data = dat,
               family = Gamma(link = "log"),
               method = fastglm_fit)
})
```

```
system.time({
    gf1 <- glm(y ~ ., data = dat,
               family = Gamma(link = "log"),
               method = fastglm_fit,
               fastmethod = 1)
})

# Different (equivalent) ways of supplying
# control arguments:
gf1 <- glm(y ~ ., data = dat,
           family = Gamma(link = "log"),
           method = fastglm_fit,
           fastmethod = 1)

gf1 <- glm(y ~ ., data = dat,
           family = Gamma(link = "log"),
           method = fastglm_fit,
           control = list(fastmethod = 1))

gf1 <- glm(y ~ ., data = dat,
           family = Gamma(link = "log"),
           method = fastglm_fit,
           control = list(method = 1))
```

`fastglm_hurdle` *Hurdle (two-part) Poisson / Negative-Binomial regression*

Description

Fits a hurdle model for count data with excess zeros: a binary (zero / non-zero) regression on the full sample combined with a zero- truncated count regression on the positive subset. The two parts have independent likelihoods and are estimated jointly. With ‘dist = "negbin"‘ the negative-binomial dispersion ‘theta‘ is estimated by an inner Brent-1D MLE inside the C++ driver, alternating with the count-side IRLS.

Usage

```
fastglm_hurdle(
  formula,
  data,
  subset,
  na.action,
  weights,
  offset,
  dist = c("poisson", "negbin"),
  zero.dist = "binomial",
  link = c("logit", "probit", "cloglog", "log"),
  init.theta = NULL,
```

```
  tol = 1e-08,
  maxit = 100L,
  outer.tol = 1e-07,
  outer.maxit = 50L,
  theta.tol = 1e-08,
  theta.maxit = 100L,
  model = TRUE,
  y = TRUE,
  x = FALSE
)
```

Arguments

`formula`	a [Formula::Formula] of the form ‘y ~ x1 + x2	z1 + z2‘ where the right-hand side after ‘|‘ specifies the zero-model design. If ‘|‘ is absent, the same RHS is used for both parts.
`data`	optional data frame / environment from which to draw model matrix variables.
`subset`, `na.action`	standard model-frame arguments.
`weights`	optional non-negative prior weights.
`offset`	optional vector or matrix. If a 2-column matrix, columns are taken as ‘(count_offset, zero_offset)‘.
`dist`	count component distribution: ‘"poisson"‘ or ‘"negbin"‘.
`zero.dist`	zero/non-zero distribution; only ‘"binomial"‘ is currently supported.
`link`	character, link for the zero/non-zero binomial. One of ‘"logit"‘ (default), ‘"probit"‘, ‘"cloglog"‘, ‘"log"‘.
`init.theta`	optional positive scalar starting value for the NB dispersion. If ‘NULL‘, a method-of-moments initializer is used.
`tol`, `maxit`	IRLS-loop convergence tolerance and iteration cap.
`outer.tol`, `outer.maxit`	‘(beta, theta)‘ outer-loop convergence tolerance and iteration cap (only used when ‘dist = "negbin"‘).
`theta.tol`, `theta.maxit`	Brent-loop tolerance / iteration cap for the inner theta MLE (only used when ‘dist = "negbin"‘).
`model`	logical; keep the model frame on the result.
`y`, `x`	logical; keep the response / design matrices on the result.

Details

This is the fastglm analogue of [pscl::hurdle()] – coefficients agree to 1e-6 on standard datasets, with all numerical loops in C++.

Value

A list of class ‘c("fastglm_hurdle", "fastglm")‘ with elements ‘coefficients‘ (a list with ‘count‘ and ‘zero‘), ‘se‘ (likewise), ‘vcov‘ (block-diagonal), ‘loglik‘, ‘loglik_count‘, ‘loglik_zero‘, ‘theta‘, ‘SE.theta‘, ‘iter_*‘, and ‘converged‘.

Examples

```
set.seed(1)
n <- 300
x1 <- rnorm(n); x2 <- rnorm(n)
eta_count <- 1.0 + 0.4*x1 - 0.2*x2
lam <- exp(eta_count)
y_count <- rpois(n, lam)
eta_zero  <- -0.5 + 0.6*x1
p_pos     <- plogis(eta_zero)
is_pos    <- rbinom(n, 1, p_pos)
# zero-truncated: resample any rare zero from positive Poisson
for (i in seq_len(n)) {
  while (y_count[i] == 0) y_count[i] <- rpois(1, lam[i])
}
y <- ifelse(is_pos == 1, y_count, 0)
df <- data.frame(y = y, x1 = x1, x2 = x2)
fit <- fastglm_hurdle(y ~ x1 + x2, data = df, dist = "poisson")
coef(fit, model = "count")
coef(fit, model = "zero")
```

---

fastglm_nb *Fit a negative-binomial GLM with simultaneous (beta, theta) MLE*

Description

‘fastglm_nb()‘ fits a negative-binomial regression model, jointly maximising the likelihood over the regression coefficients ‘beta‘ and the NB2 dispersion ‘theta‘. It is the fastglm analogue of [MASS::glm.nb()], built on top of the native NB family kernel introduced in 0.0.6 so that all numerical loops – IRLS, the inner theta MLE Brent root-find, and the outer (beta, theta) alternation – run entirely in C++.

Usage

```
fastglm_nb(
  x,
  y,
  weights = NULL,
  offset = NULL,
  start = NULL,
  init.theta = NULL,
  link = c("log", "sqrt", "identity"),
```

```
  method = 2L,
  tol = 1e-08,
  maxit = 100L,
  outer.maxit = 25L,
  outer.tol = 1e-07,
  theta.tol = 1e-08,
  theta.maxit = 100L
)
```

Arguments

x	design matrix (numeric matrix, dgCMatrix not yet supported here).
y	non-negative integer response vector.
weights	optional prior weights vector of length ‘length(y)‘.
offset	optional offset vector of length ‘length(y)‘.
start	optional starting values for ‘beta‘.
init.theta	optional starting value for ‘theta‘. If ‘NULL‘, uses the method-of-moments estimator from a Poisson pilot fit.
link	character, one of ‘"log"‘ (default), ‘"sqrt"‘, ‘"identity"‘.
method	integer; ‘0..5‘, see [fastglm()].
tol	convergence tolerance for the IRLS inner loop.
maxit	maximum number of inner-loop IRLS iterations.
outer.maxit	maximum number of ‘(beta, theta)‘ outer iterations.
outer.tol	convergence tolerance for the outer loop on the sup-norm of the beta update plus the relative change in ‘theta‘.
theta.tol	Brent tolerance for the inner theta MLE.
theta.maxit	max iterations for the inner theta MLE.

Value

A list of class ‘c("fastglm_nb", "fastglm")‘ with the usual fastglm components plus ‘theta‘, ‘SE.theta‘, ‘iter.theta‘, and ‘twologlik‘ (twice the maximised NB log-likelihood).

Examples

```
set.seed(1)
n <- 500
x <- cbind(1, matrix(rnorm(n * 2), n, 2))
eta <- x %*% c(0.3, 0.5, -0.2)
mu  <- exp(eta)
if (requireNamespace("MASS", quietly = TRUE)) {
  y   <- MASS::rnegbin(n, mu = mu, theta = 2)
  fit <- fastglm_nb(x, y)
  c(theta = fit$theta, fit$coefficients)
}
```

`fastglm_streaming` *Fit a GLM by streaming row-blocks of the design matrix*

Description

'fastglm_streaming()' runs an IRLS GLM fit where the design matrix is produced one chunk at a time by a user-supplied closure. It never holds more than a single chunk in memory at once, plus a 'p x p' accumulator. This is the front-end for fitting on data sources too large to load completely into RAM (Arrow datasets, Parquet files, DuckDB query results, on-disk CSV streams, etc.).

Usage

```
fastglm_streaming(
  chunk_callback,
  n_chunks,
  family = gaussian(),
  start = NULL,
  method = 2L,
  tol = 1e-07,
  maxit = 100L
)
```

Arguments

`chunk_callback`	a function. Called as 'chunk_callback(k)' for 'k = 1, ..., n_chunks'. Must return a list with elements **'X'** an 'n_k x p' numeric matrix (chunk of the design matrix). **'y'** numeric vector of length 'n_k' (response). **'weights'** optional; numeric vector of length 'n_k' of prior weights. **'offset'** optional; numeric vector of length 'n_k' of offsets. Every chunk must have the same number of columns, in the same order. The closure is called multiple times per IRLS iteration, so it should be reasonably cheap (e.g. an Arrow scanner that reads from columnar files).
`n_chunks`	integer; the number of chunks to iterate over.
`family`	a 'family' object describing the error distribution and link. See [stats::family()].
`start`	optional length-'p' numeric vector of starting coefficients.
`method`	integer; '2' for LLT Cholesky (default) or '3' for LDLT Cholesky. QR / SVD methods are not supported in streaming mode.
`tol`	convergence tolerance on the relative change in deviance.
`maxit`	maximum number of IRLS iterations.

Details

The IRLS loop and step-halving (Marschner 2011) run entirely in C++; the R closure is called only to *deliver* one chunk at a time. For tier-1 families (gaussian / binomial / poisson / Gamma / inverse.gaussian on their common links) the family functions are evaluated inline in C++, so the only R round-trip per iteration is the chunk fetch.

Standard errors and ‘cov.unscaled‘ come from the final ‘(X’ W X)‘ Cholesky factor, exactly as in the in-memory path.

Value

A list with class ‘"fastglm"‘ containing the same elements as [fastglm()], including ‘coefficients‘, ‘cov.unscaled‘, ‘deviance‘, ‘iter‘, ‘converged‘, etc. The design matrix is *not* attached, so ‘sandwich::vcovHC()‘ / ‘sandwich::vcovCL()‘ will require re-streaming.

Arrow / Parquet recipe

Wrap an Arrow scanner in a closure – pull each ‘RecordBatch‘ as a chunk, build the model matrix, and return it together with the response: ‘chunks(k)‘ opens the dataset, scans the ‘k‘-th batch, calls ‘as.data.frame()‘, then returns ‘list(X = model.matrix(~ x1 + x2, data = tbl), y = tbl$y)‘. Pass that closure plus ‘n_chunks = length(batches)‘ to ‘fastglm_streaming()‘.

The same recipe works for DuckDB (one ‘dbReadTable‘ per chunk), CSV streamers, and custom binary formats.

Examples

```
# Simulate a "data source" that yields the design matrix in 4 row-blocks.
set.seed(1)
n <- 1000; p <- 5
X <- cbind(1, matrix(rnorm(n * (p - 1)), n, p - 1))
y <- rbinom(n, 1, plogis(X %*% c(0.2, 0.5, -0.3, 0.4, -0.2)))
chunk_size <- 250
chunks <- function(k) {
  idx <- ((k - 1) * chunk_size + 1):(k * chunk_size)
  list(X = X[idx, , drop = FALSE], y = y[idx])
}

fit_stream <- fastglm_streaming(chunks, n_chunks = 4, family = binomial())
fit_full   <- fastglm(X, y, family = binomial(), method = 2)
max(abs(coef(fit_stream) - coef(fit_full)))
```

`fastglm_zi` *Zero-inflated Poisson / Negative-Binomial regression*

Description

Fits a zero-inflated count model: a binary inflation component combined with a Poisson or NB count component, where the count component is the *original* (non-truncated) distribution. Excess zeros above what the count model implies are absorbed by the inflation component.

Usage

```
fastglm_zi(
  formula,
  data,
  subset,
  na.action,
  weights,
  offset,
  dist = c("poisson", "negbin"),
  link = c("logit", "probit", "cloglog", "log"),
  init.theta = NULL,
  em.tol = 1e-08,
  em.maxit = 100L,
  tol = 1e-09,
  maxit = 100L,
  theta.tol = 1e-08,
  theta.maxit = 100L,
  model = TRUE,
  y = TRUE,
  x = FALSE
)
```

Arguments

formula	a [Formula::Formula] of the form ‘y ~ x1 + x2 | z1 + z2‘ where the right-hand side after ‘|‘ specifies the inflation design. If ‘|‘ is absent, the same RHS is used for both parts.
data	optional data frame / environment.
subset, na.action	standard model-frame arguments.
weights	optional non-negative prior weights.
offset	optional vector or matrix. If a 2-column matrix, columns are taken as ‘(count_offset, zero_offset)‘.
dist	count component distribution: ‘"poisson"‘ or ‘"negbin"‘.
link	character, link for the inflation binomial. One of ‘"logit"‘ (default), ‘"probit"‘, ‘"cloglog"‘, ‘"log"‘.
init.theta	optional positive scalar starting value for the NB dispersion. If ‘NULL‘, a method-of-moments pilot is used.
em.tol, em.maxit	EM convergence tolerance and iteration cap.
tol, maxit	IRLS-loop convergence tolerance and iteration cap (used inside each M-step).
theta.tol, theta.maxit	Brent-loop tolerance / iteration cap for the inner theta MLE (only used when ‘dist = "negbin"‘).
model	logical; keep the model frame on the result.
y, x	logical; keep the response / design matrices on the result.

Details

Estimation uses an EM algorithm with all numerical loops in C++. The E-step computes the posterior ‘tau_i = P(Z_i = 1 | y_i)‘ analytically; M-steps fit the inflation logit/probit/cloglog/log and the Poisson/NB count regression via the existing native fastglm IRLS (with weights ‘w_i (1 - tau_i)‘ on the count side). For NB, an inner Brent MLE re-estimates ‘theta‘ after the count-side beta step. Final vcov comes from the numerical Jacobian of the analytical observed score at the EM fixed point (block-structured for ‘(gamma, beta, theta)‘).

This is the fastglm analogue of [pscl::zeroinfl()] – coefficients agree to 1e-5 on standard datasets (slightly looser than hurdle to allow for the EM iteration), with all numerical loops in C++.

Value

A list of class ‘c("fastglm_zi", "fastglm")‘ with elements ‘coefficients‘ (a list with ‘count‘ and ‘zero‘), ‘se‘ (likewise), ‘vcov‘ (full, including theta if NB), ‘loglik‘, ‘theta‘, ‘SE.theta‘, ‘tau‘ (posterior P(Z=1|y)), ‘em_iter‘, ‘converged‘.

Examples

```
set.seed(1)
n <- 400
x  <- rnorm(n)
eta_count <- 0.8 + 0.4*x
lam <- exp(eta_count)
eta_zero <- -0.6 + 0.5*x
p_inflate <- plogis(eta_zero)
z <- rbinom(n, 1, p_inflate)
y <- ifelse(z == 1, 0, rpois(n, lam))
df <- data.frame(y = y, x = x)
fit <- fastglm_zi(y ~ x, data = df, dist = "poisson")
coef(fit, model = "count")
coef(fit, model = "zero")
```

---

negbin *Negative binomial family with known dispersion*

Description

A built-in negative-binomial family for use with [fastglm()] and [fastglmPure()] when ‘theta‘ (the NB2 dispersion) is known. Equivalent to ‘MASS::negative.binomial(theta, link)‘ but without taking a hard dependency on the **MASS** package. The variance is ‘mu + mu^2 / theta‘; as ‘theta -> Inf‘ it reduces to Poisson.

Usage

```
negbin(theta, link = "log")
```

Arguments

theta	the (known) dispersion parameter; must be positive.
link	character; one of ‘"log"‘ (default), ‘"sqrt"‘, or ‘"identity"‘.

Details

For joint estimation of ‘theta‘ together with ‘beta‘, use the dedicated (forthcoming) ‘fastglm_nb()‘ entry point. The ‘negbin()‘ family here is intended for use cases where ‘theta‘ has been pre-specified or estimated separately.

Value

A ‘family‘ object with class ‘"family"‘ and ‘family$family == "Negative Binomial(theta)"‘. The object also carries the slot ‘family$theta‘, which ‘family_code()‘ and the dispatch layer use to detect the native NB fast path.

Examples

```
set.seed(1)
n <- 500
x  <- cbind(1, matrix(rnorm(n * 2), n, 2))
mu <- exp(x %*% c(0.3, 0.4, -0.2))
y  <- rpois(n, lambda = mu * rgamma(n, shape = 2, rate = 2))   # NB(theta=2)

fit <- fastglm(x, y, family = negbin(theta = 2, link = "log"))
coef(fit)
```

---

predict.fastglm *Obtains predictions and optionally estimates standard errors of those predictions from a fitted generalized linear model object.*

Description

Obtains predictions and optionally estimates standard errors of those predictions from a fitted generalized linear model object.

Usage

```
## S3 method for class 'fastglm'
predict(
  object,
  newdata = NULL,
  type = c("link", "response"),
  se.fit = FALSE,
  dispersion = NULL,
  ...
)
```

Arguments

object	a fitted object of class inheriting from '"fastglm"'.
newdata	a matrix to be used for prediction.
type	the type of prediction required. The default is on the scale of the linear predictors; the alternative "response" is on the scale of the response variable. Thus for a default binomial model the default predictions are of log-odds (probabilities on logit scale) and type = "response" gives the predicted probabilities. The "terms" option returns a matrix giving the fitted values of each term in the model formula on the linear predictor scale. The value of this argument can be abbreviated.
se.fit	logical switch indicating if standard errors are required.
dispersion	the dispersion of the GLM fit to be assumed in computing the standard errors. If omitted, that returned by summary applied to the object is used.
...	further arguments passed to or from other methods.

---

summary.fastglm *'summary()' method for 'fastglm' fitted objects*

Description

'summary()' method for 'fastglm' fitted objects

Usage

```
## S3 method for class 'fastglm'
summary(
  object,
  dispersion = NULL,
  correlation = FALSE,
  symbolic.cor = FALSE,
  ...
)
```

Arguments

object	'fastglm' fitted object
dispersion	the dispersion parameter for the family used. Either a single numerical value or 'NULL' (the default), when it is inferred from 'object'.
correlation	logical; if 'TRUE', the correlation matrix of the estimated parameters is returned.
symbolic.cor	logical; if 'TRUE', print the correlations in a symbolic form (see 'symnum') rather than as numbers.
...	not used

Value

A ‘summary.fastglm‘ object

See Also

[summary.glm()]

Examples

```
x <- matrix(rnorm(10000 * 10), ncol = 10)
y <- 1 * (0.25 * x[,1] - 0.25 * x[,3] > rnorm(10000))

fit <- fastglm(x, y, family = binomial())

summary(fit)
```

---

vcov.fastglm *‘vcov()‘ method for ‘fastglm‘ fitted objects*

Description

‘vcov()‘ method for ‘fastglm‘ fitted objects

Usage

```
## S3 method for class 'fastglm'
vcov(object, ...)
```

Arguments

`object`	a fitted object of class inheriting from ‘"fastglm"‘.
`...`	additional arguments (currently unused).

Value

The estimated variance-covariance matrix of the fitted coefficients. For rank-deficient fits, rows and columns corresponding to aliased coefficients are filled with ‘NA‘.

`vcovHC.fastglm` *Heteroskedasticity-consistent (HC) variance estimators for 'fastglm' objects*

Description

Methods for 'sandwich::vcovHC()' on objects of class '"fastglm"' and '"fastglmFit"'. Load 'sandwich' ('library(sandwich)') before calling 'vcovHC(fit)'; otherwise no 'vcovHC' generic is in scope.

Usage

```
vcovHC.fastglm(object, type = c("HC3", "HC2", "HC1", "HC0"), ...)

vcovHC.fastglmFit(object, type = c("HC3", "HC2", "HC1", "HC0"), ...)
```

Arguments

`object`	a fitted object of class '"fastglm"' or '"fastglmFit"'.
`type`	one of '"HC0"', '"HC1"', '"HC2"', '"HC3"'. Default '"HC3"' matches 'sandwich::vcovHC.glm'.
`...`	not used.

Details

Computes the Eicker-Huber-White sandwich estimator 'bread 'cov.unscaled') and 'meat = X' diag(omega_i) X'. With 's_i = w_i^2 * r_i' the score contribution from observation 'i', the omegas are:

'HC0' 'omega_i = s_i^2'

'HC1' 'HC0' rescaled by 'n / (n - p)'

'HC2' 'omega_i = s_i^2 / (1 - h_i)'

'HC3' 'omega_i = s_i^2 / (1 - h_i)^2'

where 'r_i' is the working residual '(y - mu) / mu.eta(eta)', 'w_i^2 = prior.weight * mu.eta(eta)^2 / variance(mu)' is the IRLS working weight, and 'h_i = w_i^2 * x_i' (X' W X)^(-1) x_i' is the IRLS leverage. Equivalent to 'sandwich::vcovHC.glm'.

Requires the model matrix 'x' stored on the fitted object (set automatically by 'fastglm()', 'fastglmPure()', and 'fastglm_fit()' since version 0.0.6).

Value

A 'p x p' heteroskedasticity-consistent variance-covariance matrix.

Examples

```
if (requireNamespace("sandwich", quietly = TRUE)) {
  x <- cbind(1, matrix(rnorm(500 * 4), ncol = 4))
  y <- rbinom(500, 1, plogis(x %*% c(0.2, 0.3, -0.4, 0.1, 0.2)))
  fit <- fastglm(x, y, family = binomial())
  sandwich::vcovHC(fit)
  sandwich::vcovHC(fit, type = "HC0")
}
```

%*%,big.matrix,vector-method
big.matrix prod

Description

big.matrix prod

big.matrix prod

Usage

```
## S4 method for signature 'big.matrix,vector'
x %*% y

## S4 method for signature 'vector,big.matrix'
x %*% y
```

Arguments

x	big.matrix
y	numeric vector

Index