Package 'nbfar'

October 13, 2022

Type Package

Title Negative Binomial Factor Regression Models ('nbfar')

Version 0.1

Date 2022-02-17

Maintainer Aditya Mishra <amishra@flatironinstitute.org>

Description We developed a negative binomial factor regression model to estimate structured (sparse) associations between a feature matrix X and overdispersed count data Y. With 'nbfar', microbiome count data Y can be used, for example, to associate host or environmental covariates with microbial abundances. Currently, two models are available: a) Negative Binomial reduced rank regression (NB-RRR), b) Negative Binomial co-sparse factor regression (NB-FAR). Please refer the manuscript 'Mishra, A. K., & Müller, C. L. (2021). Negative Binomial factor regression with application to microbiome data analysis. bioRxiv.' for more details.

URL https://github.com/amishra-stats/nbfar,
https://www.biorxiv.org/content/10.1101/2021.11.29.470304v1

Depends R (>= 3.5.0), stats, utils

Imports Rcpp (>= 0.12.9), MASS, magrittr, rrpack, glmnet, RcppParallel, mpath

License GPL (>= 3.0)

Encoding UTF-8

LazyData FALSE

LinkingTo Rcpp, RcppArmadillo, RcppParallel

NeedsCompilation yes

VignetteBuilder knitr

RoxygenNote 7.1.2

Suggests rmarkdown, knitr, spelling

Language en-US

Author Aditya Mishra [aut, cre], Christian Mueller [aut]

Repository CRAN

Date/Publication 2022-02-22 18:30:03 UTC

R topics documented:

`nbfar` *Negative binomial co-sparse factor regression (NBFAR)*

Description

To estimate a low-rank and sparse coefficient matrix in large/high dimensional setting, the approach extracts unit-rank components of required matrix in sequential order. The algorithm automatically stops after extracting sufficient unit rank components.

Usage

```
nbfar(
  Yt,
  X,
  maxrank = 3,
  nlambda = 40,
  cIndex = NULL,
  ofset = NULL,
  control = list(),
  nfold = 5,
  PATH = FALSE,
  nthread = 1,
  trace = FALSE,
  verbose = TRUE
)
```

Arguments

`Yt`	response matrix
`X`	design matrix; when X = NULL, we set X as identity matrix and perform generalized sparse PCA.
`maxrank`	an integer specifying the maximum possible rank of the coefficient matrix or the number of factors
`nlambda`	number of lambda values to be used along each path
`cIndex`	specify index of control variables in the design matrix X
`ofset`	offset matrix or microbiome data analysis specific scaling: common sum scaling = CSS (default), total sum scaling = TSS, median-ratio scaling = MRS, centered-log-ratio scaling = CLR

control	a list of internal parameters controlling the model fitting
nfold	number of folds in k-fold crossvalidation
PATH	TRUE/FALSE for generating solution path of sequential estimate after cross-validation step
nthread	number of thread to be used for parallelizing the crossvalidation procedure
trace	TRUE/FALSE checking progress of cross validation error
verbose	TRUE/FALSE checking progress of estimation procedure

Value

C	estimated coefficient matrix; based on GIC
Z	estimated control variable coefficient matrix
Phi	estimted dispersion parameters
U	estimated U matrix (generalize latent factor weights)
D	estimated singular values
V	estimated V matrix (factor loadings)

References

Mishra, A., Müller, C. (2022) *Negative binomial factor regression models with application to microbiome data analysis. https://doi.org/10.1101/2021.11.29.470304*

Examples

```
## Model specification:
SD <- 123
set.seed(SD)
p <- 100; n <- 200
pz <- 0
nrank <- 3                # true rank
rank.est <- 5             # estimated rank
nlam <- 20                # number of tuning parameter
s  = 0.5
q <- 30
control <- nbfar_control()  # control parameters
#
#
## Generate data
D <- rep(0, nrank)
V <- matrix(0, ncol = nrank, nrow = q)
U <- matrix(0, ncol = nrank, nrow = p)
#
U[, 1] <- c(sample(c(1, -1), 8, replace = TRUE), rep(0, p - 8))
U[, 2] <- c(rep(0, 5), sample(c(1, -1), 9, replace = TRUE), rep(0, p - 14))
U[, 3] <- c(rep(0, 11), sample(c(1, -1), 9, replace = TRUE), rep(0, p - 20))
#
  # for similar type response type setting
```

```
  V[, 1] <- c(rep(0, 8), sample(c(1, -1), 8,
    replace =
      TRUE
  ) * runif(8, 0.3, 1), rep(0, q - 16))
  V[, 2] <- c(rep(0, 20), sample(c(1, -1), 8,
    replace =
      TRUE
  ) * runif(8, 0.3, 1), rep(0, q - 28))
  V[, 3] <- c(
    sample(c(1, -1), 5, replace = TRUE) * runif(5, 0.3, 1), rep(0, 23),
    sample(c(1, -1), 2, replace = TRUE) * runif(2, 0.3, 1), rep(0, q - 30)
  )
U[, 1:3] <- apply(U[, 1:3], 2, function(x) x / sqrt(sum(x^2)))
V[, 1:3] <- apply(V[, 1:3], 2, function(x) x / sqrt(sum(x^2)))
#
D <- s * c(4, 6, 5) # signal strength varries as per the value of s
or <- order(D, decreasing = TRUE)
U <- U[, or]
V <- V[, or]
D <- D[or]
C <- U %*% (D * t(V)) # simulated coefficient matrix
intercept <- rep(0.5, q) # specifying intercept to the model:
C0 <- rbind(intercept, C)
#
Xsigma <- 0.5^abs(outer(1:p, 1:p, FUN = "-"))
# Simulated data
sim.sample <- nbfar_sim(U, D, V, n, Xsigma, C0,disp = 3, depth = 10)  # Simulated sample
# Dispersion parameter
X <- sim.sample$X[1:n, ]
Y <- sim.sample$Y[1:n, ]
X0 <- cbind(1, X)                    # 1st column accounting for intercept

# Model with known offset
set.seed(1234)
offset <- log(10)*matrix(1,n,q)
control_nbfar <- nbfar_control(initmaxit = 5000, gamma0 = 2, spU = 0.5,
spV = 0.6, lamMinFac = 1e-10, epsilon = 1e-5)
# nbfar_test <- nbfar(Y, X, maxrank = 5, nlambda = 20, cIndex = NULL,
# ofset = offset, control = control_nbfar, nfold = 5, PATH = F)
```

`nbfar_control` *Control parameters for NBFAR and NBRRR*

Description

Default value for a list of control parameters that are used to estimate the parameters of negative binomial co-sparse factor regression (NBFAR) and negative binomial reduced rank regression (NBRRR).

Usage

```
nbfar_control(
  maxit = 5000,
  epsilon = 1e-07,
  elnetAlpha = 0.95,
  gamma0 = 1,
  spU = 0.5,
  spV = 0.5,
  lamMaxFac = 1,
  lamMinFac = 1e-06,
  initmaxit = 10000,
  initepsilon = 1e-08,
  objI = 0
)
```

Arguments

maxit	maximum iteration for each sequential steps
epsilon	tolerance value required for convergence of inner loop in GCURE
elnetAlpha	elastic net penalty parameter
gamma0	power parameter for generating the adaptive weights
spU	maximum proportion of nonzero elements in each column of U
spV	maximum proportion of nonzero elements in each column of V
lamMaxFac	a multiplier of the computed maximum value (lambda_max) of the tuning parameter
lamMinFac	a multiplier to determine lambda_min as a fraction of lambda_max
initmaxit	maximum iteration for minimizing the objective function while computing the initial estimates of the model parameter
initepsilon	tolerance value required for the convergence of the objective function while computing the initial estimates of the model parameter
objI	1 or 0 to indicate that the convergence will be on the basis of objective function or not

Value

a list of controlling parameter.

References

Mishra, A., Müller, C. (2022) *Negative binomial factor regression models with application to microbiome data analysis. https://doi.org/10.1101/2021.11.29.470304*

Examples

```
control <- nbfar_control()
```

nbfar_sim *Simulated data for testing NBFAR and NBRRR model*

Description

Simulate response and covariates for multivariate negative binomial regression with a low-rank and sparse coefficient matrix. Coefficient matrix is expressed in terms of U (left singular vector), D (singular values) and V (right singular vector).

Usage

```
nbfar_sim(U, D, V, n, Xsigma, C0, disp, depth)
```

Arguments

U	specified value of U
D	specified value of D
V	specified value of V
n	sample size
Xsigma	covariance matrix used to generate predictors in X
C0	intercept value in the coefficient matrix
disp	dispersion parameter of the generative model
depth	log of the sequencing depth of the microbiome data (used as an offset in the simulated multivariate negative binomial regression model)

Value

Y	Generated response matrix
X	Generated predictor matrix

References

Mishra, A., Müller, C. (2022) *Negative binomial factor regression models with application to microbiome data analysis. https://doi.org/10.1101/2021.11.29.470304*

Examples

```
## Model specification:
SD <- 123
set.seed(SD)
p <- 100; n <- 200
pz <- 0
nrank <- 3                # true rank
rank.est <- 5             # estimated rank
nlam <- 20                # number of tuning parameter
s  = 0.5
```

```
q <- 30
control <- nbfar_control()  # control parameters
#
#
## Generate data
D <- rep(0, nrank)
V <- matrix(0, ncol = nrank, nrow = q)
U <- matrix(0, ncol = nrank, nrow = p)
#
U[, 1] <- c(sample(c(1, -1), 8, replace = TRUE), rep(0, p - 8))
U[, 2] <- c(rep(0, 5), sample(c(1, -1), 9, replace = TRUE), rep(0, p - 14))
U[, 3] <- c(rep(0, 11), sample(c(1, -1), 9, replace = TRUE), rep(0, p - 20))
#
  # for similar type response type setting
  V[, 1] <- c(rep(0, 8), sample(c(1, -1), 8,
    replace =
      TRUE
  ) * runif(8, 0.3, 1), rep(0, q - 16))
  V[, 2] <- c(rep(0, 20), sample(c(1, -1), 8,
    replace =
      TRUE
  ) * runif(8, 0.3, 1), rep(0, q - 28))
  V[, 3] <- c(
    sample(c(1, -1), 5, replace = TRUE) * runif(5, 0.3, 1), rep(0, 23),
    sample(c(1, -1), 2, replace = TRUE) * runif(2, 0.3, 1), rep(0, q - 30)
  )
U[, 1:3] <- apply(U[, 1:3], 2, function(x) x / sqrt(sum(x^2)))
V[, 1:3] <- apply(V[, 1:3], 2, function(x) x / sqrt(sum(x^2)))
#
D <- s * c(4, 6, 5) # signal strength varries as per the value of s
or <- order(D, decreasing = TRUE)
U <- U[, or]
V <- V[, or]
D <- D[or]
C <- U %*% (D * t(V)) # simulated coefficient matrix
intercept <- rep(0.5, q) # specifying intercept to the model:
C0 <- rbind(intercept, C)
#
Xsigma <- 0.5^abs(outer(1:p, 1:p, FUN = "-"))
# Simulated data
sim.sample <- nbfar_sim(U, D, V, n, Xsigma, C0,disp = 3, depth = 10)  # Simulated sample
# Dispersion parameter
X <- sim.sample$X[1:n, ]
Y <- sim.sample$Y[1:n, ]
```

---

nbrrr *Negative binomial reduced rank regression (NBRRR)*

Description

In the range of 1 to maxrank, the estimation procedure selects the rank r of the coefficient matrix using a cross-validation approach. For the selected rank, a rank r coefficient matrix is estimated that best fits the observations.

Usage

```
nbrrr(
  Yt,
  X,
  maxrank = 10,
  cIndex = NULL,
  ofset = NULL,
  control = list(),
  nfold = 5,
  trace = FALSE,
  verbose = TRUE
)
```

Arguments

Yt	response matrix
X	design matrix; when X = NULL, we set X as identity matrix and perform generalized PCA.
maxrank	an integer specifying the maximum possible rank of the coefficient matrix or the number of factors
cIndex	specify index of control variable in the design matrix X
ofset	offset matrix or microbiome data analysis specific scaling: common sum scaling = CSS (default), total sum scaling = TSS, median-ratio scaling = MRS, centered-log-ratio scaling = CLR
control	a list of internal parameters controlling the model fitting
nfold	number of folds in k-fold crossvalidation
trace	TRUE/FALSE checking progress of cross validation error
verbose	TRUE/FALSE checking progress of estimation procedure

Value

C	estimated coefficient matrix
Z	estimated control variable coefficient matrix
PHI	estimted dispersion parameters
U	estimated U matrix (generalize latent factor weights)
D	estimated singular values
V	estimated V matrix (factor loadings)

References

Mishra, A., Müller, C. (2022) *Negative binomial factor regression models with application to microbiome data analysis. https://doi.org/10.1101/2021.11.29.470304*

Examples

```
## Model specification:
SD <- 123
set.seed(SD)
p <- 50; n <- 200
pz <- 0
nrank <- 3                # true rank
rank.est <- 5             # estimated rank
nlam <- 20                # number of tuning parameter
s  = 0.5
q <- 30
control <- nbfar_control()  # control parameters
#
#
## Generate data
D <- rep(0, nrank)
V <- matrix(0, ncol = nrank, nrow = q)
U <- matrix(0, ncol = nrank, nrow = p)
#
U[, 1] <- c(sample(c(1, -1), 8, replace = TRUE), rep(0, p - 8))
U[, 2] <- c(rep(0, 5), sample(c(1, -1), 9, replace = TRUE), rep(0, p - 14))
U[, 3] <- c(rep(0, 11), sample(c(1, -1), 9, replace = TRUE), rep(0, p - 20))
#
  # for similar type response type setting
  V[, 1] <- c(rep(0, 8), sample(c(1, -1), 8,
    replace =
      TRUE
  ) * runif(8, 0.3, 1), rep(0, q - 16))
  V[, 2] <- c(rep(0, 20), sample(c(1, -1), 8,
    replace =
      TRUE
  ) * runif(8, 0.3, 1), rep(0, q - 28))
  V[, 3] <- c(
    sample(c(1, -1), 5, replace = TRUE) * runif(5, 0.3, 1), rep(0, 23),
    sample(c(1, -1), 2, replace = TRUE) * runif(2, 0.3, 1), rep(0, q - 30)
  )
U[, 1:3] <- apply(U[, 1:3], 2, function(x) x / sqrt(sum(x^2)))
V[, 1:3] <- apply(V[, 1:3], 2, function(x) x / sqrt(sum(x^2)))
#
D <- s * c(4, 6, 5) # signal strength varries as per the value of s
or <- order(D, decreasing = TRUE)
U <- U[, or]
V <- V[, or]
D <- D[or]
C <- U %*% (D * t(V)) # simulated coefficient matrix
intercept <- rep(0.5, q) # specifying intercept to the model:
```

```
C0 <- rbind(intercept, C)
#
Xsigma <- 0.5^abs(outer(1:p, 1:p, FUN = "-"))
# Simulated data
sim.sample <- nbfar_sim(U, D, V, n, Xsigma, C0,disp = 3, depth = 10)  # Simulated sample
# Dispersion parameter
X <- sim.sample$X[1:n, ]
Y <- sim.sample$Y[1:n, ]
X0 <- cbind(1, X)                    # 1st column accounting for intercept

# Model with known offset
set.seed(1234)
offset <- log(10)*matrix(1,n,q)
control_nbrr <- nbfar_control(initmaxit = 5000, initepsilon = 1e-4)
# nbrrr_test <- nbrrr(Y, X, maxrank = 5, cIndex = NULL, ofset = offset,
#                       control = control_nbrr, nfold = 5)
```

Index