Negative Binomial factor regression with application to microbiome data analysis

Aditya Mishra, Christian L. Müller

2022-02-21

```r
library(nbfar)
```

Simulation examples

We showcase the usage of **nbfar** and **nbrrr** on simulated data. We simulate response and covariates matrix for multivariate negative binomial regression with a low-rank and sparse coefficient matrix. The coefficient matrix **C** is expressed in terms of **U** (left singular vector), **D** (singular values) and **V** (right singular vector).

Data simulation

```r
## ## -----------------------Simulation settings --------------------
## Simulation setting:
## n: sample size
## p: number of predictors
## pz: number of control variable other than intercept parameter
## q: number of outcome variables
## nrank: true rank of the model
## snr: signal to noise ratio to be used in generating gaussian outcome variables
## nlam: number of lambda values to be fitted
## rank.est: maximum estimated rank to be specified to the model
## s: multiplicative factor of singular values
## nthread: number of parallel thread can be used in  parallel for cross validation
## The simulation was replicated 100 times under each setting as detailed in the paper.

#
## Model specification:
p <- 50;
example_seed <- 123
xp = 30
set.seed(example_seed)
n <- 200
nrank <- 3                # true rank
rank.est <- 5             # estimated rank
nlam <- 40                # number of tuning parameter
s  = 0.5; q <- 30
sp  = xp/p
nthread = 1
```

```r
## -------------- Generate low-rank and sparse coefficient matrix ---
## D: singular values
## U: sparse left singular vectors
## V: sparse right singular vectors
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 = T)
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)

## ----- Simulate data -----
Xsigma <- 0.5^abs(outer(1:p, 1:p, FUN = "-"))
sim.sample <- nbfar_sim(U, D, V, n, Xsigma, C0,disp = 0.75, depth = 10)  # Simulated sample
X <- sim.sample$X[1:n, ]                    # simulated predictors (training)
Y <- sim.sample$Y[1:n, ]                    # simulated responses (training)
# 1000 test sample data
sim.sample <- nbfar_sim(U, D, V, 1000, Xsigma, C0, disp = 0.75, depth = 10)
Xt <- sim.sample$X                    # simulated predictors (test)
Yt <- sim.sample$Y                    # simulated predictors (test)
X0 <- cbind(1, X)                     # 1st column accounting for intercept


# Simulate data with 20% missing entries
miss <- 0.10          # Proportion of entries missing
t.ind <- sample.int(n * q, size = miss * n * q)
y <- as.vector(Y)
y[t.ind] <- NA
Ym <- matrix(y, n, q)   # 20% of entries are missing at random
```

Negative Binomial reduced rank regression: nbrrr

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.

```
# Model fit: (full data)
set.seed(example_seed)
control_r3 <- nbfar_control(initmaxit = 10000, initepsilon = 1e-5,
                            objI = 1)
nbrrr_test <- nbrrr(Y, X, maxrank = 5, control = control_r3, nfold = 5, trace = F)
#> Fold  1 : [Error,iteration] = [ 1.002048e-05 202 ]
#> Fold  2 : [Error,iteration] = [ 1.000633e-05 177 ]
#> Fold  3 : [Error,iteration] = [ 1.003184e-05 166 ]
#> Fold  4 : [Error,iteration] = [ 1.001567e-05 199 ]
#> Fold  5 : [Error,iteration] = [ 1.002985e-05 196 ]
#> Fold  1 : [Error,iteration] = [ 1.001932e-05 414 ]
#> Fold  2 : [Error,iteration] = [ 1.003199e-05 402 ]
#> Fold  3 : [Error,iteration] = [ 1.000342e-05 436 ]
#> Fold  4 : [Error,iteration] = [ 1.001062e-05 428 ]
#> Fold  5 : [Error,iteration] = [ 1.000717e-05 417 ]
#> Fold  1 : [Error,iteration] = [ 1.001727e-05 463 ]
#> Fold  2 : [Error,iteration] = [ 1.002563e-05 464 ]
#> Fold  3 : [Error,iteration] = [ 1.000825e-05 479 ]
#> Fold  4 : [Error,iteration] = [ 1.000612e-05 489 ]
#> Fold  5 : [Error,iteration] = [ 1.000071e-05 488 ]
#> Fold  1 : [Error,iteration] = [ 1.001721e-05 473 ]
#> Fold  2 : [Error,iteration] = [ 1.001343e-05 474 ]
#> Fold  3 : [Error,iteration] = [ 1.000438e-05 488 ]
#> Fold  4 : [Error,iteration] = [ 1.001504e-05 500 ]
#> Fold  5 : [Error,iteration] = [ 1.000496e-05 498 ]
#> Fold  1 : [Error,iteration] = [ 1.00223e-05 473 ]
#> Fold  2 : [Error,iteration] = [ 1.001623e-05 488 ]
#> Fold  3 : [Error,iteration] = [ 1.002458e-05 490 ]
#> Fold  4 : [Error,iteration] = [ 1.000614e-05 498 ]
#> Fold  5 : [Error,iteration] = [ 1.001412e-05 494 ]


# Model fit:  (missing data)
set.seed(example_seed)
control_r3 <- nbfar_control(initmaxit = 10000, initepsilon = 1e-5,
                            objI = 1)
nbrrr_testm <- nbrrr(Ym, X, maxrank = 5, control = control_r3, nfold = 5,trace = F)
#> Fold  1 : [Error,iteration] = [ 1.005434e-05 172 ]
#> Fold  2 : [Error,iteration] = [ 1.00083e-05 238 ]
#> Fold  3 : [Error,iteration] = [ 1.000397e-05 204 ]
#> Fold  4 : [Error,iteration] = [ 1.004102e-05 173 ]
#> Fold  5 : [Error,iteration] = [ 1.002187e-05 198 ]
#> Fold  1 : [Error,iteration] = [ 1.000207e-05 387 ]
#> Fold  2 : [Error,iteration] = [ 1.000388e-05 429 ]
#> Fold  3 : [Error,iteration] = [ 1.002817e-05 382 ]
#> Fold  4 : [Error,iteration] = [ 1.002338e-05 439 ]
#> Fold  5 : [Error,iteration] = [ 1.001821e-05 433 ]
#> Fold  1 : [Error,iteration] = [ 1.002984e-05 441 ]
#> Fold  2 : [Error,iteration] = [ 1.002097e-05 509 ]
#> Fold  3 : [Error,iteration] = [ 1.000156e-05 437 ]
#> Fold  4 : [Error,iteration] = [ 1.000556e-05 516 ]
#> Fold  5 : [Error,iteration] = [ 1.000915e-05 499 ]
#> Fold  1 : [Error,iteration] = [ 1.001861e-05 447 ]
#> Fold  2 : [Error,iteration] = [ 1.002143e-05 518 ]
#> Fold  3 : [Error,iteration] = [ 1.000493e-05 451 ]
#> Fold  4 : [Error,iteration] = [ 1.00226e-05 525 ]
#> Fold  5 : [Error,iteration] = [ 1.000707e-05 497 ]
#> Fold  1 : [Error,iteration] = [ 1.001012e-05 441 ]
#> Fold  2 : [Error,iteration] = [ 1.002457e-05 534 ]
#> Fold  3 : [Error,iteration] = [ 1.001708e-05 457 ]
#> Fold  4 : [Error,iteration] = [ 1.001681e-05 542 ]
#> Fold  5 : [Error,iteration] = [ 1.001137e-05 505 ]
```

Negative Binomial co-sparse factor regression: nbfar

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.

```
# Model fit: (full data)
RcppParallel::setThreadOptions(numThreads = nthread)
set.seed(example_seed)
control_nbfar <- nbfar_control(gamma0 = 1, spU = sp, spV = 20/q,
                               maxit = 2000, lamMaxFac = 1e-2,
                               lamMinFac = 1e-7, epsilon = 1e-4,
                               objI = 0,
                               initmaxit = 10000, initepsilon = 1e-7)
nbfar_test <- nbfar(Y, X, maxrank = rank.est, nlambda = nlam,
                    cIndex = NULL,
                    ofset = NULL, control = control_nbfar, nfold = 5,
                    PATH = FALSE, nthread = nthread,trace = F)
#> Initializing...
#> Initializing unit-rank unit  1 : [Error,iteration,D] = [ 1.000211e-07 4065 2.647036 ]
#> Cross validation for component: 1
#> 2.369096
#> Initializing unit-rank unit  2 : [Error,iteration,D] = [ 1.000269e-07 3044 2.939305 ]
#> Cross validation for component: 2
#> 2.940586
#> Initializing unit-rank unit  3 : [Error,iteration,D] = [ 1.000032e-07 2045 1.982996 ]
#> Cross validation for component: 3
#> 1.883394
#> Initializing unit-rank unit  4 : [Error,iteration,D] = [ 1.001446e-07 1175 0.7945658 ]
#> Cross validation for component: 4
#> 0
#> Estimated rank = 3

# Model fit: (missing data)
RcppParallel::setThreadOptions(numThreads = nthread)
set.seed(example_seed)
control_nbfar <- nbfar_control(gamma0 = 1, spU = sp, spV = 20/q,
                               maxit = 2000, lamMaxFac = 1e-2,
                               lamMinFac = 1e-7, epsilon = 1e-4,
                               objI = 0,
                                initmaxit = 10000, initepsilon = 1e-7)
nbfar_testm <- nbfar(Ym, X, maxrank = rank.est, nlambda = nlam,
                     cIndex = NULL,
                     ofset = NULL, control = control_nbfar, nfold = 5,
                     PATH = FALSE, nthread = nthread,trace = F)
#> Initializing...
#> Initializing unit-rank unit  1 : [Error,iteration,D] = [ 1.000219e-07 5945 2.699055 ]
#> Cross validation for component: 1
#> 2.315749
#> Initializing unit-rank unit  2 : [Error,iteration,D] = [ 1.000287e-07 3291 2.952874 ]
#> Cross validation for component: 2
#> 2.834746
#> Initializing unit-rank unit  3 : [Error,iteration,D] = [ 1.000118e-07 2499 1.983892 ]
#> Cross validation for component: 3
#> 1.77783
#> Initializing unit-rank unit  4 : [Error,iteration,D] = [ 1.000573e-07 1411 0.8212405 ]
#> Cross validation for component: 4
#> 0
#> Estimated rank = 3
```