Package ‘transGFM’

November 13, 2025

Type Package

Title Transfer Learning for Generalized Factor Models

Version 1.0.1

Date 2025-11-10

Maintainer Zhijing Wang <wangzhijing@sjtu.edu.cn>

Description Transfer learning for generalized factor models with support for continuous, count (Poisson), and binary data types. The package provides functions for single and multiple source transfer learning, source detection to identify positive and negative transfer sources, factor decomposition using Maximum Likelihood Estimation (MLE), and information criteria ('IC1' and 'IC2') for rank selection. The methods are particularly useful for high-dimensional data analysis where auxiliary information from related source datasets can improve estimation efficiency in the target domain.

License GPL-3

Encoding UTF-8

Depends R (>= 3.5.0)

Imports stats

Suggests testthat (>= 3.0.0), knitr, rmarkdown

RoxygenNote 7.3.2

NeedsCompilation no

Author Zhijing Wang [aut, cre]

Repository CRAN

Date/Publication 2025-11-13 18:40:02 UTC

Contents

`ic_criterion` *Information criterion (IC1/IC2) for selecting number of factors*

Description

Information criterion (IC1/IC2) for selecting number of factors

Usage

```
ic_criterion(
  X,
  r_max = 10,
  ic_type = c("IC1", "IC2"),
  data_type = "count",
  C = NULL,
  max_iter = 30,
  verbose = FALSE
)
```

Arguments

`X`	Data matrix (may contain missing values coded as NA)
`r_max`	Maximum number of factors to consider (default: 10)
`ic_type`	IC criterion type: "IC1" or "IC2" (default: "IC1")
`data_type`	Type of data: "continuous", "count", or "binary"
`C`	CJMLE projection constant (if NULL, auto-calculated)
`max_iter`	Maximum CJMLE iterations (default: 30)
`verbose`	Print progress information (default: FALSE)

Value

List with r_hat (optimal rank), ic_values, loglik_values

Examples

```
# Generate Poisson data with known rank
set.seed(2025)
n <- 100; p <- 100; r_true <- 2

# Generate true factors
F_true <- matrix(runif(n * r_true, min = -2, max = 2), n, r_true)
B_true <- matrix(runif(p * r_true, min = -2, max = 2), p, r_true)
M_true <- F_true %*% t(B_true)

# Generate Poisson observations
```

```
lambda <- exp(M_true)
X <- matrix(rpois(n * p, as.vector(lambda)), n, p)

# Add 10% missing values
n_missing <- floor(n * p * 0.1)
missing_idx <- sample(n * p, n_missing)
X[missing_idx] <- NA

# Use IC1 to select rank
result_IC1 <- ic_criterion(
  X = X,
  r_max = 6,
  ic_type = "IC1",
  data_type = "count",
  verbose = TRUE
)

print(paste("True rank:", r_true))
print(paste("Estimated rank (IC1):", result_IC1$r_hat))

# Use IC2 to select rank
result_IC2 <- ic_criterion(
  X = X,
  r_max = 6,
  ic_type = "IC2",
  data_type = "count",
  verbose = TRUE
)
```

`identify` *Identify factor decomposition via SVD*

Description

Identify factor decomposition via SVD

Usage

```
identify(M, r)
```

Arguments

`M`	Matrix to decompose
`r`	Number of factors

Value

List with F (row factors) and B (column factors)

Examples

```
# Generate Poisson data
set.seed(123)
n0 <- 50; p0 <- 50; r <- 2
F_true <- matrix(runif(n0 * r, min = -2, max = 2), n0, r)
B_true <- matrix(runif(p0 * r, min = -2, max = 2), p0, r)
F_true <- F_true / sqrt(r)
B_true <- B_true / sqrt(r)
M_true <- F_true %*% t(B_true)

# Decompose using identify
result <- identify(M_true, r = 2)
F_hat <- result$F
B_hat <- result$B

# Check reconstruction
M_reconstructed <- F_hat %*% t(B_hat)
print(max(abs(M_reconstructed - M_true)))  # Should be very small
```

`relative_error` *Calculate relative error between estimated and true matrices*

Description

Calculate relative error between estimated and true matrices

Usage

```
relative_error(M_hat, M_true)
```

Arguments

`M_hat`	Estimated matrix
`M_true`	True matrix

Value

Relative Frobenius norm error

Examples

```
M_true <- matrix(1:9, 3, 3)
M_hat <- M_true + matrix(rnorm(9, 0, 0.1), 3, 3)
relative_error(M_hat, M_true)
```

source_detection *Detect positive and negative transfer sources using ratio method*

Description

Detect positive and negative transfer sources using ratio method

Usage

```
source_detection(
  X_sources,
  X0,
  r,
  C,
  C2,
  data_type = "count",
  c_penalty = 0.1,
  verbose = TRUE
)
```

Arguments

X_sources	List of source data matrices (may contain missing values)
X0	Target data matrix (complete)
r	Number of factors
C	CJMLE projection constant
C2	Refinement projection constant
data_type	Type of data: "continuous", "count", or "binary"
c_penalty	Penalty coefficient (default: 0.1)
verbose	Print progress information (default: TRUE)

Value

List with positive_sources, negative_sources, and diagnostic info `source_potential` *Identify potential sources based on rank comparison using IC criterion*

Description

Identify potential sources based on rank comparison using IC criterion

Usage

```
source_potential(
  X_sources,
  X0,
  r_max = 10,
  ic_type = "IC1",
  data_type = "count",
  C = NULL,
  max_iter = 30,
  verbose = TRUE
)
```

Arguments

`X_sources`	List of source data matrices (may contain missing values)
`X0`	Target data matrix (may contain missing values)
`r_max`	Maximum number of factors to consider (default: 10)
`ic_type`	IC criterion type: "IC1" or "IC2" (default: "IC1")
`data_type`	Type of data: "continuous", "count", or "binary"
`C`	CJMLE projection constant (if NULL, auto-calculated)
`max_iter`	Maximum CJMLE iterations (default: 30)
`verbose`	Print progress information (default: TRUE)

Value

List with positive_potential_sources, negative_sources, r_target, r_sources

Examples

```
# Generate Poisson data
set.seed(2025)

# Generate 5 sources with different ranks
n1 <- 100; p1 <- 100
source_list <- list()

# Sources 1-2: rank 2 (same as target)
r_s <- 2
```

```
F_s <- matrix(runif(n1 * r_s, min = -2, max = 2), n1, r_s)
B_s <- matrix(runif(p1 * r_s, min = -2, max = 2), p1, r_s)
M_s <- F_s %*% t(B_s)
for (s in 1:2) {
  X_s <- matrix(rpois(n1 * p1, exp(M_s)), n1, p1)

  # Add 10% missing values
  n_missing <- floor(n1 * p1 * 0.1)
  missing_idx <- sample(n1 * p1, n_missing)
  X_s[missing_idx] <- NA

  source_list[[s]] <- X_s
}

# Sources 3-5: rank 3 (different from target)
for (s in 3:5) {
  r_s_nega <- 3
  F_s_nega <- matrix(runif(n1 * r_s_nega, min = -2, max = 2), n1, r_s_nega)
  B_s_nega <- matrix(runif(p1 * r_s_nega, min = -2, max = 2), p1, r_s_nega)
  M_s_nega <- F_s_nega %*% t(B_s_nega)
  X_s_nega <- matrix(rpois(n1 * p1, exp(M_s_nega)), n1, p1)

  n_missing <- floor(n1 * p1 * 0.1)
  missing_idx <- sample(n1 * p1, n_missing)
  X_s_nega[missing_idx] <- NA

  source_list[[s]] <- X_s_nega
}

# Target data: rank 2
n0 <- 50; p0 <- 50; r_target <- 2
M_target <- M_s[1:n0, 1:p0]
X_target <- matrix(rpois(n0 * p0, exp(M_target)), n0, p0)

# Identify potential sources
result <- source_potential(
  X_sources = source_list,
  X0 = X_target,
  r_max = 5,
  ic_type = "IC1",
  data_type = "count",
  verbose = TRUE
)

print(result$positive_potential_sources)  # Should be c(1, 2)
print(result$negative_sources)            # Should be c(3, 4, 5)
print(result$r_target)                    # Should be 2
print(result$r_sources)                   # Should be c(2, 2, 3, 3, 3)
```

transGFM   *Single source transfer learning for generalized factor models*

Description

Single source transfer learning for generalized factor models

Usage

```
transGFM(
  source_data,
  target_data,
  r,
  data_type = "count",
  lambda_seq = seq(0, 10, by = 1),
  K_cv = 3,
  sigma2 = 1,
  max_iter_cjmle = 30,
  max_iter_refine = 30,
  max_iter_nuclear = 30,
  verbose = FALSE
)
```

Arguments

`source_data`	Source data matrix (may contain missing values coded as NA)
`target_data`	Target data matrix (complete)
`r`	Number of factors
`data_type`	Type of data: "continuous", "count", or "binary"
`lambda_seq`	Sequence of lambda values for CV (default: seq(0, 10, by = 1))
`K_cv`	Number of CV folds (default: 3)
`sigma2`	Variance parameter for continuous data (default: 1)
`max_iter_cjmle`	Maximum iterations for CJMLE (default: 30)
`max_iter_refine`	Maximum iterations for refinement (default: 30)
`max_iter_nuclear`	Maximum iterations for nuclear MLE (default: 100)
`verbose`	Print progress information (default: FALSE)

Value

List containing final estimate M_trans and intermediate results

Examples

```
# Generate Poisson data
set.seed(2025)

# Source data (100 x 100 with 10% missing)
n1 <- 100; p1 <- 100; r <- 2
```

```
F_source <- matrix(runif(n1 * r, min = -2, max = 2), n1, r)
B_source <- matrix(runif(p1 * r, min = -2, max = 2), p1, r)
M_source <- F_source %*% t(B_source)
lambda_source <- exp(M_source)
X_source <- matrix(rpois(n1 * p1, as.vector(lambda_source)), n1, p1)

# Add 10% missing values to source
n_missing <- floor(n1 * p1 * 0.1)
missing_idx <- sample(n1 * p1, n_missing)
X_source[missing_idx] <- NA

# Target data (50 x 50, complete)
n0 <- 50; p0 <- 50
M_target_true <- M_source[1:n0, 1:p0]
lambda_target <- exp(M_target_true)
X_target <- matrix(rpois(n0 * p0, as.vector(lambda_target)), n0, p0)

# Run transGFM
result <- transGFM(
  source_data = X_source,
  target_data = X_target,
  r = 2,
  data_type = "count",
  lambda_seq = seq(0, 5, by = 1),
  K_cv = 3,
  verbose = FALSE
)

# Check results
print(paste("Optimal lambda:", result$optimal_lambda))
print(paste("Final relative error:",
            relative_error(result$M_trans, M_target_true)))
```

transGFM_multi — *Multiple source transfer learning for generalized factor models*

Description

Multiple source transfer learning for generalized factor models

Usage

```
transGFM_multi(
  source_data_list,
  target_data,
  r,
  data_type = "count",
  method = "AD",
  lambda_seq = seq(0, 10, by = 1),
  K_cv = 3,
  sigma2 = 1,
  max_iter_cjmle = 30,
  max_iter_refine = 30,
  max_iter_nuclear = 100,
  verbose = FALSE
)
```

Arguments

`source_data_list`	List of source data matrices (may contain missing values)
`target_data`	Target data matrix (complete)
`r`	Number of factors
`data_type`	Type of data: "continuous", "count", or "binary"
`method`	Fusion method: "AD" (Average-Debias) or "DA" (Debias-Average)
`lambda_seq`	Sequence of lambda values for CV
`K_cv`	Number of CV folds
`sigma2`	Variance parameter for continuous data
`max_iter_cjmle`	Maximum iterations for CJMLE
`max_iter_refine`	Maximum iterations for refinement
`max_iter_nuclear`	Maximum iterations for nuclear MLE
`verbose`	Print progress information

Value

List containing final estimate and intermediate results

Examples

```
# Generate Poisson data
set.seed(2025)

# Generate 3 source datasets (100 x 100 with different missing rates)
n1 <- 100; p1 <- 100; r <- 2
source_list <- list()
F_s <- matrix(runif(n1 * r, min = -2, max = 2), n1, r)
B_s <- matrix(runif(p1 * r, min = -2, max = 2), p1, r)
M_s <- F_s %*% t(B_s)
for (s in 1:3) {
  X_s <- matrix(rpois(n1 * p1, exp(M_s)), n1, p1)

  # Add missing values (10%, 12%, 14% for sources 1-3)
  missing_rate <- 0.1 + (s - 1) * 0.02
  n_missing <- floor(n1 * p1 * missing_rate)
  missing_idx <- sample(n1 * p1, n_missing)
```

```
    X_s[missing_idx] <- NA

    source_list[[s]] <- X_s
  }

  # Target data (50 x 50, complete)
  n0 <- 50; p0 <- 50
  M_target_true <- M_s[1:n0, 1:p0]
  X_target <- matrix(rpois(n0 * p0, exp(M_target_true)), n0, p0)

  # Run transGFM_multi with AD method
  result_AD <- transGFM_multi(
    source_data_list = source_list,
    target_data = X_target,
    r = 2,
    data_type = "count",
    method = "AD",
    lambda_seq = seq(0, 5, by = 1),
    K_cv = 3,
    verbose = FALSE
  )

  # Run transGFM_multi with DA method
  result_DA <- transGFM_multi(
    source_data_list = source_list,
    target_data = X_target,
    r = 2,
    data_type = "count",
    method = "DA",
    verbose = FALSE
  )

  # Compare results
  print(paste("AD method error:", relative_error(result_AD$M_trans, M_target_true)))
  print(paste("DA method error:", relative_error(result_DA$M_trans, M_target_true)))
```

Index