Package 'catfda'

September 8, 2026

Type Package

Title Statistical Analysis for Categorical Functional Data

Version 0.1.0

Description Implements methods for estimating latent Gaussian processes from categorical functional data using binomial, probit, or multinomial GAM-based approaches, and for clustering individuals via multivariate functional principal component scores. Methods are described in Champon et al. (2026) <doi:10.1080/01621459.2026.2672226>.

License MIT + file LICENSE

Encoding UTF-8

URL https://github.com/XiaoxiaChampon/catfda

BugReports https://github.com/XiaoxiaChampon/catfda/issues

Imports mgcv (>= 1.8-40), stats, rlang (>= 1.0.0), foreach (>= 1.5.0), doRNG (>= 1.8.0)

Suggests doParallel, testthat (>= 3.0.0)

RoxygenNote 7.3.2

NeedsCompilation no

Author Xiaoxia Champon [aut],
Ana-Maria Staicu [aut],
Chathura Jayalath [aut, cre] (ORCID:
<https://orcid.org/0000-0002-2056-0211>),
Anthony Weishampel [ctb],
William Rand [ctb]

Maintainer Chathura Jayalath <acj.chathura@gmail.com>

Repository CRAN

Date/Publication 2026-09-08 11:50:09 UTC

Contents

`estimate_categ_func_data`

Estimate categorical functional data using the specified link function

---

Description

Main dispatcher function for categorical functional data estimation. Chooses between probit, binomial, or multinomial approaches based on the specified method.

Usage

```
estimate_categ_func_data(
  choice,
  time_points,
  w_mat,
  n_basis = 25,
  method = "ML"
)
```

Arguments

`choice`	A string: one of "probit", "binomial", or "multinomial"
`time_points`	A numeric vector of time points (0 to 1)
`w_mat`	A matrix of categorical observations (time × individuals)
`n_basis`	Integer: number of basis functions (default = 25)
`method`	Estimation method for GAM (default = "ML")

Details

This function serves as the main entry point for categorical functional data analysis. It automatically dispatches to the appropriate estimation method based on the specified choice:

- "probit": Uses adaptive probit/binomial switching for rare events
- "binomial": Uses separate binomial GAMs for each category
- "multinomial": Uses multinomial GAM for all categories simultaneously

Value

A list of latent variables (Z) and predicted probabilities (p)

Examples

```
# Generate sample data
set.seed(123)
n_time <- 50
n_individuals <- 10
time_points <- seq(0, 1, length.out = n_time)

# Create sample categorical data
w_mat <- matrix(sample(0:2, n_time * n_individuals, replace = TRUE),
                nrow = n_time, ncol = n_individuals)

# Estimate using multinomial approach
result <- estimate_categ_func_data("multinomial", time_points, w_mat, n_basis = 10)
```

---

estimate_categ_func_data_binomial_parallel

Estimate Z and p curves using binomial link GAMs (parallel version)

Description

Estimates latent Gaussian processes using separate binomial GAM models for each category. This is the parallel implementation for improved performance with large datasets.

Usage

```
estimate_categ_func_data_binomial_parallel(
  time_points,
  x_array,
  n_basis = 25,
  method = "ML"
)
```

Arguments

time_points	Numeric vector of time values (length T)
x_array	3D array (individual × time × category) — binary one-hot encoding
n_basis	Number of basis functions (default = 25)
method	GAM optimization method (default = "ML")

Details

The binomial approach fits separate binomial GAM models for each category and then transforms the results to obtain latent processes relative to a reference category (the last category).

This parallel implementation uses the `foreach` package for efficient computation across individuals.

Value

A list with: - Z1_est, ..., Z(K-1)_est (latent curves) - p1_est, ..., pK_est (category probability curves)

Examples

```
if (requireNamespace("doParallel", quietly = TRUE)) {
  library(doParallel)
  cl <- makeCluster(2)
  registerDoParallel(cl)

  # Generate sample data
  set.seed(123)
  n_time <- 75
  n_individuals <- 15
  n_categories <- 3

  time_points <- seq(0, 1, length.out = n_time)
  x_array <- array(0, dim = c(n_individuals, n_time, n_categories))

  # Fill with random one-hot vectors
  for (i in 1:n_individuals) {
    for (t in 1:n_time) {
      cat_val <- sample(1:n_categories, 1)
      x_array[i, t, cat_val] <- 1
    }
  }

  # Parallel binomial estimation
  result <- estimate_categ_func_data_binomial_parallel(time_points, x_array)

  # Cleanup
  stopCluster(cl)
}
```

`estimate_categ_func_data_multinomial`

Estimate Z and p curves for categorical functional data (multinomial case)

Description

Estimates latent Gaussian processes and probability curves using multinomial GAM models. This approach models all categories simultaneously using a single multinomial distribution.

Usage

```
estimate_categ_func_data_multinomial(
  time_points,
  w_mat,
  n_basis = 25,
  method = "ML"
)
```

Arguments

time_points	A numeric vector of time points
w_mat	A matrix of categorical values (time × individuals)
n_basis	Number of basis functions for smoothing (default = 25)
method	GAM fitting method (default = "ML")

Details

The multinomial approach fits a single GAM with multinomial family to all categories simultaneously. This is generally more efficient and statistically coherent than fitting separate binomial models, especially when the number of categories is large.

Category labels can be arbitrary (numeric or character). Internally, labels are sorted and mapped to positions 1..K via `match()`, and then the K-th position is remapped to 0 to serve as the reference class required by `mgcv::multinom`. The reference category is always the largest (last in sorted order) original label.

Value

A named list with latent Z estimates and probability estimates

Examples

```
# Generate sample data
set.seed(123)
n_time <- 30
n_individuals <- 5
time_points <- seq(0, 1, length.out = n_time)
w_mat <- matrix(sample(1:3, n_time * n_individuals, replace = TRUE),
                nrow = n_time, ncol = n_individuals)

# Estimate using multinomial approach
result <- estimate_categ_func_data_multinomial(time_points, w_mat)
names(result)
```

`estimate_categ_func_data_multinomial_parallel` *Parallel estimation of Z and p using multinomial link GAM*

Description

Parallel version of multinomial estimation for improved performance with large datasets.

Usage

```
estimate_categ_func_data_multinomial_parallel(
  time_points,
  w_mat,
  n_basis = 25,
  method = "ML"
)
```

Arguments

`time_points`	A numeric vector of time points
`w_mat`	A matrix of categorical values (time × individuals)
`n_basis`	Number of basis functions for smoothing (default = 25)
`method`	GAM fitting method (default = "ML")

Details

This is the parallel implementation of the multinomial approach. It uses the `foreach` package with `%dorng%` for reproducible parallel computation.

Requires a parallel backend to be registered (e.g., via `doParallel`) for actual parallel execution.

Value

A named list with latent Z estimates and probability estimates

Examples

```
if (requireNamespace("doParallel", quietly = TRUE)) {
  library(doParallel)
  cl <- makeCluster(2)
  registerDoParallel(cl)

  # Generate sample data
  set.seed(123)
  n_time <- 50
  n_individuals <- 20
  time_points <- seq(0, 1, length.out = n_time)
  w_mat <- matrix(sample(1:3, n_time * n_individuals, replace = TRUE),
```

```
                  nrow = n_time, ncol = n_individuals)

  # Parallel estimation
  result <- estimate_categ_func_data_multinomial_parallel(time_points, w_mat)

  # Cleanup
  stopCluster(cl)
}
```

`estimate_categ_func_data_probit` *Estimate Z and p curves using (generalized) probit or binomial models*

Description

Estimate Z and p curves using (generalized) probit or binomial models

Usage

```
estimate_categ_func_data_probit(
  time_points,
  x_array,
  n_basis = 25,
  method = "ML",
  threshold_probability = 0.004
)
```

Arguments

`time_points`	A numeric vector of time values (length T)
`x_array`	A 3D array (individual × time × category) of one-hot encoded data
`n_basis`	Number of basis functions for smoothing (default = 25)
`method`	GAM fitting method (default = "ML")
`threshold_probability`	Probability threshold for switching to probit special case (default = 0.004)

Details

This function implements an adaptive approach that switches between probit and binomial link functions based on the sparsity of categorical events:

- Uses probit link for rare events (proportion < threshold_probability)
- Uses binomial link for common events
- Only applies special probit handling for short time series (<= 301 points)

The adaptive approach helps with numerical stability when dealing with very rare categorical events.

Value

A list containing estimated Z curves and category probabilities for each individual

Examples

```
# Generate sample one-hot encoded data
set.seed(123)
n_time <- 50
n_individuals <- 8
n_categories <- 3

time_points <- seq(0, 1, length.out = n_time)
x_array <- array(0, dim = c(n_individuals, n_time, n_categories))

# Fill with random one-hot vectors
for(i in 1:n_individuals) {
  for(t in 1:n_time) {
    cat <- sample(1:n_categories, 1)
    x_array[i, t, cat] <- 1
  }
}

# Estimate using probit approach
result <- estimate_categ_func_data_probit(time_points, x_array)
```

---

estimate_categ_func_data_probit_parallel

Parallel estimation of Z and p using (probit or binomial) link GAMs

---

Description

Parallel version of probit/binomial estimation for improved performance with large datasets.

Usage

```
estimate_categ_func_data_probit_parallel(
  time_points,
  x_array,
  n_basis = 25,
  method = "ML",
  threshold_probability = 0.004
)
```

Arguments

time_points	A numeric vector of time points (length T)
x_array	A 3D array: individual × time × category (dimensions N × T × K)

`n_basis` Number of basis functions for GAM (default = 25)

`method` GAM fitting method (default = "ML")

`threshold_probability`
If response is sparse, switch to probit (default = 0.004)

Details

This is the parallel implementation of the probit/binomial approach. It uses the `foreach` package with `%dorng%` for reproducible parallel computation.

Note: Requires a parallel backend to be registered (e.g., using doParallel) for actual parallel execution.

Value

A list of estimated latent curves (Z1, Z2, ..., ZK-1) and normalized probabilities (p1, ..., pK)

Examples

```
if (requireNamespace("doParallel", quietly = TRUE)) {
  library(doParallel)
  cl <- makeCluster(2)
  registerDoParallel(cl)

  # Generate sample data
  set.seed(123)
  n_time <- 100
  n_individuals <- 20
  n_categories <- 4

  time_points <- seq(0, 1, length.out = n_time)
  x_array <- array(0, dim = c(n_individuals, n_time, n_categories))

  # Fill with random one-hot vectors
  for (i in 1:n_individuals) {
    for (t in 1:n_time) {
      cat_val <- sample(1:n_categories, 1)
      x_array[i, t, cat_val] <- 1
    }
  }

  # Parallel estimation
  result <- estimate_categ_func_data_probit_parallel(time_points, x_array)

  # Cleanup
  stopCluster(cl)
}
```

`generate_categ_func_data` *Generate synthetic categorical functional data (W) and one-hot encoded array (X)*

Description

Generates synthetic categorical functional data from probability curves using multinomial sampling. This is useful for simulation studies and testing.

Usage

```
generate_categ_func_data(prob_curves, tol = 1e-08)
```

Arguments

`prob_curves`	A named list of T × N matrices: p1_est, ..., pK_est
`tol`	Numeric tolerance for probability sum validation (default = 1e-8)

Details

The function takes probability curves for each category and generates categorical observations by sampling from multinomial distributions at each time point for each individual.

The probability curves should be provided as a named list where each element is a matrix of dimension (time × individuals).

Value

A list with: - w_mat: Categorical matrix (T × N) - x_array: One-hot encoded array (N × T × K)

Examples

```
# Create sample probability curves
set.seed(123)
n_time <- 30
n_individuals <- 10
time_points <- seq(0, 1, length.out = n_time)

# Generate simple probability matrices that sum to 1
p1 <- matrix(0.4, nrow = n_time, ncol = n_individuals)
p2 <- matrix(0.3, nrow = n_time, ncol = n_individuals)
p3 <- matrix(0.3, nrow = n_time, ncol = n_individuals)

prob_curves <- list(p1_est = p1, p2_est = p2, p3_est = p3)

# Generate categorical data
result <- generate_categ_func_data(prob_curves)
```

`get_x_from_w` *Create one-hot encoded functional data array from categorical observations*

Description

Converts a matrix of categorical observations into a 3D array of one-hot encoded binary indicators. This is a utility function commonly needed for preprocessing categorical functional data.

Usage

```
get_x_from_w(w_mat, categories = NULL)
```

Arguments

`w_mat`	A matrix of categorical observations (time × individuals)
`categories`	Optional character or numeric vector of category labels. If `NULL` (default), labels are inferred from `w_mat`.

Details

The function automatically detects the unique categories in the input matrix and creates binary indicator variables for each category. Categories are sorted and mapped to sequential indices.

The output array has dimensions:

- Dimension 1: Individuals
- Dimension 2: Time points
- Dimension 3: Categories (one-hot encoded)

Value

A 3D array (individual × time × category)

Examples

```
# Create sample categorical data
set.seed(123)
n_time <- 20
n_individuals <- 5
w_mat <- matrix(sample(c("A", "B", "C"), n_time * n_individuals, replace = TRUE),
                nrow = n_time, ncol = n_individuals)

# Convert to one-hot encoding
x_array <- get_x_from_w(w_mat)
dim(x_array)  # Should be [5, 20, 3]

# Check that each time point has exactly one category active
apply(x_array[1, , ], 1, sum)  # Should be all 1s
```

Index