Package ‘MFLDA’

May 19, 2025

Type Package

Title Multivariate Functional Linear Discriminant Analysis

Version 0.2.0

Author

Maintainer

Description The MFLDA package provides functions for performing Multivariate Functional Linear Discriminant Analysis. This method extends traditional linear discriminant analysis to functional data, enabling the classification of multivariate observations represented as curves. By incorporating sparsity, MFLDA identifies the most relevant features while efficiently handling high-dimensional data. The package is designed for users seeking to analyze complex functional and longitudinal data with multiple predictors to gain insights into underlying patterns and relationships.

License GPL (>=3)

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Suggests knitr,
rmarkdown

VignetteBuilder knitr

R topics documented:

bs.generator *Generate B-spline Basis Matrices*

Description

This function creates B-spline basis matrices for a given input vector or matrix, with options to customize the degree of the spline, the location of knots, and whether the knots are generated based on quantiles or uniform intervals.

Usage

```
bs.generator(x, N, q = 3, KnotsLocation = "quantile", knots = NULL)
```

Arguments

`x`	A numeric vector or matrix representing the input data for which B-spline bases are to be generated. If a matrix, each column is treated as a separate variable.
`N`	An integer specifying the number of internal knots to generate.
`q`	An integer representing the degree of the spline. Default is 3 (cubic spline).
`KnotsLocation`	A character string indicating the method for generating knots. Options are ‘"quantile"‘ for quantile-based knots or ‘"uniform"‘ for uniformly spaced knots. Default is ‘"quantile"‘.
`knots`	An optional vector of custom knots. If provided, this vector will be used directly, and ‘KnotsLocation‘ will be ignored.

Details

Knots Generation: If ‘knots‘ is not provided, knots are generated based on the value of ‘KnotsLocation‘. For quantile-based knots, they are placed at quantiles of the data in ‘x‘, while for uniform spacing, knots are spread evenly across the range of ‘x‘. Matrix Handling: If ‘x‘ is a matrix, each column is treated as an independent variable, and the spline basis is constructed separately for each. This reduces memory usage by processing in blocks.

Value

A list containing:

`Bx0`	The raw B-spline basis matrix.
`Bx`	The B-spline basis matrix with centered columns.
`BxMean`	The column means of ‘Bx0‘, used for centering.
`knots`	The knots used for generating the B-spline basis.

Examples

```
# Generate a cubic B-spline basis for a vector with quantile-based knots
x <- rnorm(100)
bs.generator(x, N = 5, q = 3, KnotsLocation = "quantile")

# Generate a cubic B-spline basis for a matrix with uniform knots
x_matrix <- matrix(rnorm(1000), ncol = 10)
bs.generator(x_matrix, N = 5, q = 3, KnotsLocation = "uniform")
```

cbs — *Generate Constant B-spline Basis*

Description

This function creates a constant spline basis matrix for a given vector of input values. It assigns binary indicators for which interval each value falls into based on specified knots.

Usage

```
cbs(x, knots, Boundary.knots = range(x))
```

Arguments

`x`	A numeric vector for which the constant spline basis is to be generated.
`knots`	A numeric vector of knots that define the intervals for the constant spline basis.
`Boundary.knots`	A numeric vector of length two defining the boundary knots. Default is the range of ‘x‘. These knots represent the outer boundaries for the spline basis.

Details

The function ensures that there is no overlap between the defined intervals by assigning values only to the corresponding interval where each input falls. The first column of the output matrix corresponds to values less than the first knot, and the last column corresponds to values greater than or equal to the last knot.

Value

A matrix of binary indicators with rows corresponding to the input vector ‘x‘ and columns corresponding to the defined knots. Each entry is ‘1‘ if the corresponding value in ‘x‘ falls within the interval defined by the knots and ‘0‘ otherwise.

cvMFLDA — *Cross-Validation for Multivariate Functional Linear Discriminant Analysis (MFLDA)*

Description

This function performs cross-validation for Multivariate Functional Linear Discriminant Analysis (MFLDA). It allows for optional parallel processing and can return various metrics, including accuracy or combined metrics.

Usage

```
cvMFLDA(
  Xdata = Xdata,
  Y = Y,
  withCov = FALSE,
  plotIt = FALSE,
  metrics.choice = "Accuracy",
  Xtestdata = NULL,
  Ytest = NULL,
  isParallel = TRUE,
  ncores = NULL,
  gridMethod = "RandomSearch",
  AssignClassMethod = "Joint",
  nfolds = 5,
  ngrid = 8,
  standardize = TRUE,
  maxiteration = 20,
  weight = 0.5,
  thresh = 0.001
)
```

Arguments

`Xdata`	A data frame containing the training data in long format, with the first three columns for ‘ID‘, ‘Time‘, and ‘Group‘, followed by variable columns.
`Y`	A vector representing the group labels for the training data.
`plotIt`	A logical value indicating whether to generate discriminant plots. Default is FALSE.
`metrics.choice`	A string specifying the metric to optimize. Options are "Accuracy" or "CombinedMetrics". Default is "Accuracy".
`Xtestdata`	A data frame containing the test data. If NULL, training data is used for testing.
`Ytest`	A vector representing the group labels for the test data. Required if Xtestdata is provided.
`isParallel`	A logical value indicating whether to perform parallel processing. Default is TRUE.
`ncores`	An integer specifying the number of cores to use for parallel processing. Default is NULL, which uses half the available cores.
`nfolds`	An integer specifying the number of folds for cross-validation. Default is 5.
`ngrid`	An integer specifying the number of tuning grid values. Default is 8.
`standardize`	A logical value indicating whether to standardize the data to have mean zero and variance one for each time point and each variable. Default is TRUE.
`maxiteration`	An integer specifying the maximum number of iterations for optimization. Default is 20.
`thresh`	A numeric value indicating the convergence threshold. Default is 1e-03.

Value

A list containing:

`CVOut`	A matrix of cross-validation results for different tuning parameter values.
`mfldaerror.test`	The estimated classification error for the test data.
`sidaerror.train`	The estimated classification error for the training data.
`hatalpha`	The estimated alpha coefficients from the MFLDA.
`PredictedClass`	The predicted class labels for the test data.
`var.selected`	The variables selected by the MFLDA.
`varname.selected`	The names of the selected variables.
`optTau`	The optimal tuning parameter values.
`gridValues`	The grid of tuning parameter values used.
`myDiscPlot`	A ggplot object of the discriminant plot if `plotIt` is TRUE; otherwise NULL.
`InputData`	The original input data used.

Examples

```
# Example usage of cvMFLDA
result <- cvMFLDA(Xdata, Y, plotIt=TRUE, metrics.choice="Accuracy",
                   Xtestdata=test_data, Ytest=test_labels,
                   nfolds=5, ngrid=8)

# Accessing results
print(result$mfldaerror.test)
print(result$myDiscPlot)
```

`DiscriminantPlots` *Generate Discriminant Plots for Test Data*

Description

This function generates various discriminant plots for test data based on projections from a linear discriminant analysis. It includes a discrimination plot, density plots of the projections, and performs Kolmogorov-Smirnov tests.

Usage

```
DiscriminantPlots(Xtestdata, Ytest, myalpha, predicted.class)
```

Arguments

`Xtestdata`	A data frame containing the test data in long format, with the following columns: 'ID' (first column), 'Time' (second column), 'Group' (third column), Variables (fourth to last columns)
`Ytest`	A vector representing the group labels for the test data.
`myalpha`	A matrix of coefficients obtained from the linear discriminant analysis, where each column corresponds to a time point.
`predicted.class`	A vector of predicted class labels for each observation in the test data.

Value

A list containing:

`Projection.df`	The data frame containing the projections for the test data.
`discriminant.plot`	A ggplot object representing the discrimination plot.
`density.loess`	A ggplot object representing the density plot of the loess fit for projections.
`density.plot`	A ggplot object representing the density plot of row means.
`ks.loess.p`	The p-value from the Kolmogorov-Smirnov test on the loess predictions (if applicable).
`ks.density.p`	The p-value from the Kolmogorov-Smirnov test on the density of row means (if applicable).

Examples

```
# Example usage of DiscriminantPlots
result <- DiscriminantPlots(test_data, test_labels, coefficients_matrix, predicted_classes)

# Plot the discriminant plot
print(result$discriminant.plot)
```

`mflda` *Multivariate Functional Linear Discriminant Analysis (MFLDA)*

Description

This function performs Multivariate Functional Linear Discriminant Analysis (MFLDA) for classification of multi-class data. It trains the model on provided training data, optionally standardizes the data, and evaluates performance on test data.

Usage

```
mflda(
  Xtrain = Xtrain,
  Y = Y,
  Tau = Tau,
  Xtestdata = Xtestdata,
  Ytest = Ytest,
  plotIt = FALSE,
  standardize = TRUE,
  maxiteration = 20,
  thresh = 0.001
)
```

Arguments

`Xtrain`	A data frame containing the training data with features and identifiers. The first three columns should be identifiers, and subsequent columns should be features.
`Y`	A vector of class labels for the training data, corresponding to the rows in 'Xtrain'.
`Tau`	Regularization parameter for sLDA; determines the sparsity of the resulting discriminant functions.
`Xtestdata`	A data frame containing the test data, structured similarly to 'Xtrain'. If NULL, training data is used for testing.
`Ytest`	A vector of class labels for the test data. Required if 'Xtestdata' is provided.
`plotIt`	A logical indicating whether to plot the discriminants and densities. Default is FALSE.
`standardize`	A logical indicating whether to standardize the training and test data standardized to have mean zero and variance one for each time point and each variable. Default is TRUE.
`maxiteration`	Maximum number of iterations for convergence in optimization. Default is 20.
`thresh`	Convergence threshold for the optimization algorithm. Default is 1e-03.

Details

This function implements the MFLDA algorithm by performing the following steps: 1. Checks and preprocesses the data. 2. Optionally standardizes the data. 3. Performs the iterative optimization to obtain the discriminant functions. 4. Classifies the test data based on the learned discriminant functions. 5. Computes various performance metrics and generates plots if requested.

Value

A list containing:

`AverageError`	Average error of the classification.
`Metrics`	A vector of classification performance metrics (accuracy, balanced accuracy, F1 score, precision, recall, and Matthews correlation coefficient).
`TimeSpecificError`	Error for the test data classified based on the provided time points.
`hatalpha`	Estimated coefficients from the sLDA model.
`PredictedClass`	Predicted class labels for the test data.
`myDiscPlot`	Plot object of discriminants and densities, if 'plotIt' is TRUE.
`Projection.df`	Data frame of projected values for visualization.
`var.selected`	Indices of selected variables based on sparsity criteria.
`varname.selected`	Names of selected variables based on sparsity criteria.

Examples

```
# Example usage of mflda function:
result <- mflda(Xtrain = train_data, Y = train_labels, Tau = 0.5,
                Xtestdata = test_data, Ytest = test_labels,
                plotIt = TRUE, standardize = TRUE)
```

```
# Access predicted classes and performance metrics
predicted_classes <- result$PredictedClass
performance_metrics <- result$Metrics
```

---

mfldaclassify *Classify Test Data Using Multivariate Linear Discriminant Analysis*

Description

This function classifies test data based on learned sparse functional linear discriminant analysis parameters (features discriminant scores). It projects both training and test data onto the discriminant space and predicts the class labels for the test samples.

Usage

```
mfldaclassify(
  hatalpha = hatalpha,
  Xtestdata = Xtestdata,
  Xdata = Xdata,
  Y = trainY,
  standardize = TRUE
)
```

Arguments

`hatalpha`	A matrix or list of matrices containing the estimated coefficients for the linear discriminants obtained from the training data.
`Xtestdata`	A data frame containing the test data, structured similarly to the training data, with identifiers in the first three columns and features in the subsequent columns.
`Xdata`	A data frame containing the training data, with the same structure as ‘Xtestdata‘.
`Y`	A vector of class labels for the training data. It should match the length of the number of unique samples in ‘Xdata‘.
`standardize`	A logical value indicating whether to standardize the data to have mean zero and variance one for each time point and each variable before classification. Default is TRUE.

Details

The function performs the following steps: 1. Standardizes the training and test data if the ‘standardize‘ parameter is set to TRUE. 2. Projects both the training and test data into the discriminant space using the estimated coefficients. 3. Computes the Euclidean distances between projected test data and the class means in the discriminant space. 4. Assigns class labels to the test samples based on the closest mean in the discriminant space.

Value

A list containing:

`PredictedClass`	A vector of predicted class labels for the test data.
`Predclass.df`	A data frame of predicted class labels for each test sample across different dimensions of the discriminant space.

Examples

```
# Assuming hatalpha contains the discriminant scores, Xtestdata is your test set,
# Xdata is your training set, and Y contains the corresponding training labels.
classification_result <- mfldaclassify(hatalpha = hatalpha, Xtestdata = Xtestdata, Xdata = Xdata, Y = trainY, s

# Access predicted class labels
predicted_classes <- classification_result$PredictedClass

# Access detailed prediction data frame
predictions_df <- classification_result$Predclass.df
```

`mfldatunerange` *Generate Tuning Range for Multivariate Discriminant Analysis*

Description

This function computes the tuning ranges for the sparsity parameter in multivariate functional linear discriminant analysis (MFLDA) based on provided training data. It standardizes the data if specified and uses a grid search approach to determine optimal parameter values.

Usage

```
mfldatunerange(Xtrain = Xtrain, Y = Y, ngrid = 8, standardize = TRUE)
```

Arguments

`Xtrain`	A data frame containing the training data. The first three columns should contain identifiers (such as ‘id‘, ‘time‘, and ‘group‘), and subsequent columns should contain the features used for classification.
`Y`	A vector of class labels corresponding to the training data. It should have the same length as the number of rows in ‘Xtrain‘.
`ngrid`	An integer specifying the number of grid points to generate for the tuning range. Default is 8.
`standardize`	A logical value indicating whether to standardize the data to have mean zero and variance one for each time point and each variable. Default is TRUE.

Details

The function performs the following steps: 1. It prepares the input data, optionally standardizing it based on the ‘standardize‘ parameter. 2. It uses a custom function ‘myfastLDAnonsparse‘ to perform the preliminary analysis. 3. It generates a grid of tuning parameters by iteratively refining based on selectivity and sparsity conditions. 4. The final output contains a range of optimal tuning parameters for each class.

Value

A list containing:

`Tauvec`	A list of sequences of tuning parameters for each class dimension based on the grid search results.

Examples

```
# Assuming Xtrain is your training data frame and Y contains the corresponding labels
result <- mfldatunerange(Xtrain = Xtrain, Y = Y, ngrid = 10, standardize = TRUE)

# Access the tuning parameters
tuning_params <- result$Tauvec
```

`simulation` *Simulate Data for Classification Problems*

Description

This function simulates datasets for classification tasks based on the specified number of classes and case number. It sources external simulation scripts to generate training and testing datasets.

Usage

```
simulation(n.class, case.num)
```

Arguments

`n.class`	An integer indicating the number of classes for the simulation. Acceptable values are 2 or 3.
`case.num`	An integer specifying the case number of the simulation to execute. This should correspond to the specific simulation scenarios available for the selected number of classes.

Details

The function will source R scripts from predetermined paths based on the number of classes (2 or 3). For each class scenario, the function can simulate up to four different cases, each generating distinct datasets. It is important that the simulation scripts exist in the specified directories, and they should return a list containing ‘train.df‘ and ‘test.df‘. There are four different cases for simulation: (1) the groups are distinct across all time points, exhibiting similar average trends; (2) the groups diverge only within a specific time window, from the time point [5, 15], displaying different patterns; (3) the groups show separation over a fixed interval of 10 time points, though the exact time frame varies; (4) the groups are distinct during a random occurring period, with the length of the separation varying between 5 and 40 time points.

Value

A list containing:

`train`	A data frame of the training dataset generated from the specified simulation case.
`test`	A data frame of the testing dataset generated from the specified simulation case.

Examples

```
# Simulate data for a 2-class problem, case 1
result <- simulation(n.class = 2, case.num = 1)

# Access training and testing datasets
train_data <- result$train
test_data <- result$test
```

`spline.prediction` *Spline-Based Estimation from Dataset*

Description

This function takes a dataset of participant observations and uses spline basis functions to generate fitted values for each variable across unique time points. It constructs a linear model for each variable and predicts values using B-splines.

Usage

```
spline.prediction(olddat)
```

Arguments

`olddat`	A data frame containing the dataset with participant observations. It must include columns for ‘id‘, ‘group‘, ‘time‘, and additional variables to be predicted.

Details

The function internally identifies unique time points from the dataset and generates B-spline basis matrices for these time points. It uses linear models to predict values for each variable, storing the fitted values in a list. The function also assigns group classifications based on participant IDs if the original data has irregular time points.

Value

A list containing:

`pred.df`	A data frame with the predicted values for each variable across time points, along with participant IDs and group classifications.

Examples

```
# Assume `my_data` is a data frame with the necessary structure
prediction_results <- spline.prediction(my_data)

# View the predicted values
head(prediction_results$pred.df)
```

Index