MFLDA: Multivariate Functional Linear Discriminant Analysis

Limeng Liu, Guannan Wang, and Sandra E. Safo

Description

MFLDA is an R package designed to implement Multivariate Functional Linear Discriminant Analysis (MFLDA) for the classification of multivariate longitudinal data. This model is particularly useful in applications where the goal is to distinguish between two or three different classes from multivariate functional data while simultaneously identifying the most discriminating features.

Key Features:

- **Multivariate Functional Data:** Handles longitudinal data with multiple variables, allowing for a more comprehensive analysis of complex data structures.
- **Sparse Feature Selection:** Performs automatic feature selection, identifying the most important time-varying features that contribute to the discrimination between classes to enhance interpretability.
- **Flexible Time Representation:** Accommodates both discrete and continuous time domains, as well as irregularly sampled time points. This flexibility makes the method applicable to a wide range of real-world longitudinal data where time observations may not be equally spaced.
- **Two- or Three-Class Classification:** Supports classification tasks where the response variable consists of two or three distinct classes, making it suitable for binary and multi-class problems.

By integrating functional data analysis techniques with sparse learning methods, MFLDA offers a powerful tool for classifying multivariate longitudinal data, particularly when the data are high-dimensional and only a few features are informative for class separation. This package can be applied to diverse fields where temporal data with varying sampling frequencies and irregular time points are common.

Setup

Import the package

```
library(MFLDA)
```

Input data set should be in the form where row represents observations and column represents features. Three columns are needed as the first 3 columns: `id`, `time`, `group`, respectively represents participant unique id, time of visit, and which group the participant belongs to. Each participant can have different numbers of time points, where the time points can be irregular.

A sample dataset shown below:

```
df <- data.frame(
  id = c(1,1,1,2,2,2,2,3,3)
  time = c(1,2,3,1,2,3,4,1,2),
  group = c(1,1,1,2,2,2,2,1,1)
)
```

id	time	group
1	1	1
1	2	1
1	3	1
2	1	2
2	2	2
2	3	2
2	4	2
3	1	1
3	2	1

Apply spline smoothing estimation method to better resolve irregular time points issues. The function `spline.prediction` will take a dataframe and output a dataframe.

```
spline.prediction(dat)
```

The output dataset should have same/regular time points for each participant:

id	time	group
1	1	1
1	2	1
1	3	1
1	4	1
2	1	2
2	2	2
2	3	2
2	4	2
3	1	1
3	2	1
3	3	1
3	4	1

mfldatunerange

The function `mfldatunerange()` is part of the **MFLDA** (Multivariate Functional Linear Discriminant Analysis) framework. It is designed to help tune the range of regularization parameters (τ values) for sparse feature selection in functional linear discriminant analysis.

Usage:

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

Argument:

- `Xtrain`: The training data, where rows represent observations and columns represent features at different time points. The first three columns are `id`,`time`,`group`, while subsequent columns are feature values.
- `Y`: A vector of class labels corresponding to the observations in Xtrain.
- `ngrid`: The number of grid points to use in the τ grid search (default is 8).
- `standardize`: Logical; flag to indicate whether the data
- should be standardized to have mean zero and variance one for each time point and each feature (default is TRUE).

Value:

- `Tauvec`: A list of optimal τ values. These τ values are used to control the level of sparsity in the MFLDA model.

Example:

```
result <- mfldatunerange(Xtrain = train_data, Y = class_labels, ngrid = 8, standardize
tau_values <- result$Tauvec
```

MFLDA

The function `mflda()` performs Multivariate Functional Linear Discriminant Analysis (MFLDA) on training data and evaluates on test data if provided.

Usage:

```
mflda(Xtrain, Y, Tau, Xtestdata = NULL, Ytest = NULL, plotIt = FALSE,
standardize = TRUE, maxiteration = 20, thresh = 1e-03)
```

Arguments

- `Xtrain`: The training data, where rows represent observations and columns represent features at different time points. The first three columns are `id`,`time`,`group`, while subsequent columns are feature values.
- `Y`: A vector of class labels (1,2,...) for the training data.
- `Tau`: A regularization parameter for controlling sparsity in MFLDA.
- `Xtestdata`: (Optional) A matrix of predictor variables for test data. Rows represent samples and columns represent variables. Number of columns should be the same as training data. Defaults to Xtrain if not specified.
- `Ytest`: (Optional) A vector of class labels for the test data. Defaults to Y if not specified.
- `plotIt`: Logical; if TRUE, generates discriminant and density plots. Defaults to FALSE.
- `standardize`: Logical; if TRUE, standardizes the data to have mean zero and variance one. Defaults to TRUE.

- `maxiteration`: Maximum number of iterations for model convergence. Defaults to 20.
- `thresh`: Convergence threshold for the relative difference in estimates. Defaults to 1e-03.

Value

- `AverageError`: The mean classification error rate across time points.
- `Metrics`: A summary of classification performance, including accuracy, balanced accuracy, F1 score, precision, recall, and MCC.
- `TimeSpecificError`: Classification error calculated separately for each time point.
- `hatalpha`: The estimated sparse MFLDA discriminant weights.
- `PredictedClass`: The predicted class labels for Ytest.
- `myDiscPlot`: The discriminant plots, if plotIt = TRUE.
- `Projection.df`: A data frame containing projection scores.
- `var.selected`: Logical vector indicating selected variables based on sparsity.
- `varname.selected`: Names of the variables selected by the sparsity threshold.

Examples

```
# Run MFLDA on training and test data
result <- mflda(Xtrain = train_data, Y = train_labels, Tau = 50,
Xtestdata = test_data, Ytest = test_labels)
print(result$Metrics)
```

CVMFLDA

The function `cvmflda())` conducts cross-validation for MFLDA across a specified parameter grid to identify optimal regularization values for classification. The function can run in parallel for efficiency.

Usage

```
cvMFLDA(Xdata, Y, plotIt = FALSE, metrics.choice = "Accuracy",
        Xtestdata = NULL, Ytest = NULL, isParallel = TRUE, ncores = NULL,
        nfolds = 5, ngrid = 8, standardize = TRUE, maxiteration = 20, thresh = 1e-03)
```

Arguments

- `Xdata`: The training data, where rows represent observations and columns represent features at different time points. The first three columns are `id`,`time`,`group`, while subsequent columns are feature values.
- `Y`: A vector of class labels for the training data.
- `plotIt`: Logical; if TRUE, generates discriminant and density plots after model convergence. Defaults to FALSE.
- `metrics.choice`: Specifies the metric to optimize during cross-validation. Options are "Accuracy" (default) or "CombinedMetrics".
- `Xtestdata`: (Optional) Data frame of predictor variables for test data. Defaults to Xdata if not specified.
- `Ytest`: (Optional) Vector of class labels for the test data. Defaults to Y if not specified.
- `isParallel`: Logical; if TRUE, enables parallel processing for cross-validation. Defaults to TRUE.
- `ncores`: Number of cores for parallel processing. Defaults to half the system's available cores.
- `nfolds`: Number of folds for cross-validation. Defaults to 5.
- `ngrid`: Number of regularization values (grid points) to consider during cross-validation. Defaults to 8.

- `standardize`: Logical; if TRUE, standardizes the data. Defaults to TRUE.
- `maxiteration`: Maximum number of iterations for the MFLDA model convergence. Defaults to 20.
- `thresh`: Convergence threshold for the iterative model updates. Defaults to 1e-03.

Value

- `CVOut`: A matrix with cross-validation results for each fold and grid value.
- `mfldaerror.test`: Estimated test classification error using the optimal Tau.
- `sidaerror.train`: Estimated training classification error using the optimal Tau.
- `hatalpha`: The estimated discriminant weights from MFLDA.
- `PredictedClass`: Predicted class labels for the test set.
- `var.selected`: Logical vector indicating selected variables based on the optimal regularization parameter.
- `varname.selected`: Names of the selected variables.
- `optTau`: The optimal regularization parameter from the grid search.
- `gridValues`: All Tau values explored during cross-validation.
- `myDiscPlot`: The discriminant plots, if plotIt = TRUE.
- `InputData`: Original training data (Xdata).

Examples

```
# Run cross-validated mflda on training data
cv_result <- cvMFLDA(Xdata = training_data, Y = training_labels, nfolds = 5,
ngrid = 10, isParallel = TRUE)
print(cv_result$optTau) # Print optimal Tau value
```

DiscriminantPlot

The function `discriminantplot()` generates discriminant plots and density plots for visualizing class separation in projected test data over time.

Usage

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

Arguments

- `Xtestdata`: The testing data, where rows represent observations and columns represent features at different time points. The first three columns are `id`,`time`,`group`, while subsequent columns are feature values.
- `Ytest`: Vector of actual class labels for the test data.
- `myalpha`: Matrix of discriminant vectors, where each column represents a time point.
- `predicted.class`: Vector of predicted class labels for the test data.

Value

- `Projection.df`: Data frame containing projected values for each sample at each time point.
- `discriminant.plot`: ggplot object showing class projections over time with a loess-smoothed mean line.
- `density.loess`: ggplot density plot based on loess-smoothed projections for each class.

- `density.plot`: ggplot density plot of row means of projections, grouped by predicted class.
- `ks.loess.p`: p-value from the KS test for loess-smoothed distribution comparison.
- `ks.density.p`: p-value from the KS test for row mean distribution comparison by predicted class.

Examples

```
result <- DiscriminantPlots(Xtestdata = test_data, Ytest = y_test,
myalpha = alpha_matrix, predicted.class = pred_class)
print(result$discriminant.plot)
print(result$density.plot)
```

Data Simulation Demo -1: Binary Outcome

(the following code can be found in `main.R`)

Simulate the binary class data using the `simulation()` function

```
data <- simulation(n.class = 2, case.num = 1) ## choose case.num from 1-4
View(data$train)
View(data$test)
```

There are four different cases for binary simulation:

- 1. the two groups are distinct across all time points, exhibiting similar average trends;
- 2. the two groups diverge only within a specific time window, from the time point [5, 15], displaying different patterns;
- 3. the two groups show separation over a fixed interval of 10 time points, though the exact time frame varies;
- 4. the two groups are distinct during a random occurring period, with the length of the separation varying between 5 and 40 time points.

Next use b-spline estimation to smooth the curve

```
train.pred <- spline.prediction(data$train)
View(newdat$pred.df)
test.pred <- spline.prediction(data$test)
View(newdat$pred.df)
```

Thus, `train.pred` and `test.pred` is ready for use in MFLDA.

Data Simulation Demo -2: Multi-Classes Outcome

(the following code can be found in `main.R`)

Similarly we can simulate some multi-classes data:

```
data <- simulation(n.class = 3, case.num = 1) ## choose case.num from 1-4
View(data$train)
View(data$test)

## b-spline estimation
train.pred <- spline.prediction(data$train)
View(newdat$pred.df)
test.pred <- spline.prediction(data$test)
View(newdat$pred.df)
```

After applying spline estimation smoothing, `train.pred` and `test.pred` is ready for use in MFLDA.

MFLDA Demo - 3-Class Problem

(the following code can be found in `main.R`)

We can perform MFLDA with knwon regularization parameter τ, for example, let $\tau = 50$. By default, we will run MFLDA-D model.

We have provided some simulated data in the folder `data` after applying spline estimation. First, we need to gather some information from the data.

```
################### MFLDA-Multi-Class Problems ###################
## Read data
XTrain <- get(load("data/simTrain_spline_c3_case4.rda"))
XTest <- get(load("data/simTest_spline_c3_case4.rda"))

trainX <- XTrain
trainX$group <- as.integer(trainX$group)
trainY <- trainX$group[trainX$time == 1]
trainY <- as.integer(trainY)

testX <- XTest
testY <- testX$group[testX$time == 1]
testX$group <- as.integer(testX$group)
testY <- as.integer(testY)
```

With known parameter, we can apply `mflda()`:

```
## MFLDA with known hyper-parameter
myTau = 50 ## can choose your own
mflda.result <- mflda(Xtrain=trainX,Y=trainY,Tau=myTau,
Xtestdata=testX,Ytest=testY,
plotIt=FALSE,standardize=TRUE,maxiteration=20,thresh= 1e-03)
```

You can also check discriminant and density plots using `discriminantplot()` for the first discriminant vector:

```
myplots <- DiscriminantPlots(Xtestdata=testX,Ytest=testY,
myalpha=mflda.result$hatalpha[[1]], predicted.class = mflda.result$PredictedClass)
myplot$discriminant.plot
myplot$density.loess
myplot$density.plot
```

For unknown parameter, we can use `cvmflda()` to choose the optimal one:

```
cvmod <- cvMFLDA(Xdata=trainX,Y=trainY, metrics.choice="CombinedMetrics") ## Accuracy
cvmod$optTau
```

Then we can get some addtional information, such as predicted classes, first and second discriminant score, and selected variables.

```
## find the predicted classes
mflda.result$PredictedClass

## find the first discriminant scores
mflda.result$hatalpha[[1]]

## find the second discriminant scores
mflda.result$hatalpha[[2]]

## find the selected variables
mflda.result$varname.selected
```

Similarly, we can plot the discriminant plots for the first discriminant vector

```
## plot the discriminant plots for the first discriminant vector
myplot <- DiscriminantPlots(Xtestdata=testX,Ytest=testY,
myalpha=mflda.result$hatalpha[[1]], predicted.class = mflda.result$PredictedClass)
myplot$discriminant.plot
myplot$density.loess
myplot$density.plot
```

SFLDA Demo - 2-Class Problem

(the following code can be found in `main.R`)

SFLDA is also applicable for 2-classes problem, we have provided some simulated data in the folder `simulations` after applying spline estimation.

```
## Read data
XTrain <- get(load("data/simTrain_spline_c2_case4.rda"))
XTest <- get(load("data/simTest_spline_c2_case4.rda"))

## rest codes will be the similar to 3-class problem
```

Real-data Application: Inflammatory Bowel Disease

The Metagenomics(MGX) dataset and metadata can be download from iHMP IBD dataset website: https://ibdmdb.org/.

Preprocess the Metagenomics(MGX) dataset (code can be found in `ibd/ibd_preprocess.R`):

```
metag <- ibd_preprocess(metagenomics_df)
```

Merging the metadata (`Participant.ID`, `visit_num`, `diagnosis`, `Age.at.diagnosis`, and `site_name`) and the preprocessed metagenomics dataset to a dataframe.

Use Linear Mixed Effect Model to seelct the top 200 microbial pathways (code can be found in `R/ibd/ibd_lmms.R`).

We prepared a IBD_data demo dataset file (30% of original data) which contains in `data/ibd_sample.rda` after LMMS, and a corresponding spline prediction sample file in `data/ibd_sample_prediction.rda`.