R code for Variable Screening and Spatial Smoothing in Fréchet Regression with Application to Diffusion Tensor Imaging

Lei Yan

2023/10/19

Introduction

Within the `utility_functions` folder, you will find essential functions for implementing two crucial components of the proposed distance covariance (DC) variable screening method and spatial smoothing in Fréchet regression. These functions are instrumental in the analysis and modeling of complex data.

Let's explore the key components and files associated with these methods:

1. **Distance Covariance (DC) Variable Screening**
   - The core of the DC variable screening method can be found in the R file named `dcov.R`. This file contains the necessary code to compute the distance covariance, a fundamental measure for identifying and selecting important variables within your data.
   - To enhance computational efficiency, a C++ implementation of the core DC computation function is available in the `dcov_cpp.cpp` file. This C++ implementation is designed to accelerate the DC calculation process, making it faster and more robust.
2. **Spatial Smoothing in Fréchet Regression**
   - If your data involves spatially correlated variables, the `spatial_smooth.R` file provides functions to perform spatial smoothing. This smoothing technique is particularly useful when dealing with spatially correlated symmetric positive definite (SPD) matrices.
   - To further optimize the process, you can take advantage of the C++ implementation available in the `spatial_smooth_cpp.cpp` file. This implementation leverages C++'s performance benefits to efficiently compute weighted Fréchet means for spatial smoothing.

In addition to the primary utility functions, you'll find code in the `pred_error` and `var_sele` folders. These folders contain code related to the simulations described in Sections 3.1 and 3.2 of the accompanying paper. These simulations provide practical insights and demonstrate the application of the distance covariance screening method and spatial smoothing in real-world scenarios.

This Vignette guides you through the usage of these functions, offering practical examples and insights to empower your data analysis and modeling endeavors. Let's dive into the details and learn how to harness these tools effectively.

Usages

Distance Covariance Variable Screening

In this section, we will explore how to effectively use the distance covariance screening method. This method plays a pivotal role in the context of Fréchet regression, especially when dealing with responses represented as symmetric positive definite (SPD) matrices. For our illustrative purposes, we'll focus on the Cholesky distance between two SPD matrices. This particular example aligns with Example 1a in Section 3.2 of the associated paper.

To get started, we need to ensure that the necessary packages and source files are loaded:

```
library(here)
source(here("utility_functions", "dcov.R"))
source(here("utility_functions", "utility.R"))
Rcpp::sourceCpp(here("utility_functions", "dcov_rcpp.cpp"))
```

The key function we'll be using is compute_dcov(n, p, type_data, seed, d_y, r = 0.5). Let's break down its essential arguments:

- n: An integer representing the number of samples.
- p: An integer indicating the number of predictors.
- type_data: A string specifying the type of response data.
- seed: An integer used as a seed for random data generation.
- d_y: A function that defines the distance metric in response space.
- r: A real number ranging from 0 to 1, denoting the correlation between simulated predictors.

And the return value is a list containing the DC and generated data X, Y.

Now, let's put this method into practice and illustrate the importance of selected variables. In this demonstration, the important variables are $X_1, X_5, X_9, X_{13}, X_{17}, X_{20}$ and we let $n = 200, p = 20$.

```
n <- 200
p <- 20
seed <- 1
type_data <- "SPD"
d <- 6 # model size
d_y <- cholesky_error_C # cpp version of Cholesky distance, included in dcov_rcpp.cpp file

dcov_result <- compute_dcov(n, p, type_data, seed, d_y, r = 0.5)
dcov_SPD <- dcov_result$dcov_SPD
sort(dcov_SPD, decreasing = TRUE, index.return = TRUE)$ix[1:d]
```

Alternatively, you can separate data generation and distance covariance estimation using the `compute_dcov2(X, Y, type_data, d_y)` function. In this case, you'll first generate the data using `gendata_SPD(n, p, m, r, seed)` and then apply `compute_dcov2()` to the generated data. The arguments of `compute_dcov2()` are

- X: A n by p matrix containing the predictors
- Y: A matrix or 3D array depending on the data type of response
- d_y: A function, the distance function defined in response space

The arguments of `gendata_SPD()` are

- n: An integer, number of samples
- p: An integer, number of predictors
- m: An integer, dimension of SPD matrix outputs
- r: A real number between 0 and 1, correlation between simulated predictors
- seed: An integer, seed set for the random generation

First, generate data.

```
n <- 200
p <- 20
m <- 3
r <- 0.5
seed <- 1
dataXY <- gendata_SPD(n, p, m, r, seed)
```

Then we apply `compute_dcov2()` on the data.

```
type_data <- "SPD"
d <- 6 # model size
d_y <- cholesky_error_C # cpp version of Cholesky distance, included in dcov_rcpp.cpp file
dcov_SPD <- compute_dcov2(dataXY$x, dataXY$Min, type_data, d_y)
sort(dcov_SPD, decreasing = TRUE, index.return = TRUE)$ix[1:d]
```

Spatial Smoothing

In this section, we delve into the practical application of spatial smoothing techniques when dealing with responses represented as spatially correlated symmetric positive definite (SPD) matrices. Specifically, we'll explore how to use spatial smoothing within the context of Fréchet regression, focusing on simulating diffusion tensor imaging datasets using the Wishart process model[1].

We start by generating diffusion tensors on a 20×20 grid , with a spacing of 1 between adjacent points. Each grid point represents a voxel, and we can assign coordinates to these voxels. For any given voxel v, we denote its coordinates as $\mathbf{j}(v) = (j_1(v), j_2(v))$. To define the neighborhood of v, we consider voxels v' for which the Euclidean distance $\|\mathbf{j}(v) - \mathbf{j}(v')\|_2$ is less than or equal to 1.

Key Parameters:

- Sample Size (n): We set the sample size to 200.
- Predictors (X): We simulate predictor variables X in $\mathbb{R}^p$ using the method described in the SPD continuous example found in Section 3.2. The correlation parameter (ρ) is set to 0.5.

For the modeling of the diagonal and off-diagonal elements of $L_i(v)$, we use variables X_1, X_5, X_{10} and X_{15}, X_{20}, X_{25}.

Additional Parameters:

- Degree of Freedom (m): We set the degree of freedom (m) to 3.
- Spatial Correlation (ϕ): The spatial correlation parameter (ϕ) is set to 15.

Let's proceed with the practical application of spatial smoothing techniques and understand how they can enhance your data analysis in the context of Fréchet regression.

We start from loading the required packages and source files.

```
library(here)
library(geoR)
library(mgcv)
library(igraph)
source(here("utility_functions", "spatial_smooth.R"))
source(here("utility_functions", "utility.R"))
source(here("utility_functions", "FRiSO", "GloCovReg.R"))
source(here("utility_functions", "FRiSO", "GFRCovCholesky.R"))
Rcpp::sourceCpp(here("utility_functions", "spatial_smooth_cpp.cpp"))
```

To generate diffusion tensor data, we use function `gen_SWP(x, correlation, grid_size, m, seed_)`. The arguments are

- x: A n*p design matrix, n is the number of subject, p is the number of variables
- correlation: Correlation parameter that control the strength of spatial correlation
- grid_size: A grid_size*grid_size grid is generated
- m: Degree of freedom of the Wishart distribution of U
- seed_: Seed for reproducing results

To generate data, we first generate a design matrix using function `gendata_SPD()` and then call the `gen_SWP()` function.

[1]Learn more about the Wishart process model

```
n <- 50
p <- 30
m <- 3
r <- 0.5
seed <- 1
dataXY <- gendata_SPD(n, p, m, r, seed)
x <- dataXY$x

correlation <- 15
grid_size <- 20
Y <- gen_SWP(x, correlation, grid_size, m, seed) # dimensions m*m*n*n_voxel
```

Then for each voxel, we fit global Fréchet regression using function `GloCovReg()`.

```
n_voxel <- dim(Y)[4]
Y_hat <- array(dim = c(m, m, n, n_voxel))
for (i in seq_len(n)) {
  X_i <- x
  Y_i <- Y[,,, i]
  pred_DC_fr <- GloCovReg(as.matrix(X_i), M = Y_i,
                          xout = as.matrix(X_i),
                          optns = list(metric="cholesky"))$Mout
  Y_hat[,, i, ] <- convert_list23D(pred_DC_fr)
}
```

Now we are ready to apply spatial smoothing using function `compute_smooth_Fr(pred_Fr, weight, seed_)`. The arguments are

- pred_Fr: A m-by-m-by-n-by-n_voxel array, contains predicted SPD
- weight: A real number between 0 and 1, weight of neighborhood voxels
- seed_: Seed for reproducing results

```
weight <- 0.5
seed_ <- 1
Y_hat_smooth <- compute_smooth_Fr(Y_hat, weight = weight, seed_ = seed_)
```

Two-Stage Method

Combing DC variable screening and spatial smoothing, we have a two-stage method that first select important variables and then smooth the predictions. We continue to use the simulated diffusion tension data Y generated using function `gen_SWP()`. We set important variables to be X_1, X_5, X_{10}, X_{15}, X_{20}, X_{25}.

```
### stage one: select important variables
n_voxel <- dim(Y)[4]
type_data <- "SPD"
Y_hat <- array(dim = c(m, m, n, n_voxel))
for (i in seq_len(n)) {
  X_i <- x
  Y_i <- Y[,,, i]
  d <- 10 # keep the first d variables
  dcov <- compute_dcov2(X_i, Y_i, type_data, cholesky_error_C)
  dc_selected <- sort(dcov, decreasing = TRUE, index.return = TRUE)$ix[1:d]
  dc_selected <- dc_selected[order(dc_selected)]
  # fit global Fréchet regression using selected variables
  pred_DC_fr <- GloCovReg(as.matrix(X_i[, dc_selected]), M = Y_i,
                          xout = as.matrix(X_i[, dc_selected]),
```

```r
                          optns = list(metric="cholesky"))$Mout
  Y_hat[,, i, ] <- convert_list23D(pred_DC_fr)
}

### stage two: smoothing the results
weight <- 0.5
seed_ <- 1
Y_hat_smooth <- compute_smooth_Fr(Y_hat, weight = weight, seed_ = seed_)
```