Using the sprintr package

Guo Yu

2019-08-06

The `sprintr` package contains the implementations of a computationally efficient method, called sprinter, to fit large interaction models based on the reluctant interaction selection principle. The details of the method can be found in Yu, Bien, and Tibshirani (2019) *Reluctant interaction modeling*. In particular, `sprinter` is a multi-stage method that fits the following pairwise interaction model:

$$y = \sum_{j=1}^{p} X_j \beta_j^* + \sum_{\ell \leq k} X_\ell X_k \gamma_{\ell k}^* + \varepsilon.$$

This document serves as an introduction of using the package with a simple simulated data example.

Data simulation

We consider the following simple simulation setting, where $X \sim N(\mathbf{0}, \mathbf{I}_p)$. There are two non-trivial main effects $\beta_1 = 1$, $\beta_2 = -2$, and $\beta_j = 0$ for $j > 2$. The two important interactions are $X_1 * X_3$ with $\gamma_{13} = 3$, and $X_4 * X_5$ with $\gamma_{45} = -4$. With $\varepsilon \sim N(0, 1)$, the following code simulates $n = 100$ observation from the model above with $p = 200$.

```
library(sprintr)
set.seed(123)
n <- 100
p <- 200
x <- matrix(data = rnorm(n * p), nrow = n, ncol = p)
y <- x[, 1] - 2 * x[, 2] + 3 * x[, 1] * x[, 3] - 4 * x[, 4] * x[, 5] + rnorm(100)
```

Using `sprinter` function

The function `sprinter` implements the sprinter method (please note that the function name `sprinter` is different from the package name `sprintr`), which involves the following three main steps:

- Fit a lasso (with cross-validation) of the response y only on main effects X (if `square = FALSE`) or with both main effects and squared effects (X, X^2) (if `square = TRUE`).
- Carry out a screening procedure based on the residual from the previous step. The number of the selected candidate interactions is specified by `num_keep` .
- With a path of tuning parameter `lambda`, fit a lasso of the response on main effects, squared effects (if `square = TRUE`), and selected interactions from the previous step.

There are two tuning parameters: `num_keep` (used in Step 2) and `lambda` (used in Step 3). If `num_keep` is not specified, it will then be set to $n/\lceil \log n \rceil$ (see, e.g., Fan & Lv (2008)). If `lambda` is not specified, then `sprinter` would compute its own path of tuning parameter values based on the number of tuning parameters (`nlam`) and the range of the path (`lam_min_ratio`).

```
mod <- sprinter(x = x, y = y, square = FALSE, nlam = 100, lam_min_ratio = 0.01)
```

The output of `sprinter` is a `S3` object including several useful components. In particular, it involves a matrix `idx` that represents the index pairs of all variables considered in Step 3:

```
mod$idx[(p + 1) : nrow(mod$idx), ]
#>       index_1 index_2
#>  [1,]      96     140
#>  [2,]       5      79
#>  [3,]       7     113
#>  [4,]     113     155
#>  [5,]      94     148
#>  [6,]       5     173
#>  [7,]     108     144
#>  [8,]       7     165
#>  [9,]      17      77
#> [10,]     158     175
#> [11,]       4     102
#> [12,]      58     108
#> [13,]       5      97
#> [14,]      30     168
#> [15,]       5     182
#> [16,]       5      25
#> [17,]     135     168
#> [18,]     115     175
#> [19,]       3     177
#> [20,]       4      96
#> [21,]       1       3
#> [22,]       4       5
```

Since Step 3 of `sprinter` always includes the main effects, `mod$idx[(p + 1): nrow(mod$idx), ]` contains the indices of all the selected interactions from Step 2. The two columns of this output represents the index pair (ℓ, k) of a selected interaction $X_\ell * X_k$, where $\ell \leq k$. Note that here the last two rows are the true interactions $X_1 * X_3$ and $X_4 * X_5$. If the first entry of an index pair is zero, i.e., $(\ell = 0, k)$, then it represents a main effect X_k.

The output `mod$coef` is a `nrow(mod$idx)`-by-`length(mod$lambda)` matrix. Each column of `mod$coef` is a vector of estimate of all variable coefficients considered in Step 3 corresponding to one value of the lasso tuning parameter `lambda`. For example, for the 30-th tuning parameter, we have the corresponding coefficient estiamte:

```
estimate <- mod$coef[, 30]
cb <- cbind(mod$idx, estimate)
cb[cb[, 3] != 0, ]
#>      index_1 index_2   estimate
#> V1         0       1  0.3062352
#> V2         0       2 -0.4982855
#> V221       1       3  1.9034944
#> V222       4       5 -2.9489912
```

Using cross-validation with `cv.sprinter`

The function `cv.sprinter()` performs cross-validation to select the value of lasso tuning parameter `lambda` used in Step 3, while holding the value of `num_keep` fixed.

```
mod_cv <- cv.sprinter(x = x, y = y, square = FALSE, nlam = 100, lam_min_ratio = 0.01)
```

The output of `cv.sprinter` is a `S3` object. The most intersting information is `mod_cv$compact`, which is a matrix of three columns. The first two columns show the index pairs of all variables finally selected by the

lasso in Step 3, and the last column is the coefficient estimate corresponding to that selected variable.

```
mod_cv$compact
#>       index_1 index_2  coefficient
#>  [1,]       0       1  0.988300175
#>  [2,]       0       2 -1.454164327
#>  [3,]       0       4 -0.217993186
#>  [4,]       0      36  0.070174151
#>  [5,]       0      59 -0.028997635
#>  [6,]       0      66 -0.076081341
#>  [7,]       0      87 -0.015292763
#>  [8,]       0      94  0.071703217
#>  [9,]       0     112  0.032026436
#> [10,]       0     123 -0.028004284
#> [11,]       0     157 -0.167206693
#> [12,]     158     175  0.025534472
#> [13,]     141     168 -0.006547523
#> [14,]       3       5 -0.155987560
#> [15,]      95     194  0.049738621
#> [16,]       5      97  0.091145031
#> [17,]       4      29 -0.050334021
#> [18,]      17      77  0.023256339
#> [19,]      30     168 -0.052412220
#> [20,]     115     175 -0.027360353
#> [21,]       1       3  2.471482722
#> [22,]       4       5 -3.583816424
```

We see (from the first two rows and the last two rows) that the fit selected by cross-validation includes all the four important variables in the model, with relatively accurate estimates of their coefficients.

Finally, there is a `predict` function for the `S3` object returned by `cv.sprinter` that computes the prediction for a new data matrix of main effects:

```
newdata <- matrix(rnorm(20 * p), nrow = 20, ncol = p)
pred <- predict(mod_cv, newdata = newdata)
```