Using the sprintr package

Guo Yu

```r
library(sprintr)
```

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 (2021) *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 $p = 100$.

```r
library(sprintr)
set.seed(123)
n <- 100
p <- 100
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 (over a path of λ_1 values) of the response y only on main effects X (if `square = FALSE` by default) or with both main effects and squared effects (X, X^2) (if `square = TRUE`). Denote the residual as r_{λ_1}.
- Carry out a screening procedure based on the residual from the previous step. The number of the selected candidate interactions can be specified by a path of `num_keep` values.
- With a path of tuning parameter λ_2 values, fit a lasso of the residual r_{λ_1} on main effects, squared effects (if `square = TRUE`), and selected interactions from the previous step.

There are three tuning parameters: `lambda1` (used in Step 1), `num_keep` (used in Step 2) and `lambda3` (used in Step 3). The default value of `num_keep` is $n/\lceil \log n \rceil$ (see, e.g., Fan & Lv (2008)). If `lambda1` is not specified, then `sprinter` would compute its own path of tuning parameter values based on the number of tuning parameters `nlam1` (default to be 10), `nlam3`(default to be 100), and the range of the path (`lam_min_ratio`).

Finally, a `verbose` option (default to be `TRUE`) can be turned on to see the progress of the computation.

```
fit <- sprinter(x = x, y = y, square = FALSE)
```

`sprinter` output

The output of `sprinter` is a `S3` object including several useful components. The major components are `step1`, `step2`, and `step3`. `step1` includes the `glmnet` fit with tuning `lambda1`. The component `step 2` is a list of length `nlam1`, with `step2[[j]]` containing information about the selected interactions in Step 2 with the residual from Step 1 with tuning parameter `lambda1[j]`. The component `lambda1` is the path of tuning parameters used in Step 1. And `lambda3` is a matrix, with `lambda3[, j]` representing the path of tuning parameters used in Step 3 when Step 1 uses `lambda1[j]` as the tuning parameter.

```
fit$step2[[1]]
#>       index_1 index_2    score
#>  [1,]       4       5 421.3423
#>  [2,]       1       3 320.1576
#>  [3,]       4      96 294.3083
#>  [4,]       5      25 234.5844
#>  [5,]      95      96 224.6897
#>  [6,]      17      77 219.1827
#>  [7,]      29      95 218.0110
#>  [8,]       4      29 210.3960
#>  [9,]      76      77 209.5497
#> [10,]       5      97 205.7667
#> [11,]      43      49 202.0683
#> [12,]       4      57 195.3419
#> [13,]       4      78 195.2760
#> [14,]       3      82 192.4835
#> [15,]      96      97 192.2015
#> [16,]      46      77 191.2056
#> [17,]       1      76 190.6634
#> [18,]       4      72 185.8181
#> [19,]      13      30 183.3793
#> [20,]       8      43 182.8605
#> [21,]       1      87 181.5040
#> [22,]      13      99 181.0824
```

In particular, each element of `step2` contains the indices of all the selected interactions, and the last column represents the corresponding scores used for selection in Step 2.

The component `step3` is a list of length `nlam1`, with `step3[[j]]` containing information from Step 3 fit when the tuning parameter in Step 1 is `lambda1[j]`. Specifically, the output `fit3[[j]]$coef` is a `nrow(fit$step2[[j]]) + p`-by-`length(fit$lambda3)` matrix. Each column of `fit3[[j]]$coef` is a vector of estimates of all variable coefficients (`p` main effects + `nrow(fit$step2[[j]]` selected interactions) considered in Step 3 corresponding to the glmnet fit with one tuning parameter in `lambda3[, j]`. For example, for the 4-th tuning parameter in Step 1 and the 30-th tuning parameter of Step 3, we have the corresponding coefficient estimate:

```r
estimate <- fit$step3[[4]]$coef[, 30]
```

Cross-validating Step 1 before subsequent steps

To facilitate efficient computation, we provide the option to conduct cross-validation in Step 1 before proceeding to Step 2 and Step 3 as described in Section 3.1 in the paper. This functionality can be turned on using `cv_step1 = TRUE` argument:

```r
fit_cvstep1 <- sprinter(x = x, y = y, square = FALSE, cv_step1 = TRUE)
```

The output remains in the same format, with the only difference that only the result corresponding to the CV-selected value in Step 1 is reported.

Summarizing `sprinter` output by `print` and `plot`

The output of `sprinter` has an associated `print` function, that prints information (the number of nonzero main effects and nonzero interactions) of Step 3 fits along a path (column) of `lambda3`, for a given value of Step 1 tuning parameter (specified by `which`). For example, the following codes prints the output when the 2nd value of Step-1 tuning parameter is used:

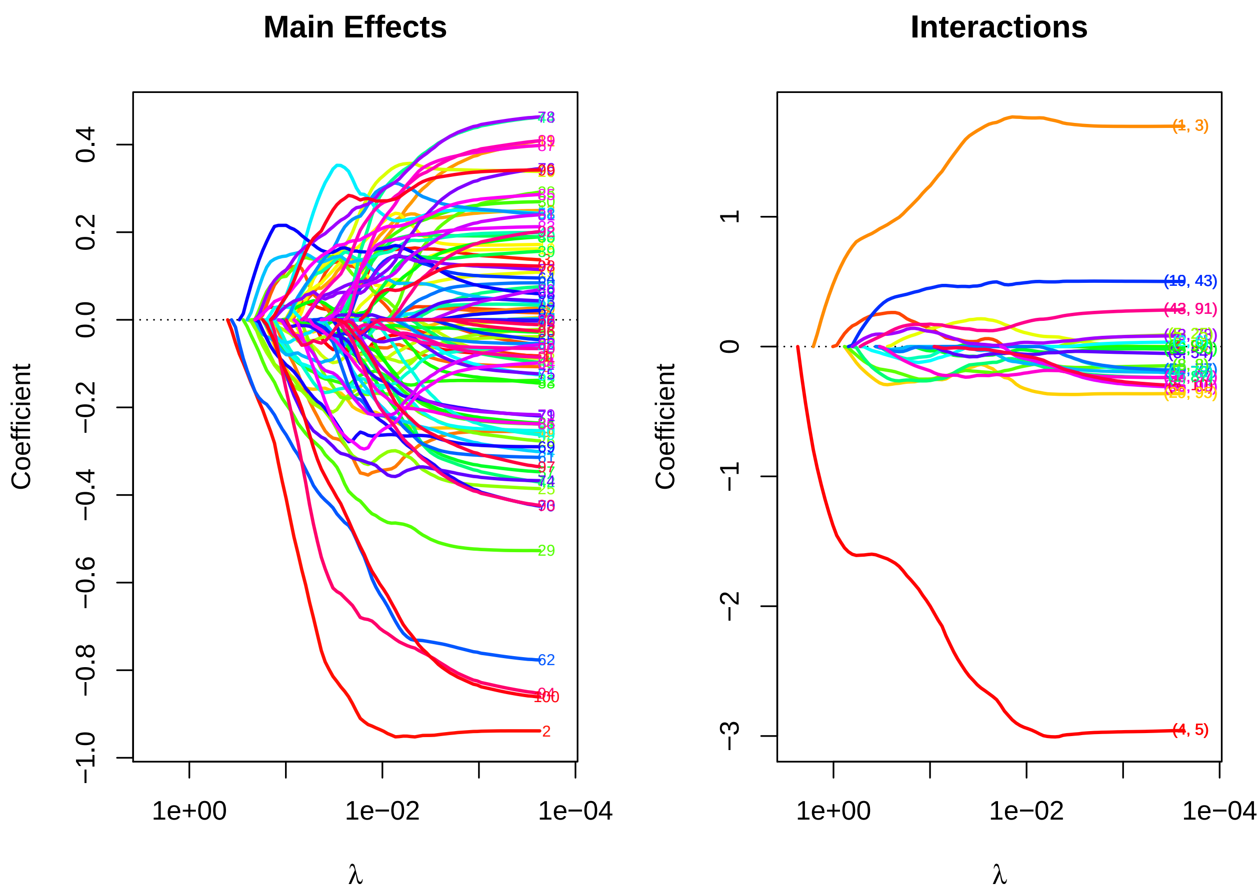
```r
print(fit, which = 2)
#> 
#> Call:  sprinter(x = x, y = y, square = FALSE) 
#> 
#>           lambda #nz main #nz interaction
#>    [1,] 3.5830000        0               0
#>    [2,] 3.2640000        0               1
#>    [3,] 2.9740000        0               1
#>    [4,] 2.7100000        0               1
#>    [5,] 2.4690000        0               2
#>    [6,] 2.2500000        0               2
#>    [7,] 2.0500000        0               2
#>    [8,] 1.8680000        0               2
#>    [9,] 1.7020000        0               2
#>   [10,] 1.5510000        0               2
#>   [11,] 1.4130000        0               2
#>   [12,] 1.2880000        0               3
#>   [13,] 1.1730000        0               3
#>   [14,] 1.0690000        1               3
#>   [15,] 0.9740000        1               3
#>   [16,] 0.8875000        1               3
#>   [17,] 0.8086000        1               3
#>   [18,] 0.7368000        1               3
#>   [19,] 0.6713000        1               3
#>   [20,] 0.6117000        1               4
#>   [21,] 0.5574000        1               4
#>   [22,] 0.5078000        1               4
#>   [23,] 0.4627000        1               5
#>   [24,] 0.4216000        1               6
#>   [25,] 0.3842000        1               6
#>   [26,] 0.3500000        2               6
#>   [27,] 0.3189000        2               8
#>   [28,] 0.2906000        4               9
```

```
#>  [29,] 0.2648000        5              10
#>  [30,] 0.2413000        7              10
#>  [31,] 0.2198000        8              10
#>  [32,] 0.2003000        8              10
#>  [33,] 0.1825000       10              10
#>  [34,] 0.1663000       11              10
#>  [35,] 0.1515000       14              10
#>  [36,] 0.1381000       19              10
#>  [37,] 0.1258000       21              10
#>  [38,] 0.1146000       23              10
#>  [39,] 0.1044000       25              10
#>  [40,] 0.0951600       27              10
#>  [41,] 0.0867100       28              11
#>  [42,] 0.0790000       31              11
#>  [43,] 0.0719900       36              11
#>  [44,] 0.0655900       38              11
#>  [45,] 0.0597600       40              12
#>  [46,] 0.0544500       41              12
#>  [47,] 0.0496200       43              13
#>  [48,] 0.0452100       45              14
#>  [49,] 0.0411900       45              14
#>  [50,] 0.0375300       49              14
#>  [51,] 0.0342000       52              16
#>  [52,] 0.0311600       54              16
#>  [53,] 0.0283900       55              14
#>  [54,] 0.0258700       56              15
#>  [55,] 0.0235700       61              15
#>  [56,] 0.0214800       63              16
#>  [57,] 0.0195700       64              15
#>  [58,] 0.0178300       66              15
#>  [59,] 0.0162500       68              15
#>  [60,] 0.0148000       68              15
#>  [61,] 0.0134900       72              15
#>  [62,] 0.0122900       76              15
#>  [63,] 0.0112000       76              16
#>  [64,] 0.0102000       74              17
#>  [65,] 0.0092970       74              17
#>  [66,] 0.0084710       77              16
#>  [67,] 0.0077190       77              16
#>  [68,] 0.0070330       80              16
#>  [69,] 0.0064080       81              16
#>  [70,] 0.0058390       81              16
#>  [71,] 0.0053200       80              16
#>  [72,] 0.0048480       80              16
#>  [73,] 0.0044170       79              16
#>  [74,] 0.0040250       80              16
#>  [75,] 0.0036670       82              16
#>  [76,] 0.0033410       84              16
#>  [77,] 0.0030440       85              16
#>  [78,] 0.0027740       85              17
#>  [79,] 0.0025280       86              18
#>  [80,] 0.0023030       86              18
#>  [81,] 0.0020980       85              18
```

```
#>   [82,] 0.0019120        85               18
#>   [83,] 0.0017420        86               19
#>   [84,] 0.0015870        87               21
#>   [85,] 0.0014460        88               20
#>   [86,] 0.0013180        88               20
#>   [87,] 0.0012010        91               20
#>   [88,] 0.0010940        92               20
#>   [89,] 0.0009969        93               21
#>   [90,] 0.0009084        93               22
#>   [91,] 0.0008277        94               22
#>   [92,] 0.0007541        95               22
#>   [93,] 0.0006871        95               22
#>   [94,] 0.0006261        95               22
#>   [95,] 0.0005705        95               22
#>   [96,] 0.0005198        96               22
#>   [97,] 0.0004736        97               22
#>   [98,] 0.0004315        98               22
#>   [99,] 0.0003932        98               22
#> [100,] 0.0003583        98               22
```

Furthermore, `plot` function shows the dependence of coefficient estimates for each main effects (left panel) and interactions(right panel) on the Step-3 tuning parameters for a particular value of `lambda1` (specified by `which`):

```r
plot(fit, which = 3)
```



Using cross-validation with `cv.sprinter`

The function `cv.sprinter()` performs a 2-dimensional cross-validation to select the best value pair of `lambda1` (if `cv_step1 == FALSE`) and `lambda3`. If `cv_step1 == TRUE`, then `cv.sprinter()` only performs a 1-dimensional CV to select best value of `lambda3` when Step 1 uses the CV selected `lambda1`.

```
fit_cv <- cv.sprinter(x = x, y = y, square = FALSE)
```

`cv.sprinter` output

The output of `cv.sprinter` is a `S3` object. Please refer to the help document for more detailed description of the output components. The most interesting information is `fit_cv$compact`, which is a matrix of three columns. The first two columns show the indices pairs of all variables finally selected by cross-validation, and the last column is the coefficient estimate corresponding to those selected variables.

```
fit_cv$compact
#>       index_1 index_2  coefficient
#>  [1,]       0       1  1.338539719
#>  [2,]       0       2 -1.856859127
#>  [3,]       0       3  0.094322730
#>  [4,]       0       4 -0.392598879
#>  [5,]       0      13 -0.020814947
#>  [6,]       0      15  0.023568197
#>  [7,]       0      20  0.014210444
#>  [8,]       0      21  0.019829941
#>  [9,]       0      25 -0.073971452
#> [10,]       0      31 -0.032587012
#> [11,]       0      34 -0.102719234
#> [12,]       0      35  0.044691171
#> [13,]       0      38 -0.076747981
#> [14,]       0      39 -0.117843688
#> [15,]       0      43  0.040567303
#> [16,]       0      44  0.109109380
#> [17,]       0      51 -0.056439409
#> [18,]       0      63 -0.014077671
#> [19,]       0      65 -0.291161610
#> [20,]       0      66 -0.003486429
#> [21,]       0      69  0.055057956
#> [22,]       0      71  0.001402845
#> [23,]       0      73  0.029520574
#> [24,]       0      76  0.126417009
#> [25,]       0      87 -0.056684590
#> [26,]       0      91  0.101240945
#> [27,]       0      99  0.213250346
#> [28,]       4       5 -3.948201270
#> [29,]       1       3  2.339323915
#> [30,]       4      96  0.081079197
#> [31,]      95      96  0.060712997
#> [32,]      29      95 -0.111464811
#> [33,]       5      97  0.004551986
#> [34,]      43      49  0.208267278
#> [35,]       4      78 -0.038651094
#> [36,]      46      77  0.002180162
```

```
#> [37,]       1      76  0.020515226
#> [38,]       4      72 -0.037080619
#> [39,]       1      87 -0.012320240
#> [40,]      13      99 -0.054249543
```

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 ($X_1, X_2, X_4 * X_5, X_1 * X_3$) in the model, with relatively accurate estimates of their coefficients.

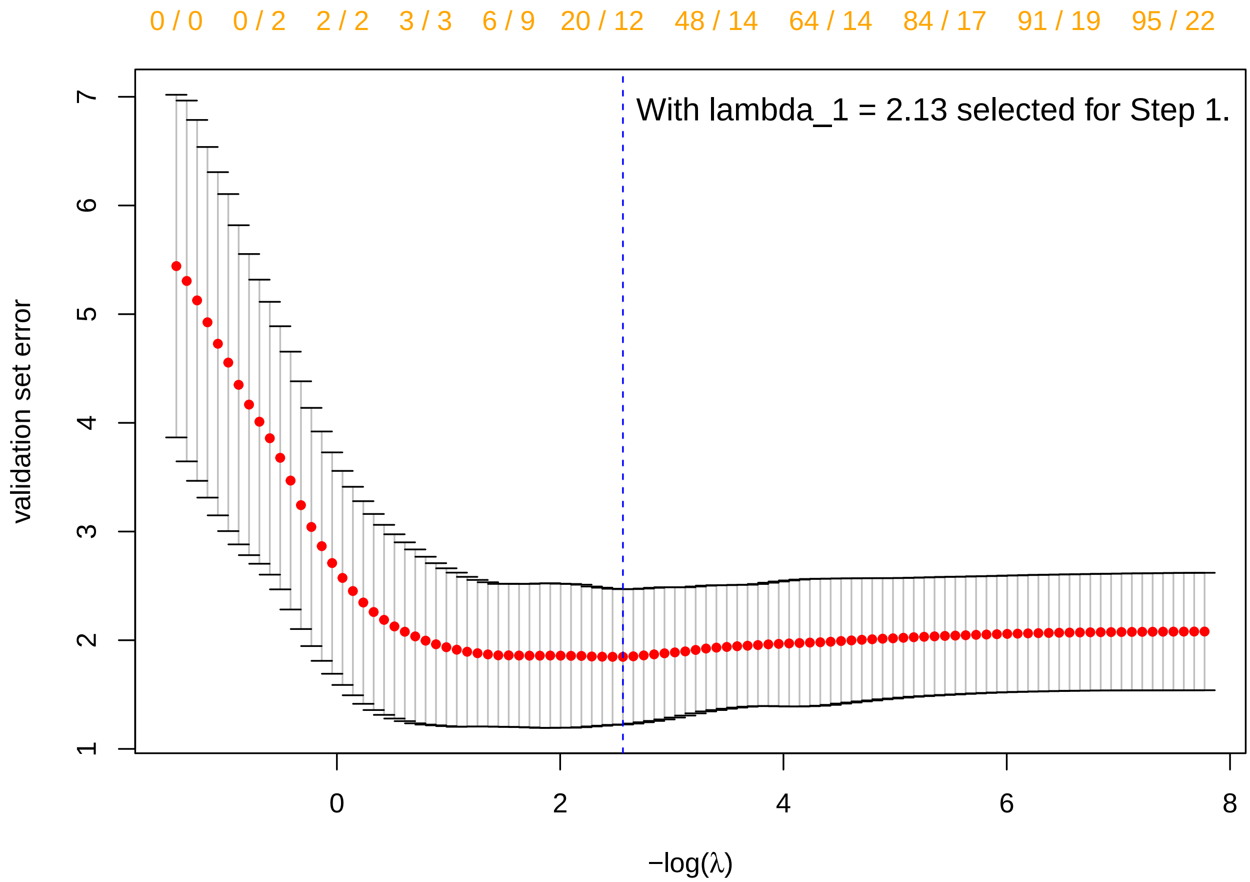
Summarizing `cv.sprinter` output by `print` and `plot`

Associated with the output of `cv.sprinter` are the `print` and `plot` functions. `print` functions can be used to the summary of the cross-validation process, indicating information such as the best number of candidate interactions in Step 2, and the validation error mean/standard errors, number of non-zero main effects/interactions for the Step-3 tuning parameter selected by `min` rule and `1se` rule.

```
print(fit_cv)
#>
#> Call:  cv.sprinter(x = x, y = y, square = FALSE)
#>
#>      lambda1 lambda3 mean(vali-err) se(vali-err) #nonzero-main #nonzero-inter
#> [1,]   2.126 0.07714          1.847       0.6225            27             13
```

The `plot` function for the output of `cv.sprinter` shows the validation error across different folds as a function of Step-3 tuning parameters (for a fixed value of Step-1 tuning parameter chosen by cross-validation). The top of the plot shows the number of nonzero main effects / nonzero interactions corresponding (in orange) to a value of Step-3 tuning parameters.

```
plot(fit_cv)
```



The blue vertical line shows the Step-3 tuning parameter selected by CV that minimizes `cvm`.

Prediction

The `predict` function is defined for both the object returned by `sprinter` and `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(fit, newdata = newdata)
```

The prediction for `sprinter` object computes the prediction at `newdata` for all the (Step-1, Step-3) tuning parameter pairs, and the prediction for `cv.sprinter` object just computes the prediction at `newdata` for the best tuning parameter pairs selected by cross-validation.

```
pred_cv <- predict(fit_cv, newdata = newdata)
```

Update

Additional support for `glmnet` function

We allow additional arguments to be passed to `glmnet` call in `sprinter` and `cv.sprinter` by using the `...` argument.