

Using the sprintr package

Guo Yu

2020-03-03

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 $p = 100$.

```
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)
```

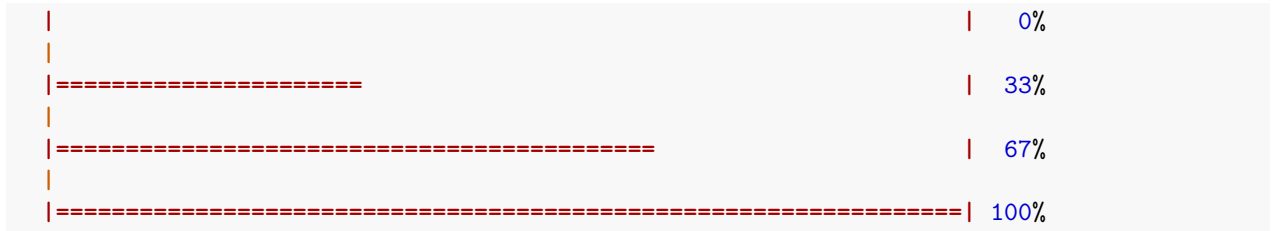
Using sprinter function

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 (with cross-validation) 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**).
- 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 **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 automatically computed as a list of decreasing values, starting from $n/\lceil \log n \rceil$ (see, e.g., Fan & Lv (2008)), based on the number of tuning parameters (**n_num_keep**, default to be 5). 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**). 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, nlam = 100, lam_min_ratio = 0.01)
#>
|
```



sprinter output

The output of `sprinter` is a S3 object including several useful components. In particular, the component `step3` is a list of length `n_num_keep`, with `step3[[j]]` containing information from Step 3 fit when the tuning parameter in Step 2 is `num_keep[j]`.

Within each `step3` component, there exists a `idx` object representing all variables considered in Step 3:

```
fit1 <- fit$step3[[1]]
fit1$idx[(p + 1) : nrow(fit1$idx), ]
#>      index_1 index_2  score
#> [1,]      4      5 336.5337
#> [2,]      4     96 250.2720
#> [3,]      1      3 248.2150
#> [4,]     29     95 181.0657
#> [5,]      5     25 178.7409
#> [6,]      5     97 163.1376
#> [7,]     13     30 161.3963
#> [8,]      5     79 158.3629
#> [9,]     17     77 157.7678
#> [10,]     95     96 152.6769
#> [11,]      4     29 152.2820
#> [12,]      4     72 150.7680
#> [13,]     76     77 147.6554
#> [14,]     43     49 144.7388
#> [15,]      4     78 144.2851
#> [16,]     30     86 143.1823
#> [17,]      1     76 142.8303
#> [18,]      4     57 139.6105
#> [19,]      5     10 139.5537
#> [20,]      2      6 136.9426
#> [21,]     51     71 136.6593
#> [22,]     13     99 136.0234
```

In particular, `fit1$idx[, 1:2]` contains the indices of all the variables indices, and the last column represents their corresponding scores used to select candidate interactions in Step 2. The two columns of `idx` represents the index pair (ℓ, k) of a selected interaction $X_\ell * X_k$, where $\ell \leq k$. If the first entry of an index pair is zero, i.e., $(\ell = 0, k)$, then it represents a main effect X_k (with zero score).

The output `fit1$coef` is a `nrow(fit1$idx)`-by-`length(fit1$lambda)` matrix. Each column of `fit1$coef` is a vector of estimates 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 estimate:

```
estimate <- fit1$coef[, 30]
cb <- cbind(fit1$idx, estimate)
cb[cb[, 3] != 0, ]
```

```

#>      index_1 index_2    score estimate
#> [1,]      4      5 336.5337 -3.162528
#> [2,]      4     96 250.2720  0.000000
#> [3,]      1      3 248.2150  1.618273
#> [4,]     29     95 181.0657  0.000000
#> [5,]      5     25 178.7409  0.000000
#> [6,]      5     97 163.1376  0.000000
#> [7,]     13     30 161.3963  0.000000
#> [8,]      5     79 158.3629  0.000000
#> [9,]     17     77 157.7678  0.000000
#> [10,]     95     96 152.6769  0.000000
#> [11,]      4     29 152.2820  0.000000
#> [12,]      4     72 150.7680  0.000000
#> [13,]     76     77 147.6554  0.000000
#> [14,]     43     49 144.7388  0.000000
#> [15,]      4     78 144.2851  0.000000
#> [16,]     30     86 143.1823  0.000000
#> [17,]      1     76 142.8303  0.000000
#> [18,]      4     57 139.6105  0.000000
#> [19,]      5     10 139.5537  0.000000
#> [20,]      2      6 136.9426  0.000000
#> [21,]     51     71 136.6593  0.000000
#> [22,]     13     99 136.0234  0.000000

```

Summarizing sprinter output by print, plot, and summary

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 of Step 3 tuning parameters, for a given value of Step 2 tuning parameter (specified by `which`). For example, the following codes prints the output when the 2nd value of Step-2 tuning parameter is used:

```

print(fit, which = 2)
#>
#> Call:  sprinter(x = x, y = y, square = FALSE, nlam = 100, lam_min_ratio = 0.01)
#>
#>      lambda #nz main #nz interaction
#> [1,] 4.21300      0      0
#> [2,] 4.02200      0      1
#> [3,] 3.83900      0      1
#> [4,] 3.66500      0      1
#> [5,] 3.49800      0      1
#> [6,] 3.33900      0      1
#> [7,] 3.18700      0      1
#> [8,] 3.04200      0      1
#> [9,] 2.90400      0      2
#> [10,] 2.77200      0      2
#> [11,] 2.64600      0      2
#> [12,] 2.52600      0      2
#> [13,] 2.41100      0      2
#> [14,] 2.30100      0      2
#> [15,] 2.19700      0      2
#> [16,] 2.09700      0      2
#> [17,] 2.00200      0      2

```

#>	[18,]	1.91100	0	2
#>	[19,]	1.82400	0	2
#>	[20,]	1.74100	2	2
#>	[21,]	1.66200	2	2
#>	[22,]	1.58600	2	2
#>	[23,]	1.51400	2	2
#>	[24,]	1.44500	2	2
#>	[25,]	1.38000	2	2
#>	[26,]	1.31700	2	2
#>	[27,]	1.25700	2	2
#>	[28,]	1.20000	2	2
#>	[29,]	1.14500	2	2
#>	[30,]	1.09300	2	2
#>	[31,]	1.04400	2	2
#>	[32,]	0.99630	2	2
#>	[33,]	0.95100	2	2
#>	[34,]	0.90780	2	2
#>	[35,]	0.86650	2	2
#>	[36,]	0.82710	2	2
#>	[37,]	0.78950	2	2
#>	[38,]	0.75360	2	2
#>	[39,]	0.71940	2	3
#>	[40,]	0.68670	2	3
#>	[41,]	0.65550	2	3
#>	[42,]	0.62570	3	3
#>	[43,]	0.59720	3	3
#>	[44,]	0.57010	3	3
#>	[45,]	0.54420	3	3
#>	[46,]	0.51950	3	3
#>	[47,]	0.49580	3	3
#>	[48,]	0.47330	3	3
#>	[49,]	0.45180	3	3
#>	[50,]	0.43130	3	4
#>	[51,]	0.41170	3	4
#>	[52,]	0.39290	3	4
#>	[53,]	0.37510	3	4
#>	[54,]	0.35800	3	4
#>	[55,]	0.34180	4	4
#>	[56,]	0.32620	4	4
#>	[57,]	0.31140	4	7
#>	[58,]	0.29720	5	7
#>	[59,]	0.28370	5	8
#>	[60,]	0.27080	5	8
#>	[61,]	0.25850	5	8
#>	[62,]	0.24680	5	8
#>	[63,]	0.23560	5	8
#>	[64,]	0.22490	5	8
#>	[65,]	0.21460	6	8
#>	[66,]	0.20490	7	9
#>	[67,]	0.19560	9	9
#>	[68,]	0.18670	11	9
#>	[69,]	0.17820	11	9
#>	[70,]	0.17010	11	9

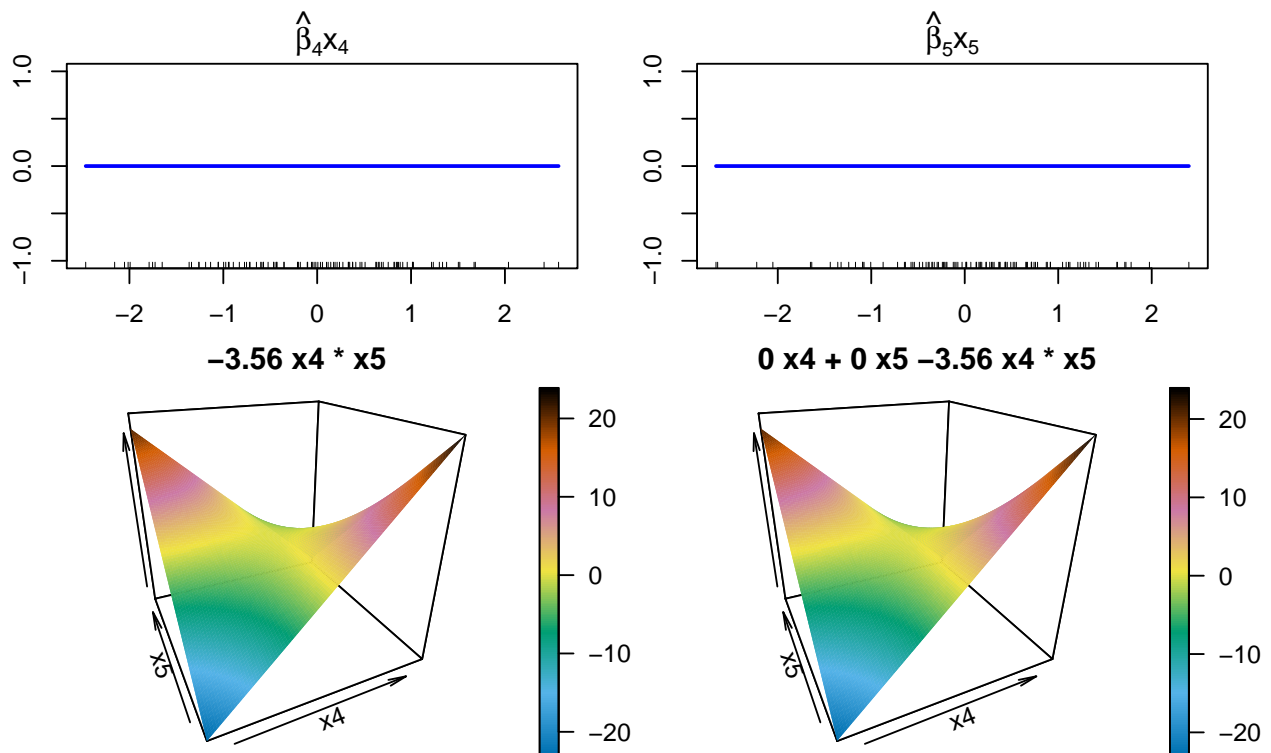
```

#> [71,] 0.16240      12      9
#> [72,] 0.15500      14      9
#> [73,] 0.14790      15      9
#> [74,] 0.14120      16     10
#> [75,] 0.13480      16     10
#> [76,] 0.12870      16     11
#> [77,] 0.12280      16     11
#> [78,] 0.11720      17     12
#> [79,] 0.11190      17     11
#> [80,] 0.10680      17     11
#> [81,] 0.10200      18     11
#> [82,] 0.09734      18     11
#> [83,] 0.09291      21     12
#> [84,] 0.08869      23     12
#> [85,] 0.08466      24     12
#> [86,] 0.08081      26     13
#> [87,] 0.07714      27     12
#> [88,] 0.07363      29     11
#> [89,] 0.07028      33     11
#> [90,] 0.06709      33     12
#> [91,] 0.06404      34     12
#> [92,] 0.06113      34     12
#> [93,] 0.05835      34     12
#> [94,] 0.05570      34     12
#> [95,] 0.05317      34     12
#> [96,] 0.05075      37     12
#> [97,] 0.04844      40     12
#> [98,] 0.04624      41     12
#> [99,] 0.04414      41     12
#> [100,] 0.04213      42     12

```

The `plot` function is also defined for `sprinter` output to look at the effects from a certain interaction (specified by `which`). For example, by examining the effect of interaction between X_4 and X_5 , we run the following `plot` function that produces 4 panels:

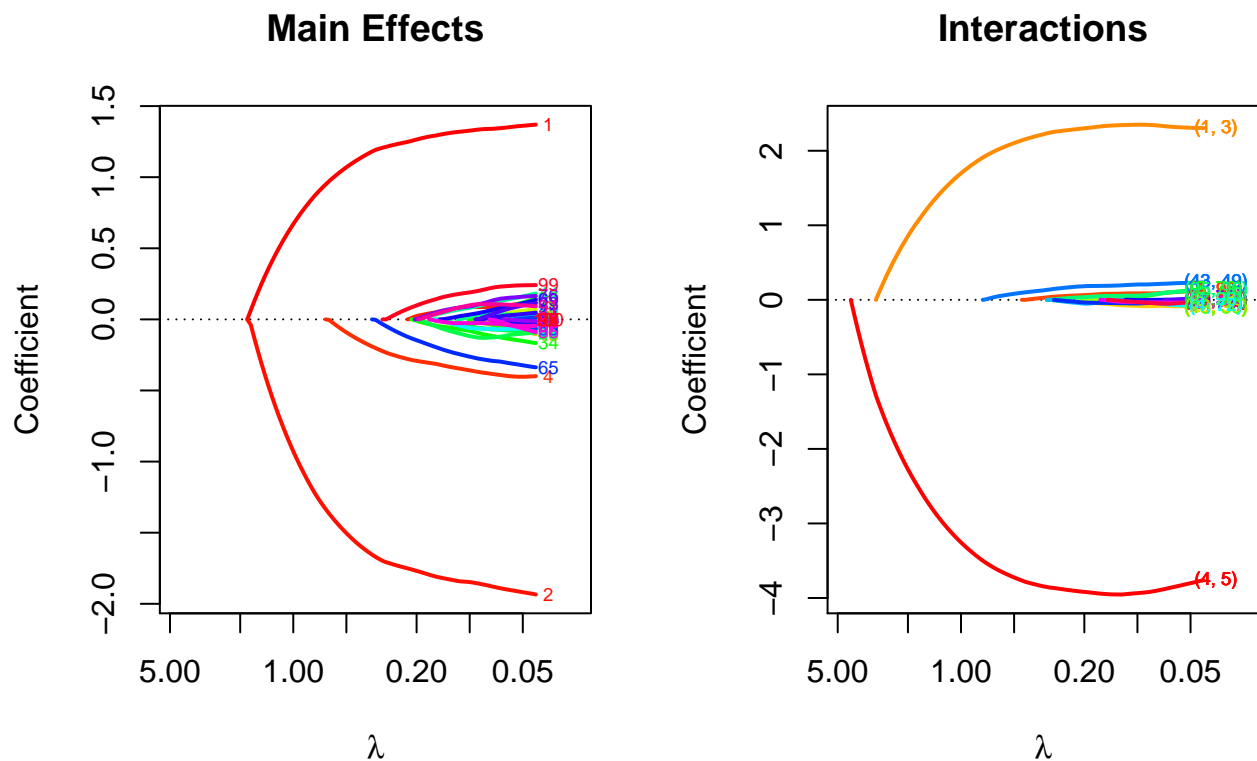
```
plot(fit, newdata = x, index = c(1, 40), which = c(4, 5))
```



The top two panels show the marginal relationship of the predicted response on the two main effects, i.e., $\hat{\beta}_4 X_4$ and $\hat{\beta}_5 X_5$ respectively. The lower-left panel shows the dependence of the predicted response on the interaction alone, i.e., $\hat{\gamma}_{45} X_4 * X_5$, and the lower-right panel shows $\hat{\beta}_4 X_4 + \hat{\beta}_5 X_5 + \hat{\gamma}_{45} X_4 * X_5$. From the plot we see that the predicted response does not depend on either of the two main effects, but depends on their interaction (with coefficient -3.56)

Finally, `summary` function shows the dependence of coefficient estimates for each main effects (left panel) and interactions(right panel) on the Step-3 tuning parameters:

```
summary(fit)
```



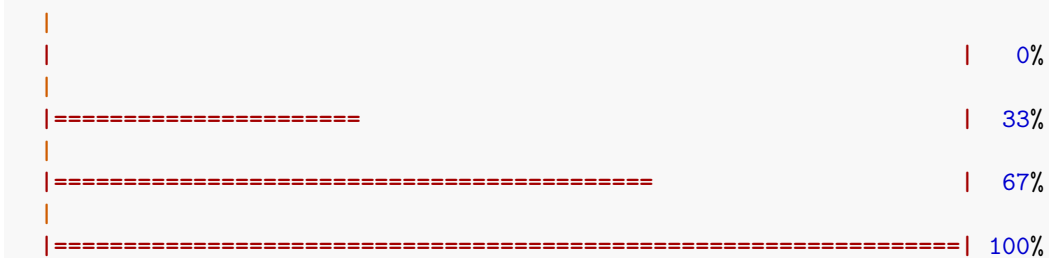
Using cross-validation with `cv.sprinter`

The function `cv.sprinter()` performs cross-validation to select the best value pairs of Step-2 and Step-3 tuning parameters.

```
fit_cv <- cv.sprinter(x = x, y = y, square = FALSE, n_num_keep = 5, nlam = 100, lam_min_ratio = 0.01)
```

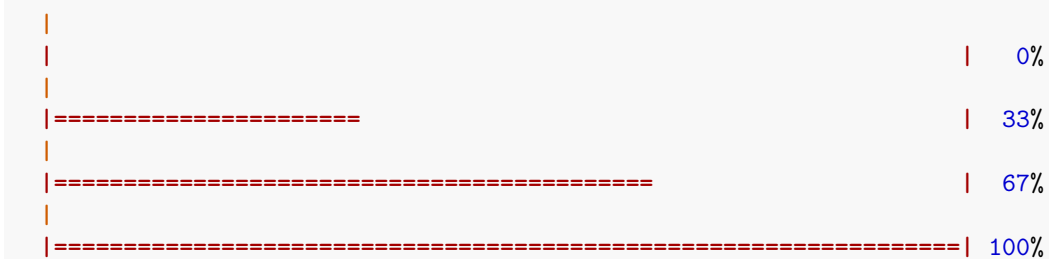
```
#> cv initial:
```

```
#>
```



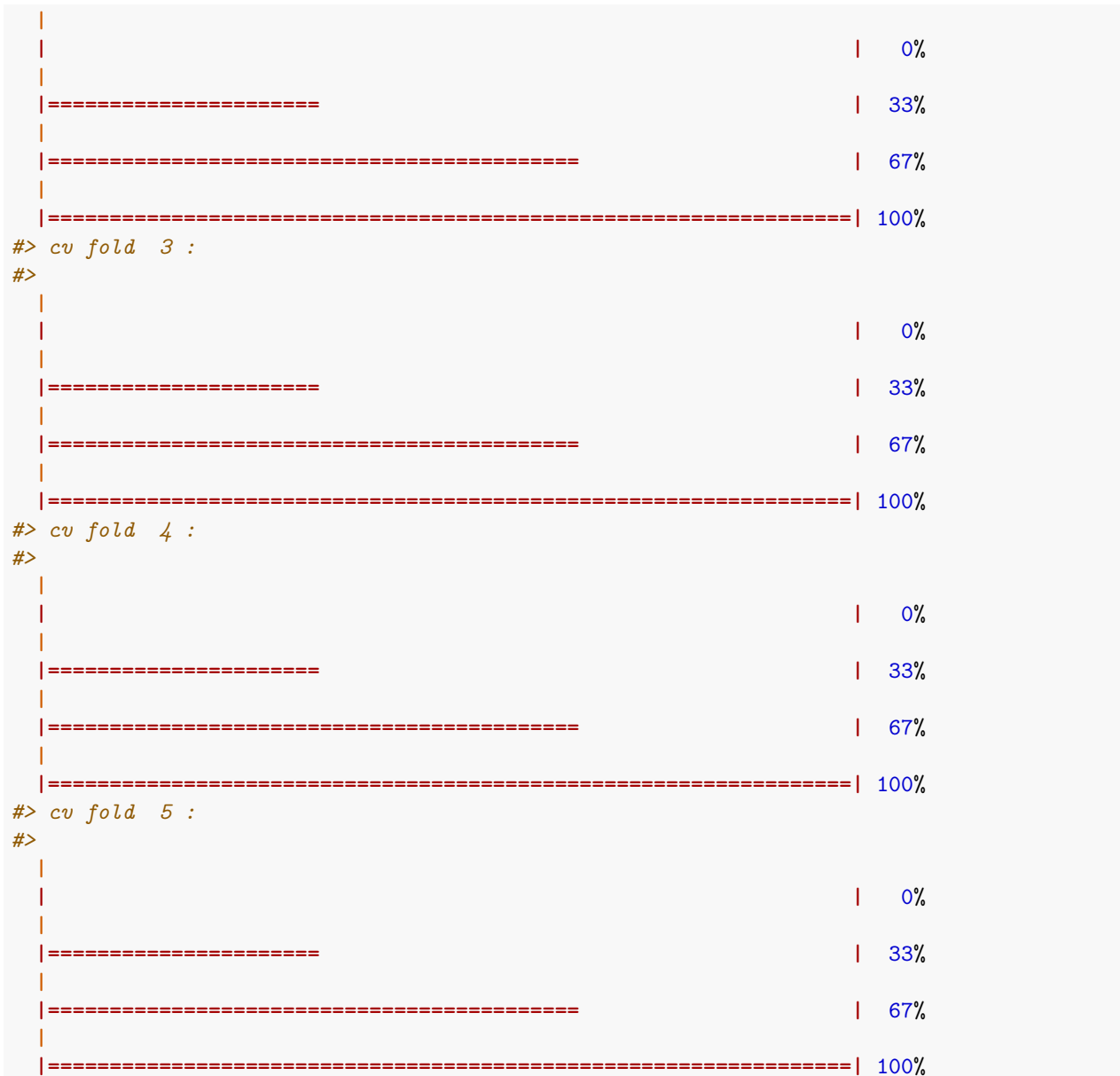
```
#> cv fold 1 :
```

```
#>
```



```
#> cv fold 2 :
```

```
#>
```



cv.sprinter output

The output of `cv.sprinter` is a S3 object. 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  1.3627135262
#> [2,]      0      2 -1.8941582052
#> [3,]      0      3  0.1050959648
#> [4,]      0      4 -0.4092385412
#> [5,]      0     21  0.0007039331
#> [6,]      0     25 -0.0838433007

```



```

#> [7,] 0 26 -0.0499478617
#> [8,] 0 31 -0.0297285465
#> [9,] 0 34 -0.0878596104
#> [10,] 0 35 0.0412542636
#> [11,] 0 38 -0.0415217706
#> [12,] 0 39 -0.1453470347
#> [13,] 0 43 0.0642984181
#> [14,] 0 44 0.1203193907
#> [15,] 0 51 -0.0126056316
#> [16,] 0 64 0.0196750028
#> [17,] 0 65 -0.2296652640
#> [18,] 0 66 -0.0175774607
#> [19,] 0 69 0.0073273880
#> [20,] 0 76 0.1071258137
#> [21,] 0 78 -0.0136436397
#> [22,] 0 87 -0.0550568118
#> [23,] 0 91 0.0885291261
#> [24,] 0 99 0.2078193532
#> [25,] 0 100 0.0225545973
#> [26,] 4 5 -4.0211027225
#> [27,] 1 3 2.4013600490
#> [28,] 4 96 0.0890808049
#> [29,] 29 95 -0.1008666063

```

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.

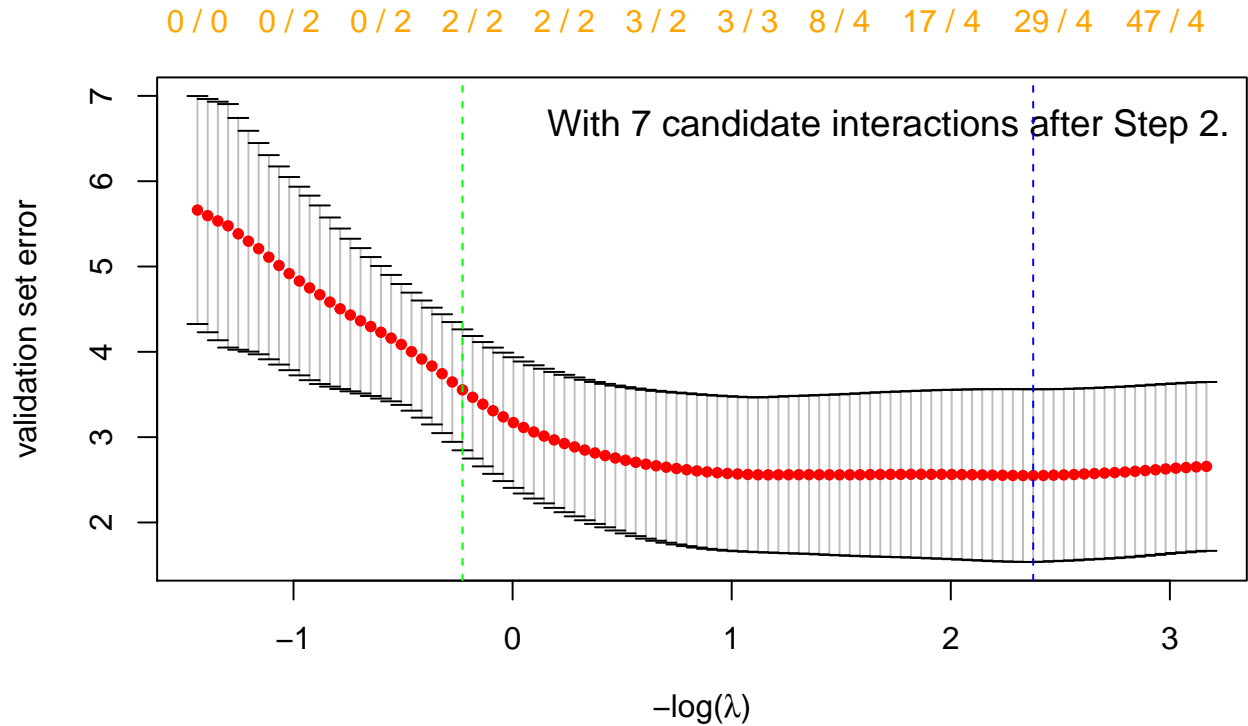
```

print(fit_cv)
#>
#> Call:  cv.sprinter(x = x, y = y, square = FALSE, n_num_keep = 5, nlam = 100,      lam_min_ratio = 0.
#>
#> Best number of candidate interactions in Step 2: 7
#>
#>      lambda mean(vali-err) se(vali-err) #nonzero-main #nonzero-inter
#> min 0.09291      4.364      0.8519      25      4
#> 1se 1.25700      5.296      1.2940      2      2

```

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-2 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 `min` rule, and the green vertical line shows the Step-3 tuning parameter selected by `1se` rule.

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-2, 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)
```

Support for other response families

Under construction