SHIM-Tweedie

February 7, 2022

1 Implementation

1.1 Initialization

$$
\begin{aligned}
g(\mathbf{x}) =& \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p + \alpha_{12}\left(x_1 x_2\right) + \alpha_{13}\left(x_1 x_3\right) + \cdots + \alpha_{p-1,p}\left(x_{p-1} x_p\right) \\
=& \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p + \gamma_{12}\beta_1\beta_2\left(x_1 x_2\right) + \cdots + \gamma_{p-1,p}\beta_{p-1}\beta_p\left(x_{p-1} x_p\right),
\end{aligned} \tag{1}
$$

1. Paper: For example, we can use the least square estimates or the simple regression estimates by regressing the response y on each of the terms.

2. For β_i, fit $y \sim \beta_i$.

3. For γ_{ij} fit $y \ \sim \beta_i * \beta_j * x_i * x_j$

1.2 γ_{ij} update

1. Weights are set to be 1.

2. $\sum_{i=1}^{p} \beta_i * x_i$ is used as an offset to fit TGLM.

3. For each γ_{ij}, $\beta_i * \beta_j * x_i * x_j$ is used as x_i'.

4. The choice of λ_γ needs to be discussed.

5. So far, I let cv.glmnet to choose λ_γ

2 β_i update

4. *Update* $\hat{\beta}_j$.

   - Let $\hat{\beta}_j^{(m)} = \hat{\beta}_j^{(m-1)}, j = 1, \ldots, p.$
   - For each j in $1, \ldots, p$, let

   $$\tilde{y}_i = y_i - \sum_{j' \neq j} \hat{\beta}_{j'}^{(m)} x_{ij'} - \sum_{j' < j'', j', j'' \neq j} \hat{\beta}_{j'}^{(m)} \hat{\beta}_{j''}^{(m)} (x_{ij'} x_{ij''}),$$

   $$i = 1, \ldots, n,$$

   $$\tilde{x}_i = x_{ij} + \sum_{j' < j} \hat{\gamma}_{j'j}^{(m)} \hat{\beta}_{j'}^{(m)} (x_{ij'} x_{ij}) + \sum_{j' > j} \hat{\gamma}_{jj'}^{(m)} \hat{\beta}_{j'}^{(m)} (x_{ij} x_{ij'}),$$

   $$i = 1, \ldots, n,$$

   then

   $$\hat{\beta}_j^{(m)} = \arg\min_{\beta_j} \sum_{i=1}^{n} \left((\tilde{y}_i - \beta_j \tilde{x}_i)^2 + \lambda_\beta w_j^\beta |\beta_j| \right).$$

Using first iteration as an example:

1. Let all β's be β_0.

2. Then follow the algorithm to update β_i, $i = 1 \ldots p$

3. From the package, it seems to create one λ_{β_i} for each β_i, here it has only one λ_β.

3 λ for γ and β

In the package, there is a function called lambda_sequence:

```r
lambda_sequence <- function(x, y, weights = NULL,
                            lambda.factor = ifelse(nobs < nvars, 0.01, 1e-06),
                            nlambda = 100, scale_x = F, center_y = F) {

  # when scaling, first you center then you standardize
  if (any(as.vector(weights) < 0)) stop("Weights must be positive")
  np <- dim(x)
  nobs <- as.integer(np[1])
  nvars <- as.integer(np[2])

  if (!is.null(weights) & length(as.vector(weights)) < nvars)
    stop("You must provide weights for every column of x")

  # scale the weights to sum to nvars
  w <- if (is.null(weights)) rep(1, nvars) else as.vector(weights) / sum(as.vector(weights)) * nvars

  sx <- if (scale_x) apply(x,2, function(i) scale(i, center = TRUE, scale = mysd(i))) else x
  sy <- if (center_y) as.vector(scale(y, center = T, scale = F)) else as.vector(y)
  lambda.max <- max(abs(colSums(sy * sx) / w)) / nrow(sx)

  rev(exp(seq(log(lambda.factor * lambda.max), log(lambda.max), length.out = nlambda)))
}
```

When $\beta = 0$, we see from (5) that β_j will stay zero if $\frac{1}{N}|\langle x_j, y\rangle| < \lambda\alpha$. Hence $N\alpha\lambda_{max} = \max_\ell |\langle x_\ell, y\rangle|$. Our strategy is to select a minimum value $\lambda_{min} = \epsilon\lambda_{max}$, and construct a sequence of K values of λ decreasing from $\lambda_{\max}$ to λ_{min} on the log scale. Typical values are $\epsilon = 0.001$ and $K = 100$.