Semiparametric Thresholding Least Squares

Pierre Chausse[*], Mihai Giurcanu[†], George Luta[‡]

Abstract

The vignette includes ideas for improving the causalTLSE package.

Introduction

This document presents one possible extension to the `causalTLSE` package. The TLSE causal effects will eventually be based on the following functions and methods. The causal effect model is

$$Y = \beta_0(1-Z) + \beta_1 Z + f_0(X) + f_1(X) + \varepsilon\,,$$

and it is approximated by the regression

$$Y = \beta_0(1-Z) + \beta_1 Z + \psi_0^T U_0(X)(1-Z) + \psi_1^T U_1(X) Z + u\,,$$

where $Y \in \mathbb{R}$ is the response variable, X is a $k\times 1$ vector of confounders, and $U_0(X) \in \mathbb{R}^{p_0}$ and $U_1(X) \in \mathbb{R}^{p_1}$ are spline vectors. We can represent $U_j(X)$, for $j=0,1$, into a block vector $\{U_{j1}(X_1)^T, U_{j2}(X_2)^T, ..., U_{jk}(X_k)^T\}^T$, where $U_{jl}(X_l) \in \mathbb{R}^{p_{jl}}$ is the vector of basis functions for group j associated with X_l, with $\sum_{l=1}^k p_{jl} = p_j$, $j=0,1$. This is equivalent to estimating the following two models separately:

$$\begin{aligned} Y &= \beta_0 + \psi_0^T U_0(X) + u(0) \\ Y &= \beta_1 + \psi_1^T U_1(X) + u(1) \end{aligned}$$

The detail on how $U_0(X)$ and $U_1(X)$ are defined and selected is presented in the causalTLSE vignette, so we won't repeat it here. It is therefore important to be familiar with this vignette to understand what follows.

What is important here is to realize that the package provides a semiparametric method for estimating the model

$$Y = \beta + f(X) + \varepsilon\,.$$

using the following basis function representation:

$$Y = \beta + \psi^T U(X) + u$$

We can create a model similar to the `tlseModel`, but without splitting the sample into two groups. We would simply have one set of knots per covariate. For now, we name the class "slse". The model constructor would be simpler, because it would not divide the sample in two. Also, the formula would not need a | operator. We would only provide the regression formula

[*]University of Waterloo, pchausse@uwaterloo.ca
[†]University of Chicago, giurcanu@uchicago.edu
[‡]Georgetown University, George.Luta@georgetown.edu

```r
slse <-  function (form, data, nbasis = function(n) n^0.3, knots,
                   userRem = NULL)
{
    mf <- model.frame(form, data)
    X <- model.matrix(form, data)
    Y <- model.response(mf)
    if (attr(terms(form), "intercept") == 1)
    {
        X <- X[, -1, drop = FALSE]
        reg <- "~Xf"
    } else {
        reg <- "~Xf-1"
    }
    formY <- as.formula(paste(all.vars(form)[1], reg, sep=""),
                        env = .GlobalEnv)
    formX <- formula(delete.response(terms(form)),  env = .GlobalEnv)
    na <- na.omit(cbind(Y, X))
    if (missing(knots)) {
        select <- "SLSE"
        crit <- ""
    } else {
        select <- "User Based"
        crit <- ""
    }
    if (missing(knots))
        knots <- as.list(rep(NA, ncol(X)))
    if (is.null(knots))
        knots <- lapply(1:ncol(X), function(i) NULL)
    if (!is.list(knots))
        stop("knots must be a list")
    if (length(knots) != ncol(X))
        stop("The length of knots must be equal to the number of covariates")
    if (!is.null(attr(na, "omit")))
    {
        na <- attr(na, "omit")
        X <- X[-na, , drop = FALSE]
        data <- data[-na, , drop = FALSE]
    } else {
        na <- NULL
    }
    if (!is.null(userRem))
    {
        w <- which(colnames(X) %in% userRem)
        if (length(w))
            knots[w] <- lapply(w, function(i) NULL)
    }
    nameX <- colnames(X)
    nameY <- all.vars(formY)[1]
    knots <- lapply(1:ncol(X), function(i)
        causalTLSE:::setKnots(X[, i], nbasis=nbasis, knots=knots[[i]]))
    names(knots) <- nameX
    obj <- list(na = na, formY = formY, formX = formX,
        nameY = nameY, knots = knots, data = data,
        nameX = nameX, method = list(select = select, crit = crit))
    class(obj) <- "slse"
    obj
}
```

The print method also needs to be simplified:

```r
print.slse <- function (x, knots = FALSE, ...)
{
    if (!knots) {
        cat("Semiparametric LSE Model\n")
        cat("************************\n\n")
        cat("Number of observations: ", nrow(x$data), "\n")
        cat("Number of missing values: ", length(x$na), "\n")
        cat("Selection Method: ", x$method$select, "\n", sep = "")
        if (x$method$crit != "")
            cat("Criterion: ", x$method$crit, "\n\n", sep = "")
        cat("Covariates approximated by semiparametric LSE:\n")
        w <- sapply(x$knots, is.null)
        selPW <- x$nameX[!w]
        nonselPW <- x$nameX[w]
        isApp <- if (length(selPW))
                     paste(selPW, collapse = ", ", sep = "")
                 else "None"
        notApp <- if (length(nonselPW))
                      paste(nonselPW, collapse = ", ", sep = "")
                  else "None"
        cat("\t", isApp, "\n", sep = "")
        cat("Covariates not approximated by semiparametric LSE:\n")
            cat("\t", notApp, "\n", sep = "")
    } else {
        cat("Lists of knots\n")
        cat("**************\n")
        for (sel in 1:length(x$knots)) {
            cat(x$nameX[sel], ":\n", sep = "")
            if (is.null(x$knots[[sel]]))
                cat("None\n")
            else print.default(format(x$knots[[sel]], ...),
                print.gap = 2L, quote = FALSE)
        }
    }
    invisible()
}
```

We can try it with the simDat4 dataset

```r
data(simDat4)
mod1 <- slse(Y~X1+X2+X3+X4, data=simDat4)
mod1
```

```
## Semiparametric LSE Model
## ************************
##
## Number of observations:  500
## Number of missing values:  0
## Selection Method: SLSE
## Covariates approximated by semiparametric LSE:
##  X1, X3
## Covariates not approximated by semiparametric LSE:
##  X2, X4
```

```
print(mod1, knots=TRUE)
```

```
## Lists of knots
## **************
## X1:
##   33.33333%          50%    66.66667%    83.33333%
##  0.03860281   1.19725489   5.69003907  15.45468686
## X2:
## None
## X3:
## 33.33333%
##         2
## X4:
## None
```

Estimation

We first simplify the internal functions:

```
.chkSelKnots <- function (model, w)
{
    wK <- "knots"
    knots <- model[[wK]]
    if (is.null(w))
        return(knots)
    if (!is.list(w))
        stop("The knots selection must be included in a list")
    if (length(w) != length(knots))
        stop(paste("The length of the knots selection list does not match the length of ",
            wK, sep = ""))
    k <- lapply(1:length(w), function(i) {
        ki <- knots[[i]]
        wi <- w[[i]]
        if (is.null(ki))
            return(NULL)
        if (is.null(wi))
            return(NULL)
        if (any(is.na(wi)))
            stop("The knots selection list cannot contain NAs")
        if (!is.integer(wi))
            stop("The knots selection list can only contain integers")
        wi <- unique(wi)
        if (any(wi < 1) | any(wi > length(ki)))
            stop(paste("Knot selection out of bound in ", wK,
                sep = ""))
        ki <- ki[wi]
    })
    names(k) <- model$nameX
    k
```

```r
}

.splineMatrix <- function (model, which)
{
    X <- causalTLSE:::model.matrix.tlseModel(model)[, which]
    knots <-  model$knots[[which]]
    if (is.null(knots))
        return(as.matrix(X))
    n <- length(X)
    p <- length(knots) + 1
    Xf <- matrix(0, nrow = n, ncol = p)
    Xf[, 1] <- X * (X <= knots[1]) + knots[1] * (X > knots[1])
    Xf[, p] <- (X - knots[p - 1]) * (X > knots[p - 1])
    if (p >= 3) {
        for (j in 2:(p - 1)) {
            Xf[, j] <- (X - knots[j - 1]) * (X >= knots[j -
                1]) * (X <= knots[j]) + (knots[j] - knots[j -
                1]) * (X > knots[j])
        }
    }
    Xf
}

multiSplines <- function (model)
{
    all <- lapply(1:length(model$nameX), function(i) {
        ans <- .splineMatrix(model, i)
        nk <- length(model$knots[[i]]) + 1
        colnames(ans) <- if (nk == 1) {
                             model$nameX[i]
                         } else {
                             paste(model$nameX[i], "_", 1:nk, sep = "")
                         }
        ans
    })
    names(all) <- model$nameX
    cnames <- lapply(all, colnames)
    names(cnames) <- names(all)
    all <- do.call(cbind, all)
    attr(all, "p") <- sapply(knots, length) + 1
    attr(all, "colnames") <- cnames
    all
}
```

Then we rewrite the estimation function

```r
estSLSE <- function(model, w = NULL)
{
    if (!inherits(model, "slse"))
        stop("model must be an object of class slse")
    model$knots <- .chkSelKnots(model, w)
    data <- model$data
    data$Xf <- multiSplines(model)
    form <- model$formY
    environment(form) <- environment()
    fit <- lm(form, data)
    obj <- list(lm.out = fit, model = model)
    class(obj) <- "slseFit"
    obj
}
print.slseFit <- function (x, ...)
{
    cat("Semiparametric LSE Estimate\n")
    cat("***************************\n")
    cat("Selection Method: ", x$model$method$select, "\n", sep = "")
    if (x$model$method$crit != "") {
        cat("Criterion: ", x$model$method$crit, "\n\n", sep = "")
    }
    else {
        cat("\n")
    }
    print.default(format(coef(x$lm.out), ...), print.gap = 2L,
        quote = FALSE)
    invisible()
}

summary.slseFit <- function(object, vcov.=vcovHC, ...)
    summary.tlseFit(object, vcov., ...)
```

Let's try it

```
fit <- estSLSE(mod1)
fit
```

```
## Semiparametric LSE Estimate
## ***************************
## Selection Method: SLSE
##
##   (Intercept)        XfX1_1        XfX1_2        XfX1_3        XfX1_4
## -0.351596632  -4.157343475   0.111423712   0.030350442  -0.016260848
##       XfX1_5          XfX2        XfX3_1        XfX3_2          XfX4
##  0.001876416   0.045432141   0.135984110  -0.070563799  -0.171961824
```

```
summary(fit)
```

```
## Semiparametric TLSE Estimate
## ****************************
## Selection method: SLSE
##
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.351597   0.379806  -0.926   0.3546
## XfX1_1      -4.157343   4.108908  -1.012   0.3116
## XfX1_2       0.111424   0.178622   0.624   0.5328
## XfX1_3       0.030350   0.048955   0.620   0.5353
## XfX1_4      -0.016261   0.022725  -0.716   0.4743
## XfX1_5       0.001876   0.004204   0.446   0.6554
## XfX2         0.045432   0.096267   0.472   0.6370
## XfX3_1       0.135984   0.112177   1.212   0.2254
## XfX3_2      -0.070564   0.104240  -0.677   0.4984
## XfX4        -0.171962   0.087088  -1.975   0.0483 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared:  0.0154, Adjusted R-squared:  -0.002686
```

Predict and Plot

Selection

```
.testKnots <- function (fit, model, whichK, whichX, treated, vcov)
{
    wK <- "knots"
    wX <- "Xf"
    if (whichX > length(model[[wK]]))
        stop("whichX exceeds the number of covariates")
    if (any(whichK > length(model[[wK]][[whichX]])))
        stop("whichK exceeds the number of knots")
    if (is.null(model[[wK]][[whichX]]))
        return(NA)
    b <- coef(fit)
    b <- na.omit(b)
    sapply(whichK, function(wi) {
        nX <- names(model[[wK]])[[whichX]]
        t <- c(paste(wX, nX, "_", wi, sep = ""), paste(wX, nX,
            "_", wi + 1, sep = ""))
        c1 <- which(names(b) == t[1])
        c2 <- which(names(b) == t[2])
        if (length(c(c1, c2)) < 2)
            return(NA)
        s2 <- vcov[c1, c1] + vcov[c2, c2] - 2 * vcov[c1, c2]
        ans <- 1 - pf((b[c1] - b[c2])^2/s2, 1, fit$df)
        names(ans) <- NULL
        ans
    })
}

.getPvalB <- function (model, vcov. = vcovHC, ...)
{
    data2 <- model$data
    data2$Xf <- multiSplines(model)
    form <- model$formY
    environment(form) <- environment()
    fit <- lm(form, data2)
    p <- attr(data2$Xf, "p")
    v <- vcov.(fit, ...)
    pval <- lapply(1:length(model$knots), function(i) {
        ki <- length(model$knots[[i]])
        if (ki == 0)
            NA
        else .testKnots(fit, model, 1:ki, i, FALSE, v)
    })
    pval1 <- lapply(1:length(model$knots1), function(i) {
        ki <- length(model$knots1[[i]])
        if (ki == 0)
            NA
        else .testKnots(fit, model, 1:ki, i, TRUE, v)
    })
    names(pval0) <- names(pval1) <- names(p0) <- names(p1) <- model$nameX
    list(pval0 = pval0, pval1 = pval1, p0 = p0, p1 = p1)
}


selSLSE <- function (model, method = c("FTLSE", "BTLSE"), crit = c("AIC",
    "BIC", "ASY"), pvalT = function(p) 1/log(p), vcov. = vcovHC,
    ...)
{
    crit <- match.arg(crit)
    method <- match.arg(method)
    critFct <- if (crit == "ASY") {
        causalTLSE.selASY
    }
    else {
        .selIC
    }
    if (method == "BTLSE")
        pval <- .getPvalB(model, vcov., ...)
    else pval <- .getPvalF(model, vcov., ...)
    model <- critFct(model, pval, pvalT, crit)
    model$method <- list(select = method, crit = crit, pval = pval)
    model
}
```