Flexible Regression: Splines and the Super Learner

Ashley I Naimi

November 2026

Contents

1 Introduction 2

2 Why Linearity Fails 2

3 Splines 5

3.1 What a Spline Is 5

3.2 Piecewise Linear Regression: Spline Precursor 6

3.3 Restricted Quadratic Splines by Hand 7

3.4 Reading a Spline Fit 8

3.5 The Spline Functions in R 9

4 Choosing the Tuning Parameters: The Super Learner 11

4.1 The Question 11

4.2 Cross-Validation and the Discrete Super Learner 11

4.3 The Super Learner 13

4.4 What We Lose: Inference 16

5 Partially Linear Models: Residual-on-Residual Regression 17

5.1 The Model and Robinson's Trick 17

5.2 The Recipe 18

5.3 Assumptions and Limits 21

6 Conclusion 22

1 Introduction

In Week 5 we split the right hand side of a regression model into a target function, coded for interpretation, and a nuisance function, coded for the performance of the target. In many situations, most of the right hand side of your regression equation will be nuisance. This happens a lot in many areas of epidemiology. Furthermore, a large part of the nuisance function will be continuous variables (e.g., confounders) entered following some functional form. This week is about what happens when the functional form is wrong, and about the tools that you can use to make this functional form much more flexible, and "data adaptive", or, how to let your data choose the functional form for you. We cover here specifically splines and the super learner (which is a tool that allows you to choose the "best" functional form, according to some criterion).

What to read: Naimi and Balzer. Stacked generalization: an introduction to super learning. *Eur J Epidemiol* 2018;33:459-464. Naimi, Jin, Yu, Parisi, and Bodnar. Residual-on-residual regression as a tool for effect estimation in observational data. arXiv 2026.

However, using a lot of these tools comes with a price: once you let the data inform chosen the fit, the standard errors we are used to constructing no longer apply. That is, when we use data adaptive methods, for the most part, we lose the ability to intepret standard errors. This is not universally true, since there are some cases that have been studied in the past where we can use data adaptive methods to introduce flexibility in regression models, and still obtain estimators with good performance and valid standard errors (Newey, 1994; Hahn, 1998). However, these tools are very specific (e.g., undersmoothed kernel regression methods with carefully selected tuning parameters), and usually require some advanced knowledge and specification strategies that are not commonly taught in regression modeling classes.

Instead, we will introduce a more general method that (under some important assumptions we'll note) can allow you to fit nuisance models using data adaptive methods, but still get valid inferences for an exposure-outcome association of interest: estimating a partially linear model (in which the exposure effect stays parametric, the nuisance function is nonparametric) using residual-on-residual regression with ordinary robust standard errors.

2 Why Linearity Fails

Write the outcome model as $E(Y \mid X, C) = \beta X + g(C)$: a binary exposure X, a continuous confounder C, and g the confounder-outcome relation on the

additive scale (note, identity link function). Entering C linearly replaces $g(C)$ with γC. Least squares, or maximum likelihood, then returns the best linear approximation it can find (Freedman, 2006).

However, there may be some leftover information in the model we're note capturing. We can write what is leftover mathematically as $g(C) - \gamma^* C$. This leftover information moves into whatever else in the model it is correlated with. And, noting that X depends on C (because C is a confounder), this means that the coefficient for X (the target function) will be contaminated by some of this leftover information.

In other words, the coefficient for X doesn't converge asymptotically to β but to $\beta + \delta$, where δ is the coefficient on X in a regression of the leftover on X and C. This is a formal mathematical representation of residual confounding (it's good to invoke the informal "water pipes" analogy of a DAG here). It is a property of the population or "data generating mechanism", and not of the sample, so it does not shrink as n grows. It is zero only when the true g is, in fact, linear or when X does not depend on C (i.e., no confounding).

We can construct a short simulation to show this. Let's draw C uniformly on $(0, 8)$, let the probability of exposure rise with C, and generate the outcome from a constant exposure effect of 2 plus a strongly curved $g(C)$ (the curve that serves as the week's known truth below) plus standard normal noise. Three models are fit to each sample: the confounder entered linearly, the confounder entered through a natural cubic spline with five degrees of freedom, and the true $g(C)$ entered as a covariate. The latter is a benchmark no analyst has, but we can compare it to see how well/poorly the others perform.

```
g_true <- function(c) {
  5 + 4 * sqrt(9 * c) * (c < 2) + (c >= 2) * abs(c - 6)^2
}


sim_once <- function(n) {
  c <- runif(n, 0, 8)
  x <- rbinom(n, 1, plogis(-2 + 0.5 * c))
  y <- 2 * x + g_true(c) + rnorm(n)
  d <- data.frame(y, x, c, g = g_true(c))
  c(linear = coef(lm(y ~ x + c, data = d))[["x"]],
```

```
    spline = coef(lm(y ~ x + ns(c, df = 5),
                     data = d))[["x"]],
    truth  = coef(lm(y ~ x + g, data = d))[["x"]])
}
```

```
##         n
## model       250   1000   4000
##   linear -0.310 -0.265 -0.279
##   spline -0.017  0.007  0.002
##   truth  -0.014  0.005  0.002
```

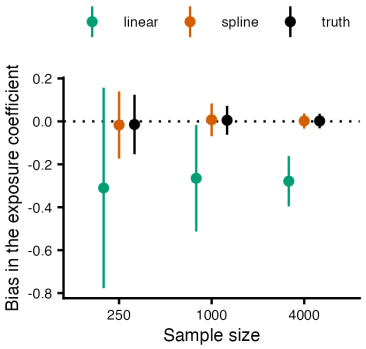


Figure 1: Bias of the exposure coefficient, with one standard deviation either side, over 300 simulated samples at each sample size, when the confounder enters the model linearly, through a five degree of freedom natural spline, or as its true function.

The linear model's bias stays where it is while its standard deviation (the bars in the margin figure) shrinks: the estimator is converging, quickly, to the wrong answer. The spline basically removes this bias (the spline model is biased by a value of 0.002 at $n = 4{,}000$ against the linear model's -0.279), with a standard deviation that nearly matches the benchmark's. This is one scenario involving model misspecification we often encounter: entering a confounder incorrectly in a regression model can leave a degree of confounding bias behind.

The second scenario is that, we may actually be interested in a continuous dose-response function for an exposure of interest. Suppose the continuous variable we used in the example simulation is, instead, the exposure of interest, and our question is about the shape of the dose-response curve itself. The week's known true dose-response is the simulated curve of the super learner handout and the Naimi and Balzer paper (Naimi and Balzer, 2018): a continuous exposure X uniform on $(0, 8)$, and an outcome that rises steeply, peaks near 2, falls to a trough at 6, and rises again (with double-exponential noise as the error source):

```
set.seed(123)
n <- 1000
x <- runif(n, 0, 8)
y <- g_true(x) + rmutil::rlaplace(n)
D <- data.frame(x, y)
```

```r
# a grid of doses, with the truth evaluated on it
xl <- seq(0, 8, 0.1)
Dl <- data.frame(x = xl, truth = g_true(xl))
```

The tools you likely already have for modeling curvilinear relationships in regression are, basically, the inclusion of polynomial terms (Week 5). We can compare some of these here: linear fit and a cubic fit, with the mean squared distance of each from the truth over the grid of doses (i.e., we can compute mean squared error for the whole function, because we know the true dose-response):

```r
fit_lin <- lm(y ~ x, data = D)
fit_cub <- lm(y ~ poly(x, 3, raw = TRUE), data = D)
sapply(list(linear = fit_lin, cubic = fit_cub), function(f)
  round(mean((predict(f, newdata = Dl) - Dl$truth)^2), 2))
```

```
## linear  cubic
##  13.07   2.36
```

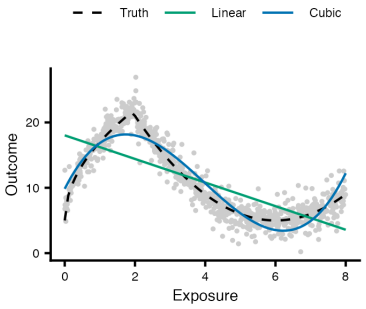


Figure 2: The simulated dose-response data (gray), the truth (dashed), and linear and cubic polynomial fits.

The linear fit (margin figure) reports a dose-response that's very wrong. The cubic does much better, but is still wrong in important ways. One reason this is the case is that, a polynomial is one function for **the whole range**, so every coefficient moves the whole curve. So while the peak is roughly in the right place (though not nearly as sharp), the trough is in the wrong place, and the tails head off on their own. Raising the polynomial degree (i.e., 4, 5, or more) makes the tails worse, not better.

Splines are another approach to curve fitting that can work better in many circumstances. In effect, they keep the polynomial idea, but make it local to a specific region of the function's domain (as we'll see).

3 Splines

3.1 What a Spline Is

The word is borrowed from engineering / architectural drafting. A spline was a thin, flexible strip of wood or metal, pinned by weights at a few fixed points

and left to bend smoothly between them, which was used to draw the curves of hulls, wings, and blueprints (margin figure). In the 1970s and 80s statisticians borrowed the idea for curve fitting: a curve that is a low-order polynomial between a few fixed points, the **knots**, with the pieces joined so that the curve has no corners at the knots.

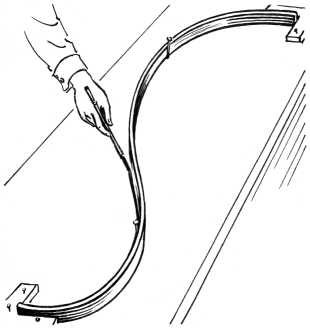


Figure 3: A drafting spline, held in a curve by weights. Source: Wikipedia.

In regression the mathematical tool used to implement splines is a **basis expansion**. The single column x in the design matrix is replaced by (expanded into) several columns (bases) $b_1(x), \ldots, b_K(x)$, each a fixed function of x, so the model becomes $E(Y \mid X) = \beta_0 + \beta_1 b_1(X) + \cdots + \beta_K b_K(X)$. It is still linear in its coefficients, so `lm()` and `glm()` fit it without modification (i.e., everything from Weeks 6 and 7 applies).

Note, a polynomial is also a basis expansion, with columns x, x^2, x^3; what makes a spline basis different is that each of its columns is active over only part of the range of x, which is what makes the fit "local". Natural cubic splines, restricted cubic splines, and B-splines are different recipes for the columns, and we'll compare them briefly in section 3.5.

Let's begin with a precursor to splines, piecewise linear regression.

3.2 Piecewise Linear Regression: Spline Precursor

The simplest way to make a fit local is to fit a different straight line to each part of the domain, and to make the lines meet. Choose a few knots, say at 1, 3, and 6, and add to the linear term one column per knot, $(x - \kappa)_+$, the positive part of $x - \kappa$: zero up to the knot, and rising with slope one after it. The model

$$E(Y \mid X) = \beta_0 + \beta_1 x + \beta_2 (x-1)_+ + \beta_3 (x-3)_+ + \beta_4 (x-6)_+$$

is a straight line with slope β_1 below the first knot, and each later coefficient is the change in slope at its knot: the slope between 1 and 3 is $\beta_1 + \beta_2$, between 3 and 6 it is $\beta_1 + \beta_2 + \beta_3$, and so on. This is an ordinary linear model with four columns (Harrell calls it a linear spline (Harrell, 2001)), so `lm()` fits it as is. Here it is on the week's dose-response data, with its mean squared distance from the truth over the grid of doses, as in the polynomial comparison above:

```
fit_pl <- lm(y ~ x + pmax(x - 1, 0) + pmax(x - 3, 0) +
               pmax(x - 6, 0), data = D)
round(coef(fit_pl), 2)
```

```
##    (Intercept)              x pmax(x - 1, 0)
##           6.90          13.65         -16.80
## pmax(x - 3, 0) pmax(x - 6, 0)
##          -0.59           6.70
```

```
round(mean((predict(fit_pl, newdata = D1) - D1$truth)^2), 2)
```

```
## [1] 2.05
```

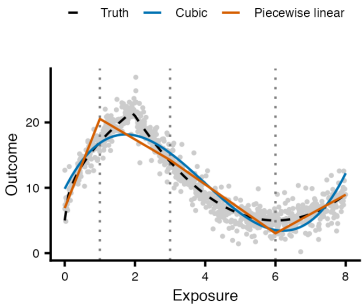


Figure 4: The dose-response data (gray), the truth (dashed), the cubic polynomial of the previous figure, and the piecewise linear fit with knots at 1, 3, and 6 (dotted lines). The four lines meet at the knots, but each meeting is a corner.

Adding up the coefficients gives the four slopes, 13.65 below the first knot, then -3.15, -3.74, and 2.96: a steep rise, a fall, and a rise again, which is the true shape. The four lines beat the cubic polynomial (2.05 against 2.36) because each only has to be right over its own piece. The price is the corners (margin figure): the pieces meet at the knots without a jump, but the slope changes abruptly there, and the true curve has no corners. Splines keep the local idea and remove the corners by replacing the lines with curved pieces that join smoothly, which is where we go next.

3.3 Restricted Quadratic Splines by Hand

Restricted quadratic splines (Howe et al., 2011) use quadratic pieces within "knots". With three knots $\kappa_1 < \kappa_2 < \kappa_3$, the basis adds two columns to the linear term. That is, we can write out the basis expansions for a three knot restricted quadratic spline as:

$$b_1(x) = (x - \kappa_1)_+^2 - (x - \kappa_3)_+^2, \qquad b_2(x) = (x - \kappa_2)_+^2 - (x - \kappa_3)_+^2,$$

where $(u)_+$ is u when $u > 0$ and zero otherwise. Each column switches on at its knot and bends quadratically; subtracting the last knot's term makes the curve linear again beyond κ_3, which is the restriction, and it keeps the tails from flying off where the data are limited. With knots at 1, 3, and 6:

```r
knots <- c(1, 3, 6)
rqs_basis <- function(x, k = knots) {
  pp <- function(u) pmax(u, 0)^2   # positive part, squared
  data.frame(b1 = pp(x - k[1]) - pp(x - k[3]),
             b2 = pp(x - k[2]) - pp(x - k[3]))
}

D  <- cbind(D,  rqs_basis(D$x))
Dl <- cbind(Dl, rqs_basis(Dl$x))

fit_rqs <- lm(y ~ x + b1 + b2, data = D)
round(coef(fit_rqs), 3)
```

```
## (Intercept)           x          b1          b2
##       9.646       7.046      -3.859       5.505
```

Each basis function is zero until its knot, bends upward, and straightens past the last knot. The margin figure continues the piecewise linear picture: the same knots, but the pieces now join without corners. The fit is in the figure below. The linear model splits the difference between the rise and the fall; the spline follows both, with three fixed knots that were chosen before we looked at the outcome. The band is the pointwise 95% interval from `predict(se.fit = TRUE)`, an ordinary standard error from an ordinary linear model: with the basis fixed in advance, the four coefficients are projection parameters in the sense of Week 4, and the model-based or sandwich variance of Week 7 is the right one for them.

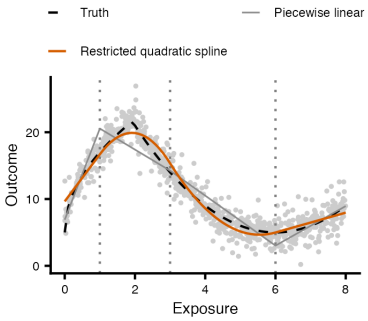


Figure 5: Continuing the previous figure: the same data (gray), truth (dashed), and knots (dotted), with the piecewise linear fit now in gray and the three-knot restricted quadratic spline in its place. The pieces still change at the knots, but they join without corners.

3.4 Reading a Spline Fit

The coefficients printed above have no interpretation worth reporting: they are weights on a linear term and two bent columns. What a spline fit returns is **predictions,** and we can report contrasts of predictions (as we did with the quadratic and cubic terms in Week 5).

For the change from 1 to 2 the spline gives 3.19 with a standard error of 0.11, against a truth of 4; from 2 to 6 it gives -14.89 (0.15) against -16.

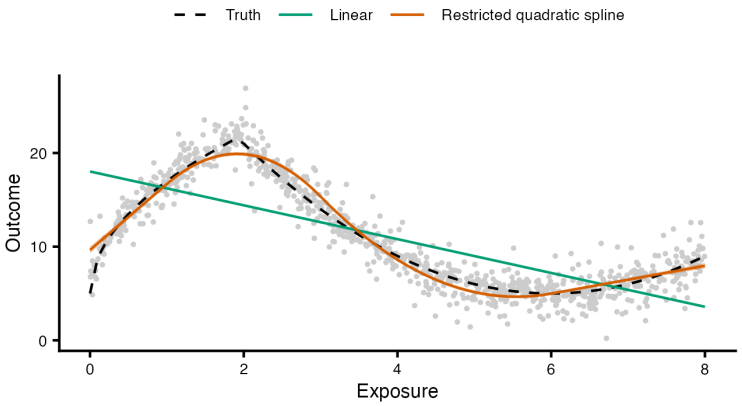


Figure 6: The simulated dose-response data (gray), the truth (dashed), a linear fit, and the three-knot restricted quadratic spline with its pointwise 95% band.

3.5 The Spline Functions in R

In practice we rarely build the spline basis by hand. Natural cubic splines, `ns()`, use cubic pieces, place the interior knots at quantiles of the variable, and force the curve to be linear beyond the boundary knots (Durrleman and Simon, 1989). Frank Harrell's restricted cubic splines, `rms::rcs()` or `Hmisc::rcspline.eval()`, span the same space in a different parametrization, with the knots at fixed quantiles of the variable's distribution by default (Harrell, 2001). B-splines, `bs()`, use polynomial pieces of a chosen degree with no restriction in the tails (Perperoglou et al., 2019). All three are variations on the restricted quadratic spline: different pieces, different columns, but the same idea. Fit to the same data, with the number of columns each adds and the knots `ns()` chose:

```
# Harrell's default positions for five knots
kn <- quantile(D$x, c(.05, .275, .5, .725, .95))

fit_ns  <- lm(y ~ ns(x, df = 4, Boundary.knots = c(0, 8)),
              data = D)
fit_bs  <- lm(y ~ bs(x, df = 6, degree = 3,
                    Boundary.knots = c(0, 8)), data = D)
rcs_x <- function(x) {
  rcspline.eval(x, knots = kn, inclx = TRUE)
}
```

```
fit_rcs <- lm(y ~ rcs_x(x), data = D)

c(ns = ncol(ns(D$x, df = 4)), bs = ncol(bs(D$x, df = 6)),
  rcs = ncol(rcs_x(D$x)))
```

```
##  ns  bs rcs
##   4   6   4
```

```
round(attr(ns(D$x, df = 4), "knots"), 2)
```

```
## [1] 2.03 3.92 5.97
```

Spline	R	Key arguments	Beyond the knots
Restricted quadratic	by hand, as above	the knots; df = knots	linear
Natural cubic	splines::ns()	df (= interior knots + 1) or knots; Boundary.knots	linear
Restricted cubic	rms::rcs(), Hmisc::rcspline.eval()	nk or knots (Harrell's quantiles); df = knots - 1	linear
B-spline	splines::bs()	df (= degree + knots), degree, knots, Boundary.knots	polynomial

The families at a glance. All of them are columns in a design matrix; they differ in what the columns look like, in how many there are for a given `df`, and in how they behave where the data run out.

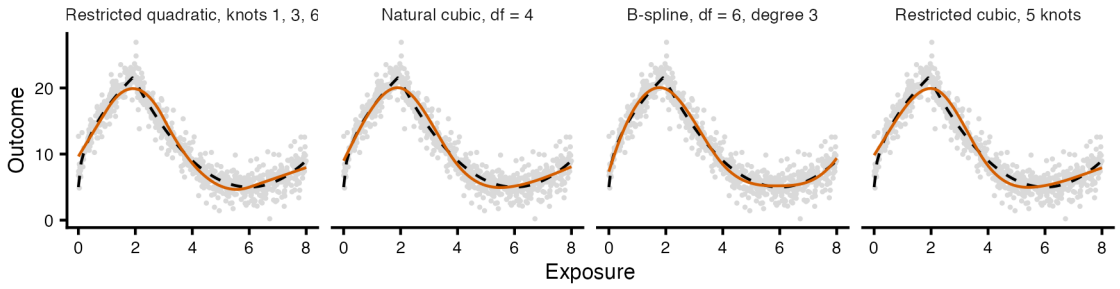


Figure 7: Four spline families fit to the same dose-response data (gray), with the truth dashed: the hand-built restricted quadratic spline, a natural cubic spline, a cubic B-spline, and Harrell's restricted cubic spline.

Two things in the table deserve a second look. What `df` counts differs by family: `ns(x, df = 4)` has four columns and three interior knots at the quartiles, while `bs(x, df = 6, degree = 3)` has six columns and the

same three knots, because the degree uses up three. So `bs(x, df = 3, degree = 3)`, a common sight in applied code, has no interior knots at all: it is the cubic polynomial of section 2 under a spline's name. And the tails differ: the natural and restricted cubic splines are linear beyond the boundary knots, which makes them the usual choice for confounders, whose extremes are sparsely populated, while the B-spline's ends in the figure are starting to curl.

4 Choosing the Tuning Parameters: The Super Learner

4.1 The Question

Every spline in section 3 came with a number we chose: three knots at 1, 3, and 6; `df = 4`; five knots at Harrell's quantiles. How do we know whether `df = 3` is right, or something higher? On the dose-response data, natural splines at nine settings of the degrees of freedom:

```
dfs  <- c(1, 2, 3, 5, 8, 12, 20, 30, 60)
fits <- lapply(dfs, function(d) {
  lm(y ~ ns(x, df = d), data = D)
})
names(fits) <- paste0("df = ", dfs)
pred_grid <- sapply(fits, predict,
                    newdata = data.frame(x = xl))
```

One degree of freedom is the linear fit; three still misses the peak; by twelve the curve is close; at sixty it has started to chase individual points. More flexibility lets the curve follow the truth and the noise more closely at the same time, and less does the reverse: Week 4's bias-variance tradeoff with the dial named. The best setting is a property of the truth and of n, which is why a fixed rule such as "use four knots" cannot be right everywhere, and why we need a way to let the data answer.

4.2 Cross-Validation and the Discrete Super Learner

Write the loss of a prediction as $L(Y, \hat{f}(X)) = \{Y - \hat{f}(X)\}^2$ and its expectation over new data as the **risk**. The average loss in the sample that produced $\hat{f}$ is an optimistic estimate of the risk, because $\hat{f}$ was fit to those same outcomes,

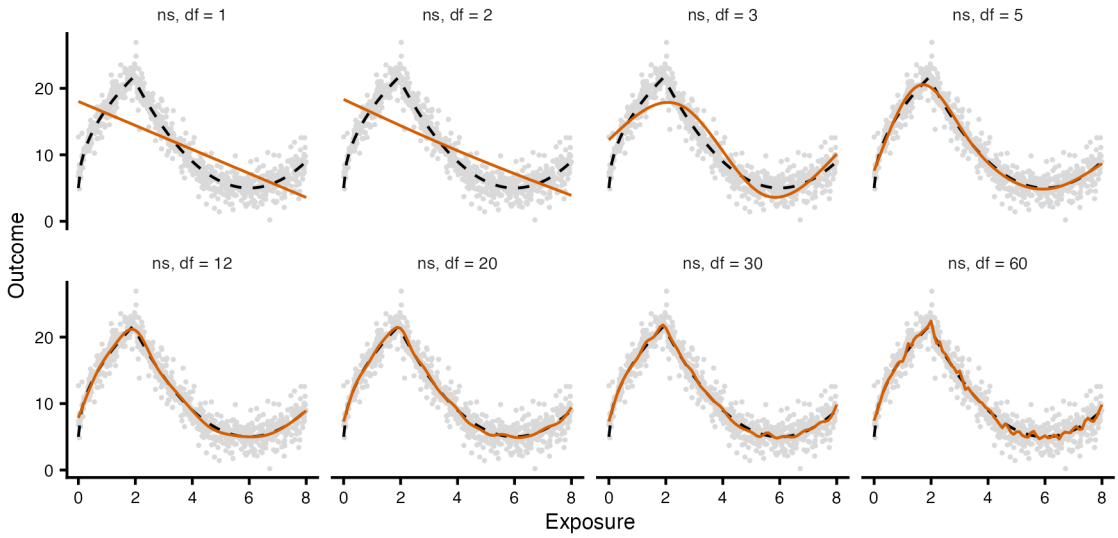


Figure 8: The dose-response data fit with natural cubic splines at eight settings of the degrees of freedom. The dashed line is the truth.

and the optimism grows with the degrees of freedom; in-sample fit cannot choose. V-fold cross-validation removes the optimism by splitting the sample into V folds, fitting on $V-1$ of them, predicting the held-out fold, and averaging the loss over held-out predictions only. We use the five folds of the super learner handout, which assigns every fifth observation to the same fold:

```
V <- 5
index <- split(1:n, 1:V)   # the handout's fold assignment
cv_pred <- matrix(NA, n, length(dfs))
for (v in 1:V) {
  train <- D[-index[[v]], ]
  for (j in seq_along(dfs)) {
    cv_pred[index[[v]], j] <- predict(
      lm(y ~ ns(x, df = dfs[j]), data = train),
      newdata = D[index[[v]], ])
  }
}
cv_risk <- colMeans((D$y - cv_pred)^2)
round(setNames(cv_risk, paste0("df", dfs)), 3)
```

```
##    df1    df2    df3    df5    df8   df12   df20
## 14.277 14.407  4.711  2.101  2.045  2.013  2.018
```

```
##   df30   df60
##  2.026  2.100
```

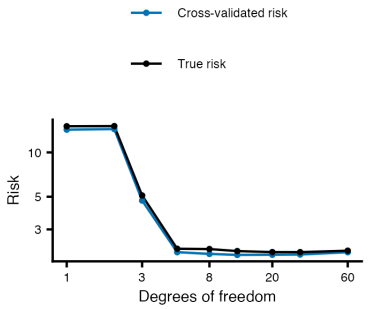


Figure 9: Five-fold cross-validated risk of each natural spline setting on the one observed sample, and the true risk (the noise variance of 2 plus the mean squared error of the curve) from 200 fresh samples. Both axes are logarithmic.

The cross-validated risk is smallest at 12 degrees of freedom, with 20 and 30 a few thousandths behind, and it tracks the true risk of each setting, which we can compute here because the truth is known (margin figure). It is a property of a procedure ("fit a natural spline with 12 degrees of freedom to a sample of this size"), not of the one curve we fitted to all the data, and that is also why it is honest: the held-out observations were never seen by the fit that predicts them. Picking the setting with the smallest cross-validated risk is called the **discrete super learner** (van der Laan et al., 2007; Naimi and Balzer, 2018): cross-validated tuning under a name that will make sense in a moment. The choice of V trades the bias of fitting on a smaller sample against the variance of a small validation set; ten folds is the usual compromise, with more in smaller samples. And the choice is itself an estimate: twenty other random fold assignments on the same data select 12 (7 times) and 20 (13 times), so which observations share a fold decides between two settings whose risks are nearly tied.

4.3 The Super Learner

Rather than choose one setting, we can let the data weight them. Wolpert called the idea stacked generalization: fit several candidate algorithms, then fit a second-level model to their predictions (Wolpert, 1992); Breiman showed that non-negative weights summing to one work best (Breiman, 1996); van der Laan, Polley, and Hubbard supplied the theory and the name: with the weights chosen on cross-validated predictions, the super learner performs asymptotically as well as the best candidate in its library, and often better (van der Laan et al., 2007). The library is a set of candidate procedures, here the nine spline settings, which is the handout's advice in action: to tune an algorithm, "include the same algorithm in the Super Learner library multiple times, with different tuning parameter values for each candidate entry" (Naimi and Balzer, 2018). The handout's five steps, in the language of the last section:

1. Split the sample into V folds.
2. For each fold, fit every candidate to the other folds and predict the held-out fold. The held-out predictions, one column per candidate, are the **level-one**

data.

3. Average the squared error of each column: the cross-validated risk of each candidate. Its minimizer is the discrete super learner.
4. Regress the outcome on the level-one columns by non-negative least squares without an intercept, and normalize the weights to sum to one.
5. Refit every candidate to all the data and combine their predictions with the weights.

Steps 1 to 3 are done: `cv_pred` is the level-one data and `cv_risk` holds its risks. Steps 4 and 5:

```
# Step 4: non-negative least squares on the level-one data
w <- nnls(cv_pred, D$y)$x
alpha <- w / sum(w)
round(setNames(alpha, paste0("df", dfs)), 3)
```

```
##   df1   df2   df3   df5   df8  df12  df20  df30  df60
## 0.000 0.000 0.005 0.081 0.000 0.474 0.088 0.229 0.123
```

```
# Step 5: the full-data fits of each candidate, combined
sl_hand <- as.numeric(pred_grid %*% alpha)
```

The weight is spread over the settings whose risks sat within a few hundredths of the minimum (5, 12, 20, 30, 60 degrees of freedom), most of it on 12, and none on the rest. The `SuperLearner` package does the same thing. To it, a learner is a function that fits and predicts, so we write one: a GLM with a natural spline on every continuous column of `X`. On one column it is `lm(y ~ ns(x, df))`, and `create.Learner()` stamps out one copy per tuning value:

```
SL.nsadd <- function(Y, X, newX, family, df = 3, ...) {
  cont <- sapply(X, function(v) length(unique(v)) > 2)
  sm   <- paste0("ns(", names(X)[cont], ", df = ", df, ")")
  rhs  <- paste(c(sm, names(X)[!cont]), collapse = " + ")
  f    <- as.formula(paste("Y ~", rhs))
  fit  <- glm(f, data = data.frame(Y = Y, X),
              family = family)
```

```
  pred <- predict(fit, newdata = newX, type = "response")
  fit  <- list(object = fit)
  class(fit) <- "SL.nsadd"
  list(pred = pred, fit = fit)
}
predict.SL.nsadd <- function(object, newdata, ...) {
  predict(object$object, newdata = newdata,
          type = "response")
}

ns_learners <- create.Learner("SL.nsadd",
                              tune = list(df = dfs))

fit_sl <- SuperLearner(
  Y = D$y, X = data.frame(x = D$x),
  SL.library = ns_learners$names, method = "method.NNLS",
  cvControl = list(V = V, validRows = index))
out <- rbind(risk = fit_sl$cvRisk, weight = fit_sl$coef)
colnames(out) <- paste0("df", dfs)
round(out, 3)
```

```
##           df1    df2   df3   df5   df8  df12  df20
## risk   14.277 14.407 4.711 2.101 2.045 2.013 2.018
## weight  0.000  0.000 0.005 0.081 0.000 0.474 0.088
##          df30  df60
## risk   2.026 2.100
## weight 0.229 0.123
```

With the same folds (`validRows = index`), the package reproduces the hand calculation's risks and weights to the digit, and its predictions on the grid of doses are identical to ours. The figure below draws the nine candidates in gray, the super learner over them, and the truth. Nothing restricts the library to one family: the sample mean, a linear model, MARS, random forests, and boosting can all join the splines, and a library should always contain the simple candidates so that the super learner can fall back on them when the flexible ones overfit (Phillips et al., 2023). The super learner is a prediction tool,

defined by a loss; it does not know a confounder from an exposure, and the causal structure of the problem remains ours to supply.

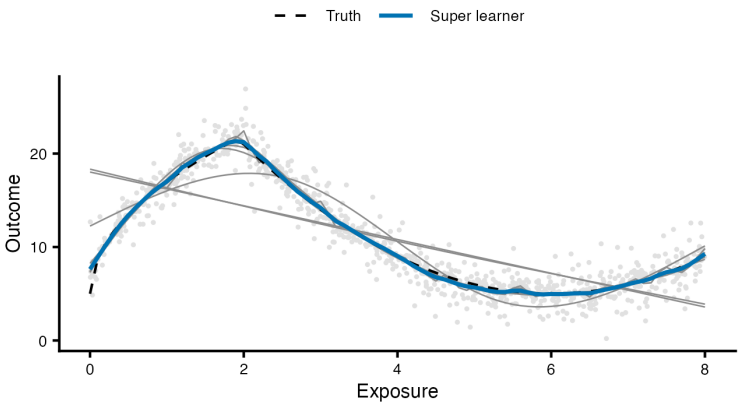


Figure 10: The nine natural spline candidates (gray), the super learner's weighted combination of them, and the truth (dashed).

4.4 What We Lose: Inference

The nuisance function is now as flexible as we like and tuned without our intervention. The price is the standard error, and it is worth being precise about why.

The discrete super learner's choice was an estimate: twenty fold assignments picked 12 (7 times) and 20 (13 times). The standard error that `lm()` prints for the chosen spline is computed as if its `df` had been fixed in advance, so it omits the variability of the choice and ignores that the choice was made to fit these outcomes: the post-selection problem, which Week 10 takes up for penalized regression. For a discrete choice among very different settings the omission is not small.

The super learner's curve has no standard error at all. It is a weighted average of fits whose weights were estimated from the same data, and its accuracy comes from a trade: cross-validation tunes for the smallest mean squared error of prediction, which balances squared bias against variance. An interval of the form estimate $\pm 1.96\times$ SE needs the opposite balance, a bias negligible next to the standard error, so a fit tuned for prediction is tuned wrongly for inference. More generally, a flexible estimate of a regression function converges to the truth more slowly than $\sqrt{n}$, at a rate that worsens with every covariate added (Györfi et al., 2002; Wasserman, 2006), and there is no limiting normal distribution centered on the truth from which to build an interval.

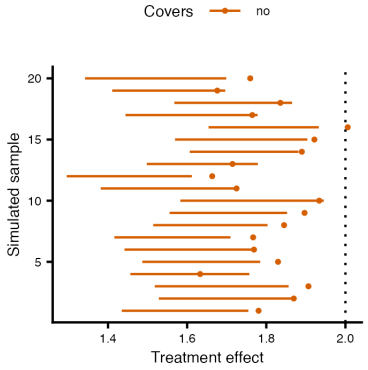


Figure 11: Percentile bootstrap intervals (200 resamples) around a super learner plug-in estimate of a treatment effect of 2 (dotted line), in twenty simulated samples of 1,200 from a design with four confounders observed only through nonlinear transformations (Kang and Schafer 2007; Naimi et al. 2023).

The consequence that matters for us is one step downstream: anything that plugs the super learner in inherits the problem. Put it in as the outcome model of Week 8's marginal standardization, or as the propensity score of inverse probability weighting, and the effect estimate's bias shrinks more slowly than its standard deviation, so an interval built from the standard deviation covers less, not more, as the sample grows. The bootstrap does not escape this, because resampling reproduces an estimator's spread and the problem is its center. The margin figure shows it on a simulation run for these notes, a miniature of the design in Naimi et al. (2023): with a super learner as the outcome model inside marginal standardization, the estimate's bias was -0.20 against a standard deviation of 0.09 at $n = 1{,}200$, and the percentile bootstrap interval covered the truth in 1% of samples at $n = 500$ and 0% at $n = 1{,}200$, although the bootstrap standard deviation itself was accurate. It measured the spread correctly around a center that was wrong. Naimi et al. (2023) and Zivich and Breskin (2021) report the same pattern at full scale.



House rule:
Do not bootstrap a super learner inside a singly robust estimator. A bootstrap interval is a statement about spread; the problem with a flexible plug-in is its center, and no number of resamples moves it. The same holds for inverse probability weighting with a super learner propensity score and for any flexible fit plugged into marginal standardization. To get inference back, change the estimator, not the interval.

5 Partially Linear Models: Residual-on-Residual Regression

5.1 The Model and Robinson's Trick

Keep the nuisance function flexible and the target parametric. The partially linear model writes the outcome as

$$Y = \psi A + g(W) + \varepsilon, \qquad E(\varepsilon \mid A, W) = 0,$$

with A the exposure, W the confounders, and g an unspecified function: linear, polynomial, a spline, a super learner, anything (Robinson, 1988; Naimi et al., 2026). Only the exposure enters linearly, which is what "partially linear" means. For a binary exposure linearity in A is automatic, so the substantive restriction is the absence of exposure-confounder interactions: one constant

effect ψ for everyone. When that holds, and the identification assumptions of Week 8 hold (exchangeability given W, positivity, consistency), ψ is the average treatment effect, the same quantity that marginal standardization targets.

Robinson's observation (Robinson, 1988) is that g can be removed from the model without ever being specified. Take the conditional expectation of both sides given W,

$$E(Y \mid W) = \psi\, E(A \mid W) + g(W),$$

and subtract it from the model:

$$Y - E(Y \mid W) = \psi\, \{A - E(A \mid W)\} + \varepsilon.$$

Nothing about the form of g was used, and g is gone. What remains is a regression through the origin of an outcome residual on an exposure residual, with the same ψ. The two conditional expectations are nuisance functions: $E(Y \mid W)$ is a regression of the outcome on the confounders alone, without the exposure, and $E(A \mid W)$ is the propensity score when A is binary. Robinson showed that if both are estimated nonparametrically, the least squares slope of the one residual on the other is still $\sqrt{n}$-consistent and asymptotically normal, with a standard error that comes from the least squares step (Robinson, 1988; Chernozhukov et al., 2018). The reason is exactly what section 4.4 lacked: the errors of the two nuisance estimates enter the estimate of ψ only through their product, so two moderately accurate flexible fits, each converging faster than $n^{-1/4}$, leave a nuisance error that is negligible on the $\sqrt{n}$ scale. The augmented inverse probability weighted and targeted maximum likelihood estimators of the average treatment effect share this property (Kennedy, 2022); it is called rate double robustness, and residual-on-residual regression is the simplest estimator that has it.

5.2 The Recipe

Four steps and a precaution. First, regress Y on W alone with the super learner and take the residual $\tilde{Y} = Y - \hat{E}(Y \mid W)$. Second, regress A on W with the super learner and take $\tilde{A} = A - \hat{E}(A \mid W)$. Third, regress $\tilde{Y}$ on $\tilde{A}$ by ordinary least squares without an intercept. Fourth, take the standard error from the

HC3 sandwich of Week 7.

The precaution is cross-fitting. A flexible fit predicts the observations it was trained on too well, so in-sample residuals are too small and carry the fit's adaptivity into the least squares step. Instead we split the sample into K folds, fit both nuisance models on $K-1$ folds, predict the held-out fold, and assemble the out-of-fold predictions into residuals for every observation (Chernozhukov et al., 2018; Zivich and Breskin, 2021). `CV.SuperLearner()` does exactly this. Its outer V-fold loop is the cross-fitting layer; inside each training set, the super learner's own cross-validation chooses the weights: two layers of cross-validation (Naimi et al., 2026). Passing the same `validRows` to both calls keeps the two residuals aligned on the same folds. The data are the NHEFS spine of Weeks 5 to 8: weight change as the outcome, quitting smoking as the exposure, and the confounders of Hernán and Robins (Hernán and Robins, 2020). With the spline learner we wrote in section 4 at three settings, the linear model, the mean, and MARS as the library:

```
# the confounders of Weeks 5 to 8, the categorical ones as
# indicator columns
W <- nhefs %>%
  transmute(age, smokeintensity, smokeyrs, wt71, sex, race,
            education = factor(education),
            exercise = factor(exercise),
            active = factor(active))
W <- as.data.frame(model.matrix(~ ., W)[, -1])

add <- create.Learner("SL.nsadd", name_prefix = "SL.add",
                      tune = list(df = c(2, 3, 5)))
lib <- c("SL.mean", "SL.glm", add$names, "SL.earth")

# one fold assignment, shared by the two nuisance fits
K <- 5
N <- nrow(W)
set.seed(2026)
folds <- split(sample(1:N), rep(1:K, length.out = N))
```

```
cv_y <- CV.SuperLearner(
  Y = nhefs$wt82_71, X = W, family = gaussian(),
  SL.library = lib, innerCvControl = list(list(V = 5)),
  cvControl = list(V = K, validRows = folds))
cv_a <- CV.SuperLearner(
  Y = nhefs$qsmk, X = W, family = binomial(),
  SL.library = lib, innerCvControl = list(list(V = 5)),
  cvControl = list(V = K, validRows = folds))

# out-of-fold residuals, then the no-intercept regression
y_til <- nhefs$wt82_71 - cv_y$SL.predict
a_til <- nhefs$qsmk - cv_a$SL.predict
ror <- lm(y_til ~ -1 + a_til)
V_ror <- vcovHC(ror, type = "HC3")
round(coeftest(ror, vcov = V_ror)[1, ], 3)
```

```
##   Estimate Std. Error    t value   Pr(>|t|)
##      3.345      0.471      7.102      0.000
```

The residual-on-residual estimate is 3.34 kg with an HC3 standard error of 0.47, beside the 3.38 (0.48) of the model with every confounder entered linearly. The two agree, which is what we hope to see when the linear model happens to be adequate; the point is that we no longer had to assume it. The weights, averaged over the outer folds, say what the library found: the least flexible spline learner carried 0.69 of the weight in the outcome model and 0.60 in the exposure model, with most of the rest on the mean and the linear model. Robinson proposed the model-based least squares standard error (here 0.44); the sandwich is preferred because the residual variance is rarely constant, and for this estimator it coincides with the influence-function variance used for AIPW and TMLE, so all three sit on the same inferential footing (Naimi et al., 2026; Mansournia et al., 2021). One more honest number: five other fold assignments give estimates from 3.37 to 3.56. Cross-fitting adds a random element of its own, about a sixth of the standard error here, which is why the seed should be fixed and reported (Naimi et al., 2024).

5.3 Assumptions and Limits

What the estimate relies on, in order of importance.

Identification. Exchangeability given W, positivity, and consistency, as for every estimator in Week 8. The super learner adjusts for what it is given; it does not find confounders.

A constant effect. The partially linear model has one ψ. If the effect varies across people, and varies with a covariate that also drives exposure, residual-on-residual regression converges not to the average treatment effect but to a variance-weighted average of individual effects, with weights $\{A-\pi(W)\}^2$ that are largest for people whose propensity score is near one half and smallest for those near the boundaries (Lal and Chou, 2026; Vansteelandt and Sjolander, 2016); the paper's heterogeneity stress test shows the drift growing with the heterogeneity while AIPW and TMLE stay put (Naimi et al., 2026). Effect modification by a few prespecified covariates V can be put back by regressing $\tilde{Y}$ on $\tilde{A}$ and $\tilde{A} \times V$, the g-estimation analogue of an interaction term; for a continuous exposure the model additionally imposes a linear dose-response, the restriction section 2 warned about.

The exposure model. Unlike AIPW and TMLE, which are consistent if either nuisance model is correct, residual-on-residual regression needs $E(A \mid W)$ to be right: if only the outcome model is correct the estimate attenuates toward the null, by the share of the residual exposure variance that is exposure-model error rather than genuine (Naimi et al., 2026). In exchange it is stable where they are not. It residualizes the exposure instead of inverting its probability, so propensity scores near 0 or 1 do not produce extreme weights, and in the paper's positivity simulations its bias and coverage held while those of AIPW and TMLE degraded.

The nuisance rates. The $\sqrt{n}$ argument needs both fits to converge faster than $n^{-1/4}$. A super learner with a sensible library and cross-fitting is the practical way to try to meet that condition, not a guarantee of it; in the paper's simulations at $n = 500$ the HC3 interval covered 92 to 94%, slightly below nominal.

6 Conclusion

This week followed one object, the nuisance function, from the wrong scale to the right estimator. A continuous confounder entered linearly leaves confounding in the exposure coefficient that no sample size removes, and a continuous exposure entered linearly reports a curve that is not there. Splines repair both by adding columns to the design matrix, which is all that natural, restricted cubic, and B-splines are. Cross-validation chooses the degrees of freedom by honest out-of-sample risk, the discrete super learner picks the best setting, and the super learner weights them. What the data choose, the usual standard error no longer covers, and a bootstrap does not restore it.

The partially linear model gets it back, and its division of labor is the lesson of the week. The super learner builds the nuisance functions; residual-on-residual regression, the simplest estimator built so that nuisance errors cancel to first order, turns them into an effect estimate whose standard error comes off the sandwich; a plug-in with a bootstrap is neither. AIPW and TMLE do the same job for the average treatment effect without the constant-effect restriction and with sensitivities of their own; they are beyond this course, but when all three are available, agreement among them is reassuring and disagreement is informative (Naimi et al., 2026).

References

Leo Breiman. Stacked regressions. *Machine Learning*, 24(1):49–64, 1996.

Victor Chernozhukov, Denis Chetverikov, Mert Demirer, Esther Duflo, Christian Hansen, Whitney Newey, and James Robins. Double/debiased machine learning for treatment and structural parameters. *The Econometrics Journal*, 21(1):C1–C68, 2018.

Sylvain Durrleman and Richard Simon. Flexible regression models with cubic splines. *Stat Med*, 8(5):551–561, 1989.

David A Freedman. On the so-called "Huber sandwich estimator" and "robust standard errors". *Am Stat*, 60(4):299–302, 2006.

L Györfi, M Kohler, A Krzyzak, and H Walk. *A Distribution-Free Theory of Nonparametric Regression*. Springer, New York, NY, 2002.

Jinyong Hahn. On the role of the propensity score in efficient semiparametric estimation of average treatment effects. *Econometrica*, 66(2):315–331, 1998.

Frank E. Harrell. *Regression Modeling Strategies: With Applications to Linear Models, Logistic Regression, and Survival Analysis*. Springer, New York, NY, 2001.

M. A. Hernán and JM Robins. *Causal Inference*. Chapman & Hall/CRC, Boca Raton, FL, 2020.

CJ Howe, SR Cole, DJ Westreich, S Greenland, S Napravnik, and JJ Eron. Splines for trend analysis and continuous confounder control. *Epidemiol*, 22(6):874–5, 2011.

Edward H. Kennedy. Semiparametric doubly robust targeted double machine learning: a review. *arXiv:2203.06469 [stat]*, 2022.

Apoorva Lal and Winston Chou. Estimating representative causal effects with double machine learning, 2026. Version 3.

Mohammad Ali Mansournia, Maryam Nazemipour, Ashley I Naimi, Gary S Collins, and Michael J Campbell. Reflection on modern methods: demystifying robust standard errors for epidemiologists. *Int J Epidemiol*, 50(1): 346–351, 2021.

Ashley I Naimi and Laura B Balzer. Stacked generalization: an introduction to super learning. *Eur J Epidemiol*, 33(5):459–464, 2018.

Ashley I Naimi, Alan E Mishler, and Edward H Kennedy. Challenges in obtaining valid causal effect estimates with machine learning algorithms. *Am J Epidemiol*, 192(9):1536–1544, 2023.

Ashley I Naimi, Ya-Hui Yu, and Lisa M Bodnar. Pseudo-random number generator influences on average treatment effect estimates obtained with machine learning. *Epidemiology*, 35(6):779–786, 2024.

Ashley I. Naimi, Qianhui Jin, Ya-Hui Yu, Sara M. Parisi, and Lisa M. Bodnar. Residual-on-Residual Regression as a Tool for Effect Estimation in Observational Data, June 2026.

Whitney K. Newey. Kernel Estimation of Partial Means and a General Variance Estimator. *Econometric Theory*, 10(2):233–253, 1994. ISSN 0266-4666.

Aris Perperoglou, Willi Sauerbrei, Michal Abrahamowicz, and Matthias Schmid. A review of spline function procedures in R. *BMC Med Res Methodol*, 19:46, 2019.

Rachael V Phillips, Mark J van der Laan, Hana Lee, and Susan Gruber. Practical considerations for specifying a super learner. *Int J Epidemiol*, 52(4):1276–1285, 2023.

P. M. Robinson. Root-N-Consistent Semiparametric Regression. *Econometrica*, 56(4):931–954, 1988. ISSN 0012-9682. DOI: 10.2307/1912705.

Mark J van der Laan, Eric C Polley, and Alan E Hubbard. Super learner. *Statistical Applications in Genetics and Molecular Biology*, 6(1):Article 25, 2007.

Stijn Vansteelandt and Arvid Sjolander. Revisiting g-estimation of the effect of a time-varying exposure subject to time-varying confounding. *Epidemiologic Methods*, 5(1):37–56, 2016. DOI: 10.1515/em-2015-0005.

Larry Wasserman. *All of nonparametric statistics*. Springer, New York; London, 2006.

DH Wolpert. Stacked generalization. *Neural Netw*, 5(2):241–59, 1992.

Paul N Zivich and Alexander Breskin. Machine learning for causal inference: on the use of cross-fit estimators. *Epidemiology*, 32(3):393–401, 2021.