Bootstrap and Splines

SYS 6018 | Spring 2022

bootstrap.pdf

Contents

1 Introduction to the Bootstrap

1.1 Required R Packages

We will be using the R packages of:

- `broom` for tidy extraction of model components
- `splines` for working with B-splines
- `tidyverse` for data manipulation and visualization

```
library(broom)
library(splines)
library(tidyverse)
```

1.2 Uncertainty in a test statistic

There is often interest to understand the uncertainty in the estimated value of a test statistic.

- For example, let p be the actual/true proportion of customers who will use your company's coupon.
- To estimate p, you decide to take a sample of $n = 200$ customers and find that $x = 10$ or $\hat{p} = 10/200 = 0.05 = 5\%$ redeemed the coupon.

1.2.1 Confidence Interval

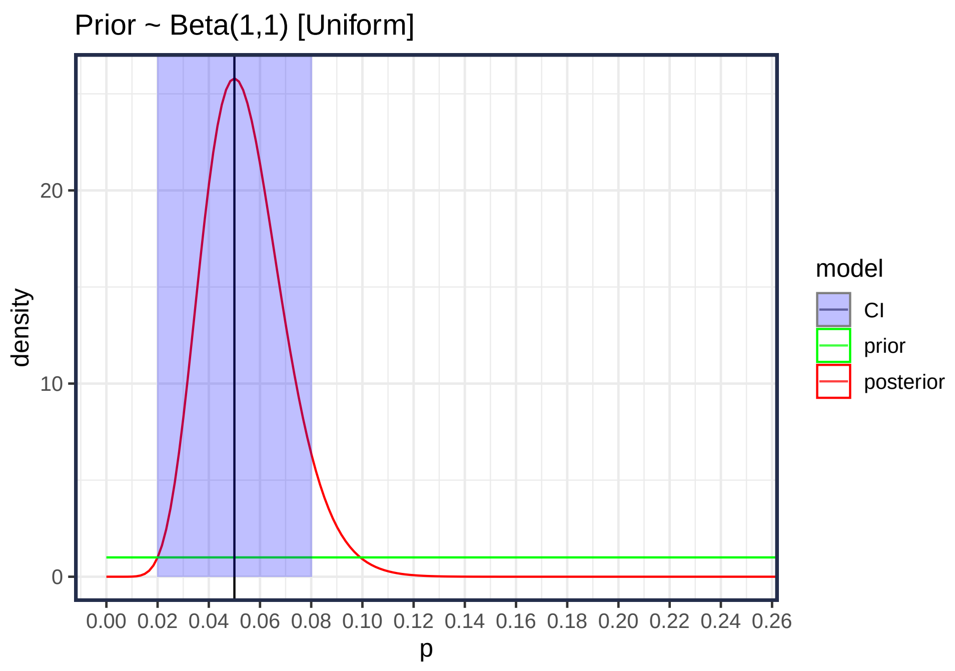
- It is common to calculate the 95% *confidence interval (CI)*

$$\begin{aligned}\text{CI}(p) &= \hat{p} \pm 2 \cdot \text{SE}(\hat{p}) \\ &= \hat{p} \pm 2\sqrt{\frac{\hat{p}(1-\hat{p})}{n}} \\ &= 0.05 \pm 0.03\end{aligned}$$

- This calculation is based on the assumption that $\hat{p}$ is approximately normally distributed with the mean equal to the *unknown* true p, i.e., $\hat{p} \sim N(p, \sqrt{\frac{p(1-p)}{n}})$.

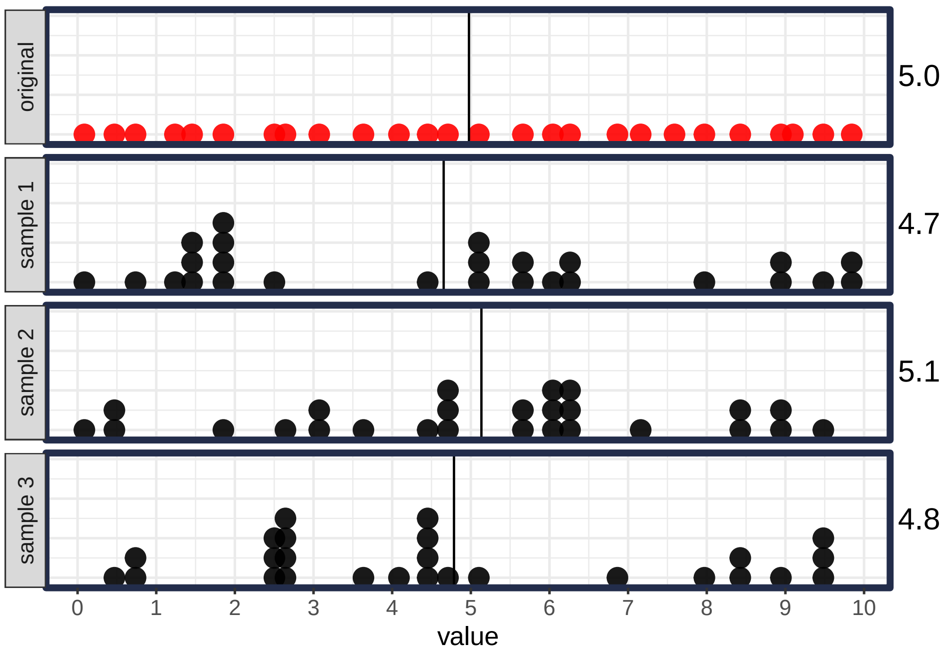
1.2.2 Bayesian Posterior Distribution

In the Bayesian world, you'd probably specify a Beta *prior* for p, i.e., $p \sim \text{Beta}(a, b)$ and calculate the *posterior* distribution $p \mid x = 10 \sim \text{Beta}(a + x, b + n - x)$ which would fully characterize the uncertainty.



1.2.3 The Bootstrap

- The Boostrap is a way to assess the uncertainty in a test statistic using *resampling*.
- The idea is to simulate the data from the *empirical distribution*, which puts a point mass of $1/n$ at each observed data point (i.e., sample the original data **with replacement**).
  - It is important to simulate n observations (same size as original data) because the uncertainty in the test statistic is a function of n



- Then, calculate the test statistic for each bootstrap sample. The variability in the collection of bootstrap test statistics should be similar to the variability in the test statistic.

Algorithm: Nonparametric/Empirical Bootstrap

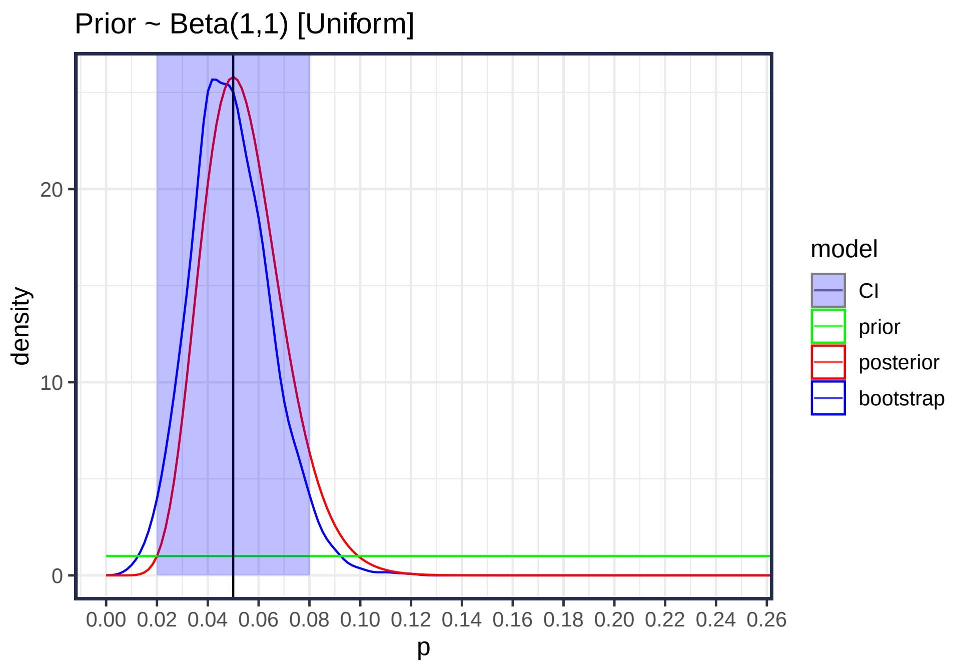
Observe data $D = [X_1, X_2, \ldots, X_n]$ (n observations).
Calculate a test statistic $\hat{\theta} = \hat{\theta}(D)$, which is a function of D.
Repeat steps 1 and 2 M times:

1. Simulate D^*, a new data set of n observations by sampling from D with replacement.
2. Calculate the bootstrap test statistic $\hat{\theta}^* = \hat{\theta}(D^*)$

The bootstrapped samples $\hat{\theta}^*_1, \hat{\theta}^*_2, \ldots, \hat{\theta}^*_M$ can be used to estimate the distribution of $\hat{\theta}$.

- Or properties of the distribution, like standard deviation (standard error), percentiles, etc.

```r
#: Original Data
x = c(rep(1, 10), rep(0, 190))      # 10 successes, 190 failures
n = length(x)                       # length of observed data

#: Bootstrap Distribution
M = 2000                            # number of bootstrap samples
p = numeric(M)                      # initialize vector for test statistic
set.seed(201910)                    # set random seed
for(m in 1:M){
  #- sample from empirical distribution
  ind = sample(n, replace=TRUE)     # sample indices with replacement
  xboot = x[ind]                    # bootstrap sample
  #- calculate proportion of successes
  p[m] = mean(xboot)                # calculate test statistic
}

#: Bootstrap Percentile based confidence Intervals
quantile(p, probs=c(.025, .975))    # 95% bootstrap interval
#>  2.5% 97.5%
#>  0.02  0.08
```



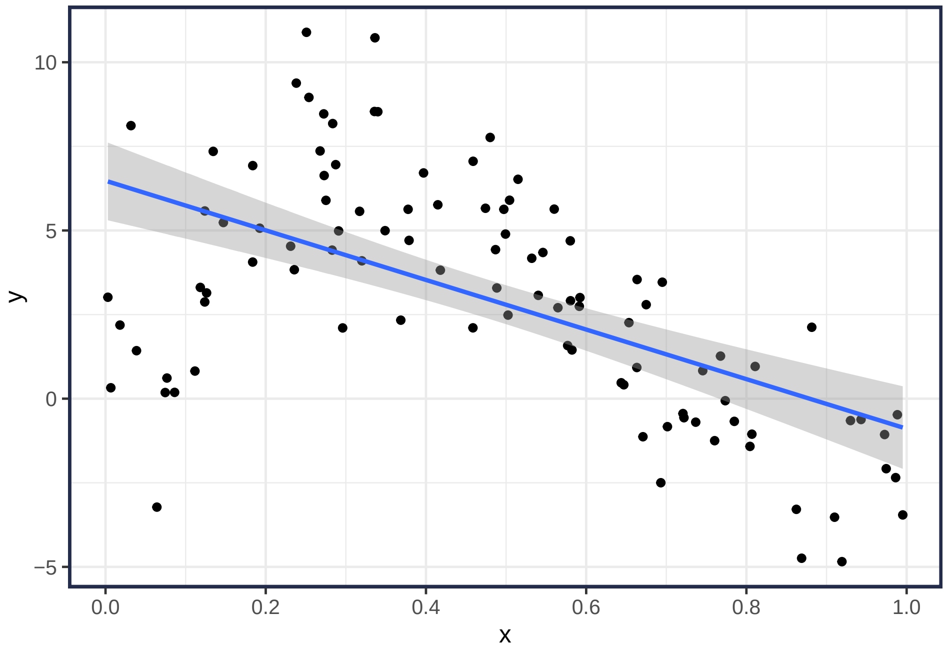
Note

- Notice that in the above example the bootstrap distribution is close to the Bayesian posterior distribution (using the uninformative Uniform prior).
- This is no accident, it turns out there is a close correspondence between the bootstrap derived distribution and the Bayesian posterior distribution under *uninformative priors*
  - See ESL 8.4 for more details

2 Bootstrapping Regression Parameters

The bootstrap is not limited to univariate test statistics. It can be used on multivariate test statistics.

Consider the uncertainty in estimates of the parameters (i.e., β coefficients) of a regression model.



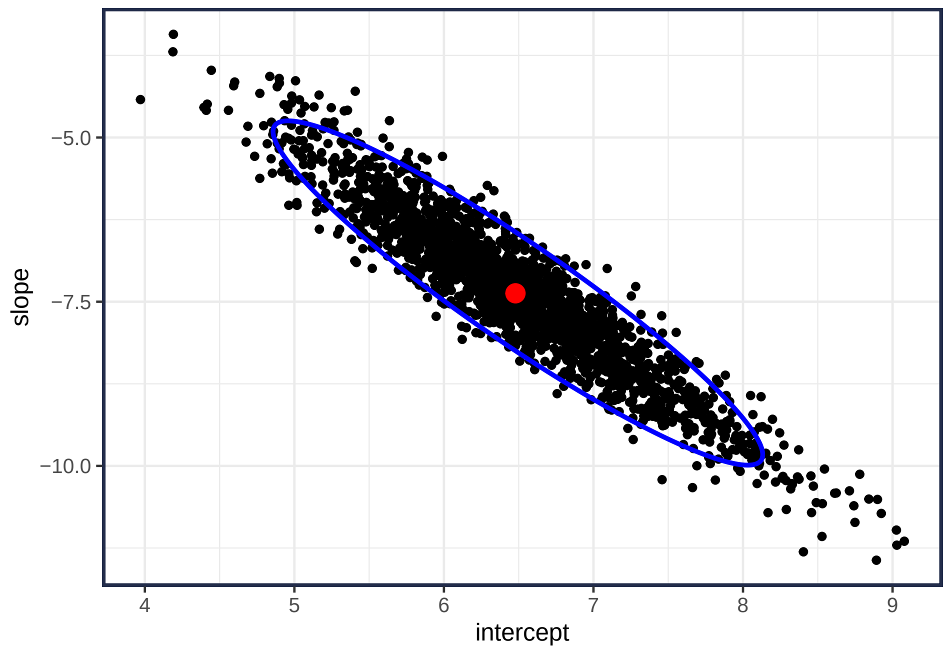
```
m1 = lm(y~x, data=data_train) # fit simple OLS
broom::tidy(m1, conf.int=TRUE)# OLS estimated coefficients
#> # A tibble: 2 x 7
#>   term        estimate std.error statistic  p.value conf.low conf.high
#>   <chr>          <dbl>     <dbl>     <dbl>    <dbl>    <dbl>     <dbl>
#> 1 (Intercept)     6.48     0.584     11.1  5.39e-19     5.32      7.64
#> 2 x              -7.37     1.06      -6.97 3.69e-10    -9.47     -5.27
vcov(m1) %>% round(2)         # variance matrix
#>             (Intercept)     x
#> (Intercept)        0.34 -0.54
#> x                 -0.54  1.12
```

2.1 Bootstrap the β's

```
#-- Bootstrap Distribution
M = 2000                          # number of bootstrap samples
beta = list()                     # initialize list for test statistics
set.seed(201910)                  # set random seed
```

```r
for(m in 1:M){
  #- sample from empirical distribution
  ind = sample(n, replace=TRUE)     # sample indices with replacement
  data.boot = data_train[ind,]      # bootstrap sample
  #- fit regression model
  m.boot = lm(y~x, data=data.boot) # fit simple OLS
  #- save test statistics
  beta[[m]] = broom::tidy(m.boot) %>% select(term, estimate)
}
#- convert to tibble (and add column names)
beta = bind_rows(beta, .id = "iteration") %>%
  pivot_wider(names_from = term, values_from=estimate) %>%
  select(intercept = "(Intercept)", slope = "x", -iteration)
```

```r
#- Plot
ggplot(beta, aes(intercept, slope)) +
  geom_point() +
  geom_point(data=tibble(intercept=coef(m1)[1], slope = coef(m1)[2]),
             color="red", size=4)
```



```r
#- bootstrap estimate
var(beta) %>% round(2)             # varaince matrix
#>           intercept slope
#> intercept      0.58 -0.87
#> slope         -0.87  1.49
apply(beta, 2, sd) %>% round(2)    # standard errors (sqrt of diagonal)
#> intercept     slope
#>      0.76      1.22
```

3 Basis Function Modeling

For a univariate x, a linear basis expansion is

$$\hat{f}(x) = \sum_j \hat{\theta}_j b_j(x)$$

where $b_j(x)$ is the value of the jth basis function at x and θ_j is the coefficient to be estimated.

- The $b_j(x)$ are sometimes specified in advanced (i.e., not estimated). But other approaches use sample data to estimate (e.g., using quantiles for knot placement).
  - Just be sure to estimate everything from the training data so there is no data leakage!

Examples:

- **Linear Regression**

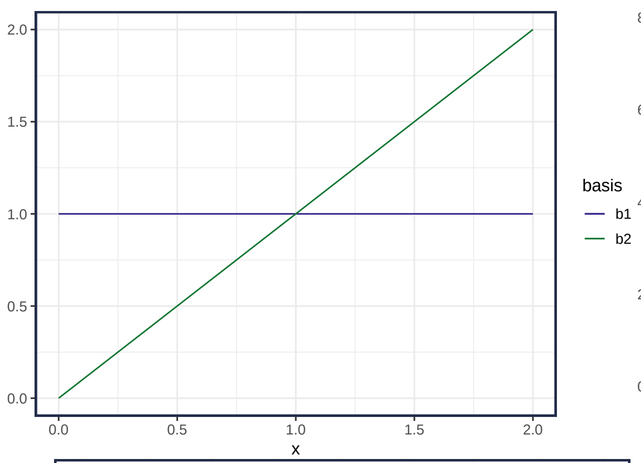
$$\hat{f}(x) = \hat{\beta}_0 + \hat{\beta}_1 x$$

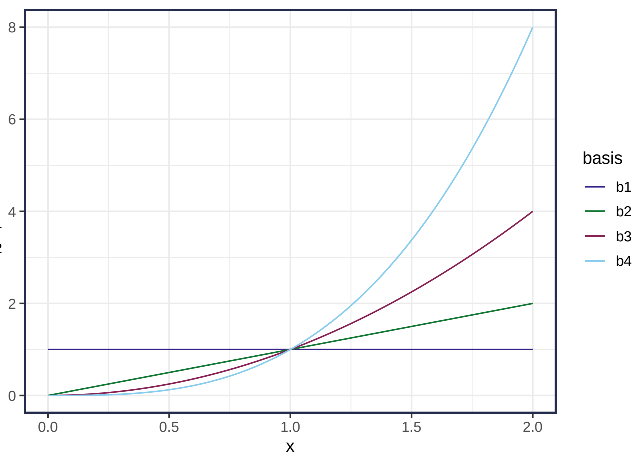
$$b_1(x) = 1$$
$$b_2(x) = x$$

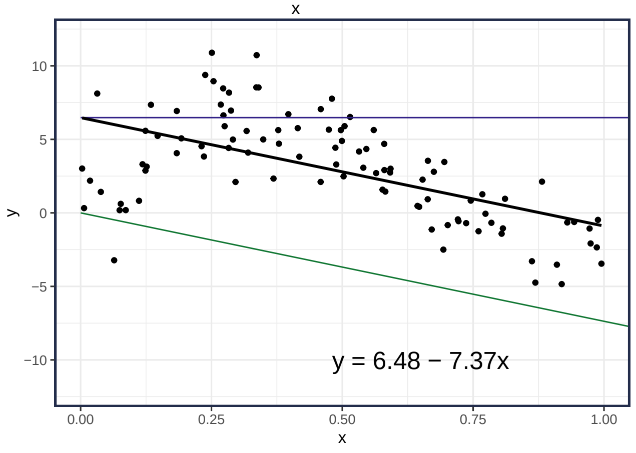
- **Polynomial Regression**

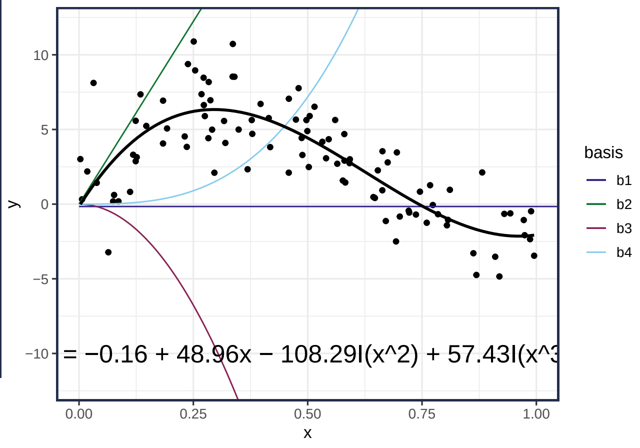
$$\hat{f}(x) = \sum_{j=1}^{d} \hat{\beta}_j x^j$$

$$b_j(x) = x^j$$





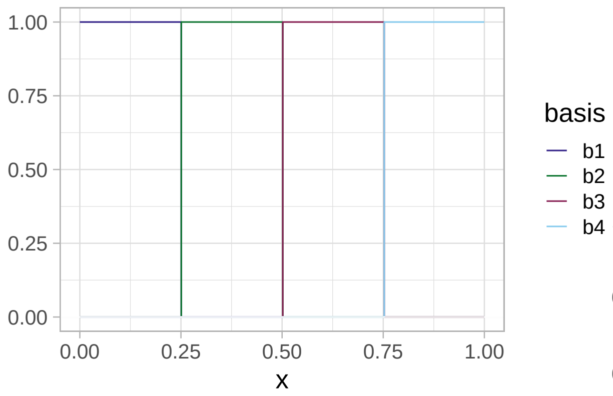


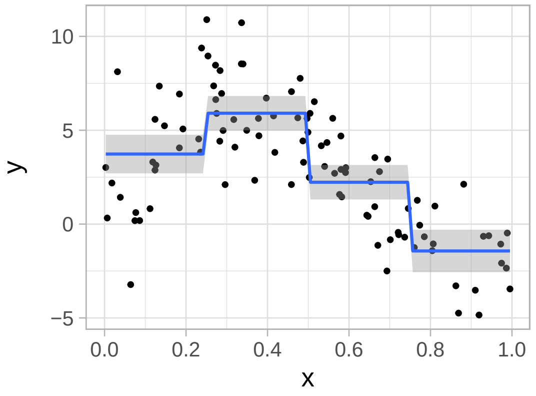


- **Piecewise Constant Regression (Regressogram)**

$$\hat{f}(x) = \sum_{j=1}^{p} \hat{\beta}_j \, \mathbb{1}(x \in R_j)$$

$$\begin{aligned} b_1(x) &= \mathbb{1}(x \in R_1) \\ b_2(x) &= \mathbb{1}(x \in R_2) \\ &\vdots \\ b_p(x) &= \mathbb{1}(x \in R_p) \end{aligned}$$



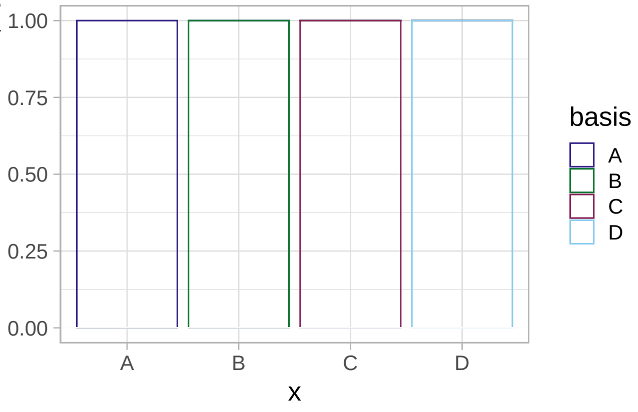


- **Categorical encoding (dummy, one-hot)**

$$x \in \{c_1, c_2, \dots, c_p\}$$

$$\begin{aligned} \hat{f}(x) &= \sum_{j=1}^{p} \hat{\beta}_j \, \mathbb{1}(x = c_j) && \text{one-hot} \\ &= \hat{\beta}_0 + \sum_{j=2}^{p} \hat{\beta}_j \, \mathbb{1}(x = c_j) && \text{dummy} \end{aligned}$$

$$\begin{aligned} b_0(x) &= 1 \\ b_1(x) &= \mathbb{1}(x = c_1) \\ b_2(x) &= \mathbb{1}(x = c_2) \\ &\vdots \\ b_p(x) &= \mathbb{1}(x = c_p) \end{aligned}$$



3.1 Piecewise Polynomials

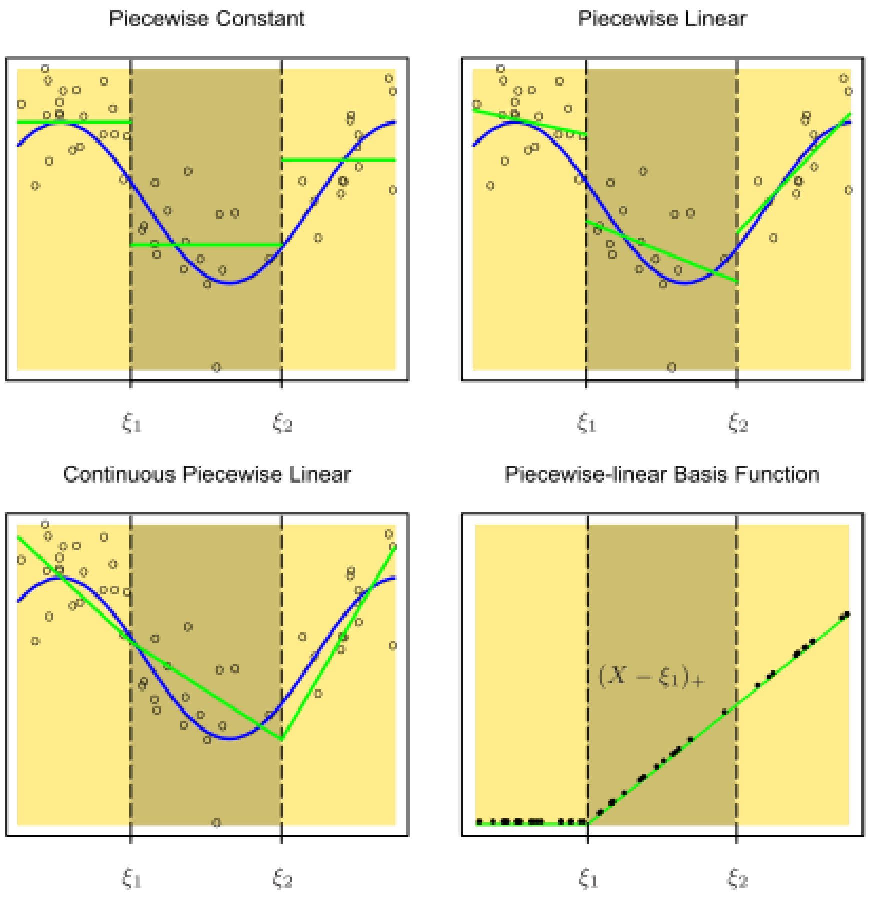


FIGURE 5.1. *The top left panel shows a piecewise constant function fit to some artificial data. The broken vertical lines indicate the positions of the two knots ξ_1 and ξ_2. The blue curve represents the true function, from which the data were generated with Gaussian noise. The remaining two panels show piecewise linear functions fit to the same data—the top right unrestricted, and the lower left restricted to be continuous at the knots. The lower right panel shows a piecewise-linear basis function, $h_3(X) = (X - \xi_1)_+$, continuous at ξ_1. The black points indicate the sample evaluations $h_3(x_i)$, $i = 1, \ldots, N$.*

3.2 B-Splines

- A degree = 0 B-spline is a *regressogram* basis. Will lead to a piecewise constant fit.
- A degree = 3 B-spline (called *cubic* splines) is similar in shape to a Gaussian pdf. But the B-spline has finite support and facilitates quick computation (due to the induced sparseness).

3.2.1 Parameter Estimation

In matrix notation,

$$\begin{aligned}\hat{f}(x) &= \sum_j \hat{\theta}_j b_j(x) \\ &= B\hat{\theta}\end{aligned}$$

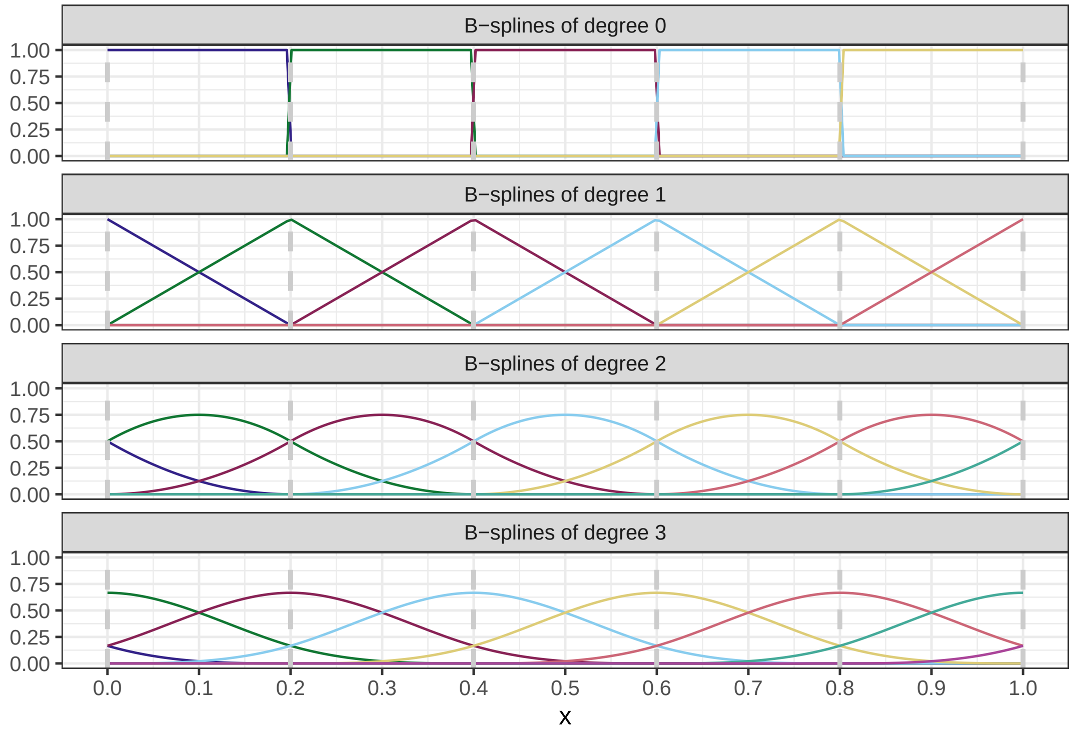


Figure 1: Like ESL Fig 5.20, B-splines (knots shown by vertical dashed lines)

where B is the *basis matrix*.

- For example, a polynomial matrix is

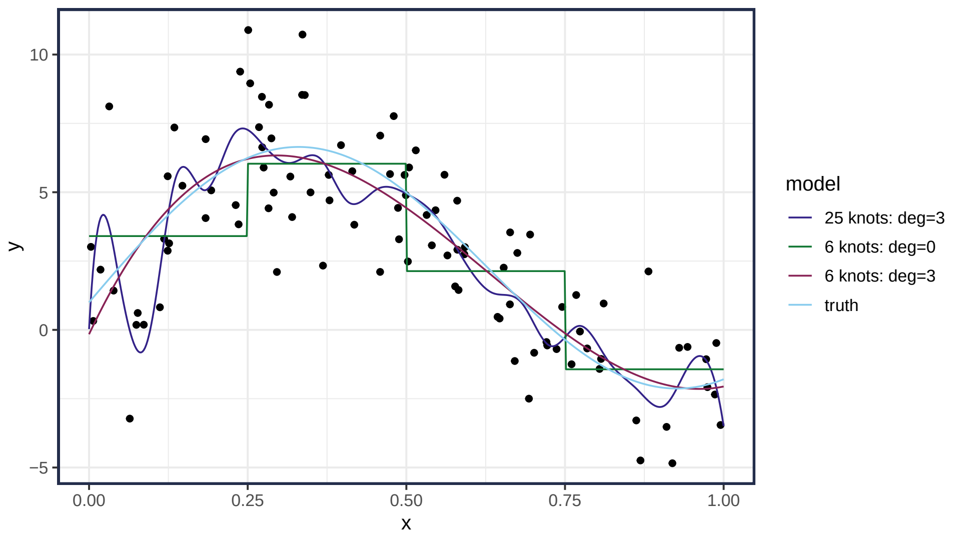
$$B = \begin{bmatrix} 1 & X_1 & X_1^2 & \dots & X_1^J \\ 1 & X_2 & X_2^2 & \dots & X_2^J \\ \vdots & \vdots & \vdots & \vdots & \\ 1 & X_n & X_n^2 & \dots & X_n^J \end{bmatrix}$$

- More generally,

$$B = \begin{bmatrix} b_1(x_1) & b_2(x_1) & \dots & b_J(x_1) \\ b_1(x_2) & b_2(x_2) & \dots & b_J(x_2) \\ \vdots & \vdots & \vdots & \vdots \\ b_1(x_n) & b_2(x_n) & \dots & b_J(x_n) \end{bmatrix}$$

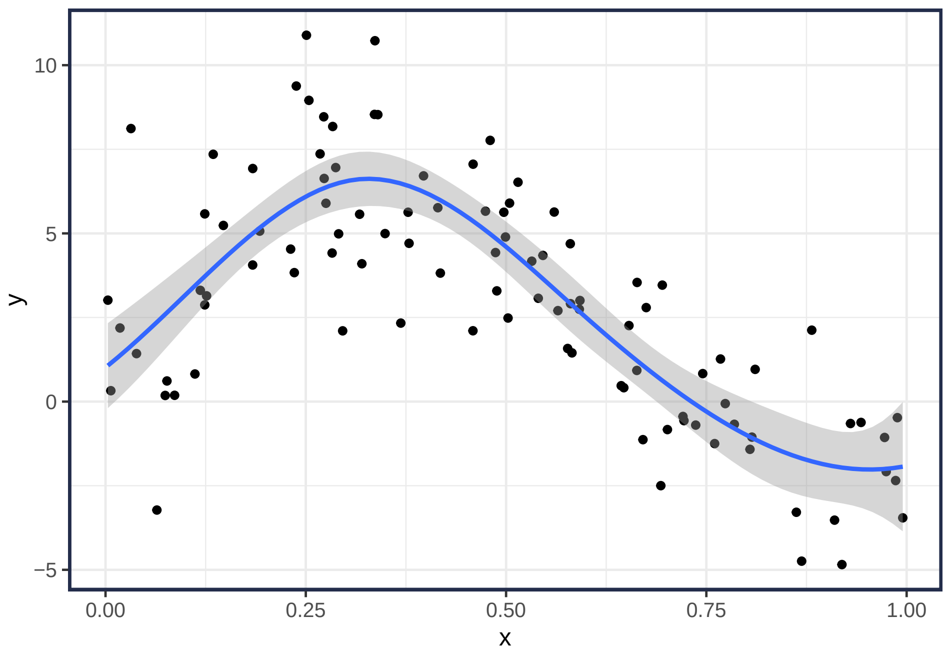
- Now, its in a form just like linear regression! Estimate with OLS

$$\hat{\theta} = (B^\mathsf{T} B)^{-1} B^\mathsf{T} Y$$

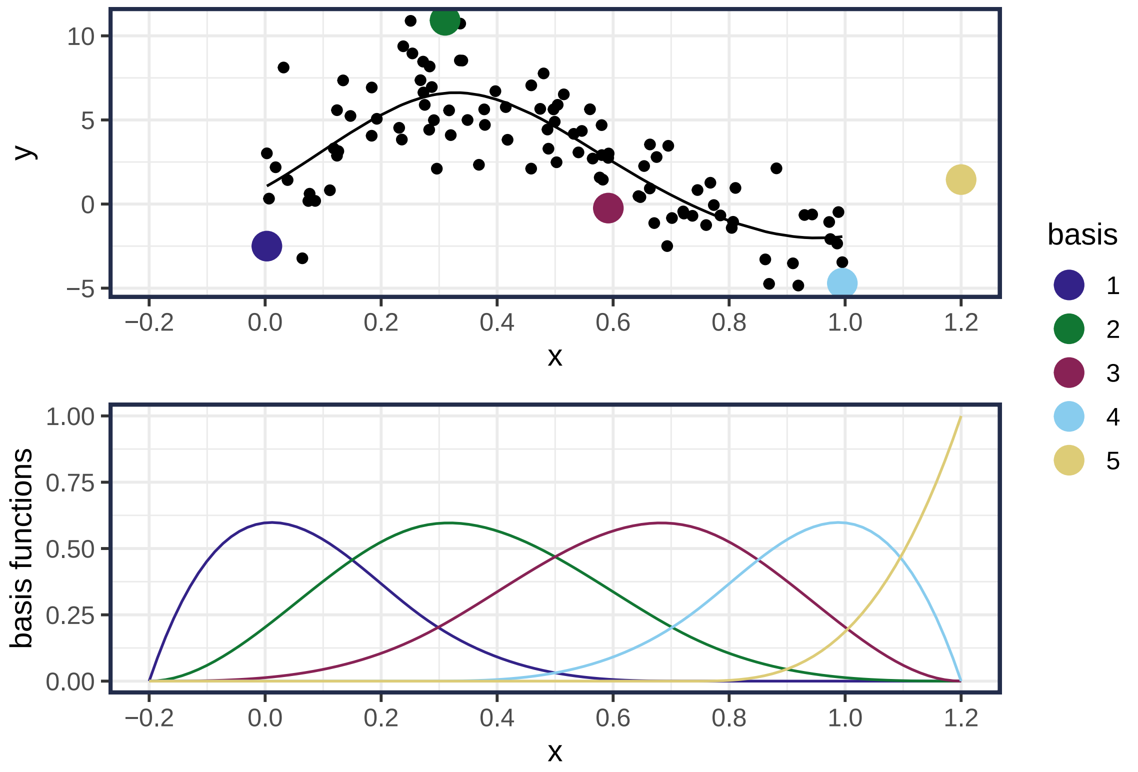


- It may be helpful to think of a basis expansion as similar to a dummy coding for categorical variables.
  - This expands the single variable x into df new variables.
- In R, the function `bs()` can be put directly in formula to make a B-spline.

```
#- fit a 5 df B-spline
# Note: don't need to include an intercept in the lm()
# Note: the boundary.knots are set just a bit outside the range of the data
#       so prediction is possible outside the range (see below for usage).
#       You probably won't need to set this in practice.
kts.bdry = c(-.2, 1.2)
model_bs = lm(y~bs(x, df=5, deg=3, Boundary.knots = kts.bdry)-1,
              data=data_train)
tidy(model_bs)
#> # A tibble: 5 x 5
#>   term                                          estimate std.error statistic  p.value
#>   <chr>                                            <dbl>     <dbl>     <dbl>    <dbl>
#> 1 bs(x, df = 5, deg = 3, Boundary.knots =~   -2.50        1.51    -1.65   1.02e- 1
#> 2 bs(x, df = 5, deg = 3, Boundary.knots =~   10.9         1.27     8.61   1.53e-13
#> 3 bs(x, df = 5, deg = 3, Boundary.knots =~   -0.241       1.53    -0.157 8.76e- 1
#> 4 bs(x, df = 5, deg = 3, Boundary.knots =~   -4.71        3.07    -1.53   1.28e- 1
#> 5 bs(x, df = 5, deg = 3, Boundary.knots =~    1.45        6.90     0.211 8.34e- 1
ggplot(data_train, aes(x,y)) + geom_point() +
  geom_smooth(method='lm', formula='y~bs(x, df=5, deg=3, Boundary.knots = kts.bdry)-1')
```



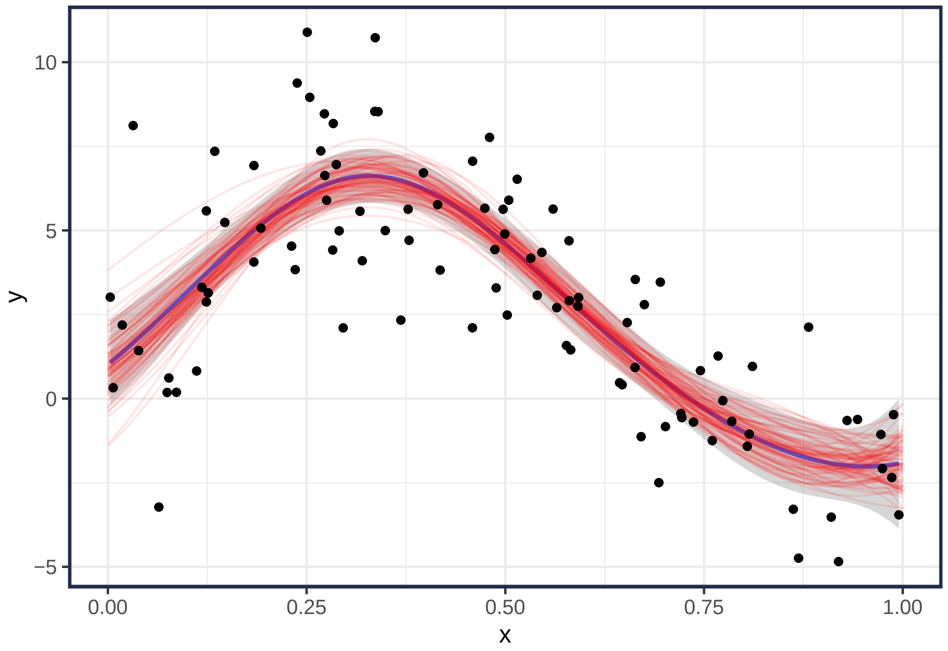
- Setting `df=5` will create a B-spline design matrix with 5 columns
  - So there are 5 basis functions
- The number of (internal) knots is equal to df-degree and at equally spaced quantiles of the data
  - With `df=5` and `deg=3`, there are 2 internal knots at the 33.33% and 66.66% percentiles of x



3.3 Bootstrap Confidence Interval for $f(x)$

Bootstrap can be used to understand the uncertainty in the fitted values

```
#-- Bootstrap CI (Percentile Method)
M = 100                                      # number of bootstrap samples
data_eval = tibble(x=seq(0, 1, length=300)) # evaluation points
YHAT = matrix(NA, nrow(data_eval), M)        # initialize matrix for fitted values

#-- Spline Settings
for(m in 1:M){
  #- sample from empirical distribution
  ind = sample(n, replace=TRUE)                # sample indices with replacement
  #- fit bspline model
  m_boot = lm(y~bs(x, df=5, Boundary.knots=kts.bdry)-1,
              data=data_train[ind,])    # fit bootstrap data
  #- predict from bootstrap model
  YHAT[,m] = predict(m_boot, data_eval)
}

#-- Convert to tibble and plot
data_fit = as_tibble(YHAT) %>%  # convert matrix to tibble
  bind_cols(data_eval) %>%      # add the eval points
  pivot_longer(-x, names_to="simulation", values_to="y") # convert to long format

ggplot(data_train, aes(x,y)) +
  geom_smooth(method='lm',
              formula=as.formula('y~bs(x, df=5, deg=3, Boundary.knots = kts.bdry)-1')) +
  geom_line(data=data_fit, color="red", alpha=.10, aes(group=simulation)) +
  geom_point()
```



```
#-- Calculate Confidence intervals
## for a 90% CI, find the upper and lower 5% values at every x location
## Homework Exercise
```

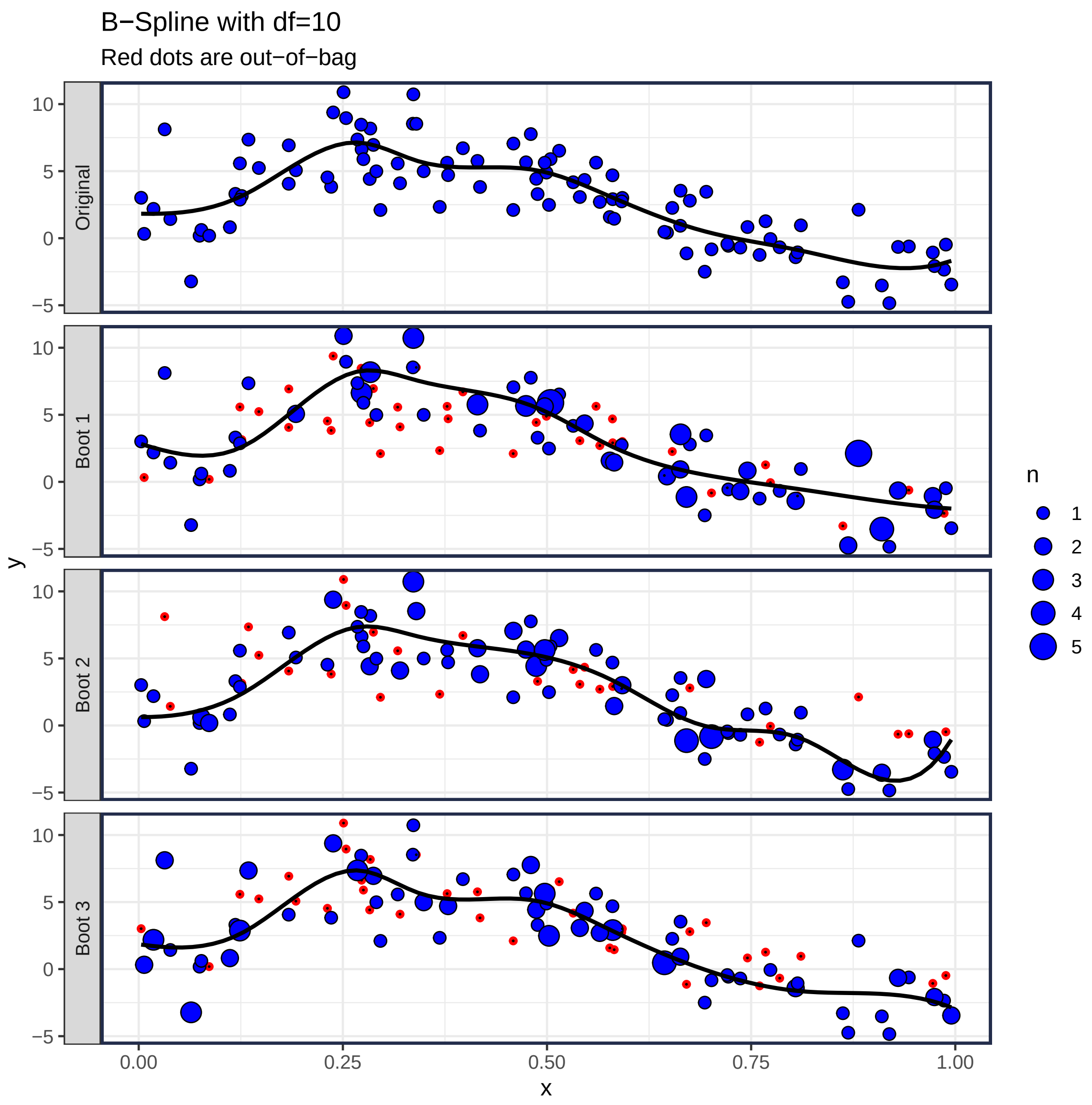
4 More Bagging

4.1 Out-of-Bag Samples

Your Turn #1 : Observations not in bootstrap sample

What is the expected number of observations that will *not* be in a bootstrap sample? Suppose n observations.

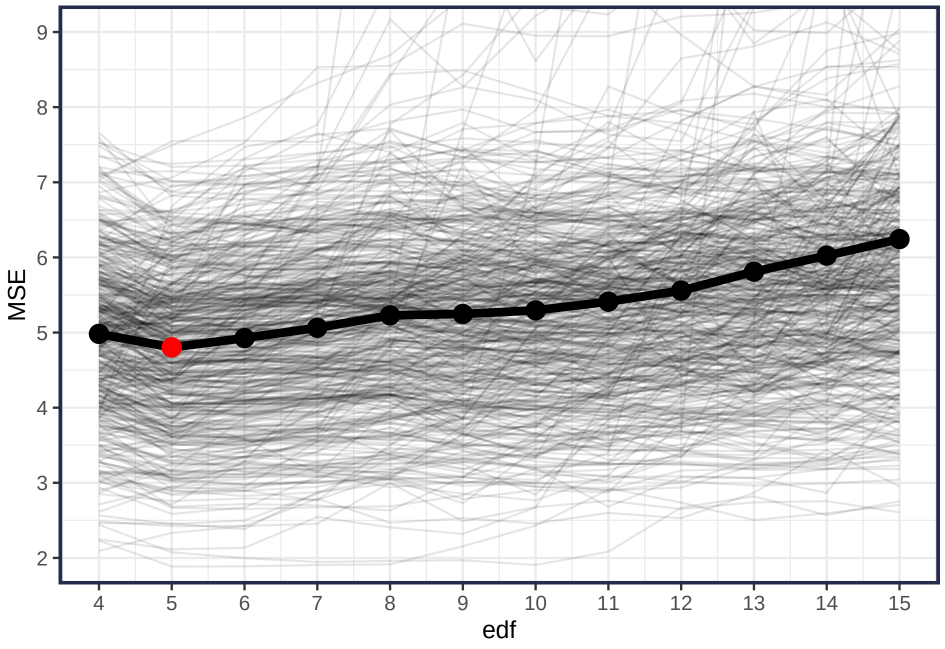
Let’s look at a few bootstrap fits:



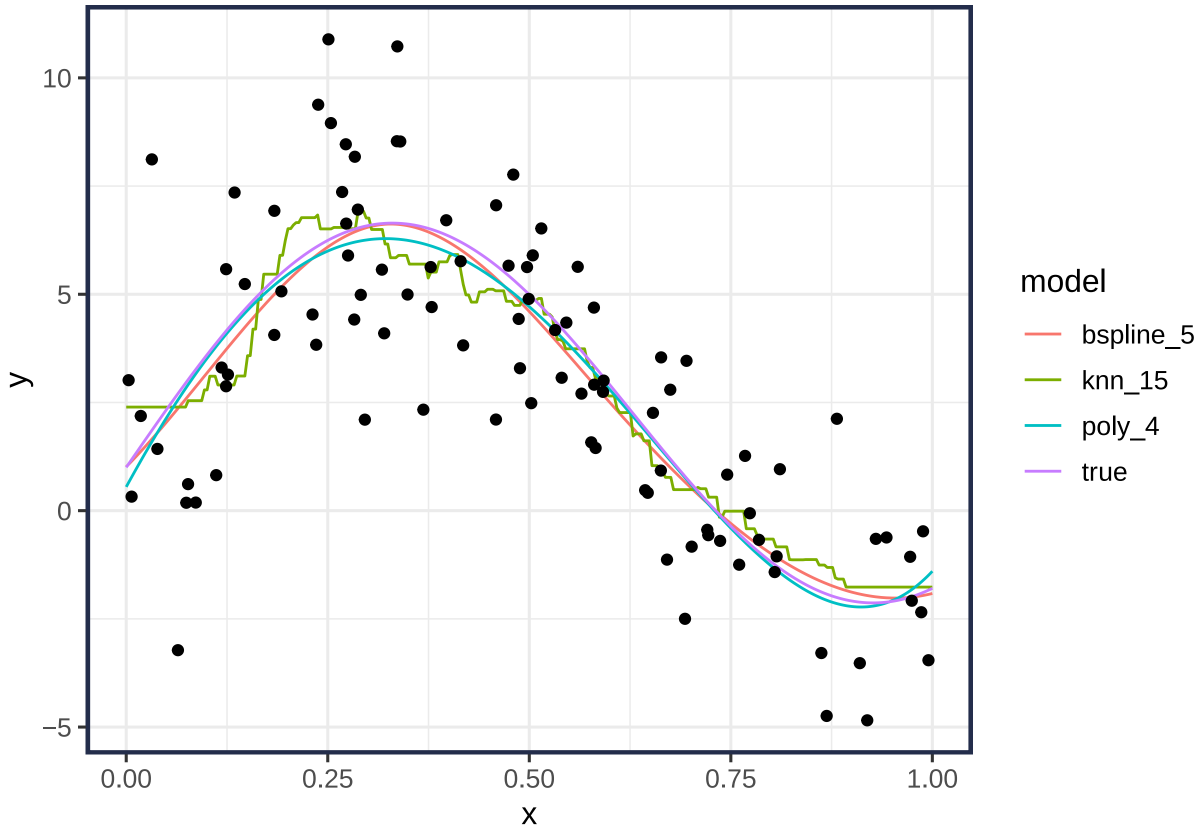
- Notice that each bootstrap sample does not include about 37% of the original observations.
- These are called *out-of-bag* samples and can be used to assess model fit
  - The out-of-bag observations were not used to estimate the model parameters, so will be sensitive to over/under fitting
- Below, we evaluate the oob error over the spline complexity (df = number of estimated coefficients)

```r
M = 500                        # number of bootstrap samples
DF = seq(4, 15, by=1)          # edfs for spline
results = list()               # initialize results list
set.seed(2019)                 # set seed so reproducible

#-- Spline Settings
for(m in 1:M){
  #- sample from empirical distribution
  ind = sample(n, replace=TRUE)        # sample indices with replacement
  oob.ind = setdiff(1:n, ind)          # out-of-bag samples
  #- fit bspline models
  for(df in DF){
    if(length(oob.ind) < 1) next
    #- fit with bootstrap data
    m_boot = lm(y~bs(x, df=df, Boundary.knots=kts.bdry)-1,
                     data=data_train[ind,])
    #- predict on oob data
    yhat.oob = predict(m_boot, data_train[oob.ind, ])
    #- get errors
    sse = sum( (data_train$y[oob.ind] - yhat.oob)^2 )
    n.oob = length(oob.ind)
    #- save results
    results = c(results, list(tibble(m, df, sse, n.oob)))
  }
}
results = bind_rows(results)     # convert from list to tibble
```

```r
avg = results %>% group_by(df) %>% summarize(mse = sum(sse)/sum(n.oob))
plot1 = results %>%
  ggplot(aes(x=df, y=sse/n.oob)) +
  geom_line(aes(group=m), alpha=.10) +
  coord_cartesian(ylim=c(2, 9)) +
  scale_x_continuous(breaks=1:20) + scale_y_continuous(breaks=1:20) +
  labs(x = "edf", y="MSE")

plot1 +
  geom_point(data=avg, aes(df,mse), size=4) + geom_line(data=avg, aes(df,mse), size=2) +
  geom_point(data=avg %>% slice_min(mse), aes(df, mse), color="red", size=4)
```



- The minimum out-of-bag error occurs at `df=5`. This matches the optimal complexity in a polynomial fit from the previous lecture notes.



4.2 Number of Bootstrap Simulations

Hesterberg recommends using $M \geq 15{,}000$ for real applications to remove most of the Monte Carlo variability.

- For the examples in class I used much less to demonstrate the principles.

5 More Resources

- Bootstrap
  - ISL 5.2
  - ESL 7.11
- Splines
  - ISL 7.2-7.5
  - ESL 5.1-5.4
- What Teachers Should Know About the Bootstrap: Resampling in the Undergraduate Statistics Curriculum, by Tim C. Hesterberg
- The `boot` package and `boot()` function provides some more advanced options for bootstrapping
- R's `tidymodels` package
  - Bootstrap resampling and tidy regression models
  - `rsample` for resampling
  - `yardstick` for evaluation metrics
  - `broom` for extracting properties (e.g., estimated parameters) of fitted models in a tidy form

5.1 Variations of the Bootstrap

- We have discussed only one type of bootstrap, *nonparametric/empirical/ordinary* where the observations are resampled
- Another option is to simulate from the *fitted model*. This is called the *parametric* bootstrap.
  - For example, in the regression setting, estimate $\hat{\theta}$ and $\hat{\sigma}$
  - Then given the original X's simulate new $y_i^* \mid x_i \sim f(x_i; \hat{\theta}) + \epsilon(\hat{\sigma})$