T02: Bias-variance trade-off

MATH 4432 Statistical Machine Learning

WANG Zhiwei

MATH, HKUST

2022-09-13

```
#> Want to understand how all the pieces fit together?
#> Read R for Data Science: https://r4ds.had.co.nz/
```

class: inverse, center, middle

Let's start by recalling what we have learned in class!

Bias–variance decomposition of squared error

When fitting a model, we want to minimize

$$\mathbb{E}_{\mathcal{D}}\left[\left(f(X) - \hat{f}\left(X;\mathcal{D}\right)\right)^2\right]$$

w.r.t $\hat{f}$.

If we add and substract $\mathbb{E}_{\mathcal{D}}\left[\hat{f}\left(X;\mathcal{D}\right)\right]$ inside the brackets, we have

$$\begin{aligned} &\mathbb{E}_{\mathcal{D}}\left[\left(f(X) - \hat{f}\left(X;\mathcal{D}\right)\right)^2\right] \\ =&\underbrace{\left(f(X) - \mathbb{E}_{\mathcal{D}}\left[\hat{f}\left(X;\mathcal{D}\right)\right]\right)^2}_{Bias^2} + \underbrace{\mathbb{E}_{\mathcal{D}}\left[\left(\mathbb{E}_{\mathcal{D}}\left[\hat{f}\left(X;\mathcal{D}\right)\right] - \hat{f}\left(X;\mathcal{D}\right)\right)^2\right]}_{Variance} \end{aligned}$$

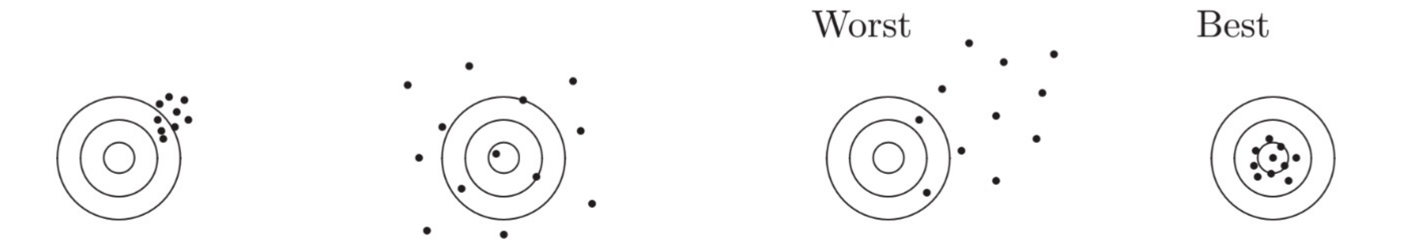
Two sources of error

- **Bias** from erroneous assumptions in the learning algorithm.

  High bias $\longrightarrow$ **underfitting**.

- **Variance** from sensitivity to small fluctuations in the training set.

  High variance $\longrightarrow$ **overfitting**.



(a) High bias, Low variance (high precision)

(b) Low bias, High variance (low precision)

(c) High bias, High variance (low precision)

(d) Low bias Low variance (high precision)

A toy example

Problem setting

Setting | Code | Plot | Fit the model | Code | Plot

- Suppose we know the ground truth of $f(\cdot) : f(x) = \sin(2\pi x)$
- Now given $\{x_n\}_{n=1}^N$, we have a set of observations $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^N$ with

$$y_n = f(x_n) + \epsilon_n,$$

where $\epsilon_n \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, 1)$ is random noise.

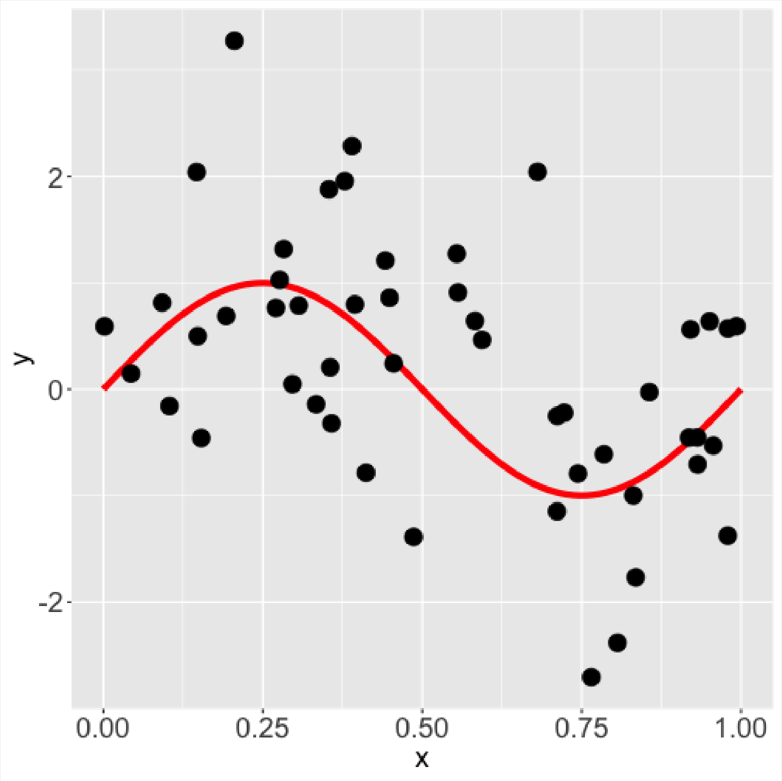
Problem setting

Setting | Code | Plot | Fit the model | Code | Plot

```
set.seed(20220913)

# Ground truth
x <- seq(0, 1, length.out = 100)
y <- sin(2 * pi * x)
# Observed data
N <- 50 # Sample size
X <- runif(N, 0, 1)
y0 <- sin(2 * pi * X)
y_obs <- y0 + rnorm(N, mean = 0, sd = 1) # Add noise

ggplot(data = NULL) +
  geom_line(aes(x = x, y = y), color = "red", size = 2) +
  geom_point(aes(x = X, y = y_obs), size = 5) +
  theme(
    text = element_text(size = 18),
    axis.text.y = element_text(size = 18),
    axis.text.x = element_text(size = 18)
    )
```

Problem setting

Setting | Code | Plot | Fit the model | Code | Plot



Problem setting

Setting | Code | Plot | **Fit the model** | Code | Plot

For a given $\mathcal{D}$, we can fit a model $\hat{f}\left(X;\mathcal{D}\right)$, e.g., linear regression.

Problem setting

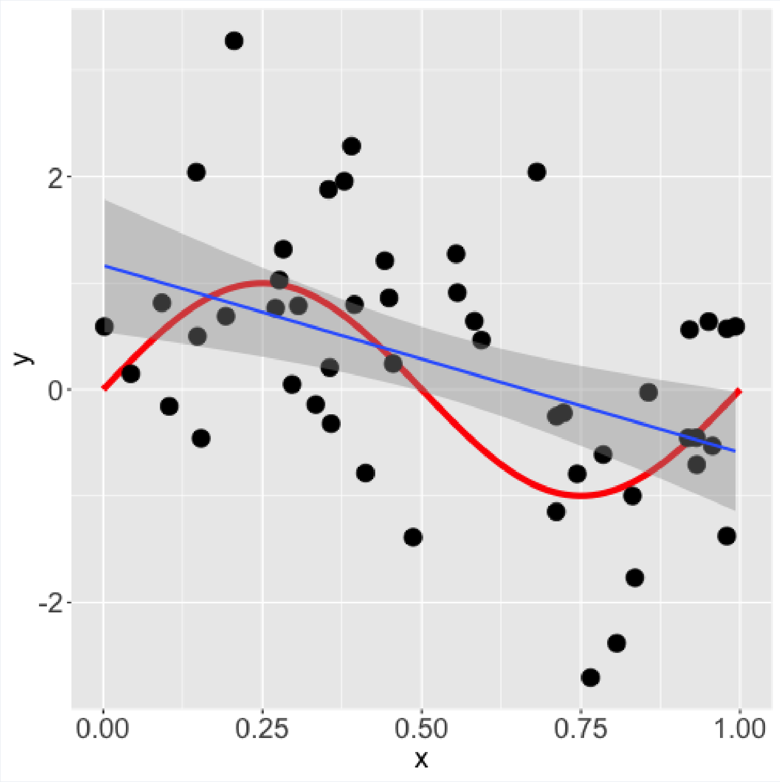
Setting Code Plot Fit the model Code Plot

```
set.seed(20220913)

ggplot(data = NULL) +
  geom_line(aes(x = x, y = y), color = "red", size = 2) +
  geom_point(aes(x = X, y = y_obs), size = 5) +
  geom_smooth(aes(x = X, y = y_obs), method = "lm") + # Linear regressi
  theme(legend.position = "none") +
  theme(
    text = element_text(size = 18),
    axis.text.y = element_text(size = 18),
    axis.text.x = element_text(size = 18)
    )
```

Problem setting

Setting | Code | Plot | Fit the model | Code | Plot

```
#> `geom_smooth()` using formula = 'y ~ x'
```



Smoothing spline [*]

Assume $f(\cdot)$ is some unknown **smooth** function, to estimate $f(\cdot)$, a smoothing spline minimizes the penalized least squares functional

$$f_\lambda = \min_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - f\left(x_i\right)\right)^2 + \lambda J_m(f),$$

where $J_m(f) = \int \left| f^{(m)}(z) \right|^2 dz$ is a penalty term that quantifies the lack of parsimony of the function estimate, and $\lambda > 0$ is the smoothing parameter that controls the influence of the penalty.

Note that $f^{(m)}(\cdot)$ denotes the m-th derivative of $f(\cdot)$, and $\mathcal{H} = \{f : J_m(f) < \infty\}$ is the space of functions with square integrable m-th derivative.

[*] See Wikipedia or Smoothing Spline Regression in R for more details if you are interested. However, this is not the point of this course!

Smoothing parameter

Smoothing parameter influence | Code | Plot

- As $\lambda \rightarrow 0$ the penalty has less influence on the penalized least squares functional. So, for very small values of λ, the function estimate f_λ essentially minimizes the residual sum of squares.

- As $\lambda \rightarrow \infty$ the penalty has more influence on the penalized least squares functional. So, for very large values of λ, the function estimate f_λ is essentially constrained to have a zero penalty, i.e., $J_m\left(f_\lambda\right) \approx 0$.

- As λ increases from 0 to ∞, the function estimate f_λ is forced to be smoother with respect to the penalty functional $J_m(\cdot)$. The goal is to find the λ that produces the "correct" degree of smoothness for the function estimate.

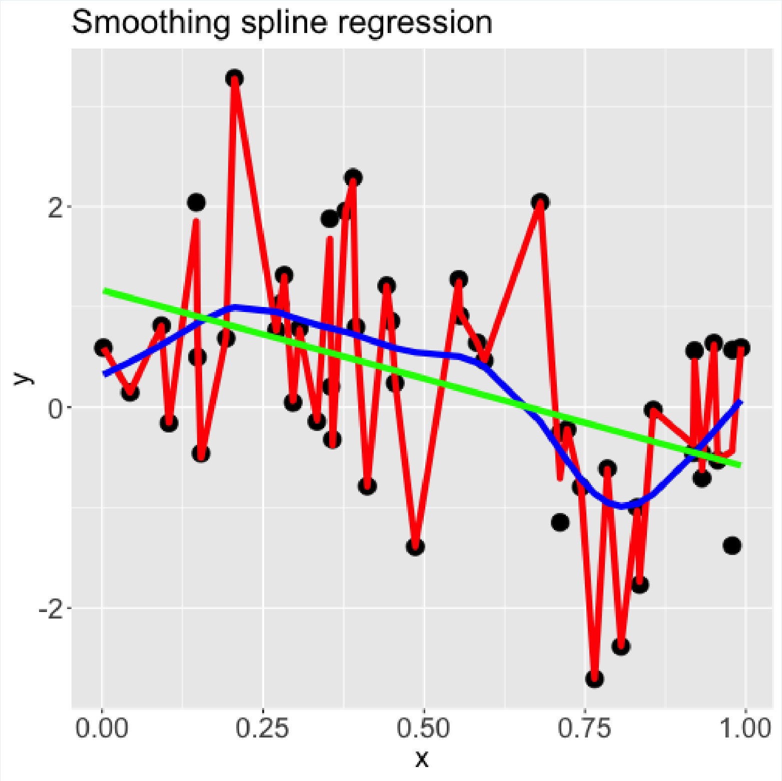
Smoothing parameter

Smoothing parameter influence | Code | Plot

```
library(ggformula)

ggplot(data = NULL, aes(x = X, y = y_obs)) +
  geom_point(size = 5) +
  geom_spline(aes(x = X, y = y_obs), spar = 1e-2, colour = "red", size =
  geom_spline(aes(x = X, y = y_obs), cv = TRUE, colour = "blue", size =
  geom_spline(aes(x = X, y = y_obs), spar = 2, colour = "green", size =
  xlab("x") +
  ylab("y") +
  ggtitle("Smoothing spline regression") +
  theme(
    text = element_text(size = 18),
    axis.text.y = element_text(size = 18),
    axis.text.x = element_text(size = 18)
    )
```

Smoothing parameter

Smoothing parameter influence | Code | **Plot**



How do we evaluate the bias-variance trade-off in this example?

We need $\mathbb{E}_{\mathcal{D}}(\hat{f}(X;\mathcal{D}))$, therefore, we need to have multiple datasets.

- For $l = 1, \ldots, L$,
  - generate $\mathcal{D}^l = \{(x_n, y_n^l)\}_{n=1}^N$, where $y_n^l = f(x_n) + \epsilon_n^l$
  - fit the l-th model $\hat{f}(X^l;\mathcal{D}^l)$ and denote the predicted value as $y^l(x_n) = \hat{f}(x_n;\mathcal{D}^l)$

  Note that $f(x_n)$ is fixed across l while the observed values y_n^l are varying due to random noise ϵ_n^l.

- Estimate $\mathbb{E}_{\mathcal{D}}[\hat{f}(X;\mathcal{D})]$ by $\bar{y}(x) = \frac{1}{L}\sum_{l=1}^L y^l(x)$
- Compute squared bias: $\frac{1}{N}\sum_{n=1}^N \left(\bar{y}(x_n) - f(x_n)\right)^2$
- Compute variance: $\frac{1}{N}\sum_{n=1}^N \frac{1}{L}\sum_{l=1}^L \left(y^l(x_n) - \bar{y}(x_n)\right)^2$

Experiments

Experiments setting | Implementation

We take the above example with $N = 20$, $L = 500$ and use smoothing spline regression with $\lambda \in [1 \times 10^{-6}, 10]$.

```
set.seed(20220913)

trial <- 500 # Number of experiment trials
N <- 20 # Number of samples for each trial
lambda_list <- exp(seq(log(1e-6), log(10), length.out = 100)) # Paramete
model_list <- list() # Model list
biasSQ <- variance <- vector(mode = "numeric", length = length(lambda_l

X <- runif(N, 0, 1) # Predictor
y0 <- sin(2 * pi * X) # True values of y
y_mat = matrix(0, nrow = N, ncol = trial) # Store the generated response
for(j in 1 : trial){
  y_mat[, j] = y0 + rnorm(N, mean = 0, sd = 1) # Add noise; each column
}
```

Experiments

Experiments setting | Implementation

```r
for(i in 1 : length(lambda_list)){
  model_list_i <- list()
  y_hat <- matrix(0, nrow = N, ncol = trial) # Predicted values

  for(j in 1 : trial){
    y <- y_mat[, j]
    fit_ss <- smooth.spline(x = X, y = y, lambda = lambda_list[i]) # Smo
    model_list_i <- c(model_list_i, list(fit_ss)) # Save the model for t
    y_hat[, j] <- predict(fit_ss, X)$y # Predicted values
  }

  model_list <- c(model_list, list(model_list_i)) # Save the model list
  y_bar <- rowMeans(y_hat) # Mean of predicted values, E(f^hat)
  biasSQ[i] <- mean((y0 - y_bar)^2)  # Bias square, E[ (f - E(f^hat))^2
  variance[i] <- mean((y_hat - y_bar)^2)   # Variance, E[ (E(f^hat) - f
  }
```

Visualization

Bias and variance | Code | Plot

Let's first take a look at how the two sources of error change as the parameter λ changes.

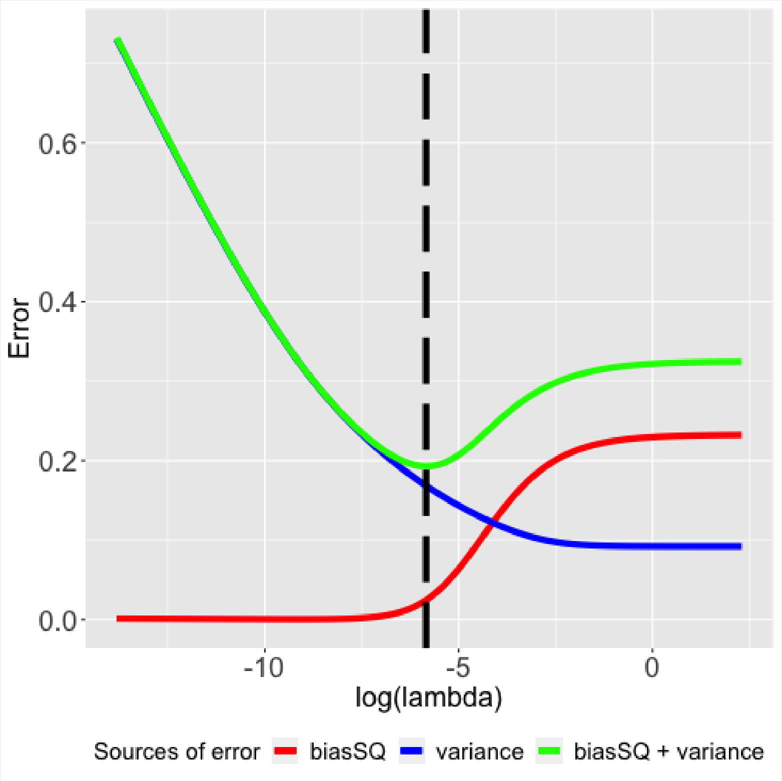
Visualization

Bias and variance | Code | Plot

```
ggplot(data = NULL) +
  geom_line(aes(x = log(lambda_list), y = biasSQ, color = "biasSQ"), si
  geom_line(aes(x = log(lambda_list), y = variance, color = "variance")
  geom_line(aes(x = log(lambda_list), y = biasSQ + variance, color = "b
  geom_vline(xintercept = log(lambda_list)[which.min(biasSQ + variance)]
  xlab("log(lambda)") +
  ylab("Error") +
  scale_color_manual(name = "Sources of error",
                     breaks = c("biasSQ", "variance", "biasSQ + variance
                     values = c("biasSQ" = "red", "variance" = "blue", "
  theme(
    text = element_text(size = 18),
    axis.text.y = element_text(size = 18),
    axis.text.x = element_text(size = 18),
    legend.title = element_text(size = 15),
    legend.text = element_text(size = 15),
    legend.position = "bottom"
    )
```

Visualization

Bias and variance | Code | Plot



More details

More details | Code | Plot

Then we chose three different values for the parameter λ (too small, suitable, too large) and visualize for more performance details.

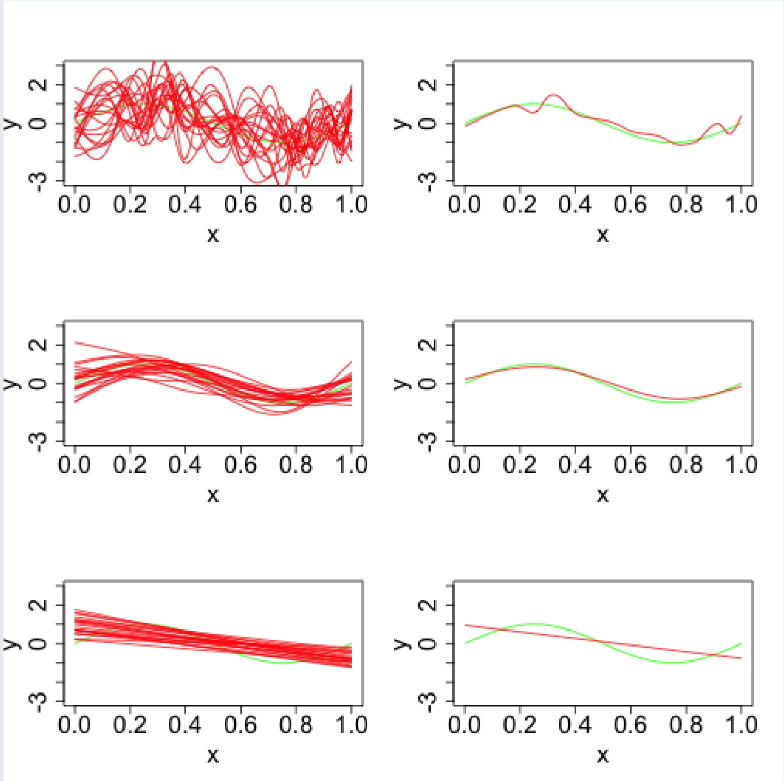
More details

More details | Code | Plot

```
par(mfrow = c(3, 2))

N <- 100
lambda_idx_list <- c(1, which.min(biasSQ + variance), 100)
x <- seq(0, 1, length.out = N)
y0 <- sin(2 * pi * x)

for(lambda_idx in lambda_idx_list){
  plot(x, y0, col = "green", type = "l", ylim = c(-3, 3), ylab = "y", ce

  y_hat <- matrix(0, nrow = N, ncol = 20)
  for(j in 1 : 20){
    y_hat[, j] <- predict(model_list[[lambda_idx]][[j]], x)$y
    lines(x, y_hat[, j], col = "red", ylim = c(-3, 3))
    }

  plot(x, y0, col = "green", type = "l", ylim = c(-3, 3), ylab = "y", ce
  lines(x, rowMeans(y_hat), col = "red", ylim = c(-3, 3))
  }
```

More details

More details | Code | **Plot**



Thank you!

Slides created via Yihui Xie's R package **xaringan**.

Theme customized via Garrick Aden-Buie's R package **xaringanthemer**.

Tabbed panels created via Garrick Aden-Buie's R package **xaringanExtra**.

The chakra comes from remark.js, **knitr**, and R Markdown.