Bias, Variance and Residual Variance

The bias-variance decomposition and how to read it

Last modified: 2026-11-19

```
# reticulate provisions the Python packages that this page imports.
library(reticulate) # nolint: undesirable_function_linter
py_packages <- c("numpy", "pandas", "matplotlib", "SyncRNG")
py_require(py_packages)
```

Source

Adapted from Brian Hutchinson's Fall 2025 lecture notes, which supplied the crime-and-income example and the bias-variance decomposition. The contrast between stochastic and deterministic noise follows the treatment of overfitting in Abu-Mostafa et al. [1, Ch. 4]; the example and its numbers are new.
The bias-variance derivation is rewritten to assume only that the noise has mean zero, finite variance, and is independent of the training set, where the notes assumed it Gaussian; see Theorem 1.4. The wider reading is Hastie et al. [2], Chapter 2, and James et al. [3], Chapter 2.

1 Bias, variance and residual variance

Exercise 1.1 (What is random when a model predicts)**.** The Generalization page says the risk cannot be computed, only estimated[a]. A different question about the same quantity is where it comes from: given a model that predicts badly, what are the distinguishable reasons it could be predicting badly? Start with the pieces. Choose an input value x, fit a model, and predict the output at x. For that process, list each thing that is random.

[a] https://morrison-lab.github.io/sds/estimation.html#def-estimate

Solution

Solution 1.1. Three things.

- The output itself.
- The training set $\mathcal{D}_{\text{train}}$.
- The fitted model $\hat{y}_{\mathcal{D}_{\text{train}}}$.

Exercise 1.2 (What is random when a model predicts (part 2))**.** Use the setup of Exercise 1.1. Say what makes each one random.

Solution

Solution 1.2.

- *The output itself.* The world supplies a true mean function μ and noise: an input[a] x has output

$Y = \mu(x) + \varepsilon$, where ε is mean-zero noise independent[b] of the training set (the noise model).

- The training set $\mathcal{D}_{\text{train}}$. It is a finite sample, and a different sample would have been just as possible.
- The fitted model. The learning algorithm turns $\mathcal{D}_{\text{train}}$ into a model, written $\hat{y}_{\mathcal{D}_{\text{train}}}$. Because $\mathcal{D}_{\text{train}}$ is random, so is $\hat{y}_{\mathcal{D}_{\text{train}}}$.

[a] https://morrison-lab.github.io/sds/model-validation.html#def-data-set
[b] https://morrison-lab.github.io/pds/independence.html#def-indpt

Theorem 1.1 (The noise has population mean zero)**.** *At any input*[a] *x, the noise ε has conditional expectation*[b] *(population mean*[c] *at x) of* 0*:*

$$\mathbb{E}[\varepsilon \mid x] = 0. \tag{1}$$

[a] https://morrison-lab.github.io/sds/model-validation.html#def-data-set
[b] https://morrison-lab.github.io/pds/expectation.html#def-cond-expectation
[c] https://morrison-lab.github.io/pds/expectation.html#def-expectation

Proof

Proof. For the outcome Y and noise ε at input[a] x, the true mean function value $\mu(x)$ does not vary once x is fixed, so its conditional expectation[b] given x is $\mu(x)$.

$$\begin{aligned}
\mathbb{E}[\varepsilon \mid x] &= \mathbb{E}[Y - \mu(x) \mid x] && \text{(definition of the noise)} \\
&= \mathbb{E}[Y \mid x] - \mu(x) && \text{(linearity of expectation)} \\
&= \mu(x) - \mu(x) && \text{(definition of the true mean function)} \\
&= 0 && \text{(a number minus itself is zero).}
\end{aligned}$$

□

[a] https://morrison-lab.github.io/sds/model-validation.html#def-data-set
[b] https://morrison-lab.github.io/pds/expectation.html#def-cond-expectation

Example 1.1 (Noise in a count outcome)**.**
Suppose the outcome Y at input[a] x is a count with a Poisson distribution[b] whose population mean[c] is the true mean function value $\mu(x)$. The noise $\varepsilon = Y - \mu(x)$ has population mean 0, by Theorem 1.1. The variance[d] of ε at x is $\text{Var}(\varepsilon) = \mu(x)$, so the noise has larger variance at an input x where $\mu(x)$ is larger. The distribution[e] of ε at x is also skewed: ε cannot fall below $-\mu(x)$ and has no upper bound. A gamma outcome behaves the same way: the noise is skewed, and the noise variance grows with the population mean $\mu(x)$.

[a] https://morrison-lab.github.io/sds/model-validation.html#def-data-set
[b] https://morrison-lab.github.io/pds/distributions.html#def-poisson
[c] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[d] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance
[e] https://morrison-lab.github.io/pds/random-variables.html#def-cdf

Example 1.2 (Noise in a binary outcome)**.** Suppose the outcome Y at input[a] x takes only the values 0 and 1, with a Bernoulli distribution[b] whose population mean[c] is the true mean function value $\mu(x)$. That value is also the probability[d] that $Y = 1$. The noise $\varepsilon = Y - \mu(x)$ then takes only two values:

- $1 - \mu(x)$ when $Y = 1$, which has probability $\mu(x)$;
- $-\mu(x)$ when $Y = 0$, which has probability $1 - \mu(x)$.

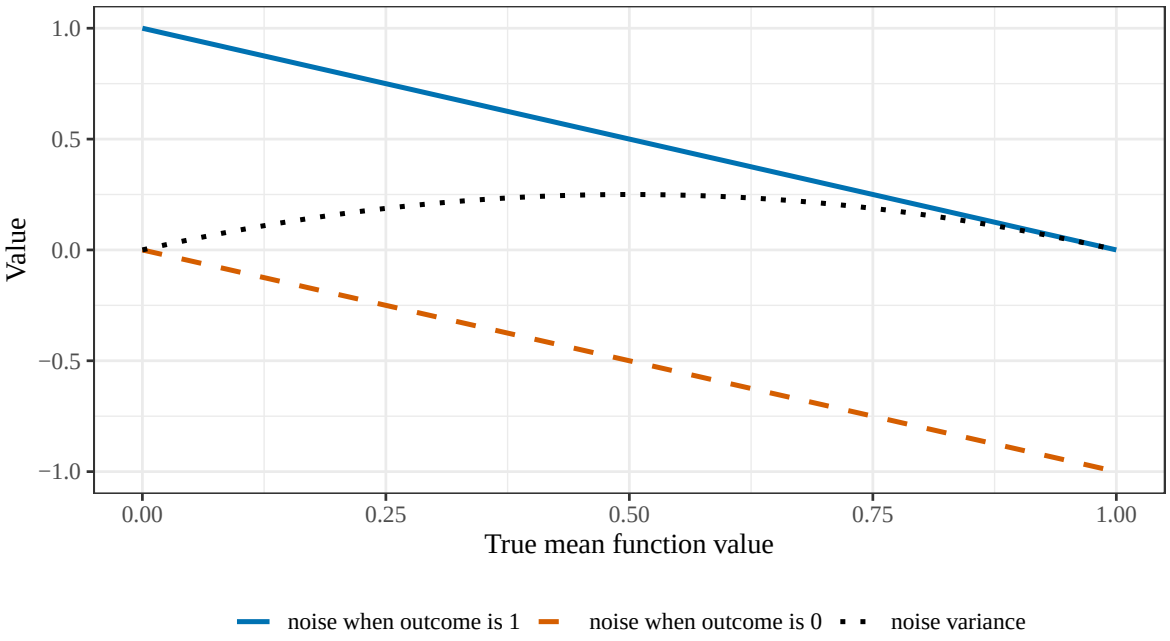
The noise has population mean 0, by Theorem 1.1. At a fixed input x, the value $\mu(x)$ is a constant[e], so ε and Y have the same variance[f]. For a Bernoulli[g] outcome, that variance is

$$\mathrm{Var}(\varepsilon) = \mu(x)(1 - \mu(x)).$$

The variance is largest at an input where $\mu(x) = 1/2$, and it shrinks toward 0 as $\mu(x)$ nears 0 or 1. So the two values the noise can take, and its variance, both depend on $\mu(x)$. Wherever $\mu(x)$ differs between inputs, the outcome is not its mean[h] plus noise with the same cumulative distribution function[i] at every input. Figure 1 draws the two noise values and the variance against $\mu(x)$.

1.0.1 R

```
nb_colours <- c(
  "noise when outcome is 1" = "#0072B2",
  "noise when outcome is 0" = "#D55E00",
  "noise variance" = "black"
)
nb_linetypes <- c(
  "noise when outcome is 1" = "solid",
  "noise when outcome is 0" = "dashed",
  "noise variance" = "dotted"
)
nb_levels <- names(nb_colours)
nb_grid <- seq(0, 1, length.out = 101)
nb_curves <- tibble::tibble(mean = nb_grid) |>
  dplyr::mutate(
    `noise when outcome is 1` = 1 - mean,
    `noise when outcome is 0` = -mean,
    `noise variance` = mean * (1 - mean)
  ) |>
  tidyr::pivot_longer(-mean, names_to = "curve", values_to = "value") |>
  dplyr::mutate(curve = factor(curve, levels = nb_levels))
nb_curves |>
  ggplot2::ggplot() +
  ggplot2::aes(x = mean, y = value, colour = curve, linetype = curve) +
  ggplot2::geom_line(linewidth = 1) +
  ggplot2::scale_colour_manual(values = nb_colours) +
  ggplot2::scale_linetype_manual(values = nb_linetypes) +
  ggplot2::labs(
    x = "True mean function value",
    y = "Value",
    colour = NULL,
    linetype = NULL
  ) +
  ggplot2::theme(legend.position = "bottom")
```

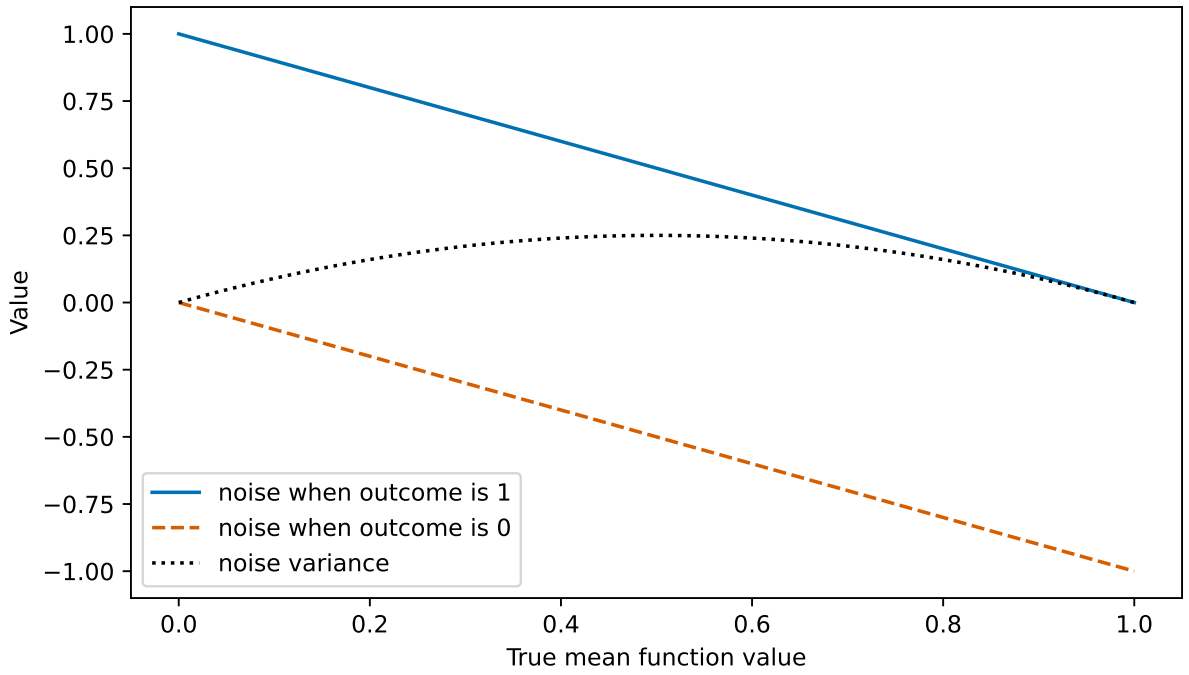

1.0
0.5
0.0
−0.5
−1.0
Value
0.00
0.25
0.50
0.75
1.00
True mean function value
noise when outcome is 1
noise when outcome is 0
noise variance

(a)

1.0.2 Python

```
import matplotlib.pyplot as plt
import numpy as np

nb_mean = np.linspace(0, 1, 101)
fig, ax = plt.subplots(figsize=(7, 4), layout="constrained")
ax.plot(nb_mean, 1 - nb_mean, color="#0072B2", linestyle="-", label="noise when outcome is 1");
ax.plot(nb_mean, -nb_mean, color="#D55E00", linestyle="--", label="noise when outcome is 0");
ax.plot(nb_mean, nb_mean * (1 - nb_mean), color="black", linestyle=":", label="noise variance");
ax.set_xlabel("True mean function value");
ax.set_ylabel("Value");
ax.legend(loc="lower left");
plt.show()
```



(b)

Figure 1: The two values the noise of a binary outcome can take, and the noise variance[j], as functions[k] of the true mean function value at an input[l] (Example 1.2).

[a] https://morrison-lab.github.io/pds/expectation.html#def-input
[b] https://morrison-lab.github.io/pds/random-variables.html#def-bernoulli
[c] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[d] https://morrison-lab.github.io/pds/probability-basics.html#def-probability
[e] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function
[f] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance
[g] https://morrison-lab.github.io/pds/random-variables.html#def-bernoulli
[h] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[i] https://morrison-lab.github.io/pds/random-variables.html#def-cdf
[j] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance
[k] https://morrison-lab.github.io/mcs/functions.html#def-function
[l] https://morrison-lab.github.io/pds/expectation.html#def-input

Remark (The noise can depend on the input)

Remark 1.1 (The noise can depend on the input). The distribution[a] of the noise ε at an input[b] x can change with x (Example 1.1). That distribution can be skewed, and the variance[c] of ε need not be the same at every input. Only the population mean[d] of ε at x must be zero (Theorem 1.1).

[a] https://morrison-lab.github.io/pds/random-variables.html#def-cdf
[b] https://morrison-lab.github.io/sds/model-validation.html#def-data-set
[c] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance
[d] https://morrison-lab.github.io/pds/expectation.html#def-expectation

Remark (The noise depends on which inputs it is conditional on)

Remark 1.2 (The noise depends on which inputs it is conditional on). The noise is defined only once the inputs in x are chosen, as the statistics notes[a] also say. Adding a feature to x can change the true mean function, and with it the noise of the same observation[b].

[a] https://morrison-lab.github.io/sds/model-validation.html#rem-noise-conditional-covariates
[b] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-observation

Remark (Independent noise draws across observations)

Remark 1.3 (Independent noise draws across observations). The noise ε is a fresh random[a] draw each time an outcome Y is observed. Each outcome Y_i in the training set $\mathcal{D}_{\text{train}}$ at input[b] x_i is $Y_i = \mu(x_i)+\varepsilon_i$, with true mean function value $\mu(x_i)$ and noise draw ε_i, and a new outcome Y at x has a new draw ε. Once the inputs[c] are fixed, $\varepsilon_i = Y_i - \mu(x_i)$ and $\varepsilon = Y - \mu(x)$ shift the outcomes by the fixed values $\mu(x_i)$ and $\mu(x)$, so conditional independence[d] of the outcomes given the inputs is equivalent to conditional independence of the noise draws given the inputs. Under that conditional independence, when the distribution[e] of ε given x does not change with x, the noise draws are independent[f] of the inputs and of one another. When the distribution of ε given x changes with x (Remark 1.1), a new observation drawn independently of $\mathcal{D}_{\text{train}}$ still has a new draw ε at the fixed input x that is independent of $\mathcal{D}_{\text{train}}$.

[a] https://morrison-lab.github.io/pds/notation.html#def-random
[b] https://morrison-lab.github.io/sds/model-validation.html#def-data-set
[c] https://morrison-lab.github.io/sds/model-validation.html#def-data-set
[d] https://morrison-lab.github.io/pds/independence.html#def-cind
[e] https://morrison-lab.github.io/pds/random-variables.html#def-cdf
[f] https://morrison-lab.github.io/pds/independence.html#def-indpt

Remark (The noise model in the probability notes)

Remark 1.4 (The noise model in the probability notes). For a single observation[a] of the outcome Y at input[b] x with true mean function value $\mu(x)$ and a finite variance[c] σ^2, the noise model equation is a consequence of the definition of noise ($Y = \mu(x) + \varepsilon$ for the noise ε) and Theorem 1.1 (conditional expectation[d] $\mathbb{E}[\varepsilon \mid x] = 0$), not an additional assumption. Only the noise's population mean[e] and variance matter here. The variance σ^2 is the variance of ε at the input x under discussion, and that variance can change with x (Remark 1.1). The probability notes[f] call the noise model equation the mean-zero noise model[g].

The noise model fixes only two parts of a distribution model[h] for the outcome: the mean function[i] is $\mu(x)$, and the variance is σ^2. Any distribution model with that mean function and that variance satisfies the noise model, such as the normal distribution model[j] with mean function $\mu(x)$ and variance σ^2.

[a] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-observation
[b] https://morrison-lab.github.io/sds/model-validation.html#def-data-set
[c] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance
[d] https://morrison-lab.github.io/pds/expectation.html#def-cond-expectation

[e] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[f] https://morrison-lab.github.io/pds/
[g] https://morrison-lab.github.io/pds/variance-covariance.html#sec-noise-model
[h] https://morrison-lab.github.io/sds/model-validation.html#def-distribution-model
[i] https://morrison-lab.github.io/sds/model-validation.html#def-mean-function
[j] https://morrison-lab.github.io/sds/model-validation.html#exm-distribution-model

Definition 1.1 (Prediction rule)**.**
A **prediction rule** is a fitted model $\hat{y}_{\mathcal{D}_{\text{train}}}$ that is used to predict the outcome at a new covariate value x. Its prediction is $\hat{y}(x)$.
In words, the prediction rule takes a new covariate value and returns the fitted model's value there.
See prediction rule[a] in the statistics notes.

[a] https://morrison-lab.github.io/sds/model-validation.html#def-prediction-rule

Example 1.3 (The fitted line as a prediction rule)**.**
Take the line $\hat{y}_{\mathcal{D}_{\text{train}}}$ fitted to the five neighborhoods of the crime data, as in the fitted-line example. Using it to predict the crime rate of a sixth neighborhood makes it a prediction rule. Using the same line only to describe the five neighborhoods makes no prediction, so there it is a fitted model and not a prediction rule.

Definition 1.2 (Average fit)**.** The **average fit** at x is the prediction $\hat{y}(x)$ of the fitted model at x, averaged over every training set $\mathcal{D}_{\text{train}}$ it might have been fitted to, where $\mathbb{E}_{\mathcal{D}_{\text{train}}}$ denotes that average over training sets. It is the population expectation

$$\mu_h(x) \stackrel{\text{def}}{=} \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}(x)], \tag{2}$$

a fixed function of x that nothing estimates, so it carries no hat. It is written like the true mean function, with the model as the subscript.

It is not a model anyone ever holds; it is a reference point for measuring how far fits are from the truth and from each other. The statistics notes define it for the fitted model of any fitting procedure[1].

Example 1.4 (The average fit of a line on cylinders)**.** Treat the 392 cars of the `Auto` data [3] as the whole world of cars. A training set is 30 cars drawn at random from it, and the learning algorithm fits the line of `mpg` on `cylinders`. Draw 2000 training sets, fit a line to each, and record each line's intercept, slope and prediction at 4 cylinders. Both tabs draw with the `SyncRNG` package [4], a random number generator with the same algorithm in R and Python, from the same seed, so they pick the same cars:

1.0.3 Python

[1] https://morrison-lab.github.io/sds/model-validation.html#def-average-fit

```python
import numpy as np
import pandas as pd
from SyncRNG import SyncRNG

auto_af = pd.read_csv("data/auto.csv").dropna(subset=["horsepower"])
cyl_af = auto_af["cylinders"].to_numpy(float)
mpg_af = auto_af["mpg"].to_numpy(float)

rng_af = SyncRNG(seed=1)
coefs_af = np.empty((2000, 2))
for s_af in range(2000):
    # One training set: the first 30 rows of a shuffle of all the rows (Python counts from 0), and the
    pick_af = rng_af.shuffle(list(range(len(mpg_af))))[:30]
    slope_af, intercept_af = np.polyfit(cyl_af[pick_af], mpg_af[pick_af], 1)
    coefs_af[s_af] = intercept_af, slope_af

preds_af = coefs_af[:, 0] + coefs_af[:, 1] * 4

# The average fit at 4 cylinders is the mean of those predictions.
# The Monte Carlo standard error of that mean is the spread of the predictions over sqrt(2000).
mc_se_af = preds_af.std(ddof=1) / np.sqrt(2000)
print(preds_af.mean().round(2), preds_af.min().round(2), preds_af.max().round(2), mc_se_af.round(3))
```

```
28.72 23.89 32.74 0.031
```

1.0.4 R

```r
auto_af <- read.csv("data/auto.csv")
auto_af <- na.omit(auto_af[, c("mpg", "cylinders", "horsepower")])

# SyncRNG is a random number generator with the same algorithm in R and Python, so the draws match the
library(SyncRNG)
rng_af <- SyncRNG(seed = 1)
coefs_af <- t(replicate(2000, {
  # One training set: the first 30 rows of a shuffle of all the rows, and the line fitted to it.
  rows <- rng_af$shuffle(seq_len(nrow(auto_af)))[1:30]
  coef(lm(mpg ~ cylinders, data = auto_af[rows, ]))
}))

preds_af <- coefs_af[, 1] + coefs_af[, 2] * 4

# The average fit at 4 cylinders is the mean of those predictions.
# The Monte Carlo standard error of that mean is the spread of the predictions over sqrt(2000).
mc_se_af <- sd(preds_af) / sqrt(2000)
c(round(c(mean = mean(preds_af), min = min(preds_af), max = max(preds_af)), 2), mc_se = round(mc_se_af
```

```
  mean    min    max  mc_se
28.720 23.890 32.740  0.031
```

No single training set produces the average fit. The 2000 lines predict anywhere from 23.9 to 32.7 miles per gallon for a four-cylinder car, and their mean, 28.7, is the average fit $\mu_h(4)$ of Equation 2, with a 95% Monte Carlo confidence interval of 28.66 to 28.78 (1.96 Monte Carlo standard errors either side). The two tabs print the same numbers up to rounding.

Figure 2 plots 100 of the 2000 lines, their pointwise average, and the true mean function:

1.0.5 Python

```
import numpy as np
import matplotlib.pyplot as plt

grid_af = np.arange(3, 9)
# The pointwise average fit is the mean of the fitted lines at each cylinder count.
avg_af = (coefs_af[:, [0]] + coefs_af[:, [1]] * grid_af).mean(axis=0)
# The true mean function is the mean mpg of the cars with each cylinder count.
truth_af = auto_af.groupby("cylinders")["mpg"].mean()

fig, ax = plt.subplots(figsize=(6, 3.5), layout="constrained")
for intercept_af, slope_af in coefs_af[:100]:
    ax.plot(grid_af, intercept_af + slope_af * grid_af, color="grey", alpha=0.3, linewidth=0.8);
# One legend entry for the 100 grey lines, with a solid line to stand for them.
ax.plot([], [], color="grey", label="One fitted line (100 shown)");
ax.plot(grid_af, avg_af, color="steelblue", linewidth=2.5, label="Average fit");
ax.plot(truth_af.index, truth_af.values, color="darkorange", linewidth=2.5, marker="o", label="True me
ax.set_xlabel("cylinders");
ax.set_ylabel("miles per gallon");
fig.legend(loc="outside lower center", ncol=3, frameon=False);
plt.show()
```

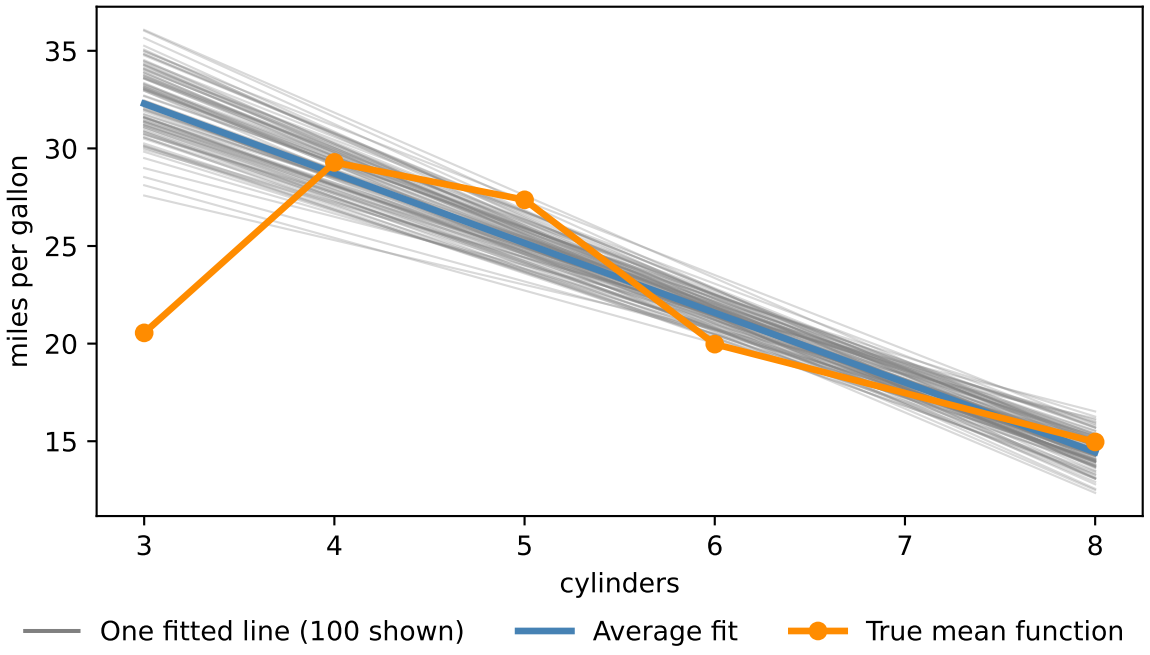


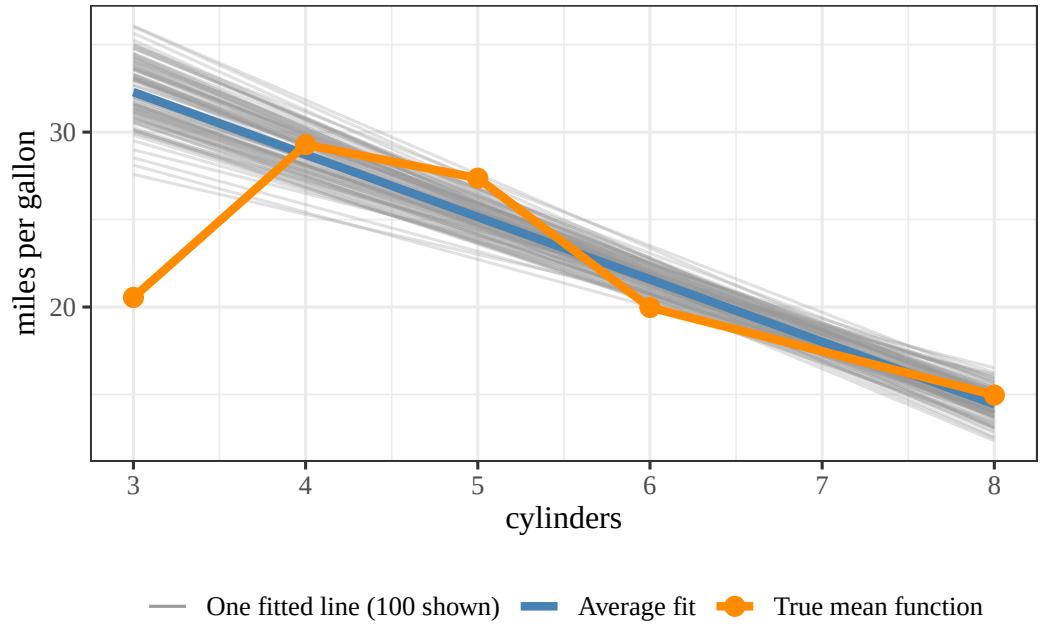
Figure 2: The fitted lines of 100 of the 2000 training sets (grey), their pointwise average fit (blue), and the true mean function (orange), for `mpg` against `cylinders`.

1.0.6 R

```r
library(ggplot2)

grid_af <- 3:8
# One row per fitted line and cylinder count; the pointwise average fit is the mean over lines.
lines_af <- data.frame(
  line = rep(seq_len(nrow(coefs_af)), each = length(grid_af)),
  cylinders = grid_af,
  fit = as.vector(t(coefs_af[, 1] + outer(coefs_af[, 2], grid_af)))
)
avg_af <- aggregate(fit ~ cylinders, data = lines_af, FUN = mean)
# The true mean function is the mean mpg of the cars with each cylinder count.
truth_af <- aggregate(mpg ~ cylinders, data = auto_af, FUN = mean)

ggplot() +
  geom_line(data = subset(lines_af, line <= 100), aes(cylinders, fit, group = line, colour = "One fitt
  geom_line(data = avg_af, aes(cylinders, fit, colour = "Average fit"), linewidth = 1.5) +
  geom_line(data = truth_af, aes(cylinders, mpg, colour = "True mean function"), linewidth = 1.5) +
  geom_point(data = truth_af, aes(cylinders, mpg, colour = "True mean function"), size = 3) +
  scale_colour_manual(
    values = c("One fitted line (100 shown)" = "grey60", "Average fit" = "steelblue", "True mean funct
    breaks = c("One fitted line (100 shown)", "Average fit", "True mean function")
  ) +
  guides(colour = guide_legend(override.aes = list(alpha = 1))) +
  labs(x = "cylinders", y = "miles per gallon", colour = NULL) +
  theme(legend.position = "bottom")
```



Definition 1.3 (Bias of a prediction)**.** The **bias** of a prediction at x is the expected[a] prediction error[b] of the prediction at x. The outcome in the prediction error[c] is a new outcome Y at x in the noise model, so the error is $\varepsilon_{\text{pred}} = (\hat{y} - Y)$. The expectation[d] is taken over training sets $\mathcal{D}_{\text{train}}$ and over the noise in Y:

$$\text{Bias}(\hat{y}) \stackrel{\text{def}}{=} \mathbb{E}_{\mathcal{D}_{\text{train}},Y}\left[\varepsilon_{\text{pred}}\right]. \tag{3}$$

[a] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[b] https://morrison-lab.github.io/sds/prediction.html#def-prediction-error
[c] https://morrison-lab.github.io/sds/prediction.html#def-prediction-error
[d] https://morrison-lab.github.io/pds/expectation.html#def-expectation

Example 1.5 (The bias of the cylinders line at four cylinders)**.** Continue Example 1.4 with the same 2000 fitted lines. At $x = 4$ cylinders, the value $\mu(4)$ of the true mean function in the noise model is the population mean[a] `mpg` of the four-cylinder cars. Compute the bias (Equation 3) of the line:

1.0.7 Python

```
import numpy as np

# mu(4) is the mean mpg of the four-cylinder cars.
f4_bv = mpg_af[cyl_af == 4].mean()

# Bias: the average fit minus the truth.
bias_bv = preds_af.mean() - f4_bv
print(np.round(bias_bv, 2))
```

```
-0.56
```

1.0.8 R

```
# mu(4) is the mean mpg of the four-cylinder cars.
f4_bv <- mean(auto_af$mpg[auto_af$cylinders == 4])

# Bias: the average fit minus the truth.
bias_bv <- mean(preds_af) - f4_bv
round(bias_bv, 2)
```

```
[1] -0.56
```

The average fit is 0.56 miles per gallon below the truth, so the bias is -0.56. The two tabs print the same number up to rounding.

Figure 3 shows the 2000 predictions with the average fit and the true mean function at four cylinders. The histogram[b] uses 40 bins[c]; an even split of the 2000 predictions would put 50 in each bin. Forty bins are enough to show the shape of the histogram, which a few wide bins would flatten and a few hundred narrow bins would break into jagged spikes.

1.0.9 Python

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(6, 3.5), layout="constrained")
ax.hist(preds_af, bins=40, color="lightgrey", edgecolor="white");
ax.axvline(preds_af.mean(), color="steelblue", linewidth=2);
ax.axvline(f4_bv, color="darkorange", linewidth=2);
ax.set_xlabel("predicted miles per gallon at 4 cylinders");
ax.set_ylabel("training sets");
plt.show()
```

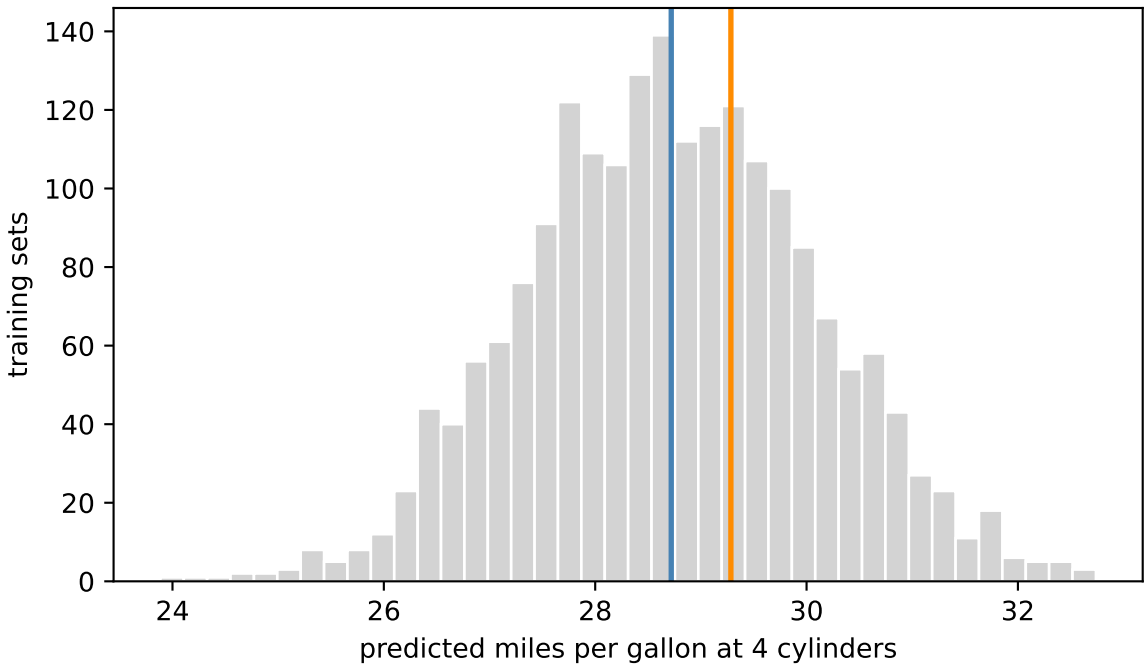
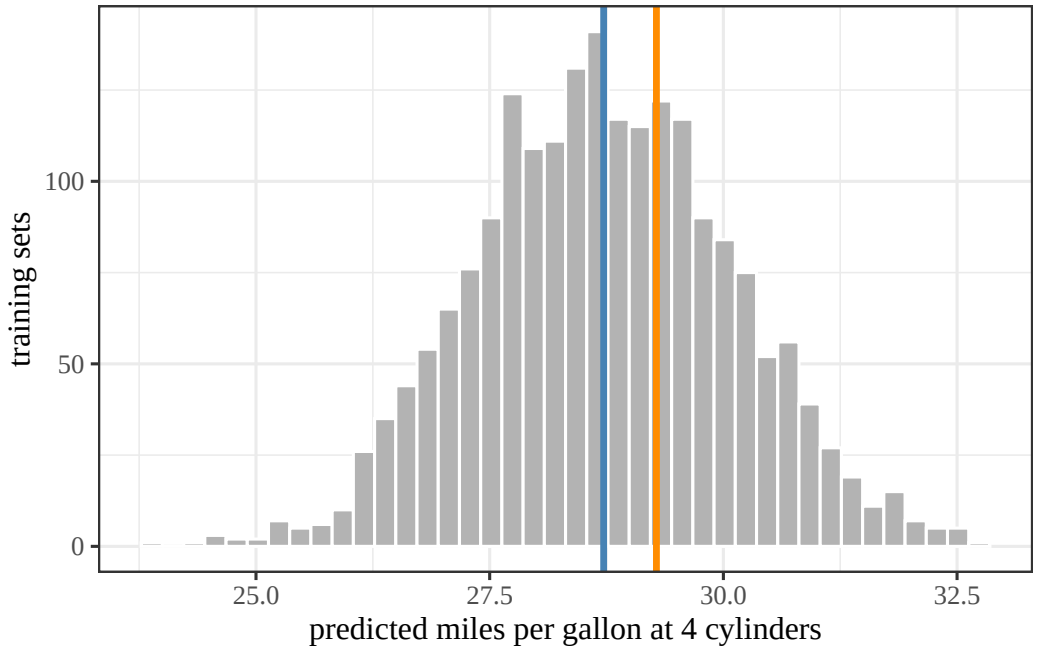


Figure 3: The 2000 predictions at four cylinders, with their average (the average fit, blue) and the true mean function at four cylinders (orange). The gap between the lines is the bias.

1.0.10 R

```
library(ggplot2)

ggplot(data.frame(prediction = preds_af), aes(prediction)) +
  geom_histogram(bins = 40, fill = "grey70", colour = "white") +
  geom_vline(xintercept = mean(preds_af), colour = "steelblue", linewidth = 1.2) +
  geom_vline(xintercept = f4_bv, colour = "darkorange", linewidth = 1.2) +
  labs(x = "predicted miles per gallon at 4 cylinders", y = "training sets")
```



[a] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[b] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-histogram
[c] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-bin

ℹ Remark (The sign of the bias)

Remark 1.5 (The sign of the bias). The bias is the expectation[a] of a prediction error[b], so it is positive when the prediction $\hat{y}$ exceeds the new outcome Y in expectation, and negative when the prediction falls short of the new outcome in expectation. The risk is the expected squared prediction error[c], so it is never negative. It adds the variance of the prediction and the residual variance to the square[d] of the bias.

[a] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[b] https://morrison-lab.github.io/sds/prediction.html#def-prediction-error
[c] https://morrison-lab.github.io/sds/model-validation.html#def-epe
[d] https://morrison-lab.github.io/mds/algebra-basics.html#def-square

Definition 1.4 (Variance of a prediction)**.** The **variance** of a prediction at x is the expected square of the deviation[a] of the prediction at x from its population mean[b] over training sets $\mathcal{D}_{\text{train}}$, which is the average fit $\mu_h(x)$:

$$\text{Var}(\hat{y}) = \mathbb{E}_{\mathcal{D}_{\text{train}}}\left[\left(\mathbb{d}_{\hat{y}}\right)^2\right], \quad \text{where } \mathbb{d}_{\hat{y}} = \hat{y} - \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}]. \tag{4}$$

[a] https://morrison-lab.github.io/pds/variance-covariance.html#def-deviation
[b] https://morrison-lab.github.io/pds/expectation.html#def-expectation

Example 1.6 (The variance of the cylinders line at four cylinders)**.** Continue Example 1.5 with the same 2000 fitted models. Compute the variance (Equation 4) of their four-cylinder prediction:

1.0.11 Python

```
# Variance: the mean squared deviation of a prediction from the average fit.
dev_var = preds_af - preds_af.mean()
var_bv = np.mean(dev_var ** 2)
print(np.round(var_bv, 2))
```

```
1.86
```

1.0.12 R

```
# Variance: the mean squared deviation of a prediction from the average fit.
avg_fit_var <- mean(preds_af)
dev_var <- preds_af - avg_fit_var
var_bv <- mean(dev_var^2)
round(var_bv, 2)
```

```
[1] 1.86
```

The variance is 1.86 squared miles per gallon: the sample mean[a], over the 2000 fitted models, of the squared deviation[b] of a prediction from the average fit. The two tabs print the same number up to rounding.

Figure 4 shows the 2000 deviation[c]s of the predictions from the average fit, in the same 40 bins[d] as Figure 3 and for the same reason.

1.0.13 Python

```
fig, ax = plt.subplots(figsize=(6, 3.5), layout="constrained")
ax.hist(dev_var, bins=40, color="lightgrey", edgecolor="white");
ax.axvline(0, color="steelblue", linewidth=2);
ax.set_xlabel("prediction minus average fit (miles per gallon)");
ax.set_ylabel("training sets");
plt.show()
```

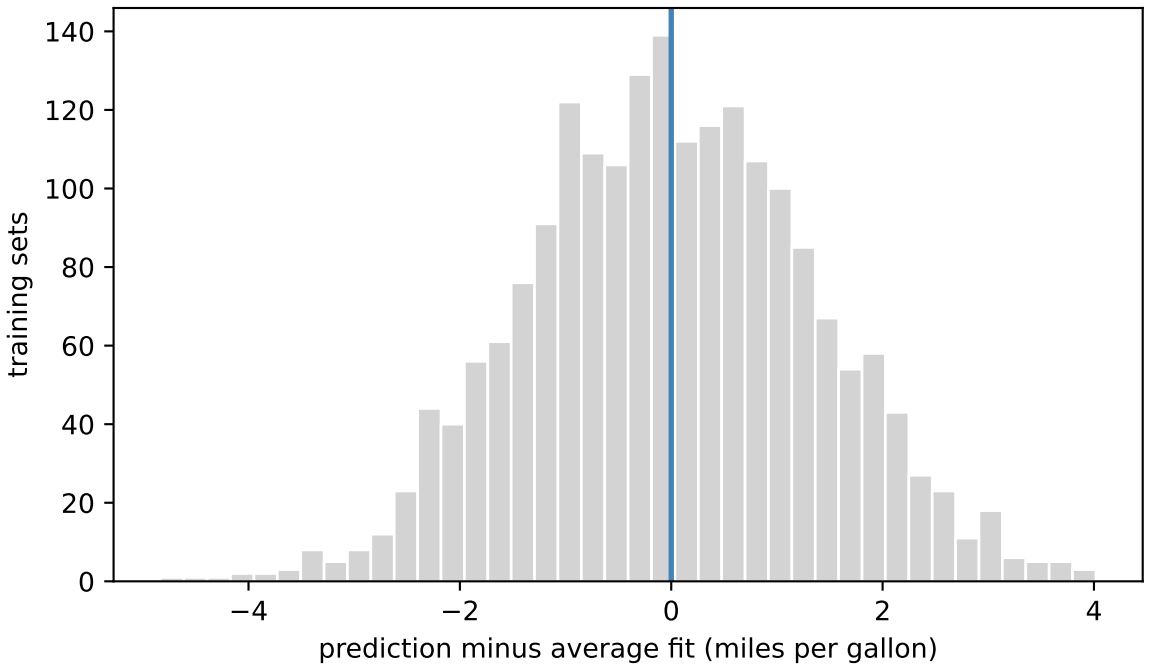
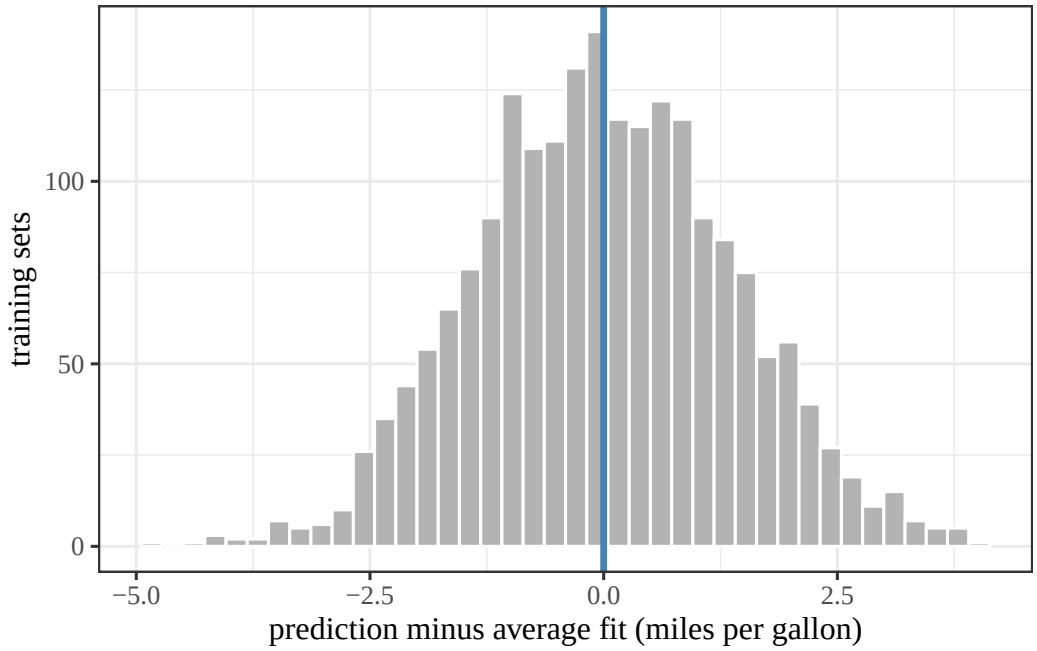


Figure 4: The deviations of the 2000 predictions at four cylinders from their average fit. The variance is the mean of their squares, so a wider histogram means a larger variance.

1.0.14 R

```
dev_tbl_var <- tibble::tibble(deviation = dev_var)

dev_tbl_var |>
  ggplot() +
  aes(x = deviation) +
  geom_histogram(bins = 40, fill = "grey70", colour = "white") +
  geom_vline(xintercept = 0, colour = "steelblue", linewidth = 1.2) +
  labs(
    x = "prediction minus average fit (miles per gallon)",
    y = "training sets"
  )
```



[a] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean
[b] https://morrison-lab.github.io/pds/variance-covariance.html#def-deviation
[c] https://morrison-lab.github.io/pds/variance-covariance.html#def-deviation
[d] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-bin

Remark (The variance of a fitted model's prediction is the variance of a random variable)

Remark 1.6 (The variance of a fitted model's prediction is the variance of a random variable). The prediction of the fitted model at x is a random variable[a], because it depends on the training set $\mathcal{D}_{\text{train}}$. The variance of that prediction is therefore the variance[b] of that random variable.

[a] https://morrison-lab.github.io/pds/random-variables.html#def-random-variable
[b] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance

Theorem 1.2 (The bias as the average fit minus the true mean function)**.** *The bias of the prediction at x equals the average fit minus the true mean function value:*

$$\text{Bias}(\hat{y}) = \mu_h(x) - \mu(x).$$

Proof

Proof. The mean[a] of the new outcome Y is the true mean function value $\mu(x)$ because the noise has mean zero by the noise model.

$$\begin{aligned}
\text{Bias}(\hat{y}) &= \mathbb{E}_{\mathcal{D}_{\text{train}},Y}\left[\varepsilon_{\text{pred}}\right] && \text{(definition of bias)} \\
&= \mathbb{E}_{\mathcal{D}_{\text{train}},Y}[(\hat{y} - Y)] && \text{(definition of prediction error)} \\
&= \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}] - \mathbb{E}_{Y}[Y] && \text{(linearity of expectation)} \\
&= \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}] - \mu(x) && \text{(the noise has mean zero)} \\
&= \mu_h(x) - \mu(x) && \text{(definition of the average fit).}
\end{aligned}$$

□

[a] https://morrison-lab.github.io/pds/expectation.html#def-expectation

Example 1.7 (The bias as the average fit minus the true mean function at four cylinders)**.** Continue Example 1.5 with the same 2000 fitted lines. The bias is the expectation[a] of the prediction error[b] against a new outcome (Equation 3). The new outcome has mean[c] $\mu(4)$, so the bias equals the average estimation error[d] of the lines. Compute that average, and compare it with the average fit minus the truth:

1.0.15 Python

```
# The bias from its definition: the mean error over the lines, taking the new outcome at its mean.
bias_def_baf = np.mean(preds_af - f4_bv)
# The bias from the theorem: the average fit minus the truth.
bias_thm_baf = preds_af.mean() - f4_bv
print(np.round([bias_def_baf, bias_thm_baf], 2))
```

```
[-0.56 -0.56]
```

1.0.16 R

```
# The bias from its definition: the mean error over the lines, taking the new outcome at its mean.
bias_def_baf <- mean(preds_af - f4_bv)
# The bias from the theorem: the average fit minus the truth.
bias_thm_baf <- mean(preds_af) - f4_bv
round(c(definition = bias_def_baf, theorem = bias_thm_baf), 2)
```

```
definition    theorem
     -0.56      -0.56
```

Both routes give -0.56 miles per gallon, as Theorem 1.2 says.

[a] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[b] https://morrison-lab.github.io/sds/prediction.html#def-prediction-error
[c] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[d] https://morrison-lab.github.io/sds/estimation.html#def-estimation-error

Theorem 1.3 (The variance equals the variance of the estimation error)**.** *The variance of the prediction of the fitted model at x equals the variance of the estimation error[a] of that prediction:*

$$\mathrm{Var}(\hat{y}) = \mathrm{Var}(\varepsilon(\hat{y})).$$

[a] https://morrison-lab.github.io/sds/estimation.html#def-estimation-error

Proof

Proof. The mean[a] of the estimation error[b] is the average fit minus the true mean function value, which is the bias by Theorem 1.2. Subtracting the bias from the error leaves the fit minus the average fit.

$$\begin{aligned}
\varepsilon(\hat{y}) - \mathrm{Bias}(\hat{y}) &= (\hat{y} - \mu(x)) - \left(\mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}] - \mu(x)\right) && \text{(definition of estimation error; @thm-bias-average-fit)} \\
&= \hat{y} - \mu(x) - \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}] + \mu(x) && \text{(removing parentheses)} \\
&= \hat{y} - \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}] + (\mu(x) - \mu(x)) && \text{(reordering)} \\
&= \hat{y} - \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}] + 0 && \text{(a number minus itself is zero)} \\
&= \hat{y} - \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}] && \text{(adding zero changes nothing).}
\end{aligned}$$

The error's variance is its expected squared deviation from its own mean, and its mean is the bias. So the variance of the error is the expected square of the difference above. By Equation 4, the variance of the fit is the expected square of the same difference. So the two variances are equal. □

[a] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[b] https://morrison-lab.github.io/sds/estimation.html#def-estimation-error

Example 1.8 (The variance as the spread around the average fit at four cylinders)**.** Continue Example 1.5 with the same 2000 fitted lines. Compute the variance of the lines (Equation 4), and compare it with the variance of their estimation error[a]:

1.0.17 Python

```
# The variance from the definition: the spread of the lines around the average fit.
var_def_vaf = np.mean((preds_af - preds_af.mean()) ** 2)
# The variance from the theorem: the spread of the estimation error around its mean, the bias.
err_vaf = preds_af - f4_bv
var_thm_vaf = np.mean((err_vaf - err_vaf.mean()) ** 2)
print(np.round([var_def_vaf, var_thm_vaf], 2))
```

```
[1.86 1.86]
```

1.0.18 R

```
# The variance from the definition: the spread of the lines around the average fit.
var_def_vaf <- mean((preds_af - mean(preds_af))^2)
# The variance from the theorem: the spread of the estimation error around its mean, the bias.
err_vaf <- preds_af - f4_bv
var_thm_vaf <- mean((err_vaf - mean(err_vaf))^2)
round(c(definition = var_def_vaf, theorem = var_thm_vaf), 2)
```

```
definition    theorem
      1.86       1.86
```

Both routes give 1.86 squared miles per gallon, as Theorem 1.3 says.

[a] https://morrison-lab.github.io/sds/estimation.html#def-estimation-error

Five training sets of size five each get four fitted models, one for each polynomial degree[2] from 1 to 4. Both tabs draw the training sets with the `SyncRNG` package [4] from the same seed[3], so they draw the same data, and add normal[4] noise (standard deviation[5] 0.13) to the true mean function:

1.0.19 Python

```
import numpy as np
import matplotlib.pyplot as plt
from SyncRNG import SyncRNG


def truth_bv(x):
    return 0.55 - 0.9 * x + 0.6 * np.sin(6 * x)


x_bv = np.array([0.1, 0.3, 0.5, 0.7, 0.9])
grid_bv = np.linspace(0.05, 0.95, 100)

rng_bv = SyncRNG(seed=37)
# The first draws from a small seed are poorly mixed, so skip the first 100.
```

[2] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-polynomial-degree
[3] https://morrison-lab.github.io/sds/model-validation.html#def-random-seed
[4] https://morrison-lab.github.io/pds/random-variables.html#def-normal
[5] https://morrison-lab.github.io/pds/variance-covariance.html#def-sd

```
# With seed 37 the spread rises with the degree, and the line and quadratic have about the same bias; th
for _ in range(100):
    _ = rng_bv.rand()


def std_normal_bv():
    # One standard normal draw from two uniform draws (the Box-Muller transform).
    u1, u2 = rng_bv.rand(), rng_bv.rand()
    return np.sqrt(-2 * np.log(u1)) * np.cos(2 * np.pi * u2)


# Five training sets of five points.
samples_bv = [truth_bv(x_bv) + 0.13 * np.array([std_normal_bv() for _ in x_bv]) for _ in range(5)]
titles_bv = ["A line: high bias, low variance", "A quadratic", "A cubic", "A quartic: low bias, high var

fig, axes = plt.subplots(2, 2, figsize=(6.5, 5), sharex=True, sharey=True, layout="constrained")
for degree, ax, title in zip(range(1, 5), axes.flat, titles_bv):
    # One fit per training set, evaluated on the grid.
    fits = np.array([np.polyval(np.polyfit(x_bv, y, degree), grid_bv) for y in samples_bv])
    for fit in fits:
        ax.plot(grid_bv, fit, color="#0072B2", linewidth=1);
    ax.plot(grid_bv, truth_bv(grid_bv), color="black", linestyle="--", linewidth=1.2);
    ax.set_title(title);
for ax in axes[1]:
    ax.set_xlabel("Feature, x");
for ax in axes[:, 0]:
    ax.set_ylabel("Response, y");
fig.legend(
    handles=[
        plt.Line2D([], [], color="#0072B2", label="One fit per training set"),
        plt.Line2D([], [], color="black", linestyle="--", label="The true mean function"),
    ],
    loc="outside lower center",
    ncol=2,
    frameon=False,
);
plt.show()
```

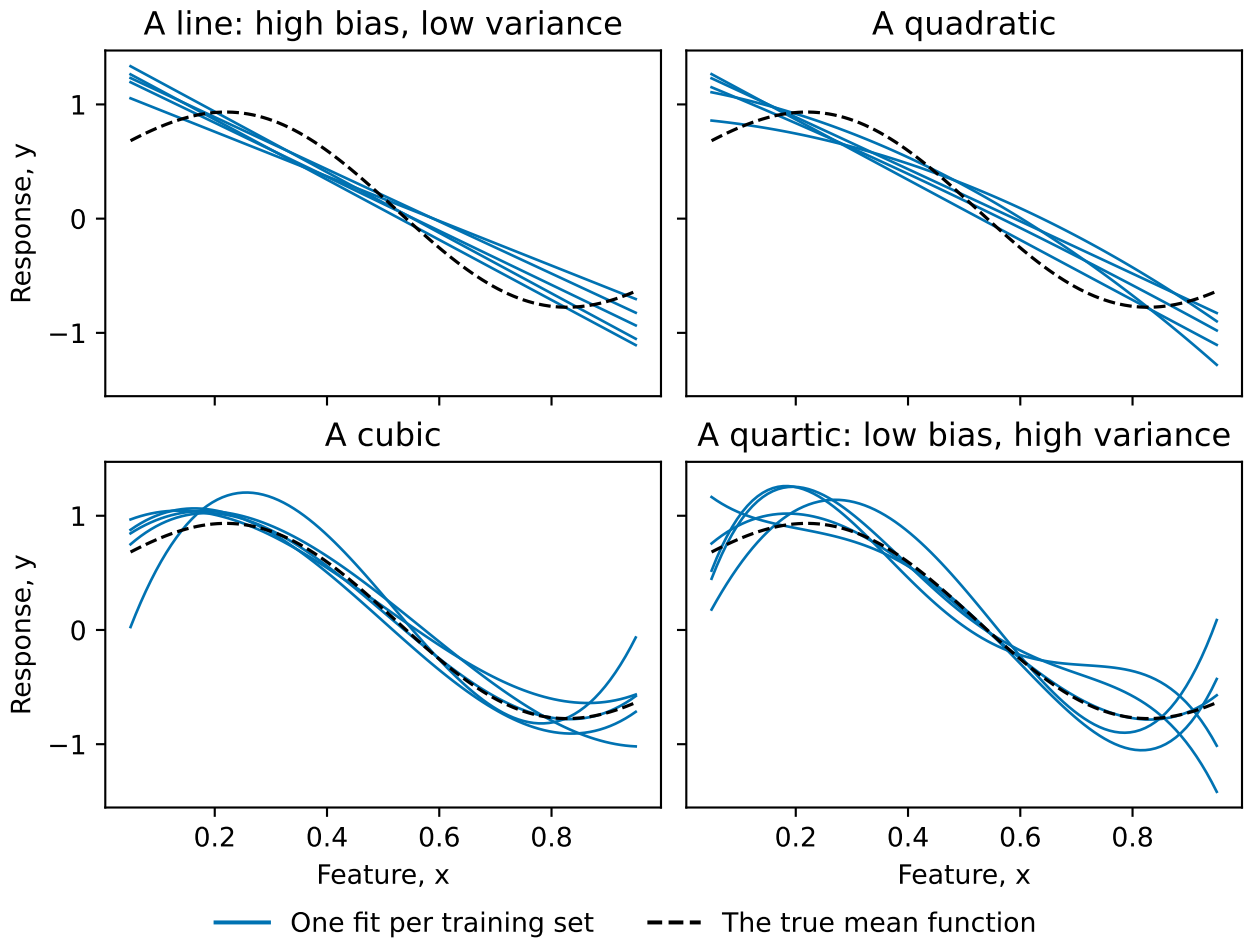


Figure 5: The same five training sets, fitted four times: by a line, a quadratic, a cubic and a quartic. The line and the quadratic lie close together and miss the true mean function (dashed) in the same places. The cubic fits follow it closely except near the ends of the range. The quartic fits, each passing through all five points of its training set, spread widely. Each curve is one training set's fit, so the spread among the five is the variability itself.

1.0.20 R

```r
library(ggplot2)
library(SyncRNG)

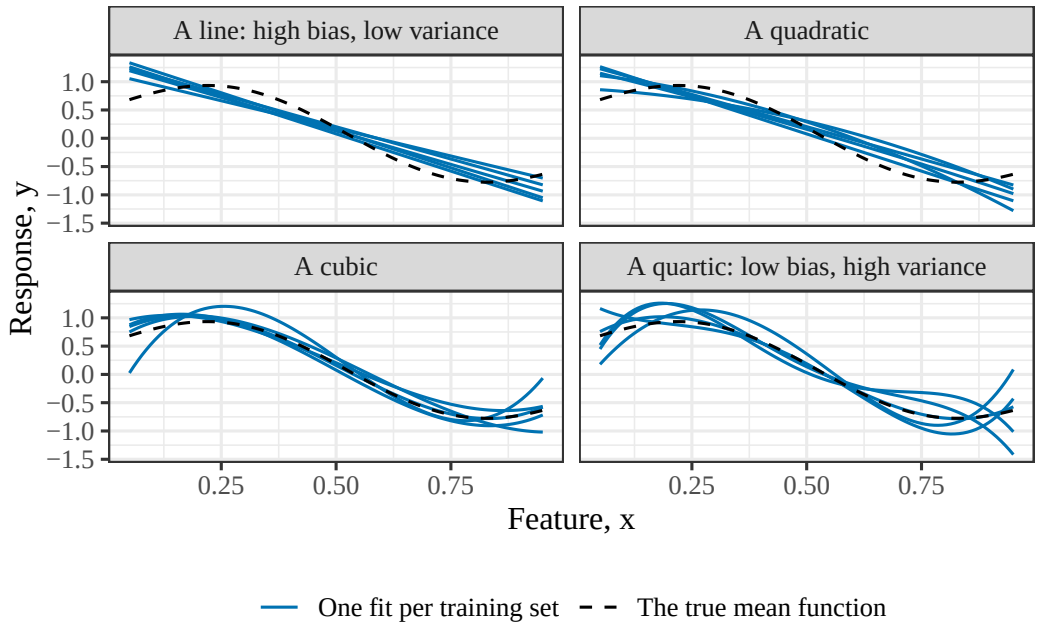
truth_bv <- function(x) 0.55 - 0.9 * x + 0.6 * sin(6 * x)
x_bv <- c(0.1, 0.3, 0.5, 0.7, 0.9)
grid_bv <- seq(0.05, 0.95, length.out = 100)

rng_bv <- SyncRNG(seed = 37)
# The first draws from a small seed are poorly mixed, so skip the first 100.
# With seed 37 the spread rises with the degree, and the line and quadratic have about the same bias; th
for (i in 1:100) rng_bv$rand()
# One standard normal draw from two uniform draws (the Box-Muller transform).
std_normal_bv <- function() {
  u <- c(rng_bv$rand(), rng_bv$rand())
  sqrt(-2 * log(u[1])) * cos(2 * pi * u[2])
}
```

```r
# Five training sets of five points, each fitted by a polynomial of degree 1 to 4.
fits_bv <- do.call(rbind, lapply(1:5, function(set) {
  y <- truth_bv(x_bv) + 0.13 * replicate(5, std_normal_bv())
  do.call(rbind, lapply(1:4, function(degree) {
    fit <- lm(y ~ poly(x_bv, degree, raw = TRUE))
    data.frame(
      set = set,
      degree = factor(degree, levels = 1:4, labels = c("A line: high bias, low variance", "A quadratic",
      x = grid_bv,
      fit = predict(fit, data.frame(x_bv = grid_bv))
    )
  }))
}))
truth_df_bv <- data.frame(x = grid_bv, fit = truth_bv(grid_bv))

# The spread of the five fits: their standard deviation at each x, averaged over the grid.
spread_bv <- sapply(split(fits_bv, fits_bv$degree), function(d) mean(tapply(d$fit, d$x, sd)))

ggplot(fits_bv, aes(x, fit)) +
  geom_line(aes(group = set, colour = "One fit per training set", linetype = "One fit per training set")
  geom_line(data = truth_df_bv, aes(colour = "The true mean function", linetype = "The true mean functio
  facet_wrap(~degree, nrow = 2) +
  scale_colour_manual(values = c("One fit per training set" = "#0072B2", "The true mean function" = "bla
  scale_linetype_manual(values = c("One fit per training set" = "solid", "The true mean function" = "das
  labs(x = "Feature, x", y = "Response, y", colour = NULL, linetype = NULL)
```



Each quartic has five parameters, one for each input[6] in its training set, so it fits that training set exactly and swings between its inputs, which is the variability the bottom-right panel shows. The line has two parameters and barely moves, and the quadratic and the cubic sit between the two. The spread of the five fits, averaged over the grid, is 0.08 for the line, 0.10 for the quadratic, 0.13 for the cubic and 0.18 for the quartic, so here it rises with every step in the degree[7].

[6] https://morrison-lab.github.io/sds/model-validation.html#def-data-set
[7] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-polynomial-degree

Theorem 1.4 (Separating the noise from the expected error)**.** *Choose an input value x, and let the output variable $Y = \mu(x) + \varepsilon$ as in the noise model, where ε has mean 0 and variance σ^2 and is independent of the training set $\mathcal{D}_{train}$. Then the expected squared error*[a] *at x is the noise variance σ^2 plus the model's own expected squared error:*

$$\mathbb{E}_{\mathcal{D}_{train},\varepsilon}\left[(Y - \hat{y})^2\right] = \sigma^2 + \mathbb{E}_{\mathcal{D}_{train}}\left[(\mu(x) - \hat{y})^2\right]. \tag{5}$$

[a] https://morrison-lab.github.io/sds/model-validation.html#def-epe

Proof

Proof. This is the statistics notes' theorem that the expected squared prediction error is the noise variance plus the expected squared estimation error[a], applied to the prediction $\hat{y}$ of the mean $\mu(x)$. □

[a] https://morrison-lab.github.io/sds/model-validation.html#thm-epe-split-noise

Example 1.9 (Separating the noise at four cylinders)**.** Continue Example 1.4 with the same 2000 fitted lines, at $x = 4$ cylinders, where $\mu(4)$ is the population mean `mpg` of the four-cylinder cars and the noise is how far a single car sits from that population mean. Compute the left side of Equation 5 directly, and then the noise variance plus the model's own expected squared error:

1.0.21 Python

```
y4_sn = mpg_af[cyl_af == 4]
f4_sn = y4_sn.mean()

# The left side: average the squared error over every line and every car.
left_sn = np.mean((y4_sn[None, :] - preds_af[:, None]) ** 2)
noise_sn = np.mean((y4_sn - f4_sn) ** 2)
model_sn = np.mean((f4_sn - preds_af) ** 2)
print(np.round([left_sn, noise_sn, model_sn, noise_sn + model_sn], 2))
```

```
[34.18 31.99  2.18 34.18]
```

1.0.22 R

```
y4_sn <- auto_af$mpg[auto_af$cylinders == 4]
f4_sn <- mean(y4_sn)

left_sn <- mean(outer(preds_af, y4_sn, function(h, y) (y - h)^2))
noise_sn <- mean((y4_sn - f4_sn)^2)
model_sn <- mean((f4_sn - preds_af)^2)
round(c(left = left_sn, noise = noise_sn, model = model_sn,
        sum = noise_sn + model_sn), 2)
```

```
 left noise model   sum
34.18 31.99  2.18 34.18
```

The expected squared error is 34.18, and the noise variance 31.99 plus the model's own expected squared error 2.18 add to 34.18, as Theorem 1.4 says.

Theorem 1.5 (Separating the bias from the model's own error)**.** *Choose an input value x, let $\mu(x)$ be the true mean function of the outcome variable given the input value x in the noise model, and let $\mu_h(x)$ be the average fit of Equation 2. Then the expected squared distance between the truth and the fitted model is the squared bias plus the variance:*

$$\mathbb{E}_{\mathcal{D}_{train}}\left[(\mu(x)-\hat{y})^2\right] = \left(\mu(x)-\mathbb{E}_{\mathcal{D}_{train}}[\hat{y}(x)]\right)^2 + \mathbb{E}_{\mathcal{D}_{train}}\left[\left(\hat{y}-\mathbb{E}_{\mathcal{D}_{train}}[\hat{y}(x)]\right)^2\right]. \tag{6}$$

Proof

Proof. The statistics notes work this as an exercise, separating the bias[a]. □

[a] https://morrison-lab.github.io/sds/model-validation.html#sol-epe-split-bias

Example 1.10 (Separating the bias at four cylinders)**.** Continue Example 1.4 with the same 2000 fitted lines, at $x = 4$ cylinders, where $\mu(4)$ is the population mean `mpg` of the four-cylinder cars. Compute the left side of Equation 6 directly, and then the squared bias plus the variance:

1.0.23 Python

```
f4_sb = mpg_af[cyl_af == 4].mean()

# The left side: average the squared distance from mu(4) over the lines.
left_sb = np.mean((f4_sb - preds_af) ** 2)
bias2_sb = (f4_sb - preds_af.mean()) ** 2
variance_sb = np.mean((preds_af - preds_af.mean()) ** 2)
print(np.round([left_sb, bias2_sb, variance_sb, bias2_sb + variance_sb], 2))
```

```
[2.18 0.32 1.86 2.18]
```

1.0.24 R

```
f4_sb <- mean(auto_af$mpg[auto_af$cylinders == 4])

left_sb <- mean((f4_sb - preds_af)^2)
bias2_sb <- (f4_sb - mean(preds_af))^2
variance_sb <- mean((preds_af - mean(preds_af))^2)
round(c(left = left_sb, bias2 = bias2_sb, variance = variance_sb,
        sum = bias2_sb + variance_sb), 2)
```

```
    left    bias2 variance      sum
    2.18     0.32     1.86     2.18
```

The model's own expected squared error is 2.18, and the squared bias 0.32 plus the variance 1.86 add to 2.18, as Theorem 1.5 says.

Theorem 1.6 (Bias–variance decomposition)**.** *Choose an input value x. By the noise model, a new output variable at x is $Y = \mu(x) + \varepsilon$, where $\mu(x)$ is the value of the true mean function at x and ε is the noise in this new output, whose population mean[a] is 0 by Theorem 1.1. A learning algorithm fits a model to the training set $\mathcal{D}_{train}$. Let $\hat{y}$ denote the model's prediction for the input x, which varies with $\mathcal{D}_{train}$, and let $\mu_h(x)$ denote the average fit of that prediction. Assume that*

- *ε has variance[b] $\sigma^2 < \infty$, and*
- *ε is independent[c] of $\mathcal{D}_{train}$, whose outputs contain their own separate noise draws.*

Let $\mathbb{B}[\hat{y}]$ denote $\mathrm{Bias}(\hat{y})$, the prediction's bias, let $\mathbb{V}[\hat{y}]$ denote $\mathrm{Var}(\hat{y})$, its variance, and let $\mathrm{ESE}(x)$ denote $\mathbb{E}_{\mathcal{D}_{train},\varepsilon}\left[(Y-\hat{y})^2\right]$, the expected squared error[d] of the prediction $\hat{y}$ of Y.
Then $\mathrm{ESE}(x)$ splits into three terms:

$$\mathrm{ESE}(x) = (\mathbb{B}[\hat{y}])^2 + \mathbb{V}[\hat{y}] + \sigma^2. \tag{7}$$

[a] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[b] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance

[c] https://morrison-lab.github.io/pds/independence.html#def-indpt
[d] https://morrison-lab.github.io/sds/model-validation.html#def-epe

Proof

Proof. By Theorem 1.4 and Theorem 1.5, the expected squared error[a] is the sum[b] of the noise variance σ^2, $\left(\mu(x) - \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}(x)]\right)^2$, and $\mathbb{E}_{\mathcal{D}_{\text{train}}}\left[\left(\hat{y} - \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}(x)]\right)^2\right]$. The first is σ^2 itself. The second is the square of the bias of the prediction, because that bias is the average fit of the prediction minus the value of the true mean function of Y given $X = x$, by Theorem 1.2. A square does not change when its base changes sign. The third is the variance of the prediction, by Equation 4. □

[a] https://morrison-lab.github.io/sds/model-validation.html#def-epe
[b] https://morrison-lab.github.io/mds/algebra-sums.html#def-summation

Remark (The decomposition holds for any algorithm)

Remark 1.7 (The decomposition holds for any algorithm). Equation 7 holds for any learning algorithm, and ε need not be Gaussian. The statistics notes prove it as the bias, variance and noise decomposition[a] of the expected squared prediction error.

[a] https://morrison-lab.github.io/sds/model-validation.html#thm-epe-decomposition

Remark (The error decomposition over training sets)

Remark 1.8 (The error decomposition over training sets). The reducible error and residual variance decomposition holds the training set $\mathcal{D}_{\text{train}}$ fixed. Then the fitted model $\hat{y}_{\mathcal{D}_{\text{train}}}$ is fixed, and so is its prediction. That proposition writes the prediction at the input[a] $\underset{\sim}{x}$ as $\hat{y}$. This page writes the prediction at the input x as $\hat{y}$. The prediction $\hat{y}$ depends on $\mathcal{D}_{\text{train}}$, so it changes from one training set to the next. The expected squared error[b] $\text{ESE}(x)$ also takes the expectation[c] over the noise and over training sets. There, the reducible error $(\hat{y} - \mu(x))^2$ varies with $\mathcal{D}_{\text{train}}$. So the expected squared error depends on the reducible error from every training set. Theorem 1.4 splits that expected squared error into two terms. The first term is the noise's variance[d] σ^2. The second term is $\mathbb{E}_{\mathcal{D}_{\text{train}}}\left[(\mu(x) - \hat{y})^2\right]$. By Theorem 1.5, that second term is $(\mathbb{B}[\hat{y}])^2 + \mathbb{V}[\hat{y}]$, the squared bias plus the prediction's variance.

[a] https://morrison-lab.github.io/sds/model-validation.html#def-data-set
[b] https://morrison-lab.github.io/sds/model-validation.html#def-epe
[c] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[d] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance

Example 1.11 (Splitting the error of the cylinders line at four cylinders)**.**
Continue Example 1.4 with the same 2000 fitted lines. The input is $x = 4$ cylinders, and the output Y is the `mpg` of a four-cylinder car, so $\mu(4)$ is the population mean `mpg` of the four-cylinder cars and the noise is how far a single car sits from that population mean. Compute the three terms of Equation 7 and, separately, the expected squared error itself:

1.0.25 Python

```
# Four-cylinder cars: their mean mpg is mu(4), and their spread is the noise.
y4_bv = mpg_af[cyl_af == 4]

bias2_bv = (y4_bv.mean() - preds_af.mean()) ** 2
variance_bv = np.mean((preds_af - preds_af.mean()) ** 2)
noise_bv = np.mean((y4_bv - y4_bv.mean()) ** 2)

# The left side: average the squared error over every line and every car.
error_bv = np.mean((y4_bv[None, :] - preds_af[:, None]) ** 2)
# Its Monte Carlo standard error: each line's mean squared error over the cars is one draw.
line_err_bv = np.mean((y4_bv[None, :] - preds_af[:, None]) ** 2, axis=1)
mc_se_bv = line_err_bv.std(ddof=1) / np.sqrt(len(line_err_bv))
print(np.round([bias2_bv, variance_bv, noise_bv, bias2_bv + variance_bv + noise_bv, error_bv], 2), rou
```

```
[ 0.32  1.86 31.99 34.18 34.18] 0.067
```

1.0.26 R

```
y4_bv <- auto_af$mpg[auto_af$cylinders == 4]

bias2_bv <- (mean(y4_bv) - mean(preds_af))^2
variance_bv <- mean((preds_af - mean(preds_af))^2)
noise_bv <- mean((y4_bv - mean(y4_bv))^2)

# The left side: average the squared error over every line and every car.
err_mat_bv <- outer(preds_af, y4_bv, function(h, y) (y - h)^2)
error_bv <- mean(err_mat_bv)
# Its Monte Carlo standard error: each line's mean squared error over the cars is one draw.
line_err_bv <- rowMeans(err_mat_bv)
mc_se_bv <- sd(line_err_bv) / sqrt(length(line_err_bv))
c(round(c(bias2 = bias2_bv, variance = variance_bv, noise = noise_bv,
          sum = bias2_bv + variance_bv + noise_bv, error = error_bv), 2),
  mc_se = round(mc_se_bv, 3))
```

```
   bias2 variance    noise      sum    error    mc_se
   0.320    1.860   31.990   34.180   34.180    0.067
```

The three terms add to 34.18, which is the expected squared error 34.18 computed directly, as Theorem 1.6 says. That identity is exact for the 2000 lines and the cars used here. As an estimate of the expected squared error over all training sets, 34.18 has a 95% Monte Carlo confidence interval of 34.05 to 34.31. Most of it is the noise variance[a], 32.0: even a model with no bias and no variance would have that expected squared error on a four-cylinder car. The squared bias is only 0.32 and the variance 1.86. The decomposition holds for any learning algorithm, so the same arithmetic would split the error of a tree or a neural network.

[a] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance

Watch

Abu-Mostafa derives the same decomposition in his Caltech lecture on the bias–variance trade-off, from 16:43[a], and he reads bias and variance from learning curves, from 46:16[b] [5], Lecture 8.

[a] https://www.youtube.com/watch?v=zrEyxfl2-a8&t=1003s
[b] https://www.youtube.com/watch?v=zrEyxfl2-a8&t=2776s

Watch

Hastie and Tibshirani follow the bias–variance trade-off[a] [6], video 2.3.

[a] https://www.youtube.com/watch?v=pvcEQfcO3pk

Watch

Weinberger explains the bias–variance decomposition[a] [7], Lecture 19.

[a] https://www.youtube.com/watch?v=zUJbRO0Wavo

Watch

Starmer explains bias and variance[a] [8] without the algebra.

[a] https://www.youtube.com/watch?v=EuBBz3bI-aA

Read

GeeksforGeeks explains the bias–variance trade-off[a] [9].

[a] https://www.geeksforgeeks.org/machine-learning/ml-bias-variance-trade-off/

1.1 Reading the decomposition

Exercise 1.3 (Read the three terms)**.** Equation 7 splits the expected squared error at x into three terms. For each one, say what it measures, which panel of Figure 5 shows it large (if any), and whether a different learning algorithm or more data could make it smaller.

Solution

Solution 1.3. Only two of the three can be changed.

The first term, $\left(\mu(x) - \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}(x)]\right)^2$, is the square of the bias (Definition 1.3) of $\hat{y}$ as an estimate of $\mu(x)$: how far the algorithm's average fit is from the truth. It is a property of the *family* of models, not of any one fit. A straight line fitted to a curve has bias no amount of data will remove, because the average of many lines is still a line. That is the top-left panel of Figure 5, and the quadratic in the top-right panel behaves the same way: the five fits agree with each other and all differ from the truth in the same way. A different family can shrink it; more data cannot. A model whose error is mostly this term is said to *underfit*.

The second term, $\mathbb{E}_{\mathcal{D}_{\text{train}}}\left[\left(\hat{y} - \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}(x)]\right)^2\right]$, is the variance (Definition 1.4) of $\hat{y}$: how far a single fit is from the average fit. It is a property of how strongly the algorithm reacts to its training data. The bottom-right panel of Figure 5 shows five quartics fitted to five samples from the same world, tracking the truth on average and differing strongly between the data points. More data or a less flexible family can shrink it. A model whose error is mostly this term is said to *overfit*.

The third term, σ^2, is the noise in Y itself. No panel of Figure 5 shows it, because the figure plots only fitted curves and the truth, never the noisy data, and σ^2 is the same for every family. It is not a fact about the model or the algorithm, and no method in these notes reduces it. It sets a lower limit: no model, however good, predicts better than σ^2 on average. When a reported error is near that limit, little further improvement is possible; when a reported error is *below* it, something is wrong with how it was measured.

Definition 1.5 (Underfitting)**.** A model is **underfitting** when the bias term of Equation 7 is large enough to dominate its error.

Definition 1.6 (Overfitting)**.** A model is **overfitting** when the variance term of Equation 7 is large enough to dominate its error.
See overfitting[a] in the statistics notes.

[a] https://morrison-lab.github.io/sds/model-validation.html#def-overfitting

rme's overfitting[8] names the same failure by its symptom: a model that fits its training data well but predicts new observations poorly.

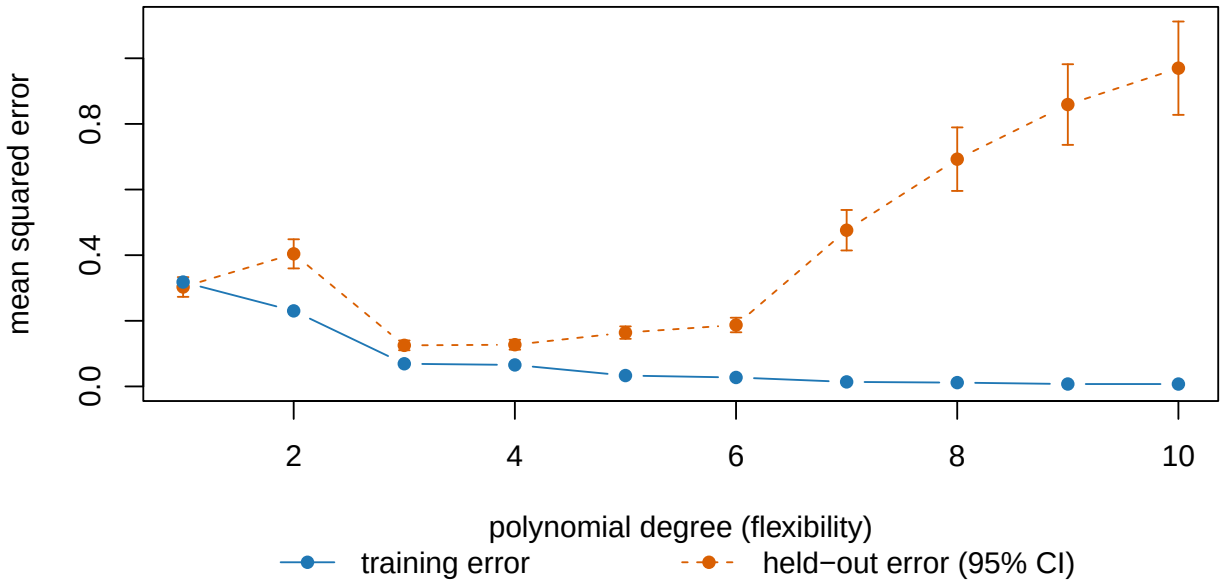


Figure 6: Training error and held-out error of polynomial fits of degree 1 to 10, fitted to 15 noisy points from a smooth curve. Training error only falls as the degree grows. Held-out error, estimated on 500 fresh points with a 95% confidence interval, falls while the fit is underfitting, then rises again as it overfits.

Read

GeeksforGeeks explains underfitting and overfitting[a] [10].

[a] https://www.geeksforgeeks.org/machine-learning/underfitting-and-overfitting-in-machine-learning/

Read

Domingos, a University of Washington professor, gives twelve lessons on generalization, overfitting and feature engineering in A Few Useful Things to Know About Machine Learning[a] [11].

[a] https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf

[8] https://morrison-lab.github.io/rme/chapters/predictor-selection.html#def-overfitting

Exercise 1.4 (Predict weight from height)**.** Suppose we predict a person's weight from their height. Could any model, given enough data, drive its expected squared error to zero?

Solution

Solution 1.4. No. The two are related, and yet no function of height alone will ever return weight exactly, because two people of the same height differ in ways that height does not record. That leftover spread is called the *residual variance*: the part of the output that the inputs do not determine.

Exercise 1.5 (Predict weight from height (part 2))**.** Use the setup of Exercise 1.4.
If not, what would lower the minimum possible error?

Solution

Solution 1.5. Residual variance depends on the features and not on the model, so a better model leaves it unchanged. Record frame size too, and some of the spread can move out of the residual variance and into the part a model can learn. Averaged over the inputs, the residual variance does not rise when a feature is added, and it falls exactly when the feature changes the conditional mean of weight given height.

Definition 1.7 (Residual variance (residual noise variance, irreducible error))**.**
The **residual variance** (also called the **residual noise variance** or the **irreducible error**) at x is the variance[a] of the noise ε in the noise model: the variance of the output Y around its true mean function at x,

$$\sigma^2 \stackrel{\text{def}}{=} \operatorname{Var}(\varepsilon) = \operatorname{Var}(Y \mid X = x) = \mathbb{E}\big[(Y - \mu(x))^2 \mid X = x\big]. \tag{8}$$

[a] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance

Remark (What "irreducible" means)

Remark 1.9 (What "irreducible" means). The residual variance is the part of the expected[a] squared error in Equation 7 that no choice of model or algorithm[b] changes. It is a variance[c], a component of the expected loss. Textbooks call it the "irreducible error" [3, Sec. 2.1.1] [2], equation 7.9. That name can suggest that it is a prediction error[d], although it is a variance.
It is irreducible only given the inputs. It depends on which inputs the model observes. Averaged over new inputs, the residual variance is $\mathbb{E}[\operatorname{Var}(Y \mid X)]$. By the law of total variance[e], adding a feature never raises this average, and lowers it exactly when the feature changes the conditional mean of Y given the other inputs. At a single x, the residual variance can rise or fall when a feature is added.

[a] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[b] https://morrison-lab.github.io/mcs/expressions-algorithms.html#def-algorithm
[c] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance
[d] https://morrison-lab.github.io/sds/prediction.html#def-prediction-error
[e] https://morrison-lab.github.io/pds/variance-covariance.html#thm-total-variance

Remark (Estimating the residual variance)

Remark 1.10 (Estimating the residual variance). The residual variance is a population[a] quantity, which $\hat{\sigma}^2$, computed from the residuals of a fitted model, estimates[b].

[a] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-population
[b] https://morrison-lab.github.io/sds/estimation.html#def-estimate

Remark (Residual variance in the statistics notes)

Remark 1.11 (Residual variance in the statistics notes). The statistics notes define the same residual variance[a] for the prediction at one covariate value x. It is the σ^2 term in their bias, variance and noise decomposition[b] of the expected squared prediction error, which they prove step by step.

[a] https://morrison-lab.github.io/sds/model-validation.html#def-irreducible-error
[b] https://morrison-lab.github.io/sds/model-validation.html#thm-epe-decomposition

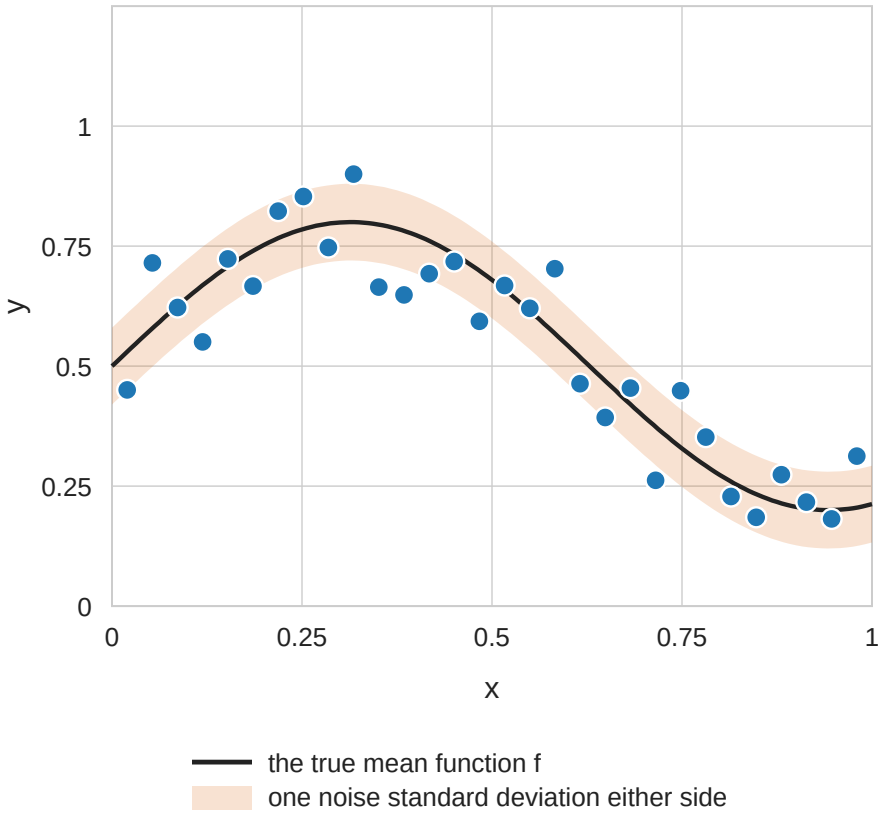


Figure 7: Thirty points drawn from a true mean function plus noise. The band reaches one standard deviation of the noise above and below the true mean function, so its squared half-width is the residual variance σ^2. Even a model that matches the true mean function exactly misses a typical point by about the half-width of the band, because the noise moves each point off the function. No model can predict the observed values more closely than the spread of the noise allows.

Exercise 1.6 (Error without noise)**.** Imagine a data-generating process with no noise at all: $Y = \mu(x)$ exactly, with $\mu(x) = \sin(2.5\pi x)$ on $-1 \le x \le 1$, a curve with two and a half full waves. A student fits polynomials of degree 1, 3, 5 and 7 by least squares to training sets of $n = 8$ points, and tests each fit on new points from the same curve.
The degree-7 polynomial[a] passes through all eight training points. Which polynomial do you expect to predict best on new points?

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-polynomial

Solution

Solution 1.6. Fit each degree to 200 training sets of $n = 8$ points, each an equally spaced grid on $[-1, 1]$ shifted a little further along from one set to the next, and split the error on 401 evenly spaced test points with Equation 7. The grids are fixed, not random, so the R and Python tabs compute the same numbers.

1.1.1 Python

```python
import numpy as np
import pandas as pd

# The target, with no noise: the test labels are exactly mu(x).
def f_nn(x):
    return np.sin(2.5 * np.pi * x)

n_nn, sets_nn = 8, 200
x_test_nn = np.linspace(-1, 1, 401)
rows_nn = []
for degree in (1, 3, 5, 7):
    fits, train_err = [], []
    for k in range(sets_nn):
        # Each training set is the same equally spaced grid, shifted a little.
        # Shifting by hand, rather than drawing random points,
        # makes the numbers identical in the R tab.
        x = -1 + 2 / n_nn * (np.arange(n_nn) + (k + 0.5) / sets_nn)
        coef = np.linalg.lstsq(np.vander(x, degree + 1), f_nn(x), rcond=None)[0]
        fits.append(np.vander(x_test_nn, degree + 1) @ coef)
        train_err.append(np.mean((np.vander(x, degree + 1) @ coef - f_nn(x)) ** 2))
    fits = np.array(fits)
    avg_fit = fits.mean(axis=0)
    # The three terms of the decomposition. Noise is zero, so there are only two.
    bias2 = np.mean((avg_fit - f_nn(x_test_nn)) ** 2)
    variance = np.mean(fits.var(axis=0))
    rows_nn.append((degree, np.mean(train_err), bias2, variance, bias2 + variance))
print(pd.DataFrame(rows_nn, columns=["degree", "train", "bias2", "variance", "error"]).round(3))
```

```
   degree  train  bias2  variance  error
0       1  0.494  0.500     0.006  0.506
1       3  0.430  0.441     0.016  0.457
2       5  0.102  0.743     0.389  1.132
3       7  0.000  1.465     0.862  2.327
```

1.1.2 R

```r
# The target, with no noise: the test labels are exactly mu(x).
f_nn <- function(x) sin(2.5 * pi * x)

n_nn <- 8
sets_nn <- 200
x_test_nn <- seq(-1, 1, length.out = 401)
nn_results <- do.call(rbind, lapply(c(1, 3, 5, 7), function(degree) {
  # Each training set is the same equally spaced grid, shifted a little.
  # Shifting by hand, rather than drawing random points,
  # makes the numbers identical in the Python tab.
  fits <- sapply(seq_len(sets_nn) - 1, function(k) {
    x <- -1 + 2 / n_nn * (seq_len(n_nn) - 1 + (k + 0.5) / sets_nn)
    design <- outer(x, 0:degree, "^")
    coef <- qr.solve(design, f_nn(x))
    c(test = drop(outer(x_test_nn, 0:degree, "^") %*% coef),
      train = mean((design %*% coef - f_nn(x))^2))
  })
  test_fits <- fits[seq_along(x_test_nn), ]
  avg_fit <- rowMeans(test_fits)
  # The three terms of the decomposition. Noise is zero, so there are only two.
  bias2 <- mean((avg_fit - f_nn(x_test_nn))^2)
  variance <- mean(apply(test_fits, 1, var) * (sets_nn - 1) / sets_nn)
  data.frame(degree = degree, train = mean(fits["train", ]),
             bias2 = bias2, variance = variance, error = bias2 + variance)
}))
round(nn_results, 3)
```

```
# A tibble: 4 x 5
  degree train bias2 variance error
   <dbl> <dbl> <dbl>    <dbl> <dbl>
1      1 0.494 0.5      0.006 0.506
2      3 0.43  0.441    0.016 0.457
3      5 0.102 0.743    0.389 1.13
4      7 0     1.46     0.862 2.33
```

The cubic has the lowest error, 0.46, and the line is close behind at 0.51. The septic, which fits all eight training points exactly (training error 0.00), has the highest, 2.33.

Expect the low degrees[a] to win, though the degree-7 fit has the best training error. With eight points, the degree-7 fit has no freedom left over, so each shifted training set bends it differently (variance[b] 0.86).

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-polynomial-degree
[b] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance

Exercise 1.7 (Error without noise (part 2))**.** Use the setup of Exercise 1.6.
The residual variance σ^2 of Definition 1.7 is zero here. Where, then, does the error of the best polynomial[a] come from?

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-polynomial

Solution

Solution 1.7. The error comes from the model, not from the world. A line, or even a cubic, cannot follow two and a half waves, so every fit misses the curve in the same way (squared bias[a] 0.50 for the line). A flexible polynomial[b] passes through the curve at the training points and still goes wrong between them, because eight points are too few to pin it down there. Either way, the part of μ the

model cannot learn from this much data acts on the fit like noise in the labels, which the definition of deterministic noise below makes precise.

[a] https://morrison-lab.github.io/sds/estimation.html#def-bias
[b] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-polynomial

Figure 8 draws the fits behind the table above: the curves each training set produces, their average, and the error by degree.

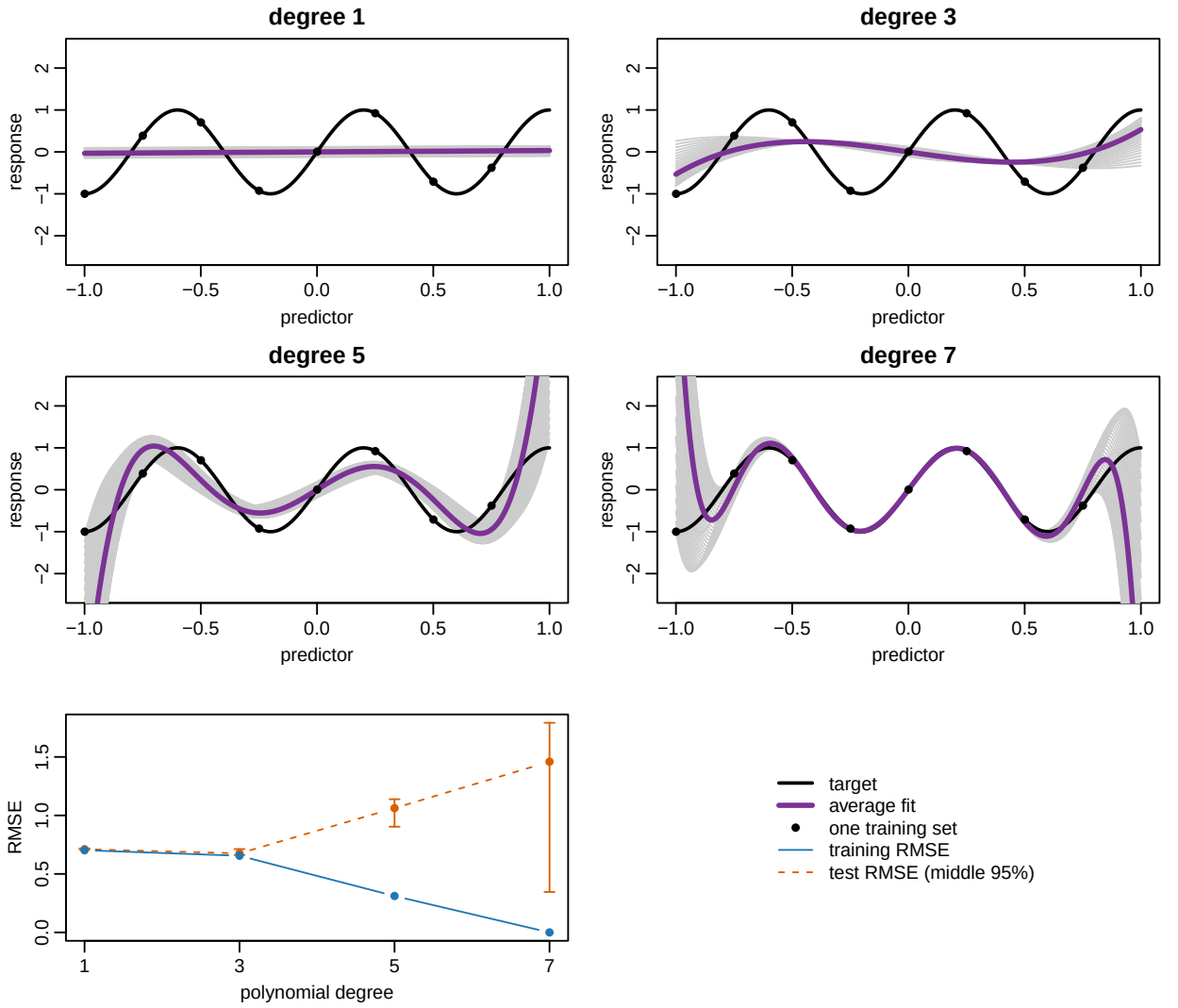


Figure 8: Fits to a noiseless curve, μ, from 200 shifted grids of 8 points. Top: for each polynomial degree, 40 of the 200 fitted curves (grey), their average (colour) and one training set (dots). The line and the cubic miss the curve in the same way every time; the degree-7 polynomial passes through every training set and still swings wildly between the points. Bottom: training and test root mean squared error (RMSE) by degree, with the bars spanning the middle 95% of the 200 training sets.

Definition 1.8 (Best member of a model family)**.** The **best member** h^* of a mean-function model family[a] $\mathcal{H}$ for a target function μ is the member h of $\mathcal{H}$ with the smallest mean squared difference from μ, $\mathbb{E}[(\mu(X) - h(X))^2]$, where the random predictor X is a draw of the predictor x from the population[b].

[a] https://morrison-lab.github.io/sds/model-validation.html#def-model-family
[b] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-population

Example 1.12 (The best line for a curved target)**.** Let the target μ be a sine curve, and let $\mathcal{H}$ be the family of straight lines. No line follows a sine curve, but some lines are closer to it than others, averaging the squared gap over the population of x. The closest line is the best member h^*. A line fitted to a training set is a different object, because it depends on the data and so usually differs from h^*.

Definition 1.9 (Deterministic noise)**.** The **deterministic noise** of a mean-function model family[a] $\mathcal{H}$ at x is the part of $\mu(x)$ that the family cannot express: $\mu(x) - h^*(x)$, where h^* is the best member of $\mathcal{H}$ (Definition 1.8).

[a] https://morrison-lab.github.io/sds/model-validation.html#def-model-family

Remark (Deterministic versus stochastic noise)

Remark 1.12 (Deterministic versus stochastic noise). Deterministic noise differs from the noise ε in the labels of the noise model, which we call *stochastic noise* when we need to tell the two apart, in two ways:

- Deterministic noise depends on the family: a richer family leaves less of μ unexpressed.
- Deterministic noise is fixed for a given x: drawing a new training set does not change it.

Watch

Abu-Mostafa separates stochastic from deterministic noise in his Caltech lecture on overfitting [5], Lecture 11; the same chapter of his book, Abu-Mostafa et al. [1, Ch. 4], gives the written treatment.

Exercise 1.8 (Why the ranking was wrong)**.** The table of empirical risk by degree ranked the fitted polynomials[a] by their training error, and the quartic came first. Which of the three terms of Equation 7 does training-set error measure?

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-polynomial

Solution

Solution 1.8. None of them. Training-set error is an optimistic estimate of the expected error on the left-hand side of Equation 7.

Exercise 1.9 (Why the ranking was wrong (part 2))**.** Use the setup of Exercise 1.8.
Why does the ranking by training-set error favor the most flexible family?

Solution

Solution 1.9. Training error grows more optimistic with flexibility[a], because a more flexible model can fit the particular noise in its training set. So the quantity the table of empirical risk by degree ranks by improves for exactly the reason the variance[b] term worsens.

[a] https://morrison-lab.github.io/sds/model-validation.html#def-flexibility
[b] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance

Remark (The bias-variance tradeoff)

Remark 1.13 (The bias-variance tradeoff). Across the families of Figure 5, more flexibility gives lower bias but higher variance: the line and the quadratic miss the truth in the same way every time, the cubic

follows it closely, and the quartics swing apart from one another. Because of that trade, model selection needs careful study and is not a routine step. For the crime data, ranking the polynomials by training error put the quartic first because it is the most flexible, not because it predicts best (Exercise 1.8), so choosing the degree takes more than comparing the training errors.

Exercise 1.10 (Changes along the tradeoff)**.** List changes you could make to the data or the model that shrink one of the two correctable terms of Equation 7. For each, say in which direction it changes the bias term and in which direction it changes the variance term.

Solution

Solution 1.10. Both correctable terms change in opposite directions as flexibility changes, so the usual changes are:

- more data, which shrinks variance and leaves bias alone;
- a smaller model family, which lowers variance and raises bias;
- a larger model family, which does the reverse;
- keeping the family but penalizing the size of the fitted parameters, which deliberately accepts a little bias in exchange for a larger reduction in variance.

Exercise 1.11 (Why would anyone choose the less flexible model?)**.** A straight line can only tilt; a high-degree polynomial can bend almost any way. Give two reasons the city in the crime example might still prefer the straight line: one that applies if it wants to *explain* how crime varies with income, and one that applies even if it only wants to *predict*.

Solution

Solution 1.11.

- To explain, the line is readable. Its slope says how much the predicted crime rate changes per unit of income, and its sign says in which direction. The quartic of the exercise "Which model was actually better?" has no single number like that: each of its five coefficients multiplies a different power of income, so none of them alone says how crime changes with income, even when the curve predicts well. James et al. [12, Sec. 2.1.3] calls this the trade-off between flexibility and interpretability.
- To predict, the line may simply be more accurate. With five neighborhoods, a flexible fit spends its freedom chasing those five points. Figure 5 shows what that costs: flexible fits to different samples from the same population disagree wildly with each other, so a prediction for a sixth neighborhood depends on which five happened to be sampled. That is the variance term of Equation 7, so the quartic's perfect fit to its training set in the exercise "Which model was actually better?" was no evidence that it predicts well. James et al. [12, Sec. 2.1.3] makes the same point: even when only prediction matters, a less flexible method often predicts more accurately, because a very flexible one can overfit (Definition 1.6).

Neither reason makes the line the right choice by default. If the true relationship bends, the line's bias does not shrink with more data.

Definition 1.10 (Regularization)**. Regularization** is fitting a model by minimizing its empirical risk plus a penalty on the size of its parameters [13]. With a penalty function $\text{pen}(\theta) \geq 0$ and a penalty weight $\lambda \geq 0$, regularization minimizes the objective plus the weighted penalty,

$$\mathcal{J}(\theta) + \lambda \,\text{pen}(\theta). \tag{9}$$

We cover regularization in detail when we study models whose parameters should be penalized.

Example 1.13 (Regularizing the crime-data quartic)**.**
For the polynomial[a] fits to the crime data in the table of empirical risk by degree, a penalty on the coefficients makes a large coefficient on x^4 costly, so the fitted quartic moves closer to a flatter curve that bends to fit each of the five neighborhoods less closely.

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-polynomial

Definition 1.11 (Penalty weight (regularization strength))**.** A **penalty weight** $\lambda \geq 0$, also called the **regularization strength**, is the number that scales a penalty $\text{pen}(\theta) \geq 0$, a nonnegative measure of the size of a model's parameters, when the penalty is added to the training objective in regularization. The value $\lambda = 0$ leaves the objective unchanged, and a larger λ penalizes large parameters more heavily.

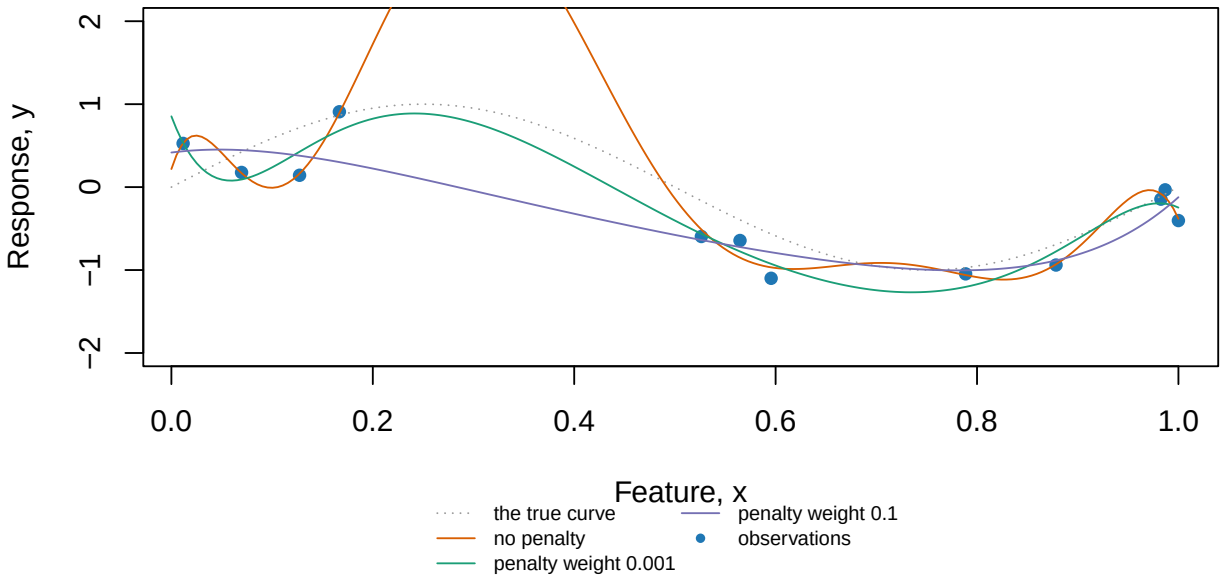


Figure 9: A degree-9 polynomial fitted to 12 noisy points from a smooth curve, with a penalty on the size of its coefficients. With no penalty, the fit passes close to every point and swings wildly between them. A larger penalty weight pulls the fit toward a smoother curve that follows the points less closely.

Read

GeeksforGeeks explains regularization[a] [14].

[a] https://www.geeksforgeeks.org/machine-learning/regularization-in-machine-learning/

Exercise 1.12 (Read the response to a change, name the term)**.** Each of the following describes what happened when you changed one thing about a model and scored it honestly again. For each, say which term of Equation 7 the response points at, and what you would try next.
Strengthening the regularization penalty raises training error a little and lowers held-out error.

Solution

Solution 1.12. **Variance**[a]**.** A stronger penalty limits how closely the fitted model can adapt to the peculiarities of one training set, so it shrinks $\mathbb{E}_{\mathcal{D}_{\text{train}}}\left[\left(\hat{y} - \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}(x)]\right)^2\right]$ at the cost of a little bias[b]. Held-out error fell, so the variance removed outweighed the bias added. Keep raising the penalty while held-out error keeps falling, and stop where it starts to rise.

[a] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance
[b] https://morrison-lab.github.io/sds/estimation.html#def-bias

Exercise 1.13 (Read the response to a change, name the term (part 2))**.** Use the setup of Exercise 1.12. Doubling the training set does not change held-out error, but switching to a more flexible model family[a] lowers training and held-out error together.

[a] https://morrison-lab.github.io/sds/model-validation.html#def-model-family

Solution

Solution 1.13. **Bias**[a]**.** More data cuts variance[b], because each fit then depends less on the peculiarities of one training set, but it does not move the average fit μ_h toward the truth. A held-out error that does not move with twice the data is therefore not mainly coming from variance. A more flexible family can place μ_h closer to the truth, so both errors fell when you switched. Keep adding flexibility, or better features, while held-out error keeps falling.

[a] https://morrison-lab.github.io/sds/estimation.html#def-bias
[b] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance

Exercise 1.14 (Read the response to a change, name the term (part 3))**.** Use the setup of Exercise 1.12. Training error and held-out error are both small and roughly equal, and collecting ten times as much data changes neither.

Solution

Solution 1.14. **Residual variance**, most likely. Both errors agree, so variance[a] is small; both are small, so bias[b] is small; and more data moving neither says the remaining error is not a property of the model at all. There is nothing left to try on the modeling side. The remaining options are outside the model: measure the inputs[c] more precisely, or find a feature that carries information the current ones do not.
Case 3 is the one students most often keep working on. Recognizing a finished model is as much a skill as improving an unfinished one, and σ^2 is unknown, so "most likely" is as strong as the diagnosis gets.

[a] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance
[b] https://morrison-lab.github.io/sds/estimation.html#def-bias
[c] https://morrison-lab.github.io/sds/model-validation.html#def-data-set

Exercise 1.15 (A perfect algorithm)**.** Suppose an algorithm has zero bias and zero variance at every x, so every training set gives it the same fit, and that fit is f. Can the risk of its fitted model still be nonzero? Use Equation 7 to say why or why not, and say what each of the two zeros means in words.

Solution

Solution 1.15. Yes: the expected squared error at x is then σ^2, because the other two terms of Equation 7 are zero. The risk is nonzero whenever the noise is.
Zero variance means the algorithm returns the same fitted model on every training set, so it ignores its

data. Zero bias means that model is the truth, $\mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}(x)] = \mu(x)$. Both zeros together describe an algorithm with nothing left to learn, and even that algorithm cannot predict Y more closely than the noise allows.
Students tend to read "zero bias and zero variance" as "zero error". Point back to Definition 1.7.

Remark (Bias and variance for a classifier)

Remark 1.14 (Bias and variance for a classifier). The decomposition of Theorem 1.6 is for squared error, and it does not carry over to a classifier's zero-one loss as an exact sum. For a classifier the words are an analogy. The Bayes error plays the part of the residual variance: no classifier using the same features does better. Under zero-one loss, bias and variance combine multiplicatively rather than additively [15, Sec. 6]. A classifier's error at x depends on the bias only through the boundary bias, which records how far the average fit lies from the decision threshold and on which side. A fit that is very biased but on the correct side of the threshold is not penalized for that bias. In that case, reducing variance alone moves the error toward the Bayes error. Friedman puts it as "variance tends to dominate bias for classification" [15, Sec. 7], p. 63. He also shows the opposite case: where the average fit is on the wrong side of the threshold, reducing variance makes the error worse [15, Sec. 7], p. 62.

The course website's version of this page has an interactive plot here.

Reading

- Hastie et al. [2], Section 2.9, is the bias-variance trade-off[9]. It is worth reading after the bias-variance decomposition rather than before.
- James et al. [3], Section 2.2.2, is the bias-variance decomposition, in a less technical way.

Watch

Andrew Ng's short lectures for DeepLearning.AI cover the same ideas for neural networks: how to read underfitting and overfitting as bias and variance [16], and a basic recipe for deciding what to change next [17].

References

[1] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, *Learning from data: A short course*. AMLBook, 2012. Available: https://amlbook.com/

[2] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: Data mining, inference, and prediction*, 2nd ed. Springer, 2009. Available: https://hastie.su.domains/ElemStatLearn/

[3] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning with applications in R*, 2nd ed. Springer, 2021. Available: https://www.statlearning.com/

[4] G. van den Burg, *SyncRNG: A synchronized Tausworthe RNG for R and Python*. 2024. doi: 10.32614/CRAN.package.SyncRNG[10].

[5] Y. S. Abu-Mostafa, "Learning from data: CS 156 lectures." Caltech, lecture videos on YouTube, 2012. Available: https://www.youtube.com/playlist?list=PLD63A284B7615313A

[6] T. Hastie, R. Tibshirani, and J. Taylor, "Statistical learning with python." Stanford Online, lecture videos on YouTube, 2023. Available: https://www.youtube.com/playlist?list=PLoROMvodv4rPP6braWoRt5UCXYZ71GZIQ

[7] K. Weinberger, "CS4780: Machine learning for intelligent systems, spring 2017." Cornell University, lecture videos on YouTube, 2017. Available: https://www.youtube.com/playlist?list=PLl8OlHZGYOQ7bkVbuRthEsaLr7bONzbXS

[9] https://morrison-lab.github.io/sds/model-validation.html#def-bias-variance-tradeoff
[10] https://doi.org/10.32614/CRAN.package.SyncRNG

[8] J. Starmer, "Machine learning fundamentals: Bias and variance." StatQuest, video on YouTube, 2018. Available: https://www.youtube.com/watch?v=EuBBz3bI-aA

[9] GeeksforGeeks, "Bias-variance trade off - machine learning." GeeksforGeeks, tutorial article, 2026. Available: https://www.geeksforgeeks.org/machine-learning/ml-bias-variance-trade-off/

[10] GeeksforGeeks, "Underfitting and overfitting in ML." GeeksforGeeks, tutorial article, 2025. Available: https://www.geeksforgeeks.org/machine-learning/underfitting-and-overfitting-in-machine-learning/

[11] P. Domingos, "A few useful things to know about machine learning," *Communications of the ACM*, vol. 55, no. 10, pp. 78–87, 2012, doi: 10.1145/2347736.2347755[11].

[12] G. James, D. Witten, T. Hastie, R. Tibshirani, and J. Taylor, *An introduction to statistical learning: With applications in Python*. Springer, 2023. doi: 10.1007/978-3-031-38747-0[12].

[13] Wikipedia contributors, "Regularization (mathematics) — Wikipedia, the free encyclopedia." 2026. Available: https://en.wikipedia.org/wiki/Regularization_(mathematics)

[14] GeeksforGeeks, "Regularization in machine learning." GeeksforGeeks, tutorial article, 2026. Available: https://www.geeksforgeeks.org/machine-learning/regularization-in-machine-learning/

[15] J. H. Friedman, "On bias, variance, 0/1—loss, and the curse-of-dimensionality," *Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 55–77, 1997, doi: 10.1023/A:1009778005914[13].

[16] A. Ng, "Bias/variance." DeepLearning.AI, Deep Learning Specialization, Course 2; video on YouTube, 2017. Available: https://www.youtube.com/watch?v=SjQyLhQIXSM

[17] A. Ng, "Basic recipe for machine learning." DeepLearning.AI, Deep Learning Specialization, Course 2; video on YouTube, 2017. Available: https://www.youtube.com/watch?v=C1N_PDHuJ6Q

[11] https://doi.org/10.1145/2347736.2347755
[12] https://doi.org/10.1007/978-3-031-38747-0
[13] https://doi.org/10.1023/A:1009778005914