Validating Predictions

Last modified: 2026-11-12 22:02:32 (PST)

Data sets, input matrices and outcome vectors

Definition 0.1 (Data set)**.** A **data set** is a finite collection of n labelled observations, that is, n pairs $(\underset{\sim}{x}_i, y_i)$, where $\underset{\sim}{x}_i$ is an input vector and y_i is the corresponding outcome:

$$\mathcal{D} \stackrel{\text{def}}{=} \{(\underset{\sim}{x}_i, y_i) : i = 1, \dots, n\}. \tag{1}$$

Remark (The sets we split a data set into)

Remark 0.1 (The sets we split a data set into). We divide a data set into subsets, each with its own job. "Set" in the names training set, validation set and test set is short for "subset of the data set". A training/validation/test split assigns each observation to one of them, and the training set is the subset a fitting procedure is applied to.

Definition 0.2 (Input matrix)**.** The **input matrix X** of a data set $\mathcal{D}$ stacks the n inputs $\underset{\sim}{x}_i$ as rows, with one column for each of the p input variables:

$$\mathbf{X} = \begin{bmatrix} \underset{\sim}{x}_1^\top \\ \vdots \\ \underset{\sim}{x}_n^\top \end{bmatrix}. \tag{2}$$

Definition 0.3 (Outcome vector)**.** The **outcome vector** $\underset{\sim}{y}$ of a data set $\mathcal{D}$ stacks its n outcomes y_i:

$$\underset{\sim}{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}. \tag{3}$$

Example 0.1 (The input matrix and outcome vector of the first five cars)**.** In the `mtcars` data frame, take the fuel economy `mpg` as the outcome y and the gross horsepower `hp` and weight `wt` as the $p = 2$ inputs. The first five cars give a data set of $n = 5$ observations.

```
# take the first five cars from the mtcars data frame
cars5 <- head(mtcars, 5)

# stack the two inputs as columns, one row per car
X_cars5 <- as.matrix(cars5[, c("hp", "wt")])

# stack the outcomes into one vector
y_cars5 <- cars5$mpg

# show the input matrix and the outcome vector side by side
cbind(X_cars5, mpg = y_cars5)
#>                    hp    wt  mpg
#> Mazda RX4         110 2.620 21.0
#> Mazda RX4 Wag     110 2.875 21.0
#> Datsun 710         93 2.320 22.8
#> Hornet 4 Drive    110 3.215 21.4
#> Hornet Sportabout 175 3.440 18.7
```

The input matrix `X_cars5` has 5 rows, one per car, and 2 columns, one per input variable. The outcome vector `y_cars5` has 5 entries, one per car.

Fitting and scoring models

Definition 0.1 (Covariate)**.** A **covariate** x is a variable whose value is used to predict the outcome y.

Example 0.1 (A car's weight)**.** To predict a car's fuel economy y, the car's weight is a covariate x.

Definition 0.2 (Mean function)**.** The **mean function** μ of an outcome Y given a covariate X maps each covariate value x to the mean of Y given $X = x$:

$$\mu(x) \stackrel{\text{def}}{=} \mathbb{E}[Y \mid X = x]. \tag{4}$$

Definition 0.3 (Mean-function model family)**.** A **mean-function model family** $\mathcal{H}$ is a set of functions, each of which maps a covariate value x to a candidate for the value $\mu(x)$ of the mean function at x.

Example 0.2 (Polynomials of degree at most d)**.** The polynomials of degree at most d form a mean-function model family $\mathcal{H}$: each polynomial maps a covariate value x to a candidate for the value of the mean function at x, and the members differ in their coefficients.

Definition 0.4 (Objective function)**.** An **objective function** $\mathcal{J}$ is a rule that gives each member h of a model family $\mathcal{H}$ a number computed from a data set $\mathcal{D}$, where a smaller number means a better fit to the data:

$$h \mapsto \mathcal{J}(h; \mathcal{D}), \quad h \in \mathcal{H}. \tag{5}$$

Example 0.3 (Residual sum of squares)**.** For the polynomials h in Example 0.2, the residual sum of squares is an objective function $\mathcal{J}$: it gives each polynomial the sum over the observations of its squared difference from the observed outcome.

Example 0.4 (Exact and exhaustive searches)**.** For the residual sum of squares over the polynomials in Example 0.3, solving the normal equations is an optimization algorithm $\mathcal{A}$ that finds the unique minimizer in one step.
Trying every member of a finite model family and keeping the one with the smallest value of the objective function is another, and it ends after the last member.

Definition 0.5 (Random seed)**.** A **random seed** $\mathfrak{s}$ is a number that fixes the sequence of random numbers a computer generates: the same seed gives the same sequence.
Source: Wikipedia contributors [1].

Example 0.5 (Reproducing a random split)**.** Randomly splitting a data set in half twice with the same random seed $\mathfrak{s}$ gives the same two halves. Splitting it with two different seeds gives two different splits.

Definition 0.6 (Preprocessing)**.** The **preprocessing** D of a fitting procedure is the map from covariate values x to the features $D(x)$ that the procedure's model family uses as inputs.

Example 0.6 (Standardizing a covariate)**.** Standardization is a preprocessing D: it maps each covariate value x to its standardized value.

Definition 0.7 (Feature)**.** A **feature** is one entry of the output $D(x)$ of the preprocessing D: a value that the model family receives as input. With no preprocessing, the features are the covariates.

Example 0.7 (Features of a quadratic fit)**.** Let the one covariate be a patient's age x. The preprocessing $D(x) = [1, x, x^2]^\top$ has three features: 1, x and x^2. The covariate is one number, and the model family receives three.

Definition 0.8 (Tuning parameter)**.** A **tuning parameter** is a setting λ that the analyst fixes before the model is fitted, rather than one computed from the data, and that selects one prediction function from a family:

$$\hat{y}(x) = \hat{y}_\lambda(x). \tag{6}$$

Source: Wikipedia contributors [2].

Definition 0.9 (Fitted model)**.** A **fitted model** $\hat{h}$ is a member of a model family $\mathcal{H}$ that has been chosen to describe a data set $\mathcal{D}$.

Example 0.8 (A fitted line)**.** A line chosen to describe the weights and fuel economies of the cars in the `mtcars` data set $\mathcal{D}$ is a fitted model $\hat{h}$.

Definition 0.10 (Prediction rule)**.**
A **prediction rule** is a fitted model $\hat{h}$ that is used to predict the outcome at a new covariate value x. Its prediction is $\hat{h}(x)$.
In words, the prediction rule takes a new covariate value and returns the fitted model's value there.

Example 0.9 (A fitted line used to predict)**.** Take the line from Example 0.8. Using it to predict the fuel economy of a car that is not in the `mtcars` data set makes it a prediction rule. Using it only to describe the cars in the data set makes no prediction, so there it is a fitted model and not a prediction rule.

Definition 0.11 (Fitting procedure)**.** A **fitting procedure** g is a function that, given any data set $\mathcal{D}$, produces the fitted model $\hat{h} = g(\mathcal{D})$:

$$\mathcal{D} \mapsto \hat{h} = g(\mathcal{D}). \tag{7}$$

A fitting procedure is specified by these inputs:

- model family $\mathcal{H}$;
- objective function $\mathcal{J}$;
- optimization algorithm $\mathcal{A}$;
- random seed $\mathfrak{s}$;
- tuning parameters λ;
- preprocessing D.

Example 0.10 (Inputs of least-squares polynomial fitting)**.** The least-squares fitting procedure g for polynomials of degree d is specified by these inputs:

- model family $\mathcal{H}$: the polynomials of degree at most d;
- objective function $\mathcal{J}$: the residual sum of squares;
- optimization algorithm $\mathcal{A}$: solving the normal equations;
- random seed $\mathfrak{s}$: none, since nothing is random;
- tuning parameter λ: the degree d;
- preprocessing D: none.

Definition 0.12 (Training set)**.** A **training set** is the data to which a fitting procedure is applied: n observations, written as an ordered list indexed by a set I of n observation numbers:

$$\mathcal{D}_{\text{train}} \stackrel{\text{def}}{=} ((\underset{\sim}{x}_i, y_i))_{i \in I}, \quad |I| = n. \tag{8}$$

Source: Wikipedia contributors [3].

Remark (Source: Training set)

Remark 0.1 (Source: Training set). Hastie et al. [4, Sec. 7.2], who also write the training set as $\mathcal{T}$.

Remark (Notation for the fitted model from a training set)

Remark 0.2 (Notation for the fitted model from a training set). The fitted model $\hat{y}_{\mathcal{D}_{\text{train}}}$ that a fitting procedure g produces from the training set $\mathcal{D}_{\text{train}}$ is written $\hat{y}$, without the

subscript when the training set is the one in use:

$$\hat{y} \stackrel{\text{def}}{=} \hat{y}_{\mathcal{D}_{\text{train}}}. \tag{9}$$

Definition 0.13 (Prediction function family (hypothesis class, hypothesis space))**.** A **prediction function family** $\mathcal{Y}$ (also called a **hypothesis class** or a **hypothesis space**) is a set of functions, each of which maps a covariate value x to a prediction of the outcome.

Example 0.11 (Polynomials as prediction functions)**.** Predicting the outcome at a covariate value x by the value at x of a polynomial of degree at most d makes the polynomials of Example 0.2 a prediction function family $\mathcal{Y}$. The same functions are the members of a mean-function model family $\mathcal{H}$ when each one is a candidate for the mean function, and the members of $\mathcal{Y}$ when each one is a candidate for how to predict.

Definition 0.14 (New observation)**.** A **new observation** (X_*, Y_*) relative to a training set $\mathcal{D}_{\text{train}}$ is a covariate value and an outcome drawn from the same population as the observations of $\mathcal{D}_{\text{train}}$, and statistically independent of $\mathcal{D}_{\text{train}}$. The subscript $*$ marks a quantity of the new observation, as in X_* and Y_*, and distinguishes it from the indexed observations (x_i, y_i) of $\mathcal{D}_{\text{train}}$. It is the new observation of the pds notes, where the data used to choose the prediction function is $\mathcal{D}_{\text{train}}$, to which a fitting procedure is applied.

Example 0.12 (A new observation for the first five cars)**.** Take the first five cars of `mtcars` as the training set, with the gross horsepower `hp` and weight `wt` as the covariate values and the fuel economy `mpg` as the outcome, as in Example 0.1. Treat the sixth car as a new observation: a car from the same population that was not used to form the training set.

```
# the sixth car is not among the five training cars
new_car <- mtcars |>
  dplyr::slice(6) |>
  dplyr::select(hp, wt, mpg)
new_car
#>          hp   wt  mpg
#> Valiant 105 3.46 18.1
```

Its covariate value is $X_* = (105, 3.46)$ and its outcome is $Y_* = 18.1$.

Definition 0.15 (Predictive model)**.** A **predictive model** is a model family or fitted model whose purpose is to predict the outcomes of new observations. Its quality is judged by its prediction error on new observations, not by whether its form describes how the outcome arises.

Example 0.13 (Polynomials as predictive models)**.** The polynomials of Example 0.11 are a predictive model when we choose among them by their prediction error on new observations. They are not a scientific model of the outcome: a polynomial of high degree can predict well without describing any physical condition or change.

Definition 0.16 (Noise)**.** Let (X, Y) be the covariate and the outcome of one observation from a population.
The **noise** ε of the observation is the deviation of the outcome Y from the value of the mean function $\mu(X)$ evaluated at Y's own covariate, X:

$$\varepsilon \stackrel{\text{def}}{=} Y - \mu(X). \tag{10}$$

Example 0.14 (The noise of one observation)**.** Suppose an observation has the covariate value $X = x$ with mean function value $\mu(x) = 5$. If its outcome is $Y = 5.3$, its noise is $\varepsilon = 5.3 - 5 = 0.3$. If its outcome is $Y = 4.6$, its noise is $\varepsilon = 4.6 - 5 = -0.4$. The observation may belong to a training set or be a new observation: the definition does not distinguish them.

Theorem 0.1 (The noise has population mean zero)**.** *Given $X_* = x$, the noise ε has population mean 0:*

$$\mathbb{E}[\varepsilon \mid X_* = x] = 0. \tag{11}$$

Proof

Proof. Take every expectation to be conditional on $X_* = x$. The value $\mu(x)$ is a constant given $X_* = x$.

$$\begin{aligned}
\mathbb{E}[\varepsilon] &= \mathbb{E}[Y_* - \mu(x)] && \text{(definition of the noise)} \\
&= \mathbb{E}[Y_*] - \mu(x) && \text{(linearity of expectation; a constant is its own mean)} \\
&= \mu(x) - \mu(x) && \text{(definition of the mean function)} \\
&= 0 && (a - a = 0)
\end{aligned}$$

□

Example 0.15 (A new outcome that is 4 or 6 with equal probability)**.** Suppose a new outcome at x is 4 or 6, each with probability $1/2$. Its population mean is $\mu(x) = 5$, so its noise is -1 or $+1$, each with probability $1/2$. The population mean of the noise is $(-1) \cdot \frac{1}{2} + 1 \cdot \frac{1}{2} = 0$, as Theorem 0.1 says.

Definition 0.17 (Loss of a fitted model on an observation)**.** The **loss** of a fitted model $\hat{y}$ on an observation (x_i, y_i) is the cost, under a loss function L, of its prediction $\hat{y}(x_i)$ of the outcome y_i:

$$\ell_i(\hat{y}) \stackrel{\text{def}}{=} L(y_i, \hat{y}(x_i)). \tag{12}$$

The subscript names the observation, a member of a family of losses, and the parentheses hold the fitted model the loss is evaluated at. The index $*$ names the new observation (X_*, Y_*), so $\ell_*(\hat{y}) = L(Y_*, \hat{y}(X_*))$. When the fitted model is clear from context, we write ℓ_i for $\ell_i(\hat{y})$.

Remark (Squared-error loss)

Remark 0.3 (Squared-error loss). Each error defined below holds for any loss. The examples, exercises and the bias-variance decomposition use *squared-error loss*, $L(y, \hat{y}) = (\hat{y} - y)^2$, unless they say otherwise. Then the loss on an observation is the square of its prediction error, $\ell_i = \varepsilon_{\text{pred}_i}^2$, and each average loss below is a sample mean squared error.

Definition 0.18 (Generalization error)**.** The **generalization error** of a fitted model $\hat{y}$ fitted to a training set $\mathcal{D}_{\text{train}}$ is its expected loss ℓ_* on a new observation (X_*, Y_*), with $\mathcal{D}_{\text{train}}$ held fixed:

$$\text{Err}_{\mathcal{D}_{\text{train}}} \stackrel{\text{def}}{=} \mathbb{E}[\ell_* \mid \mathcal{D}_{\text{train}}]. \tag{13}$$

Source: Wikipedia contributors [5].

Remark (Source: Generalization error)

Remark 0.4 (Source: Generalization error). The risk form follows Brian Hutchinson's Fall 2025 lecture notes; the conditional form, with the training set held fixed, follows Hastie et al. [4, Sec. 7.2].

Definition 0.19 (Held-out set (hold-out set))**.** A **held-out set** for a training set $\mathcal{D}_{\text{train}}$ is a list of m further observations, none of them in $\mathcal{D}_{\text{train}}$, that the fitting procedure does

not use and on which the fitted model $\hat{y}$ is scored:

$$((x_{*,1}, y_{*,1}), \dots, (x_{*,m}, y_{*,m})). \tag{14}$$

Remark (Source: Held-out set (hold-out set))

Remark 0.5 (Source: Held-out set (hold-out set)). Also called a **hold-out set** [6, Sec. 5.1.1].

Definition 0.20 (Root mean squared error of predictions)**.** The **sample root mean squared error (sRMSE)** of predictions, also called their **root mean squared error**, $\hat{y}_1, \dots, \hat{y}_n$ of observed outcomes $y_1, \dots, y_n$ is the square root of their sMSE:

$$\text{sRMSE}(\hat{y}) \stackrel{\text{def}}{=} \sqrt{\text{sMSE}(\hat{y})}. \tag{15}$$

Definition 0.21 (Empirical risk)**.** The **empirical risk** of a fitted model $\hat{y}$ on a set of n observations is the sample mean of its loss on those observations, the data counterpart of the risk, which is the expected loss:

$$\hat{R}(\hat{y}) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^{n} \ell_i(\hat{y}). \tag{16}$$

A subscript names the set the empirical risk is evaluated on, as in $\hat{R}_{\mathcal{D}_{\text{train}}}$ for a training set $\mathcal{D}_{\text{train}}$.

Example 0.16 (Empirical risk of a birth-weight model under squared error)**.**
The `birthwt` data record the birth weight in grams of babies born at Baystate Medical Center in Springfield, Massachusetts [7], as distributed in R's `MASS` package [8]. We fit a linear regression of birth weight on the mother's weight at her last menstrual period and her smoking status. We use every third row as the held-out set and fit on the remaining rows, the training set. The loss is the squared error, so the empirical risk is the mean of the squared prediction errors.

R

```r
birthwt <- read.csv("data/birthwt.csv")
is_held_out <- seq_len(nrow(birthwt)) %% 3 == 0

fit_bwt <- lm(bwt ~ lwt + smoke, data = birthwt[!is_held_out, ])

empirical_risk <- function(data) {
  mean((data$bwt - predict(fit_bwt, newdata = data))^2)
}

empirical_risks <- c(
  training = empirical_risk(birthwt[!is_held_out, ]),
  held_out = empirical_risk(birthwt[is_held_out, ])
)
empirical_risks
#> training held_out
#>   488365   503649
```

Python

```python
import numpy as np
import pandas as pd
import statsmodels.formula.api as smf

birthwt = pd.read_csv("data/birthwt.csv")
is_held_out = (np.arange(1, len(birthwt) + 1) % 3 == 0)

fit_bwt = smf.ols("bwt ~ lwt + smoke", data=birthwt[~is_held_out]).fit()


def empirical_risk(data):
    return ((data["bwt"] - fit_bwt.predict(data)) ** 2).mean()


empirical_risks = {
    "training": empirical_risk(birthwt[~is_held_out]),
    "held_out": empirical_risk(birthwt[is_held_out]),
}
empirical_risks
#> {'training': np.float64(488364.9956483022), 'held_out': np.float64(503648.75351911323)}
```

Of the 189 babies, 126 are in the training set. The fitted model has an empirical risk of

488,365 squared grams on the training set and 503,649 squared grams on the held-out set. On the training set, the root mean squared error is 699 grams, the root of the training empirical risk.

Definition 0.22 (Training error)**.** The **training error** of a fitted model $\hat{y}$ is its empirical risk $\hat{R}_{\mathcal{D}_{\text{train}}}(\hat{y})$ on the training set $\mathcal{D}_{\text{train}}$ used to fit it:

$$\overline{\text{err}} \stackrel{\text{def}}{=} \hat{R}_{\mathcal{D}_{\text{train}}}(\hat{y}). \tag{17}$$

Remark (Source: Training error)

Remark 0.6 (Source: Training error). Hastie et al. [4, Sec. 7.2], who write the training error as $\overline{\text{err}}$ and define it for any loss.

Overfitting

Exercise 0.1 (Training error versus error on new cars)**.** The built-in `mtcars` data frame has $n = 32$ cars. Fit four models for fuel efficiency `mpg` as a polynomial in weight `wt`, of degree 1, 2, 3, and 4, using only the first 20 rows as the training set. For each model, compute the sRMSE of its predictions on the 20 training rows and on the remaining 12 rows, a held-out set.

1. Which degree has the smallest training sRMSE?
2. Which degree has the smallest sRMSE on the 12 held-out rows?
3. Do the terms added beyond degree 2 improve predictions for the held-out cars?

Solution

Solution 0.1.

```r
train_cars <- mtcars[1:20, ]
held_out_cars <- mtcars[21:32, ]

prediction_mse <- function(model, data) {
  response_only <- stats::update(stats::formula(model), . ~ 1)
  y <- stats::model.response(stats::model.frame(response_only, data))
  mean((predict(model, newdata = data) - y)^2)
}

rmse <- function(model, data) {
  sqrt(prediction_mse(model, data))
}

overfitting_results <- tibble::tibble(degree = 1:4) |>
  dplyr::mutate(
    fit = purrr::map(
      degree,
      \(d) lm(mpg ~ poly(wt, d, raw = TRUE), data = train_cars)
    ),
    training_RMSE = purrr::map_dbl(fit, rmse, data = train_cars),
    held_out_RMSE = purrr::map_dbl(fit, rmse, data = held_out_cars)
  ) |>
  dplyr::select(-fit)

overfitting_results
#> # A tibble: 4 x 3
#>   degree training_RMSE held_out_RMSE
#>    <int>         <dbl>         <dbl>
#> 1      1          2.91          3.45
#> 2      2          2.46          3.18
#> 3      3          2.45          3.21
#> 4      4          2.44          3.19
```

```r
best_train <- overfitting_results$degree[
  which.min(overfitting_results$training_RMSE)
]
best_held_out <- overfitting_results$degree[
  which.min(overfitting_results$held_out_RMSE)
]

c(best_train = best_train, best_held_out = best_held_out)
#>    best_train best_held_out
#>             4             2
```

1. The smallest training sRMSE is at degree 4. Each model contains the previous one as a special case, so its training sRMSE cannot increase as the degree grows.
2. The smallest held-out sRMSE is at degree 2.
3. No. Going from degree 2 to degree 3 or 4 lowers the training sRMSE a little but leaves the held-out sRMSE essentially unchanged (it does not fall below its degree-2 value). The terms beyond degree 2 improve the fit to the 20 training cars without improving predictions for the 12 held-out cars. Degree 1 to degree 2 is different: there the held-out sRMSE also falls.

Definition 0.1 (Overfitting)**. Overfitting** occurs when a fitted model fits its training set well but predicts poorly for new observations.
Source: Wikipedia contributors [9].

Example 0.1 (Overfitting in polynomial fits to cars)**.** In Exercise 0.1, the terms beyond degree 2 fit the 20 training cars more closely but do not predict the 12 held-out cars better. The effect is small here: it is the beginning of overfitting, not a dramatic case.

Example 0.2 (Polynomial degree)**.** In Exercise 0.1, least squares computes the coefficients of each polynomial from the 20 training cars, but the degree d is fixed before the fit. So the degree is a tuning parameter of least-squares polynomial fitting, and the four fits are g_1, g_2, g_3, g_4 with $\lambda = d$.

Generalization error

Exercise 0.1 (Score a fitted model on a car it has not seen)**.** A fitted model $\hat{y}$ has been fitted to a training set $\mathcal{D}_{\text{train}}$ of n observations, such as the 20 training cars of Exercise 0.1. A new car (X_*, Y_*) is then drawn from the same population, independently of $\mathcal{D}_{\text{train}}$.

1. Holding $\mathcal{D}_{\text{train}}$ fixed, write the average loss ℓ_* that $\hat{y}$ would incur over all such new cars, as an expectation. Then write it for squared-error loss, as the average squared prediction error.
2. Which parts of your expression do you know, and which do you not know?
3. The training sRMSE of Exercise 0.1 is the square root of the training error $\overline{\text{err}}$ under squared-error loss. What does $\overline{\text{err}}$ use in place of your expectation?

Solution

Solution 0.1.

1. Average the loss over the distribution of (X_*, Y_*), with $\mathcal{D}_{\text{train}}$, and so the fitted model $\hat{y}$, held fixed:

   $$\mathbb{E}[\ell_* \mid \mathcal{D}_{\text{train}}].$$

   For squared-error loss, the loss is the squared prediction error:

   $$\mathbb{E}\left[\varepsilon_{\text{pred}}{}_*^2 \mid \mathcal{D}_{\text{train}}\right].$$

2. The fitted model $\hat{y}$ is known, because we fitted it. The joint distribution of (X_*, Y_*) is unknown, so the expectation cannot be computed. If we knew that distribution, we would not need data to estimate the true mean function $\mu(x)$ of the outcome variable given the input value x.

3. The training error $\overline{\text{err}}$ replaces the expectation over new cars with an average over the n training cars. Those cars are draws from the same population, but they are not independent of $\mathcal{D}_{\text{train}}$: they *are* $\mathcal{D}_{\text{train}}$, and $\hat{y}$ was chosen to fit them.

Definition 0.1 (Expected generalization error (expected prediction error, expected test error))**.** The **expected generalization error** of a fitting procedure g at training-set size n is the mean, over training sets $\mathcal{D}_{\text{train}}$ of n observations drawn independently from the population, of the generalization error of the fitted model that g produces from $\mathcal{D}_{\text{train}}$:

$$\mathrm{Err}^g(n) \stackrel{\text{def}}{=} \mathbb{E}\Big[\mathrm{Err}_{\mathcal{D}_{\text{train}}}\Big], \quad \mathcal{D}_{\text{train}} = ((X_i, Y_i))_{i \in I}, \quad |I| = n. \tag{18}$$

ℹ Remark (Source: Expected generalization error (expected prediction error, expected test error))

Remark 0.1 (Source: Expected generalization error (expected prediction error, expected test error)). Also called the **expected prediction error** or **expected test error** [4, Sec. 7.2]. Hastie et al. write the quantity as Err, without the argument n.

Definition 0.2 (Distribution model)**.** A **distribution model** P for an outcome Y given a covariate X gives each covariate value x a probability distribution for Y given $X = x$, specified by its cumulative distribution function. The mean function μ_P of P maps each covariate value x to the mean of that distribution:

$$\mu_P(x) \stackrel{\text{def}}{=} \mathbb{E}_P[Y \mid X = x]. \tag{19}$$

Example 0.1 (A normal distribution model with a mean function)**.** Let h be a mean-function model (a line, for instance) and $\sigma^2 > 0$ a variance. The distribution model P that gives each covariate value x the normal distribution with mean $h(x)$ and variance σ^2 is

$$Y \mid X = x \sim \mathrm{N}(h(x), \sigma^2). \tag{20}$$

The mean parameter of a normal distribution is its mean, so Equation 23 gives

$$\mu_P(x) = h(x). \tag{21}$$

The mean function h is the component of P that says where the outcome is centered, and σ^2 is the component that says how far it spreads.

Definition 0.3 (Distribution model family)**.** A **distribution model family** $\mathcal{P}$ is a set of distribution models. The mean functions of its members form the set

$$\{\mu_P : P \in \mathcal{P}\}. \tag{22}$$

Example 0.2 (Normal distribution models with polynomial mean functions)**.** The normal distribution models of Example 0.4 whose mean function h is a polynomial of degree at most d, with any variance $\sigma^2 > 0$, form a distribution model family $\mathcal{P}$. By Equation 25, the set of mean functions in Equation 26 is the set of polynomials of degree at

most d of Example 0.2.

Definition 0.4 (Fitted distribution model)**.** A **fitted distribution model** $\hat{P}$ is a member of a distribution model family $\mathcal{P}$ that has been chosen to describe a data set $\mathcal{D}$.

Example 0.3 (A fitted normal distribution model)**.** Choosing both a line and a variance to describe the weights and fuel economies of the cars in the `mtcars` data set $\mathcal{D}$ gives a fitted distribution model $\hat{P}$ in the family of Example 0.5 with polynomial degree $d = 1$. Its mean function $\mu_{\hat{P}}$ is the fitted line of Example 0.8.

Definition 0.5 (Distribution model)**.** A **distribution model** P for an outcome Y given a covariate X gives each covariate value x a probability distribution for Y given $X = x$, specified by its cumulative distribution function. The mean function μ_P of P maps each covariate value x to the mean of that distribution:

$$\mu_P(x) \stackrel{\text{def}}{=} \mathbb{E}_P[Y \mid X = x]. \tag{23}$$

Example 0.4 (A normal distribution model with a mean function)**.** Let h be a mean-function model (a line, for instance) and $\sigma^2 > 0$ a variance. The distribution model P that gives each covariate value x the normal distribution with mean $h(x)$ and variance σ^2 is

$$Y \mid X = x \sim \mathrm{N}(h(x), \sigma^2). \tag{24}$$

The mean parameter of a normal distribution is its mean, so Equation 23 gives

$$\mu_P(x) = h(x). \tag{25}$$

The mean function h is the component of P that says where the outcome is centered, and σ^2 is the component that says how far it spreads.

Definition 0.6 (Distribution model family)**.** A **distribution model family** $\mathcal{P}$ is a set of distribution models. The mean functions of its members form the set

$$\{\mu_P : P \in \mathcal{P}\}. \tag{26}$$

Example 0.5 (Normal distribution models with polynomial mean functions)**.** The normal distribution models of Example 0.4 whose mean function h is a polynomial of degree

at most d, with any variance $\sigma^2 > 0$, form a distribution model family $\mathcal{P}$. By Equation 25, the set of mean functions in Equation 26 is the set of polynomials of degree at most d of Example 0.2.

Definition 0.7 (Fitted distribution model)**.** A **fitted distribution model** $\hat{P}$ is a member of a distribution model family $\mathcal{P}$ that has been chosen to describe a data set $\mathcal{D}$.

Example 0.6 (A fitted normal distribution model)**.** Choosing both a line and a variance to describe the weights and fuel economies of the cars in the `mtcars` data set $\mathcal{D}$ gives a fitted distribution model $\hat{P}$ in the family of Example 0.5 with polynomial degree $d = 1$. Its mean function $\mu_{\hat{P}}$ is the fitted line of Example 0.8.

Definition 0.8 (Expected prediction loss at a covariate value)**.** The **expected prediction loss at** x of a fitting procedure g at training-set size n is the expected loss $\ell_*(\hat{y})$ of the fitted model $\hat{y} = \hat{y}_{\mathcal{D}_{\text{train}}}$ that g produces from $\mathcal{D}_{\text{train}}$, on a new outcome Y_* with $X_* = x$, averaged over random training sets $\mathcal{D}_{\text{train}}$ of n observations drawn independently of (X_*, Y_*), and over Y_*. The fitted model $\hat{y}_{\mathcal{D}_{\text{train}}}$ is random because $\mathcal{D}_{\text{train}}$ is:

$$\text{EPL}_n^g(x) \stackrel{\text{def}}{=} \mathbb{E}\Big[\ell_*\Big(\hat{y}_{\mathcal{D}_{\text{train}}}\Big) \mid X_* = x\Big]. \tag{27}$$

ℹ Remark (The expected prediction loss and the generalization error)

Remark 0.2 (The expected prediction loss and the generalization error). The generalization error $\text{Err}_{\mathcal{D}_{\text{train}}}$ holds the training set fixed and averages the loss over new covariate values too. The expected prediction loss at x averages over training sets as well, and holds the covariate value fixed. Neither is the test error: the test error is an average over a finite test set, so it estimates the generalization error without equalling it.

Definition 0.9 (Expected prediction error at a covariate value)**.** The **expected prediction error at** x (**EPE**; also the **expected squared prediction error**, **ESPE**) of a fitting procedure g at training-set size n is its expected prediction loss under squared-error loss, the expected value of the squared prediction error $\varepsilon_{\text{pred}_*}(\hat{y}) = \hat{y}(X_*) - Y_*$ of the fitted model $\hat{y} = \hat{y}_{\mathcal{D}_{\text{train}}}$ that g produces from $\mathcal{D}_{\text{train}}$:

$$\text{EPE}_n^g(x) \stackrel{\text{def}}{=} \mathbb{E}\Big[\varepsilon_{\text{pred}_*}\Big(\hat{y}_{\mathcal{D}_{\text{train}}}\Big)^2 \mid X_* = x\Big]. \tag{28}$$

Remark (The expected prediction error is a population mean squared error)

Remark 0.3 (The expected prediction error is a population mean squared error). The expected prediction error is the pMSE of the prediction $\hat{y}(x)$ as a prediction of the new outcome Y_*. With squared-error loss, $\mathrm{EPL}_n^g(x) = \mathrm{EPE}_n^g(x)$.
It differs from the pMSE of the same prediction as an estimator of the mean function value $\mu(x)$: Theorem 0.4 shows that it exceeds that pMSE by the noise variance. The acronym omits "squared" because "error" already means squared error here. Another loss gives an expected prediction loss, not an EPE.

Remark (Source: Expected prediction error at a covariate value)

Remark 0.4 (Source: Expected prediction error at a covariate value). Hastie et al. [4, Sec. 7.3] write this quantity as $\mathrm{Err}(x_0)$ for squared-error loss.

Exercise 0.2 (The expected generalization error, covariate value by covariate value)**.** Let g be a fitting procedure, and let the random training set $\mathcal{D}_{\text{train}}$ of n observations be independent of the new observation (X_*, Y_*). Derive an expression for the expected generalization error $\mathrm{Err}^g(n)$ in terms of the expected prediction loss at x, $\mathrm{EPL}_n^g(x)$, and the distribution of X_*.

Solution

Solution 0.2.

$$
\begin{aligned}
&\mathrm{Err}^g(n) \\
&\quad = \mathbb{E}\left[\mathrm{Err}_{\mathcal{D}_{\text{train}}}\right] && \text{(definition)} \\
&\quad = \mathbb{E}[\mathbb{E}[\ell_* \mid \mathcal{D}_{\text{train}}]] && \text{(definition)} \\
&\quad = \mathbb{E}[\ell_*] && \text{(iterated expectations)} \\
&\quad = \mathbb{E}[\mathbb{E}[\ell_* \mid X_*]] && \text{(iterated expectations)} \\
&\quad = \mathbb{E}[\mathrm{EPL}_n^g(X_*)] && \text{(definition)}
\end{aligned}
$$

The steps labeled "definition" apply Equation 18, Equation 13 and Equation 27, in that order. Here ℓ_* abbreviates $\ell_*(\hat{y})$, the loss of the fitted model $\hat{y}(X_*) = \hat{y}_{\mathcal{D}_{\text{train}}}(X_*)$ (Equation 9) on the new observation. The two steps labeled "iterated expectations" use the law of iterated expectations, the first conditioning on $\mathcal{D}_{\text{train}}$ and the second on X_*. The last line needs $\mathcal{D}_{\text{train}}$ to be independent of (X_*, Y_*): then, given $X_* = x$, $\mathcal{D}_{\text{train}}$ still has its unconditional distribution, as Definition 0.8 requires.

Theorem 0.1 (The expected generalization error averages the loss at each covariate value)**.** *If the random training set $\mathcal{D}_{train}$ is independent of the new observation (X_*, Y_*), the expected generalization error of a fitting procedure g is the mean of its expected prediction loss at x over the distribution of X_*:*

$$\mathrm{Err}^g(n) = \mathbb{E}[\mathrm{EPL}_n^g(X_*)]. \tag{29}$$

Proof

Proof. This is Solution 0.2. □

Definition 0.10 (Expected prediction)**.** Let g be a fitting procedure, and let $\mathcal{D}_{\text{train}}$ be a random training set. The **expected prediction** of g at a covariate value x is the value $\hat{y}(x)$ of the fitted model $\hat{y}$ at x, averaged over the population distribution of all training sets that might have been collected:

$$\mu_h(x) \stackrel{\text{def}}{=} \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}(x)]. \tag{30}$$

Remark

Remark. The expected prediction is not the prediction of any one analysis, because an analysis has one training set, not the whole distribution of them. It is a reference point for measuring how far the predictions are from the mean function and from each other.

Example 0.7 (Computing the expected prediction)**.** Suppose the predictions $\hat{y}(x)$ at x take the values 9, 11 and 13 with equal probability. Then

$$\begin{aligned} \mu_h(x) &= \frac{9 + 11 + 13}{3} \\ &= 11. \end{aligned}$$

Example 0.8 (The expected prediction of a line on cylinders)**.** Treat the 392 cars of the `Auto` data [6] 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 [10], a random number generator with the same algorithm in R and Python, from the same seed, so they pick the same cars:

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
    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 expected prediction 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(20
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.
#> 28.72 23.89 32.74 0.031
```

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 Python tab.
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 expected prediction 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)
summary_af <- c(
  mean = mean(preds_af),
  min = min(preds_af),
  max = max(preds_af)
)
c(round(summary_af, 2), mc_se = round(mc_se_af, 3))
#>   mean    min    max  mc_se
#> 28.720 23.890 32.740  0.031
```

No single training set produces the expected prediction. 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 expected prediction $\mu_h(4)$ of Equation 30, 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 1 plots 100 of the 2000 lines, their pointwise average, and the true mean function:

Python

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

grid_af = np.arange(3, 9)
# The pointwise expected prediction 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=
# 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="Expected prediction")
ax.plot(truth_af.index, truth_af.values, color="darkorange", linewidth=2.5, marker="o", lab
ax.set_xlabel("cylinders")
ax.set_ylabel("miles per gallon")
fig.legend(loc="outside lower center", ncol=3, frameon=False)
plt.show()
```

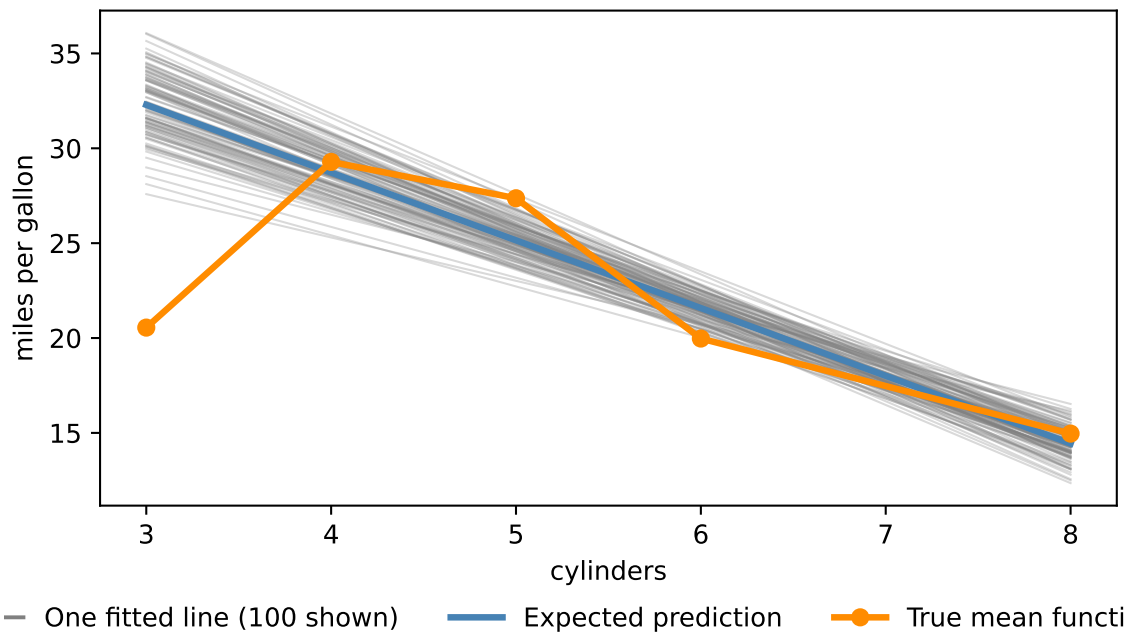


Figure 1: The fitted lines of 100 of the 2000 training sets (grey), their pointwise expected prediction (blue), and the true mean function (orange), for `mpg` against `cylinders`.

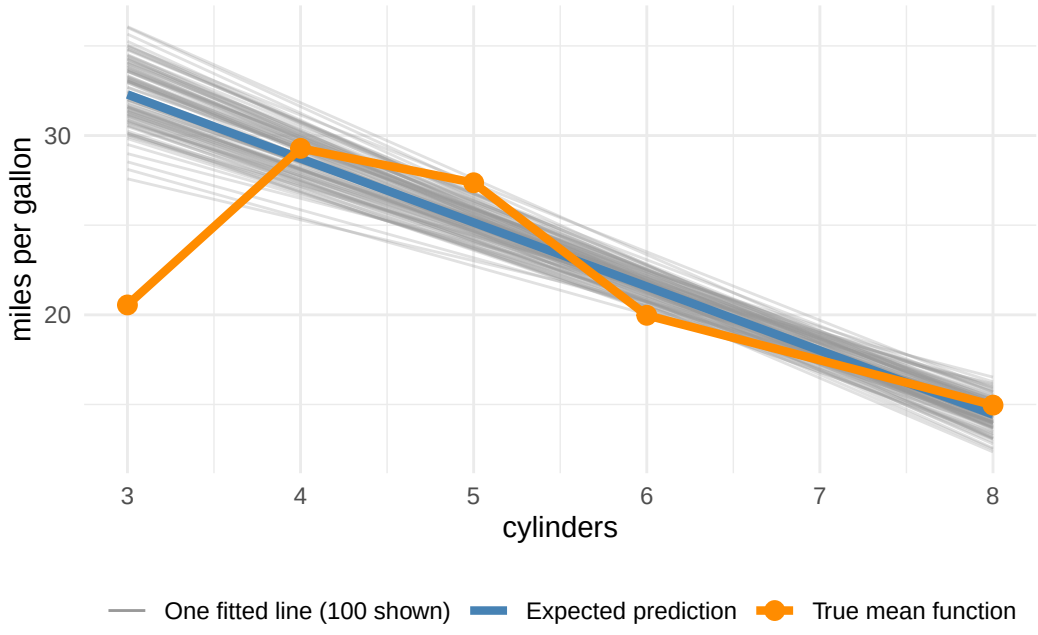
R

```r
library(ggplot2)

grid_af <- 3:8
# One row per fitted line and cylinder count;
# the pointwise expected prediction 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)

line_name_af <- "One fitted line (100 shown)"
avg_name_af <- "Expected prediction"
truth_name_af <- "True mean function"

ggplot() +
  geom_line(
    data = subset(lines_af, line <= 100),
    aes(cylinders, fit, group = line, colour = line_name_af),
    alpha = 0.3
  ) +
  geom_line(
    data = avg_af,
    aes(cylinders, fit, colour = avg_name_af),
    linewidth = 1.5
  ) +
  geom_line(
    data = truth_af,
    aes(cylinders, mpg, colour = truth_name_af),
    linewidth = 1.5
  ) +
  geom_point(
    data = truth_af,
    aes(cylinders, mpg, colour = truth_name_af),
    size = 3
  ) +
  scale_colour_manual(
    values = c("grey60", "steelblue", "darkorange"),
    breaks = c(line_name_af, avg_name_af, truth_name_af)
  ) +
  guides(colour = guide_legend(override.aes = list(alpha = 1))) +
  labs(x = "cylinders", y = "miles per gallon", colour = NULL) +
  theme_minimal() +
  theme(legend.position = "bottom")
```



Definition 0.11 (Bias of a prediction)**.** Let g be a fitting procedure, and let $\mathcal{D}_{\text{train}}$ be a random training set. At a covariate value x, the prediction $\hat{y}(x)$ is a random variable, because it depends on the random training data. Let (X_*, Y_*) be a new observation with $X_* = x$. The **bias** of the prediction is the expected prediction error of the prediction $\hat{y}(x)$ of the new outcome Y_*. The expectation averages over training sets and over the new outcome:

$$\text{Bias}(\hat{y}(x)) = \mathbb{E}\left[\varepsilon_{\text{pred}_*}\left(\hat{y}_{\mathcal{D}_{\text{train}}}\right) \mid X_* = x\right]. \tag{31}$$

Remark (The bias is signed, not squared)

Remark 0.5 (The bias is signed, not squared). The bias is an expected prediction error, so it is signed, not squared. It is not the expected squared prediction error.

Remark (The bias as an iterated expectation and as integrals)

Remark 0.6 (The bias as an iterated expectation and as integrals). The new observation is independent of the training set. So by the law of iterated expectations, the expectation in Equation 31 can be taken in two steps: first over the new outcome with the training set fixed, then over the training sets.

$$\operatorname{Bias}(\hat{y}(x)) = \mathbb{E}_{\mathcal{D}_{\text{train}}}\left[\mathbb{E}_{Y_*}\left[\varepsilon_{\text{pred}_*}\left(\hat{y}_{\mathcal{D}_{\text{train}}}\right) \mid \mathcal{D}_{\text{train}}, X_* = x\right]\right]. \tag{32}$$

The training set consists of n independent and identically distributed pairs (X_i, Y_i), so its density is the product of the densities of its pairs. When the variables are continuous, the two steps are integrals. The inner integral runs over the new outcome y, and the outer integral runs over the values (x_i, y_i) of the training pairs, which determine the prediction $\hat{y}(x)$:

$$\operatorname{Bias}(\hat{y}(x)) = \int \cdots \int \left(\int (\hat{y}(x) - y) \mathbb{p}(Y_* = y \mid X_* = x)\, dy \right) \prod_{i=1}^{n} \mathbb{p}(X_i = x_i,\, Y_i = y_i)\, dx_1\, dy_1 \cdots dx_n\, dy_n. \tag{33}$$

For discrete variables, the integrals are sums.

Theorem 0.2 (The bias as the expected prediction minus the mean function)**.** *The bias of the prediction at x is the expected prediction minus the mean function value:*

$$\operatorname{Bias}(\hat{y}(x)) = \mathbb{E}_{\mathcal{D}_{train}}[\hat{y}] - \mu(x). \tag{34}$$

Proof

Proof. The new outcome is independent of the training set, and its mean is $\mu(x)$ because the noise has mean zero.

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

□

Remark

Remark. In expected prediction error decompositions [6, Sec. 2.2.2], the bias reflects the approximation error that comes from approximating a real-world relationship by a model.

Example 0.9 (Computing the bias of a prediction)**.** Suppose $\mu(x) = 10$, and the predictions $\hat{y}(x)$, over repeated training sets, average 11. Then

$$\begin{aligned}\operatorname{Bias}(\hat{y}) &= \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}] - \mu(x) \quad \text{(@thm-prediction-bias-average-fit)} \\ &= 11 - 10 \\ &= 1.\end{aligned}$$

The squared bias is $1^2 = 1$, the value in Exercise 0.3.

Definition 0.12 (Residual variance)**.** Let (X, Y) be the covariate and the outcome of one observation from a population, and let ε be its noise.
The **residual variance** (or **irreducible error**) at the covariate value x is the conditional population variance of the noise given $X = x$:

$$\sigma^2 \stackrel{\text{def}}{=} \operatorname{Var}(\varepsilon \mid X = x). \tag{35}$$

It is a variance, a component of the expected squared prediction error, and not an error. It is irreducible only given the covariates the prediction uses: a further covariate that predicts the outcome moves part of the spread out of the noise and into the mean function, so the residual variance depends on which covariates are measured. Because $\operatorname{Var}(\varepsilon \mid X = x) = \operatorname{Var}(Y \mid X = x)$, the residual variance averaged over the covariate values of new observations is $\mathbb{E}[\operatorname{Var}(Y \mid X)]$. It is a population quantity, not the estimate computed from the residuals of a fitted model.
Source: Wikipedia contributors [11]; James et al. [6, Sec. 2.2.2].

Example 0.10 (The residual variance of a normal population)**.** Take a population whose outcome given the covariate value x has the normal distribution $\mathrm{N}(\sin(2\pi x), \sigma^2)$, where $\sigma^2 = \sigma^2$ and $\sigma = 0.3$. Its noise $\varepsilon = Y - \sin(2\pi X)$, given $X = x$, has the normal distribution with mean 0 and conditional variance σ^2. The residual variance at every x is therefore $\sigma^2 = \sigma^2 = 0.09$.

Remark (What the residual variance does and does not depend on)

Remark 0.7 (What the residual variance does and does not depend on). The residual variance σ^2 does not depend on the fitting procedure g, so no choice of prediction using the same covariates can remove it. It is defined at the single covariate value x, so it may differ from one covariate value to another: it does not assume constant noise variance across covariate values. For example, in Example 0.10, every prediction has expected squared prediction error at least σ^2.

Exercise 0.3 (Expected squared prediction error from three sources)**.** Let:

- (X_*, Y_*) be a new observation relative to the random training set $\mathcal{D}_{\text{train}}$, and

- $\hat{y}(x)$ be a prediction at the covariate value x.

Assume:

- $X_* = x$,
- the squared bias of $\hat{y}(x)$ is $(\text{Bias}(\hat{y}(x)))^2 = 1$,
- the variance of $\hat{y}(x)$ is $\text{Var}(\hat{y}(x)) = 2$, and
- the population variance σ^2 of the noise of Y_* is 3.

What is the expected squared prediction error $\text{ESPE}(\hat{y}(x))$?

Solution

Solution 0.3. The expected squared prediction error is the sum of the three sources: $1 + 2 + 3 = 6$.

Exercise 0.4 (Expected squared prediction error without bias or variance)**.** Make the assumptions of Exercise 0.3. If we could remove all bias and all variance of $\hat{y}(x)$, what would the expected squared prediction error at x be?

Solution

Solution 0.4. $0 + 0 + 3 = 3$. The population variance $\sigma^2 = 3$ of the noise remains, so 3 is the smallest expected squared prediction error any prediction can have here.

Exercise 0.5 (Which sources of error a better method can reduce)**.** Make the assumptions of Exercise 0.3. Which of the three quantities (the squared bias, the variance of $\hat{y}(x)$, and the population variance σ^2 of the noise) can a better prediction method reduce, and which cannot?

Solution

Solution 0.5. A better method can reduce the squared bias and the variance of $\hat{y}(x)$, because both depend on how g is built. It cannot reduce σ^2, which is the variance of the noise ε in the new outcome.

Exercise 0.6 (Separate the noise)**.** Let:

- (X_*, Y_*) be a new observation relative to the random training set $\mathcal{D}_{\text{train}}$,
- ε be the noise of Y_*, and
- $\sigma^2 \stackrel{\text{def}}{=} \text{Var}(\varepsilon)$ be its population variance.

Assume:

- the new covariate value X_* equals the covariate value x of the prediction and
- the population variance σ^2 is finite.

Take every expectation below to be conditional on $X_* = x$. Add and subtract $\mu(x)$ inside the square of the expected prediction error,

$$\mathbb{E}\left[(Y_* - \hat{y}(x))^2\right],$$

expand it, and take the expectation term by term.
Two of the three terms simplify. Say what each becomes, and name which facts about the noise hold by definition and which hypothesis makes the cross term zero.

Solution

Solution 0.6. Express the prediction error as noise plus the error of the prediction, used as an estimator of the mean function value $\mu(x)$:

$$\begin{aligned} Y_* - \hat{y}(x) &= (Y_* - \mu(x)) + (\mu(x) - \hat{\mu}(x)) && \text{(add and subtract } \mu(x)\text{)} \\ &= \varepsilon + (\mu(x) - \hat{\mu}(x)) && \text{(definition of } Y_*\text{)} \end{aligned}$$

Square it:

$$(Y_* - \hat{y}(x))^2 = \varepsilon^2 + 2\varepsilon(\mu(x) - \hat{\mu}(x)) + (\mu(x) - \hat{\mu}(x))^2 \quad \text{(expand the square of a sum)}$$

Take the expectation of each term in turn, using linearity of expectation to distribute the expectation over the sum.
For the noise term, the variance identity gives $\sigma^2 = \mathbb{E}[\varepsilon^2] - (\mathbb{E}[\varepsilon])^2$:

$$\begin{aligned} \mathbb{E}[\varepsilon^2] &= \sigma^2 + (\mathbb{E}[\varepsilon])^2 && \text{(variance identity)} \\ &= \sigma^2 + 0^2 && \text{(the noise has population mean 0)} \\ &= \sigma^2 && (0^2 = 0, \text{ and } \sigma^2 + 0 = \sigma^2) \end{aligned}$$

For the cross term, $\varepsilon = Y_* - \mu(x)$ shifts Y_* by a constant, and Y_* is independent of $\mathcal{D}_{\text{train}}$, so ε is independent of $\mathcal{D}_{\text{train}}$, and hence of $\hat{y}(x)$ and of $\mu(x) - \hat{\mu}(x)$. The expectation of a product of independent random variables is the product of their expectations:

$$\begin{aligned} \mathbb{E}[2\varepsilon(\mu(x) - \hat{\mu}(x))] &= 2\mathbb{E}[\varepsilon(\mu(x) - \hat{\mu}(x))] && \text{(constant factor 2)} \\ &= 2\mathbb{E}[\varepsilon]\mathbb{E}[\mu(x) - \hat{\mu}(x)] && \text{(independence)} \\ &= 2 \cdot 0 \cdot \mathbb{E}[\mu(x) - \hat{\mu}(x)] && \text{(the noise has population mean 0)} \\ &= 0 && \text{(zero times a number)} \end{aligned}$$

Adding the three expectations,

$$\mathbb{E}\Big[(Y_* - \hat{y}(x))^2\Big] = \sigma^2 + \mathbb{E}\Big[(\mu(x) - \hat{\mu}(x))^2\Big]. \tag{36}$$

The noise has population mean 0 by Theorem 0.1, so the algebra used only one hypothesis about the new observation: it is independent of the training set. Independence holds for a fresh observation, and it fails for an observation the procedure was trained on, whose noise helped determine the prediction. It also fails when the new observation depends on the training data (for example, a later time in the same series), or when the same test set was used to choose the procedure.

The algebra did not use that the new pair has the same population distribution as the training data. Suppose the new pair came from a population whose mean function differed from the training population's. Then Equation 36 would still hold, with the new population's mean function and noise variance. Its estimation-error term would include the bias from that difference. So it would no longer measure how well the procedure estimates the training population's mean function.

Theorem 0.3 (The pMSE of a prediction is the pMSE of the mean estimate plus the noise variance)**.** *Let:*

- (X_*, Y_*) *be a new observation relative to the random training set* $\mathcal{D}_{train}$*,*
- ε *be the noise of* Y_**, and*
- $\sigma^2 \overset{def}{=} \mathrm{Var}(\varepsilon)$ *be its population variance.*

Assume:

- *the new covariate value* X_* *equals the covariate value* x *of the prediction and*
- *the population variance* σ^2 *is finite.*

Then, given $X_* = x$*, the expected squared prediction error (ESPE) of* $\hat{y}(x)$ *is the sum of two terms. The first is the residual variance* σ^2*. The second is the expected squared estimation error (ESEE) of* $\hat{y}(x)$ *as an estimator of* $\mu(x)$*:*

$$\mathrm{ESPE}(\hat{y}(x)) = \sigma^2 + \mathrm{ESEE}(\hat{\mu}(x)). \tag{37}$$

Proof

Proof. This is Solution 0.6. □

Example 0.11 (The two pMSEs differ by the noise variance)**.** Take a population whose mean function is known: $Y \mid X = x \sim \mathrm{N}(\sin(2\pi x), \sigma^2)$ with $\sigma = 0.3$. Let the covariate value be $x = 0.25$, where the mean function value is $\mu(x) = 1$, and fit a cubic polynomial by least squares to $n = 30$ observations. We draw 5,000 training sets, each with its own new outcome at x, and average the squared errors of the two targets over them.

```
split_x0 <- 0.25
split_reps <- 5000
split_sigma <- 0.3
split_truth <- function(x) sin(2 * pi * x)
split_draw <- function(n) {
  x <- stats::runif(n)
  tibble::tibble(x = x, y = split_truth(x) + stats::rnorm(n, sd = split_sigma))
}

set.seed(2)
split_errors <- purrr::map(seq_len(split_reps), \(i) {
  train <- split_draw(30)
  fit <- predict(lm(y ~ poly(x, 3), data = train), tibble::tibble(x = split_x0))
  new_y <- split_truth(split_x0) + stats::rnorm(1, sd = split_sigma)
  tibble::tibble(
    new_outcome = (new_y - fit)^2,
    mean_estimate = (split_truth(split_x0) - fit)^2
  )
}) |>
  dplyr::bind_rows()

split_pmse <- colMeans(split_errors)
```

The pMSE of the prediction of the new outcome is 0.099. The pMSE of the prediction as an estimate of the mean is 0.011. Their difference, 0.087, is close to $\sigma^2 = 0.09$, as Theorem 0.3 says. It is not exactly equal because 5,000 training sets give only a Monte Carlo approximation to each expectation.

Exercise 0.7 (Separate the bias)**.** Equation 36 leaves one term that still mixes two ideas:

$$\mathbb{E}\left[(\mu(x) - \hat{y})^2\right].$$

A prediction can miss $\mu(x)$ because its expected prediction is off, or because it gives a different result for every training set. These are different failures with different fixes, so the term should split into one piece for each.

Split it with the same technique as before: add and subtract the expected prediction $\mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}]$ (Equation 30), expand, and take the expectation term by term.
Which of the three terms is not random at all, and what makes the cross term zero this time?

Solution

Solution 0.7. Insert the expected prediction:

$$\mu(x) - \hat{y} = \big(\mu(x) - \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}]\big) + \big(\mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}] - \hat{y}\big) \quad \text{(add and subtract } \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}]\text{)}$$

Square it:

$$(\mu(x) - \hat{y})^2 = \big(\mu(x) - \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}]\big)^2 + 2\big(\mu(x) - \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}]\big)\big(\mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}] - \hat{y}\big) + \big(\mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}] - \hat{y}\big)^2 \quad \text{(expand the squ}$$

Take the expectation of each term in turn.
The first term does not depend on $\mathcal{D}_{\text{train}}$: $\mu(x)$ is a constant, and $\mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}]$ has already been averaged over training sets. The expectation of a constant is that constant:

$$\begin{aligned}
\mathbb{E}\Big[\big(\mu(x) - \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}]\big)^2\Big] &= \big(\mu(x) - \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}]\big)^2 && \text{(expectation of a constant)} \\
&= \big(\mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}] - \mu(x)\big)^2 && \text{(a number and its negative have the same square)} \\
&= (\text{Bias}(\hat{y}))^2 && (\text{Bias}(\hat{y}) = \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}] - \mu(x), \text{ @thm-prediction-bias-average-fi}
\end{aligned}$$

The cross term vanishes, and this time not by independence but because the expected prediction is the expectation of the prediction. The first factor is a constant, so it comes out of the expectation:

$$\begin{aligned}
\mathbb{E}\Big[2\big(\mu(x) - \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}]\big)\big(\mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}] - \hat{y}\big)\Big] &= 2\big(\mu(x) - \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}]\big)\mathbb{E}\Big[\mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}] - \hat{y}\Big] && \text{(constant factors)} \\
&= 2\big(\mu(x) - \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}]\big)\big(\mathbb{E}\Big[\mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}]\Big] - \mathbb{E}[\hat{y}]\big) && \text{(linearity)} \\
&= 2\big(\mu(x) - \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}]\big)\big(\mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}] - \mathbb{E}[\hat{y}]\big) && \text{(expectation of a cons} \\
&= 2\big(\mu(x) - \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}]\big)\big(\mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}] - \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}]\big) && \text{(definition of the expe} \\
&= 2\big(\mu(x) - \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}]\big) \cdot 0 && (a - a = 0) \\
&= 0 && \text{(a number times zero)}
\end{aligned}$$

The third term is the expectation of the squared deviation of the prediction from its mean, which is the variance of the prediction:

$$\begin{aligned}\mathbb{E}\Big[\big(\mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}]-\hat{y}\big)^2\Big] &= \mathbb{E}\Big[\big(\hat{y}-\mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}]\big)^2\Big] && \text{(a number and its negative have the same square)}\\ &= \text{Var}(\hat{y}) && \text{(definition of variance)}\end{aligned}$$

Adding the three expectations,

$$\mathbb{E}\Big[(\mu(x)-\hat{y})^2\Big] = (\text{Bias}(\hat{y}))^2 + \text{Var}(\hat{y}). \tag{38}$$

Neither cross term needed the noise to be normal, and neither assumed anything about the fitting procedure: g can be a line, a tree or a neural network. This split is specific to squared-error loss, the loss of this chapter's examples.

Theorem 0.4 (Bias, variance and noise)**.** *Let:*

- (X_*, Y_*) *be a new observation relative to the random training set* $\mathcal{D}_{train}$*,*
- ε *be the noise of* Y_**, and*
- $\sigma^2 \overset{def}{=} \text{Var}(\varepsilon)$ *be its population variance, and*
- $\mathbb{B}[\hat{y}]$ *and* $\mathbb{V}[\hat{y}]$ *be short forms of the bias* $\text{Bias}(\hat{y})$ *and the variance* $\text{Var}(\hat{y})$ *of the prediction.*

Assume:

- *the new covariate value* X_* *equals the covariate value* x *of the prediction,*
- *the population variance* σ^2 *is finite, and*
- *the expected squared prediction* $\mathbb{E}\Big[(\hat{y})^2\Big]$ *is finite.*

Then the expected prediction error at x *is the squared bias of the prediction* $\hat{y}$*, plus its variance, plus* σ^2*:*

$$\text{EPE}_n^g(x) = (\mathbb{B}[\hat{y}])^2 + \mathbb{V}[\hat{y}] + \sigma^2. \tag{39}$$

Proof

Proof. Every expectation below is taken given $X_* = x$. By Theorem 0.3, the expected squared prediction error is the variance of the noise plus the pMSE of the prediction, used as an estimator of $\mu(x)$. That pMSE is the squared bias plus the variance. To see this, add and subtract the expected prediction $\mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}]$ inside the square and expand it:

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

This choice splits the error about $\mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}]$, the center of the prediction's distribution,

so the pieces are the bias and the variance. Take the expectation of the three terms, one at a time:

$$\begin{aligned}
\mathbb{E}\Big[\big(\mu(x) - \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}]\big)^2\Big] &= \big(\mu(x) - \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}]\big)^2 && \text{(a constant is its own expe} \\
&= \big(\mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}] - \mu(x)\big)^2 = (\mathbb{B}[\hat{y}])^2 && \text{(squaring ignores the sign;} \\
\mathbb{E}\Big[2\big(\mu(x) - \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}]\big)\big(\mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}] - \hat{y}\big)\Big] &= 2\big(\mu(x) - \mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}]\big)\big(\mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}] - \mathbb{E}[\hat{y}]\big) && \text{(constants come out of the} \\
&= 0 && (\mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}] \text{ is the expectatio} \\
\mathbb{E}\Big[\big(\mathbb{E}_{\mathcal{D}_{\text{train}}}[\hat{y}] - \hat{y}\big)^2\Big] &= \mathbb{V}[\hat{y}] && \text{(definition of the variance)}
\end{aligned}$$

Substituting the split of the pMSE into Equation 37:

$$\begin{aligned}
\text{EPE}_n^g(x) &= \sigma^2 + \mathbb{E}\Big[(\mu(x) - \hat{y})^2\Big] && \text{(separating the noise)} \\
&= \sigma^2 + (\mathbb{B}[\hat{y}])^2 + \mathbb{V}[\hat{y}] && \text{(separating the bias)} \\
&= (\mathbb{B}[\hat{y}])^2 + \mathbb{V}[\hat{y}] + \sigma^2 && \text{(reorder the terms)}
\end{aligned}$$

□

ℹ Remark (Other names, and how the expected error splits)

Remark 0.8 (Other names, and how the expected error splits). Hastie et al. [4, Sec. 7.2] also call $\text{Err}_{\mathcal{D}_{\text{train}}}$ the *test error*, and in sec. 7.12 the *conditional* test error. It is the risk of the prediction $\hat{y}(X_*)$ under the loss L, computed with the training set held fixed. The argument n of the expected generalization error $\text{Err}^g(n)$ of a fitting procedure g records that it depends on how many observations each training set has. By Theorem 0.1, $\text{Err}^g(n)$ averages the expected prediction error at each covariate value, and Theorem 0.4 splits each of those expected prediction errors, under squared-error loss, into a squared bias, a variance and a noise variance.

Definition 0.13 (Flexibility)**.** Let g_1 and g_2 be fitting procedures that each choose, by least squares, a fitted model from a set of candidate functions, $\mathcal{G}_1$ and $\mathcal{G}_2$ respectively. The procedure g_2 is **at least as flexible** as g_1 if every candidate of g_1 is also a candidate of g_2:

$$\mathcal{G}_1 \subseteq \mathcal{G}_2. \tag{40}$$

Remark (Source: Flexibility)

Remark 0.9 (Source: Flexibility). James et al. [6, Sec. 2.1.3] use flexibility without a formal definition.

Remark (Polynomial degree orders flexibility)

Remark 0.10 (Polynomial degree orders flexibility). Least-squares polynomial fits are ordered by degree: a polynomial of degree d is also a polynomial of degree $d+1$ whose coefficient of x^{d+1} is zero, so the fit of degree $d+1$ is at least as flexible as the fit of degree d. The four fits of Exercise 0.1 are ordered this way. Definition 0.13 is narrower than the informal use of the word in James et al. [6, Sec. 2.1.3]: it orders only procedures whose candidate sets are nested. Other procedures can be compared by other measures, such as the effective number of parameters [4, Sec. 7.6].

Definition 0.14 (Bias–variance trade-off)**.** Let $g_1, \dots, g_L$ be fitting procedures, each at least as flexible as the one before, and let $\mathcal{D}_{\text{train}}$ be a random training set. At a covariate value x, with μ the mean function, there is a **bias–variance trade-off** among them when, from each procedure to the next, the squared bias of the prediction does not rise and its variance does not fall:

$$\begin{aligned} \left(\operatorname{Bias}(\hat{y}_{l+1}(x))\right)^2 &\le \left(\operatorname{Bias}(\hat{y}_l(x))\right)^2, \\ \operatorname{Var}(\hat{y}_{l+1}(x)) &\ge \operatorname{Var}(\hat{y}_l(x)), \qquad l = 1, \dots, L-1. \end{aligned} \tag{41}$$

Source: James et al. [6, Sec. 2.2.2]; Wikipedia contributors [11].

Remark (Overfitting and the trade-off)

Remark 0.11 (Overfitting and the trade-off). By Theorem 0.4, $\operatorname{EPE}_n^{g_l}(x)$ is the squared bias plus the variance plus σ^2. Under a bias–variance trade-off among $\hat{y}_1, \dots, \hat{y}_L$, moving from $\hat{y}_l$ to $\hat{y}_{l+1}$ lowers the expected squared prediction error at x when the squared bias falls by more than the variance rises, and raises it otherwise. The smallest of $\operatorname{EPE}_n^{g_1}(x), \dots, \operatorname{EPE}_n^{g_L}(x)$ is often at some g_l with $1 < l < L$ [6, Sec. 2.2.2], but not always: if the mean function μ is among the functions that candidate 1 can fit, the squared bias of $\hat{y}_1$ can be zero, and then no more flexible $\hat{y}_l$ has a smaller squared bias and, under the trade-off, none has a smaller variance, so none has a smaller expected squared prediction error. Past the candidate with the smallest expected squared prediction error, the more flexible candidates have lowered their squared bias by less than they have raised their variance, and their fitted models tend to overfit: they fit their training sets more closely than the fitted model of the candidate with the smallest expected squared prediction error but predict new observations worse.

Remark (Coefficients per observation)

Remark 0.12 (Coefficients per observation). Adding coefficients to a least-squares fit makes it at least as flexible as before. With few observations per coefficient, the variance this adds can outweigh the squared bias it removes, which is the overfitting side of Remark 0.11. In Exercise 0.1, each added power of `wt`, like an added predictor, is one more coefficient fitted to the same 20 cars: the degree-4 fit has five coefficients for those 20 cars.

Exercise 0.8 (Held-out predictions and the generalization error)**.** A fitted model $\hat{y}$ is fitted to a training set $\mathcal{D}_{\text{train}}$. A held-out set of m observations $(X_{*,1}, Y_{*,1}), \dots, (X_{*,m}, Y_{*,m})$ is drawn from the same population as the new observation (X_*, Y_*) of Definition 0.18, each independently of $\mathcal{D}_{\text{train}}$. Write $\ell_{*,j}$ for the loss of $\hat{y}$ on held-out observation j.
Find, given $\mathcal{D}_{\text{train}}$, the expected sample mean loss of the held-out predictions, in terms of the generalization error $\text{Err}_{\mathcal{D}_{\text{train}}}$:

$$\mathbb{E}\left[\frac{1}{m}\sum_{j=1}^{m}\ell_{*,j} \mid \mathcal{D}_{\text{train}}\right].$$

Solution

Solution 0.8. Given $\mathcal{D}_{\text{train}}$, the fitted model $\hat{y}$ is fixed. Each held-out pair has the same distribution as (X_*, Y_*) and is independent of $\mathcal{D}_{\text{train}}$, so each loss has conditional expectation $\mathbb{E}\left[\ell_{*,j} \mid \mathcal{D}_{\text{train}}\right] = \text{Err}_{\mathcal{D}_{\text{train}}}$ by Equation 13. Then

$$
\begin{aligned}
&\mathbb{E}\left[\frac{1}{m}\sum_{j=1}^{m}\ell_{*,j} \mid \mathcal{D}_{\text{train}}\right] \\
&\quad= \frac{1}{m}\mathbb{E}\left[\sum_{j=1}^{m}\ell_{*,j} \mid \mathcal{D}_{\text{train}}\right] && \text{(constant } 1/m \text{ out)} \\
&\quad= \frac{1}{m}\sum_{j=1}^{m}\mathbb{E}\left[\ell_{*,j} \mid \mathcal{D}_{\text{train}}\right] && \text{(linearity of } \mathbb{E}\text{)} \\
&\quad= \frac{1}{m}\sum_{j=1}^{m}\text{Err}_{\mathcal{D}_{\text{train}}} && \text{(each term, as above)} \\
&\quad= \frac{1}{m}\left(m\ \text{Err}_{\mathcal{D}_{\text{train}}}\right) && (m \text{ equal terms}) \\
&\quad= \left(\frac{1}{m}\cdot m\right)\text{Err}_{\mathcal{D}_{\text{train}}} && \text{(regroup the product)} \\
&\quad= 1\cdot\text{Err}_{\mathcal{D}_{\text{train}}} && (\tfrac{1}{m}\cdot m = 1) \\
&\quad= \text{Err}_{\mathcal{D}_{\text{train}}} && \text{(multiplying by 1 changes nothing)}
\end{aligned}
$$

The step labeled "each term, as above" is the one that needs the held-out set to be independent of $\mathcal{D}_{\text{train}}$.

Theorem 0.5 (The held-out sample mean loss estimates the generalization error)**.** *Let a fitted model $\hat{y}$ be fitted to a training set $\mathcal{D}_{train}$, and let a held-out set of m observations $(X_{*,1}, Y_{*,1}), \ldots, (X_{*,m}, Y_{*,m})$ be drawn from the same population as (X_*, Y_*), each independently of $\mathcal{D}_{train}$. Write $\ell_{*,j}$ for the loss of $\hat{y}$ on held-out observation j. Then, given $\mathcal{D}_{train}$, the expected sample mean loss of the held-out predictions is the generalization error:*

$$
\mathbb{E}\left[\frac{1}{m}\sum_{j=1}^{m}\ell_{*,j} \mid \mathcal{D}_{train}\right] = \text{Err}_{\mathcal{D}_{train}}.
$$

Proof

Proof. This is Solution 0.8. □

Remark (Training error is optimistic)

Remark 0.13 (Training error is optimistic). The same argument fails for the training set. The training pairs are not independent of $\mathcal{D}_{\text{train}}$: they are $\mathcal{D}_{\text{train}}$, and $\hat{y}$ was fitted to make their prediction errors small. So the argument of Theorem 0.5, that each loss has

conditional expectation $\mathrm{Err}_{\mathcal{D}_{\text{train}}}$, does not hold for them, and the training error $\overline{\mathrm{err}}$ is typically *smaller* than $\mathrm{Err}_{\mathcal{D}_{\text{train}}}$ [4, Sec. 7.4]. Exercise 0.1 shows the pattern: training sRMSE keeps falling as the degree grows, while held-out sRMSE does not.

Example 0.12 (Training error and generalization error by degree)**.** Real data never reveal $\mathrm{Err}_{\mathcal{D}_{\text{train}}}$, but a simulation can, because it can draw as many new observations as we like. Here the population is known: $X \sim \mathrm{Uniform}(0,1)$ and $Y \mid X = x \sim \mathrm{N}(\sin(2\pi x), \sigma^2)$, with $\sigma = 0.3$. We draw one training set of $n = 30$ observations, fit a polynomial in x of each degree from 1 to 10 by least squares, and score each fit on the training set and on 10,000 new observations. By Theorem 0.5, the second score has expectation $\mathrm{Err}_{\mathcal{D}_{\text{train}}}$ given the training set, and with 10,000 observations it is a precise estimate of it.

```r
sim_sigma <- 0.3
sim_truth <- function(x) sin(2 * pi * x)
sim_draw <- function(n) {
  x <- stats::runif(n)
  tibble::tibble(x = x, y = sim_truth(x) + stats::rnorm(n, sd = sim_sigma))
}

set.seed(1)
sim_train <- sim_draw(30)
sim_new <- sim_draw(10000)
sim_degrees <- 1:10

sim_errors <- tibble::tibble(degree = sim_degrees) |>
  dplyr::mutate(
    fit = purrr::map(degree, \(d) lm(y ~ poly(x, d), data = sim_train)),
    training = purrr::map_dbl(fit, rmse, data = sim_train),
    generalization = purrr::map_dbl(fit, rmse, data = sim_new)
  ) |>
  dplyr::select(-fit)

sim_errors
#> # A tibble: 10 x 3
#>    degree training generalization
#>     <int>    <dbl>          <dbl>
#>  1      1    0.567          0.547
#>  2      2    0.560          0.557
#>  3      3    0.231          0.315
#>  4      4    0.231          0.315
#>  5      5    0.216          0.306
#>  6      6    0.195          0.324
#>  7      7    0.194          0.326
#>  8      8    0.184          0.335
#>  9      9    0.183          0.335
#> 10     10    0.180          0.341
```

```r
sim_errors |>
  tidyr::pivot_longer(
    c(training, generalization),
    names_to = "error",
    values_to = "rmse"
  ) |>
  ggplot2::ggplot(ggplot2::aes(degree, rmse, colour = error, shape = error)) +
  ggplot2::geom_hline(yintercept = sim_sigma, linetype = "dashed") +
  ggplot2::geom_line() +
  ggplot2::geom_point(size = 2) +
  ggplot2::scale_x_continuous(breaks = sim_degrees) +
  ggplot2::scale_y_log10() +
  ggplot2::labs(
    x = "polynomial degree",
    y = "root mean squared error (log scale)",
    colour = NULL,
    shape = NULL
  ) +
  ggplot2::theme_bw() +
  ggplot2::theme(legend.position = "top")
```

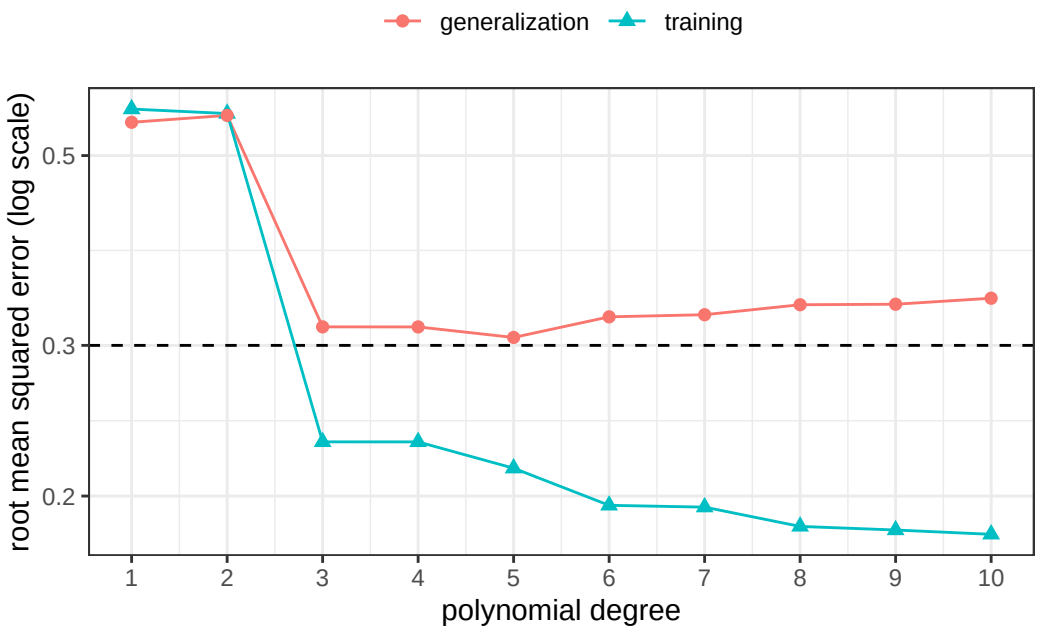


Figure 2: Training sRMSE and estimated generalization error (sRMSE on 10,000 new observations) of polynomial fits to one simulated training set of $n = 30$. The dashed line is the noise standard deviation σ, the square root of the residual variance.

```r
sim_best_generalization <-
  sim_errors$degree[which.min(sim_errors$generalization)]
sim_best_training <- sim_errors$degree[which.min(sim_errors$training)]
sim_n_below <- c(
  training = sum(sim_errors$training < sim_sigma),
  generalization = sum(sim_errors$generalization < sim_sigma)
)
c(
  best_by_training = sim_best_training,
  best_by_generalization = sim_best_generalization,
  n_below_sigma2 = sim_n_below
)
#>              best_by_training        best_by_generalization
#>                            10                             5
#>       n_below_sigma2.training n_below_sigma2.generalization
#>                             8                             0
```

The training error is smallest at degree 10 and never rises as the degree grows, because each polynomial family contains the one before it. The estimated generalization error is smallest at degree 5. No fitted model can have generalization pRMSE below $\sigma = 0.3$. Yet the training sRMSE falls below σ at 8 of the ten degrees, while the estimated generalization error falls below it at 0 of them: the training error is optimistic.

Training, validation, and test sets

Exercise 0.1 (Sizes of a three-way split)**.** The `mtcars` data frame has $n = 32$ cars. Suppose we assign $\lfloor 0.6n \rfloor$ cars to a training set, $\lfloor 0.2n \rfloor$ cars to a validation set, and all remaining cars to a test set. How many cars are in each set?

Solution

Solution 0.1.

$$\begin{aligned} \lfloor 0.6 \cdot 32 \rfloor &= \lfloor 19.2 \rfloor \\ &= 19 \end{aligned}$$

training cars,

$$\begin{aligned} \lfloor 0.2 \cdot 32 \rfloor &= \lfloor 6.4 \rfloor \\ &= 6 \end{aligned}$$

validation cars, and $32 - 19 - 6 = 7$ test cars.

Definition 0.1 (Training/validation/test split)**.** A **training/validation/test split** divides the observations $1, \dots, n$ of a data set into three disjoint sets of indices, I_{train}, I_{val} and I_{test}, and takes the observations in I_{train} as the training set:

$$\begin{aligned} &I_{\text{train}} \cup I_{\text{val}} \cup I_{\text{test}} = \{1, \dots, n\}, \\ &I_{\text{train}} \cap I_{\text{val}} = \emptyset, \\ &I_{\text{train}} \cap I_{\text{test}} = \emptyset, \\ &I_{\text{val}} \cap I_{\text{test}} = \emptyset, \\ &\mathcal{D}_{\text{train}} = ((x_i, y_i))_{i \in I_{\text{train}}}. \end{aligned} \tag{42}$$

Source: James et al. [6, pp. 198–201]; Wikipedia contributors [3].

Example 0.1 (Split sizes for 32 cars)**.** In Exercise 0.1, a rule of 60% / 20% / 20% on $n = 32$ cars gives sets of 19, 6, and 7 cars, and the 7 test cars are used only once, after the model is chosen.

Definition 0.2 (Model selection)**. Model selection** is choosing one fitted model from a set of candidate fitted models from different model families, using observations that were held out from fitting.
Source: Wikipedia contributors [12].

Example 0.2 (Choosing a polynomial degree)**.** In Example 0.2, the candidates are the least-squares fitted models from four polynomial model families, the families of degree 1 ("linear"), 2 ("quadratic"), 3 ("cubic") and 4 ("quartic"). Choosing among these four fitted models is model selection, and the observations it uses are cars held out from fitting.

Definition 0.3 (Validation set)**.** The **validation set** of a training/validation/test split is the held-out set of observations indexed by I_{val}, used for model selection among candidate fitted models fitted to the training set:

$$((x_i, y_i))_{i \in I_{\text{val}}}. \tag{43}$$

Source: Wikipedia contributors [3].

Definition 0.4 (Validation error)**.** The **validation error** of a fitted model $\hat{y}$ fitted to a training set $\mathcal{D}_{\text{train}}$ is its sample mean loss ℓ_i on the validation set, indexed by I_{val}:

$$\overline{\text{err}}_{\text{val}} \stackrel{\text{def}}{=} \frac{1}{|I_{\text{val}}|} \sum_{i \in I_{\text{val}}} \ell_i. \tag{44}$$

Definition 0.5 (Validation choice)**.** Given candidate fitting procedures $g_1, \ldots, g_L$, the **validation choice** $\hat{l}$ is the index of the candidate whose fitted model, fitted to the training set $\mathcal{D}_{\text{train}}$, has the smallest validation error:

$$\hat{l} \stackrel{\text{def}}{=} \arg\min_l \frac{1}{|I_{\text{val}}|} \sum_{i \in I_{\text{val}}} L(y_i, \hat{y}_l(x_i)). \tag{45}$$

Definition 0.6 (Test set)**.** The **test set** of a training/validation/test split is the held-out set of observations indexed by I_{test}, on which the fitted model $\hat{y}_{\hat{l}}$ that the validation choice picks is scored:

$$((x_i, y_i))_{i \in I_{\text{test}}}. \tag{46}$$

Source: Wikipedia contributors [3].

Definition 0.7 (Test error)**.** The **test error** of a fitted model $\hat{y}$ fitted to a training set $\mathcal{D}_{\text{train}}$ is its sample mean loss ℓ_i on the test set, indexed by I_{test}:

$$\overline{\text{err}}_{\text{test}} \stackrel{\text{def}}{=} \frac{1}{|I_{\text{test}}|} \sum_{i \in I_{\text{test}}} \ell_i. \tag{47}$$

Definition 0.8 (Error rate (misclassification rate))**.** The **error rate** of a fitted model $\hat{y}$ for a categorical outcome on a set of observations is the sample mean of the loss under 0-1 loss, $L(y, \hat{y}) = \mathbb{1}_{y \neq \hat{y}}$, which is 1 for a wrong prediction and 0 for a correct one. The error rate is the proportion of observations in the set whose predicted category differs from the observed category.

Definition 0.9 (Training error rate)**.** The **training error rate** of a fitted model is its error rate on the training set, which is the training error under 0-1 loss.

Definition 0.10 (Test error rate)**.** The **test error rate** of a fitted model is its error rate on the test set, which is the test error under 0-1 loss.

Example 0.3 (Test error rate of a low-birth-weight classifier)**.** The `birthwt` data hold births at Baystate Medical Center in Springfield, Massachusetts [7], as distributed in R's `MASS` package [8]. The outcome `low` is 1 for a birth weight under 2.5 kg and 0 otherwise. We fit a logistic regression of `low` on the mother's age, weight at her last menstrual period, smoking status, history of hypertension, and presence of uterine irritability. The fitted model predicts `low = 1` when its fitted probability exceeds 0.5. We use every third row as the test set and fit on the remaining rows, the training set.

R

```
birthwt <- read.csv("data/birthwt.csv")
is_test <- seq_len(nrow(birthwt)) %% 3 == 0

fit_low <- glm(
  low ~ age + lwt + smoke + ht + ui,
  family = binomial,
  data = birthwt[!is_test, ]
)

predicted_low <- function(data) {
  as.integer(predict(fit_low, newdata = data, type = "response") > 0.5)
}

error_rates <- c(
  training = mean(predicted_low(birthwt[!is_test, ]) != birthwt$low[!is_test]),
  test = mean(predicted_low(birthwt[is_test, ]) != birthwt$low[is_test])
)
error_rates
#> training     test
#> 0.261905 0.396825
```

Python

```python
import numpy as np
import pandas as pd
import statsmodels.formula.api as smf

birthwt = pd.read_csv("data/birthwt.csv")
is_test = (np.arange(1, len(birthwt) + 1) % 3 == 0)

fit_low = smf.logit(
    "low ~ age + lwt + smoke + ht + ui", data=birthwt[~is_test]
).fit(disp=False)


def predicted_low(data):
    return (fit_low.predict(data) > 0.5).astype(int)


error_rates = {
    "training": (predicted_low(birthwt[~is_test]) != birthwt["low"][~is_test]).mean(),
    "test": (predicted_low(birthwt[is_test]) != birthwt["low"][is_test]).mean(),
}
error_rates
#> {'training': np.float64(0.2619047619047619), 'test': np.float64(0.3968253968253968)}
```

Of the 189 births, 63 are in the test set. The covariates are chosen for simplicity, not for predictive quality. The fitted model gets the wrong birth-weight category for 39.7% of the test births, so its test error rate is 0.397. Its training error rate is 0.262.

Remark (What each set is for)

Remark 0.1 (What each set is for). Each set of a training/validation/test split has one job:

- the training set is used to fit each candidate model, which estimates its parameters;
- the validation set is used to compare the candidates, choose transformations, or choose values of a tuning parameter;
- the test set is used once, after the validation choice is made, to estimate the generalization error of the chosen fitted model.

The chosen fitted model won on the validation set partly because of that particular validation sample, so its prediction error on the validation set is optimistic as an estimate of its generalization error. Keeping the test set out of every choice is what makes its score free of that optimism.

For further reading, the machine learning notes define the same three sets in their own notation: the training set, the validation set, and the test set. They use a different symbol for the training set and a parenthesized observation index, but the same three-way split.

Remark (What the validation and test errors estimate)

Remark 0.2 (What the validation and test errors estimate). The validation error and the test error differ from the training error $\overline{\text{err}}$ only in the set on which the fitted model is evaluated. The model was not fitted to the validation or test set, so each estimates the generalization error of the fitted model without the optimism of the training error. The validation choice selects the model with the smallest validation error, so the validation error of the chosen model is still optimistic; the test error is computed once, after the choice, so it is not.

Hastie et al. [4, Sec. 7.2] use "test error" for the generalization error of the fitted model itself. Here the test error is the number computed from the test set, and the generalization error is what it estimates.

Each definition holds for any loss. Under squared-error loss, they are sample mean squared errors; under absolute-error loss, they are sample mean absolute errors.

Example 0.4 (Numerical example)**.** This example (Example 0.4) uses R's built-in `mtcars` dataset ($n = 32$ cars) to predict fuel efficiency (`mpg`) from vehicle weight (`wt`). It uses one random split to compare linear, quadratic, cubic, and quartic models (`mpg ~ wt`, `mpg ~ wt + I(wt^2)`, `mpg ~ wt + I(wt^2) + I(wt^3)`, and `mpg ~ wt + I(wt^2) + I(wt^3) + I(wt^4)`) on a validation set, then reports the chosen model's test sRMSE on untouched test data [6, p. 213].

Table 1: Sizes of the training, validation, and test splits

```
set.seed(108)
n <- nrow(mtcars)

n_train <- floor(0.6 * n)
n_valid <- floor(0.2 * n)

idx_train <- sample.int(n, size = n_train)
idx_remaining <- setdiff(seq_len(n), idx_train)
idx_valid <- sample(idx_remaining, size = n_valid)
idx_test <- setdiff(idx_remaining, idx_valid)

split_sizes <- tibble::tibble(
  split = c("training", "validation", "test"),
  n = c(length(idx_train), length(idx_valid), length(idx_test))
)

split_sizes
#> # A tibble: 3 x 2
#>   split          n
#>   <chr>      <int>
#> 1 training      19
#> 2 validation     6
#> 3 test           7
```

```
train_dat <- mtcars[idx_train, ]
valid_dat <- mtcars[idx_valid, ]
test_dat <- mtcars[idx_test, ]

model_linear <- lm(mpg ~ wt, data = train_dat)
model_quadratic <- lm(mpg ~ wt + I(wt^2), data = train_dat)
model_cubic <- lm(mpg ~ wt + I(wt^2) + I(wt^3), data = train_dat)
model_quartic <- lm(
  mpg ~ wt + I(wt^2) + I(wt^3) + I(wt^4),
  data = train_dat
)

candidate_models <- list(
  linear = model_linear,
  quadratic = model_quadratic,
  cubic = model_cubic,
  quartic = model_quartic
)

validation_results <- tibble::tibble(
  model = names(candidate_models)
) |>
  dplyr::mutate(
    training_RMSE = purrr::map_dbl(
      model,
      \(model_name) rmse(candidate_models[[model_name]], train_dat)
    ),
    validation_RMSE = purrr::map_dbl(
      model,
      \(model_name) rmse(candidate_models[[model_name]], valid_dat)
    )
  )

model_labels <- c(
  linear = "linear model",
  quadratic = "quadratic model",
  cubic = "cubic model",
  quartic = "quartic model"
)

chosen_model_name <-
  validation_results |>
  dplyr::arrange(validation_RMSE) |>
  dplyr::slice(1) |>
  dplyr::pull(model)
```



```
chosen_model_label <- model_labels[[chosen_model_name]]

chosen_model <- candidate_models[[chosen_model_name]]
chosen_model_validation_rmse <- rmse(chosen_model, valid_dat)

performance_comparison <- tibble::tibble(
```

```r
partition_colors <- c(
  training = "#1b9e77",
  validation = "#d95f02",
  test = "#7570b3"
)
partition_shapes <- c(
  training = 16,
  validation = 17,
  test = 15
)
wt_range <- range(c(train_dat$wt, valid_dat$wt, test_dat$wt))
mpg_range <- range(c(train_dat$mpg, valid_dat$mpg, test_dat$mpg))
wt_grid <- seq(wt_range[1], wt_range[2], length.out = 200)

partition_data <- dplyr::bind_rows(
  training = train_dat,
  validation = valid_dat,
  test = test_dat,
  .id = "series"
)

plot_partitions_with_fit <- function(model, fit_label) {
  series_levels <- c(names(partition_colors), fit_label)
  partition_data <- partition_data |>
    dplyr::mutate(
      series = factor(partition_data$series, levels = series_levels)
    )
  grid_data <- tibble::tibble(wt = wt_grid)
  fitted_curve <- tibble::tibble(
    wt = wt_grid,
    mpg = predict(model, newdata = grid_data),
    series = factor(fit_label, levels = series_levels)
  )
  # nolint start: object_usage_linter.
  ggplot2::ggplot(
    partition_data,
    ggplot2::aes(
      x = wt,
      y = mpg,
      color = series,
      shape = series
    )
  ) +
    ggplot2::geom_point() +
    ggplot2::geom_line(
      ggplot2::aes(linetype = series),
      data = fitted_curve,
      linewidth = 1
    ) +
    ggplot2::scale_color_manual(
      limits = series_levels,
      values = c(partition_colors, stats::setNames("blue", fit_label))
    ) +
```

```
plot_partitions_with_fit(model_linear, "linear fit")
```

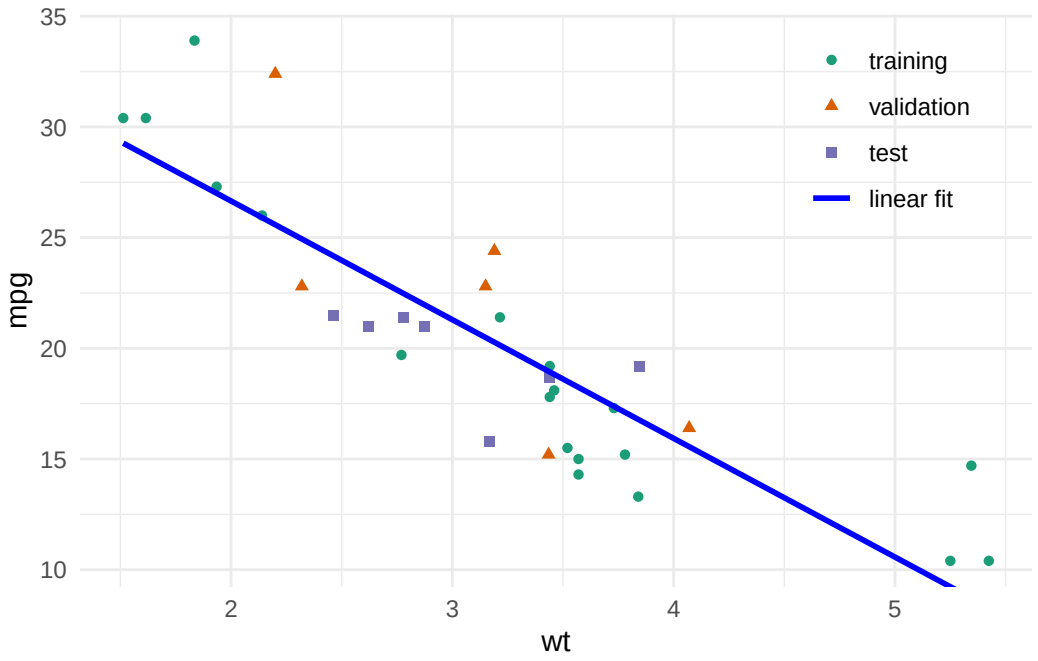


Figure 3: Linear model fit superimposed on data partitions

```
plot_partitions_with_fit(model_quadratic, "quadratic fit")
```

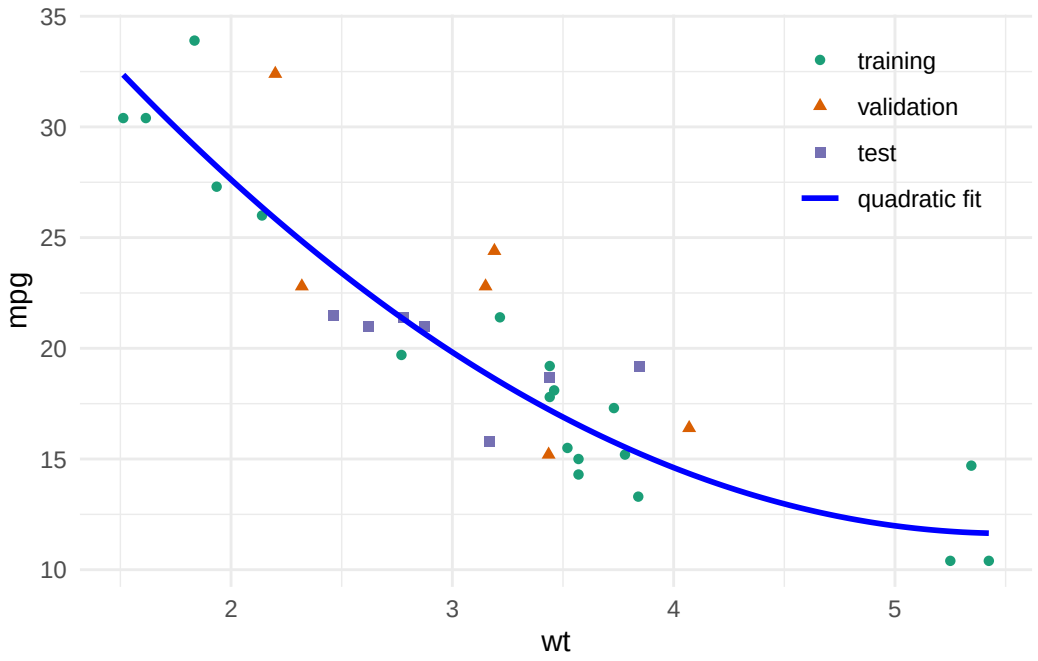


Figure 4: Quadratic model fit superimposed on data partitions

```
plot_partitions_with_fit(model_cubic, "cubic fit")
```

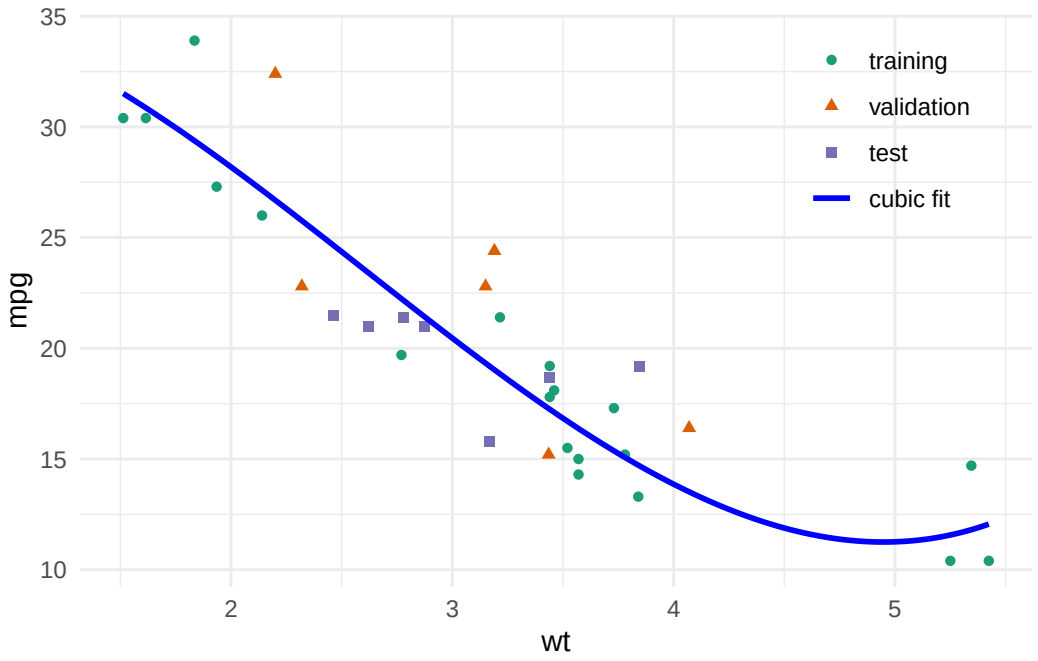


Figure 5: Cubic model fit superimposed on data partitions

```
plot_partitions_with_fit(model_quartic, "quartic fit")
```

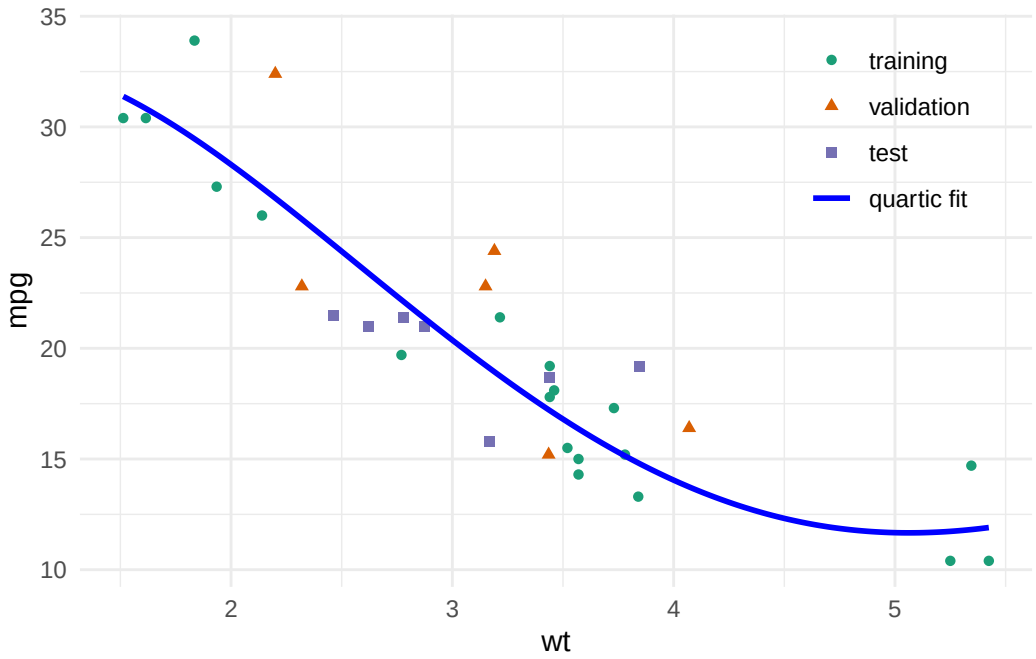


Figure 6: Quartic model fit superimposed on data partitions

Table 2: Training and validation sRMSE for candidate models

```
validation_results
#> # A tibble: 4 x 3
#>   model     training_RMSE validation_RMSE
#>   <chr>             <dbl>           <dbl>
#> 1 linear             2.80            3.84
#> 2 quadratic          1.96            4.16
#> 3 cubic              1.91            3.98
#> 4 quartic            1.90            3.97
```

The chosen model is the linear model, because it has the lowest validation sRMSE of the four. This split uses different cars from Exercise 0.1, and its validation set has only 6 cars, so the model it picks can differ from the one chosen there.

Table 3: Validation and test sRMSE for the chosen model

```
performance_comparison
#> # A tibble: 2 x 3
#>   split      model   RMSE
#>   <chr>      <chr>  <dbl>
#> 1 validation linear  3.84
#> 2 test       linear  2.44
```

Source

The validation-set lab of James et al. [6, p. 213].

k-fold cross-validation

Exercise 0.1 (Folds of the `mtcars` data)**.** Suppose we use $k = 4$-fold cross-validation on the $n = 32$ cars in `mtcars`.

1. How many cars are in each fold?
2. Each time a model is fit, how many cars are used to fit it?
3. How many models are fit in total?
4. How many times is a given car used for fitting, and how many times for prediction?

Solution

Solution 0.1.

1. $32/4 = 8$ cars per fold.
2. Each fit leaves out one fold, so it uses $32 - 8 = 24$ cars.
3. One model per fold, so 4 models.
4. A given car is in exactly one fold. It is used for prediction once, when its fold is left out, and for fitting in the other 3 fits.

Definition 0.1 (Folds)**.** A division of a training set $\mathcal{D}_{\text{train}}$, indexed by I, into k **folds** is a partition of I into k nonempty, disjoint subsets $\mathcal{F}_1, \ldots, \mathcal{F}_k$:

$$\mathcal{F}_1 \cup \cdots \cup \mathcal{F}_k = I. \tag{48}$$

Remark (Source: Folds)

Remark 0.1 (Source: Folds). James et al. [6, Sec. 5.1.3] and Hastie et al. [4, Sec. 7.10.1], who split the data into k roughly equal-sized parts.

Definition 0.2 (Fold size)**.** The **size** n_j of fold $\mathcal{F}_j$ is the number of observations it contains:

$$n_j \stackrel{\text{def}}{=} |\mathcal{F}_j|. \tag{49}$$

Definition 0.3 (Fold assignment)**.** Given folds $\mathcal{F}_1, \ldots, \mathcal{F}_k$, the **fold assignment** $\kappa(i)$ of observation i is the index of the fold that contains it:

$$i \in \mathcal{F}_{\kappa(i)}. \tag{50}$$

Remark (Source: Fold assignment)

Remark 0.2 (Source: Fold assignment). The fold-assignment function $\kappa(i)$ follows Hastie et al. [4, Sec. 7.10.1].

Definition 0.4 (Fold-out fitted model)**.** For fold $\mathcal{F}_j$, the **fold-out fitted model** $\hat{y}^{(-j)}(\cdot)$ is the fitted model that a fitting procedure g produces from the observations outside that fold:

$$\hat{y}^{(-j)}(x) \stackrel{\text{def}}{=} \hat{y}_{((x_i,y_i))_{i\in I\setminus\mathcal{F}_j}}(x). \tag{51}$$

Remark (Source: Fold-out fitted model)

Remark 0.3 (Source: Fold-out fitted model). Hastie et al. [4, Sec. 7.10.1], who write the fitted function with the kth part of the data removed as $\hat{f}^{-k}$.

Definition 0.5 (k-fold cross-validation)**.** In **k-fold cross-validation**:

1. The data are randomly divided into k folds of equal, or nearly equal, size.
2. For each fold $\mathcal{F}_j$, $j = 1, \ldots, k$:
   - Fit the model to all data *except* fold $\mathcal{F}_j$, giving the fold-out fitted model $\hat{y}^{(-j)}(\cdot)$.
   - Compute predicted values for the observations in fold $\mathcal{F}_j$.

3. Compute a summary prediction-error measure across all folds.

Source: Wikipedia contributors [13].

Remark (Cross-validation instead of a single validation set)

Remark 0.4 (Cross-validation instead of a single validation set). If data are limited, k-fold cross-validation can replace the single validation set of Definition 0.1 for more stable tuning. In Exercise 0.1, $k = 4$ folds of 8 cars each give 4 fits on 24 cars each.

Definition 0.6 (Cross-validation estimate)**.** The **k-fold cross-validation estimate** $\widehat{\mathrm{Err}}_{\mathrm{CV}(k)}$ is the sMSE of the predictions that each observation $i \in I$ of the training set receives from the fold-out fitted model of its fold assignment $\kappa(i)$ in k-fold cross-validation:

$$\widehat{\mathrm{Err}}_{\mathrm{CV}(k)} \stackrel{\mathrm{def}}{=} \frac{1}{n} \sum_{i \in I} \left(\hat{y}^{(-\kappa(i))}(x_i) - y_i \right)^2. \tag{52}$$

Remark (Source: Cross-validation estimate)

Remark 0.5 (Source: Cross-validation estimate). Hastie et al. [4, Sec. 7.10.1], for the per-observation form with the fold index $\kappa(i)$.

Exercise 0.2 (The cross-validation estimate, fold by fold)**.** Rewrite the cross-validation estimate Equation 52 as $1/n$ times a double sum: an outer sum over the fold numbers $j = 1, \ldots, k$ and an inner sum over the observations i in fold $\mathcal{F}_j$.

Solution

Solution 0.2. The folds $\mathcal{F}_1, \ldots, \mathcal{F}_k$ partition I, so summing over the observations of each fold and then over the folds adds each term of Equation 52 exactly once:

$$\begin{aligned} \widehat{\mathrm{Err}}_{\mathrm{CV}(k)} &= \frac{1}{n} \sum_{i \in I} \left(\hat{y}^{(-\kappa(i))}(x_i) - y_i \right)^2 && \text{(definition)} \\ &= \frac{1}{n} \sum_{j=1}^{k} \sum_{i \in \mathcal{F}_j} \left(\hat{y}^{(-\kappa(i))}(x_i) - y_i \right)^2 && \text{(group by fold)} \end{aligned}$$

Exercise 0.3 (Simplify the summand of the cross-validation estimate)**.** In the double sum from Exercise 0.2, simplify the summand so that the summand no longer uses the

fold assignment $\kappa(i)$.

Solution

Solution 0.3. For $i \in \mathcal{F}_j$, Equation 50 also puts i in $\mathcal{F}_{\kappa(i)}$, and the folds are disjoint, so $\kappa(i) = j$. Starting from the grouped sum of Solution 0.2:

$$\begin{aligned}\widehat{\mathrm{Err}}_{\mathrm{CV}(k)} &= \frac{1}{n}\sum_{j=1}^{k}\sum_{i\in\mathcal{F}_j}\left(\hat{y}^{(-\kappa(i))}(x_i) - y_i\right)^2 \quad \text{(grouped by fold)} \\ &= \frac{1}{n}\sum_{j=1}^{k}\sum_{i\in\mathcal{F}_j}\left(\hat{y}^{(-j)}(x_i) - y_i\right)^2 \qquad (\kappa(i) = j \text{ for } i \in \mathcal{F}_j)\end{aligned}$$

In the last line, the fitted model $\hat{y}^{(-j)}(\cdot)$ depends only on the outer index, so each inner sum uses a single fitted model.

Theorem 0.1 (The cross-validation estimate, fold by fold)**.** *The cross-validation estimate sums, fold by fold, the squared errors of each fold's observations under the fitted model fitted without that fold, and divides the total by n:*

$$\widehat{\mathrm{Err}}_{CV(k)} = \frac{1}{n}\sum_{j=1}^{k}\sum_{i\in\mathcal{F}_j}\left(\hat{y}^{(-j)}(x_i) - y_i\right)^2. \tag{53}$$

Proof

Proof. The proof combines Solution 0.2 and Solution 0.3. □

Definition 0.7 (Fold mean squared error)**.** The **fold mean squared error** MSE_j of fold $\mathcal{F}_j$ is the sMSE of the fold-out fitted model's predictions for the n_j (fold size) observations in that fold:

$$\mathrm{MSE}_j \overset{\text{def}}{=} \frac{1}{n_j}\sum_{i\in\mathcal{F}_j}\left(\hat{y}^{(-j)}(x_i) - y_i\right)^2. \tag{54}$$

Remark (Source: Fold mean squared error)

Remark 0.6 (Source: Fold mean squared error). James et al. [6, Sec. 5.1.3], who write the fold mean squared errors as $\mathrm{MSE}_1, \ldots, \mathrm{MSE}_k$.

Remark (What the cross-validation estimate estimates)

Remark 0.7 (What the cross-validation estimate estimates). Each prediction in Equation 52 comes from a fitted model that did not see the observation it predicts, so each squared error is scored as in Theorem 0.5. But the model for fold $\mathcal{F}_j$ was fitted to $n - n_j$ observations, not n, and the k models differ from one another and from the model fitted to all n. So $\widehat{\mathrm{Err}}_{\mathrm{CV}(k)}$ estimates the expected generalization error of the fitting *procedure* g at the smaller training-set sizes $n - n_j$, which is $\mathrm{Err}^g(n(k-1)/k)$ when every fold has n/k observations, rather than $\mathrm{Err}_{\mathcal{D}_{\mathrm{train}}}$ for the one model fitted to all the data [4, Sec. 7.12].
When every fold has $n_j = n/k$ observations, $\widehat{\mathrm{Err}}_{\mathrm{CV}(k)}$ is also the average of the k fold sMSEs $\mathrm{MSE}_1, \ldots, \mathrm{MSE}_k$, the form James et al. [6, Sec. 5.1.3] use.

Remark (Choosing the number of folds)

Remark 0.8 (Choosing the number of folds). The number of folds k trades bias against variance in the cross-validation estimate itself [6, Sec. 5.1.4]:

- **Bias.** Each fitted model in Equation 52 is fitted to about $n(k-1)/k$ observations. A model fitted to fewer observations usually has a larger generalization error, so $\widehat{\mathrm{Err}}_{\mathrm{CV}(k)}$ tends to overstate the expected generalization error of a model fitted to all n. The overstatement shrinks as k grows.
- **Variance.** As k grows, the k training sets overlap more, so the k fitted models are more alike and their errors may be more strongly correlated. An average of strongly correlated errors varies more from one data set to another than an average of weakly correlated ones. This is a heuristic, not a theorem [4, Sec. 7.10]. How correlated the errors are depends on how much the fitted model changes when one observation changes, so for a stable fitting method the variance need not rise with k.
- **Computation.** The model is fitted k times.

$k = 5$ or $k = 10$ is the usual compromise.

Example 0.1 (Cross-validation versus the generalization error)**.** This example applies k-fold cross-validation with $k = 5$ to the simulated training set of Example 0.12, using only those 30 observations. The function `assign_folds()` assigns each observation to a fold at random, `cv_squared_errors()` fits the model once per fold without that fold and returns each observation's squared prediction error, and `cv_mse()` averages them, which is Equation 52.

```r
assign_folds <- function(n, k) {
  sample(rep_len(seq_len(k), n))
}

cv_squared_errors <- function(formula, data, folds) {
  y <- stats::model.response(stats::model.frame(formula, data))
  sq_errors <- numeric(nrow(data))
  for (j in unique(folds)) {
    in_fold <- folds == j
    fit <- lm(formula, data = data[!in_fold, ])
    pred <- predict(fit, newdata = data[in_fold, ])
    sq_errors[in_fold] <- (pred - y[in_fold])^2
  }
  sq_errors
}

cv_mse <- function(formula, data, folds) {
  mean(cv_squared_errors(formula, data, folds))
}

set.seed(5)
sim_folds <- assign_folds(nrow(sim_train), k = 5)
table(fold = sim_folds)
#> fold
#> 1 2 3 4 5
#> 6 6 6 6 6
```

The same fold assignment is used for every degree, so the degrees are compared on the same splits. The table and figure report the square root of each cross-validation estimate, so that it is in the same unit as the training sRMSE and generalization pRMSE of Example 0.12.

```r
sim_errors_cv <- sim_errors |>
  dplyr::mutate(
    cv_5 = purrr::map_dbl(
      degree,
      \(d) sqrt(cv_mse(y ~ poly(x, d), sim_train, sim_folds))
    )
  )

sim_errors_cv
#> # A tibble: 10 x 4
#>    degree training generalization  cv_5
#>     <int>    <dbl>          <dbl> <dbl>
#>  1      1    0.567          0.547 0.597
#>  2      2    0.560          0.557 0.610
#>  3      3    0.231          0.315 0.287
#>  4      4    0.231          0.315 0.313
#>  5      5    0.216          0.306 0.323
#>  6      6    0.195          0.324 0.307
#>  7      7    0.194          0.326 0.395
#>  8      8    0.184          0.335 0.316
#>  9      9    0.183          0.335 0.536
#> 10     10    0.180          0.341 1.39
```

```
sim_errors_cv |>
  tidyr::pivot_longer(
    c(training, generalization, cv_5),
    names_to = "error",
    values_to = "rmse"
  ) |>
  dplyr::mutate(
    error = factor(
      error,
      levels = c("training", "cv_5", "generalization"),
      labels = c("training", "5-fold CV", "generalization")
    )
  ) |>
  ggplot2::ggplot(ggplot2::aes(degree, rmse, colour = error, shape = error)) +
  ggplot2::geom_hline(yintercept = sim_sigma, linetype = "dashed") +
  ggplot2::geom_line() +
  ggplot2::geom_point(size = 2) +
  ggplot2::scale_x_continuous(breaks = sim_degrees) +
  ggplot2::scale_y_log10() +
  ggplot2::labs(
    x = "polynomial degree",
    y = "root mean squared error (log scale)",
    colour = NULL,
    shape = NULL
  ) +
  ggplot2::theme_bw() +
  ggplot2::theme(legend.position = "top")
```

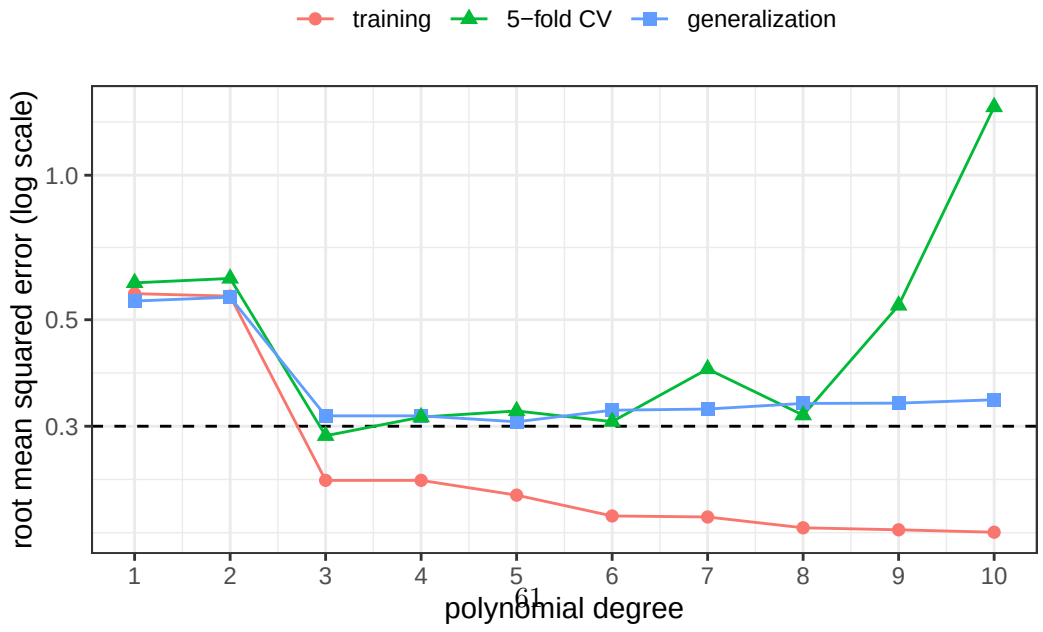


Figure 7: Figure 2 with the 5-fold cross-validation estimate added. Only the training and cross-validation curves could be computed from real data.

```r
sim_best_cv <- sim_errors_cv$degree[which.min(sim_errors_cv$cv_5)]
sim_n_fit <- nrow(sim_train) - max(table(sim_folds))
sim_top <- dplyr::filter(sim_errors_cv, degree == max(sim_degrees))
sim_cv_ratio_top <- sim_top$cv_5 / sim_top$generalization
c(
  best_by_cv = sim_best_cv,
  best_by_generalization = sim_best_generalization,
  n_per_fit = sim_n_fit,
  cv_over_generalization_at_top_degree = sim_cv_ratio_top
)
#>                            best_by_cv              best_by_generalization
#>                               3.00000                             5.00000
#>                             n_per_fit cv_over_generalization_at_top_degree
#>                              24.00000                             4.07316
```

The cross-validation estimate is smallest at degree 3, and the estimated generalization error at degree 5. Unlike the training error, the cross-validation estimate does not keep falling as the degree grows, so it does not simply reward higher degrees. It is not an estimate of $\text{Err}_{\mathcal{D}_{\text{train}}}$ for the fit to all 30 observations, though (Remark 0.7). Each of its fits uses only 24 observations. At degree 10, polynomials fitted to 24 points predict their held-out points much worse than the fit to all 30 predicts new data: the cross-validation estimate is 4.1 times the estimated generalization pRMSE.

Part of that gap is expected: by Remark 0.7, with g the least-squares fit of a degree-10 polynomial, the cross-validation estimate targets $\text{Err}^{g}(24)$, not the expected generalization error of a fit to 30 points. A simulation of many training sets of each size shows how much the size alone matters at this degree.

```
sim_generalization_draws <- function(
  n_train, degree, n_reps, draw, new_data, score
) {
  purrr::map_dbl(seq_len(n_reps), \(r) {
    fit <- lm(y ~ poly(x, degree), data = draw(n_train))
    score(fit, new_data)
  })
}

sim_n_reps <- 500
set.seed(24)
sim_size_effect <- tibble::tibble(
  n_train = c(sim_n_fit, nrow(sim_train))
) |>
  dplyr::mutate(
    draws = purrr::map(
      n_train,
      \(n) {
        sim_generalization_draws(
          n, max(sim_degrees), sim_n_reps,
          draw = sim_draw, new_data = sim_new, score = prediction_mse
        )
      }
    ),
    median = purrr::map_dbl(draws, stats::median),
    quantile_90 = purrr::map_dbl(draws, \(v) stats::quantile(v, 0.9)),
    mean = purrr::map_dbl(draws, mean),
    largest = purrr::map_dbl(draws, max),
    top_two_share = purrr::map_dbl(
      draws,
      \(v) sum(sort(v, decreasing = TRUE)[1:2]) / sum(v)
    )
  ) |>
  dplyr::select(-draws)

sim_size_effect
#> # A tibble: 2 x 6
#>   n_train median quantile_90   mean   largest top_two_share
#>     <int>  <dbl>       <dbl>  <dbl>     <dbl>         <dbl>
#> 1      24  1.14        207.  39068. 18318345.         0.965
#> 2      30  0.274        40.6   116.    24568.         0.508
```

Over 500 simulated training sets of each size, degree-10 fits to 24 points have median generalization pMSE 1.14, against 0.27 for fits to 30 points, and 90th percentiles of 207.31 and 40.65. The sample means are no guide to $\mathrm{Err}^g(24)$ here: the largest single value among the fits to 24 points is 18,000,000, from one fit that is wildly wrong away from the data, and the two largest values supply 97% and 51% of the two means. So fits of this degree to 24 points are much worse than fits to 30 points, and a large cross-validation estimate here is no surprise. How much of this particular gap comes from the training-set size, and how much from the chance of this one data set, a single run cannot say. The estimate also jumps around from one degree to the next, because it rests on only 30 observations rather than on 10,000 new ones.

Example 0.2 (Four-fold cross-validation of the `mtcars` polynomials)**.** This example applies the $k = 4$ folds of Exercise 0.1 to the four polynomial models of Exercise 0.1, now using all $n = 32$ cars and the `cv_mse()` function of Example 0.1. The fold assignment is random, so the estimate is too; to show how much, we repeat the whole procedure with 5 different random fold assignments.

Table 4: Square root of the 4-fold cross-validation estimate, by degree, for each of 5 random fold assignments

```
mtcars_degrees <- 1:4
n_repeats <- 5

set.seed(32)
mtcars_cv <-
  tidyr::expand_grid(repeat_id = seq_len(n_repeats), degree = mtcars_degrees) |>
  dplyr::mutate(
    rmse = {
      folds <- assign_folds(nrow(mtcars), k = 4)
      purrr::map_dbl(
        degree,
        \(d) sqrt(cv_mse(mpg ~ poly(wt, d), mtcars, folds))
      )
    },
    .by = repeat_id
  ) |>
  tidyr::pivot_wider(
    names_from = degree,
    values_from = rmse,
    names_prefix = "degree_"
  )

mtcars_cv
#> # A tibble: 5 x 5
#>   repeat_id degree_1 degree_2 degree_3 degree_4
#>       <int>    <dbl>    <dbl>    <dbl>    <dbl>
#> 1         1     3.42     2.67     2.69     2.92
#> 2         2     3.10     2.73     2.74     2.82
#> 3         3     3.14     2.82     2.86     3.00
#> 4         4     3.18     2.65     2.76     3.00
#> 5         5     3.38     2.74     2.78     2.81
```

Table 5: Degree with the smallest 4-fold cross-validated sRMSE in each of 5 random fold assignments

```
mtcars_cv_best <-
  mtcars_cv |>
  tidyr::pivot_longer(
    -repeat_id,
    names_to = "degree",
    names_prefix = "degree_",
    values_to = "rmse"
  ) |>
  dplyr::slice_min(rmse, n = 1, by = repeat_id)

mtcars_cv_best
#> # A tibble: 5 x 3
#>   repeat_id degree  rmse
#>       <int> <chr>  <dbl>
#> 1         1 2       2.67
#> 2         2 2       2.73
#> 3         3 2       2.82
#> 4         4 2       2.65
#> 5         5 2       2.74
```

```r
mtcars_cv_range <-
  mtcars_cv |>
  dplyr::summarise(
    dplyr::across(dplyr::starts_with("degree_"), \(v) diff(range(v)))
  )
mtcars_chosen <- sort(unique(mtcars_cv_best$degree))
mtcars_choice_text <- if (length(mtcars_chosen) == 1) {
  paste0(
    "but every assignment chose degree ", mtcars_chosen,
    ": the estimates moved, and the winner did not"
  )
} else {
  paste0(
    "and the chosen degree was ",
    knitr::combine_words(mtcars_chosen, and = " or "),
    ", depending on how the cars happened to be split"
  )
}
mtcars_cv_range
#> # A tibble: 1 x 4
#>   degree_1 degree_2 degree_3 degree_4
#>      <dbl>    <dbl>    <dbl>    <dbl>
#> 1    0.320    0.168    0.164    0.195
```

Each row of Table 4 holds, for each degree, the square root of $\widehat{\mathrm{Err}}_{\mathrm{CV}(4)}$ (Equation 52) for one fold assignment. Across fold assignments, the estimate for one degree varied by as much as 0.32 mpg, but every assignment chose degree 2: the estimates moved, and the winner did not. A single run of k-fold cross-validation reports one of these rows, so a small difference between two models' estimates may not survive a different split.

Leave-one-out cross-validation

Exercise 0.1 (The largest number of folds)**.** Suppose we take $k = n$ in k-fold cross-validation on the $n = 32$ cars in `mtcars`.

1. How many cars are in each fold?
2. How many models are fit, and how many cars is each fitted to?
3. If we run the procedure twice, with two different random fold assignments, can the two values of $\widehat{\mathrm{Err}}_{\mathrm{CV}(32)}$ (Equation 52) differ?

Solution

Solution 0.1.

1. $32/32 = 1$ car per fold.
2. One model per fold, so 32 models, each fitted to the $32 - 1 = 31$ other cars.
3. No. With one car per fold, every assignment of cars to folds produces the same 32 fits, only numbered differently, and Equation 52 does not depend on the numbering. Unlike Example 0.2, the estimate is not random.

Definition 0.1 (Leave-one-out cross-validation)**. Leave-one-out cross-validation (LOOCV)** is *k*-fold cross-validation with $k = n$, so that each of the n folds holds one observation, and the fold-out fitted model $\hat{y}^{(-\kappa(i))}(\cdot)$ of observation i's fold is fitted to all observations except observation i. Its cross-validation estimate (Equation 52 with $k = n$) is

$$\widehat{\mathrm{Err}}_{\mathrm{CV}(n)} = \frac{1}{n} \sum_{i \in I} \left(\hat{y}^{(-\kappa(i))}(x_i) - y_i \right)^2. \tag{55}$$

Source: James et al. [6, Sec. 5.1.2]; Wikipedia contributors [13].

Lemma 0.1 (Leave-one-out cross-validation does not depend on the fold assignment)**.** *Let $\mathcal{D}_{train} = ((x_i, y_i))_{i \in I}$ be a training set of $n = |I|$ observations, indexed by I. Then the cross-validation estimate $\widehat{\mathrm{Err}}_{CV(n)}$ of leave-one-out cross-validation ($k = n$) is the same for every fold assignment $\kappa(i)$.*

Proof

Proof. This is the solution to Exercise 0.1, part 3, worked in Solution 0.1. It uses Equation 52 and Equation 51.
Each step below changes one thing.

1. With $k = n$, the n nonempty, disjoint folds of Equation 48 partition the n indices of I, so each fold holds exactly one index. (A partition of n indices into n nonempty, disjoint subsets has one index per subset.)
2. For observation i, Equation 50 gives $i \in \mathcal{F}_{\kappa(i)}$, so $\mathcal{F}_{\kappa(i)} = \{i\}$. (Fold $\kappa(i)$ holds one index, and that index is i.)
3. Let g be the fitting procedure, which maps a training set to a fitted model. The fold-out fitted model of observation i is then $\hat{y}^{(-\kappa(i))}(x) = \hat{y}_{((x_{i'}, y_{i'}))_{i' \in I \setminus \{i\}}}(x)$. (Equation 51 with $\mathcal{F}_{\kappa(i)} = \{i\}$.)

4. This model is fitted to every observation except observation i, and it does not involve $\kappa(i)$ itself. (Step 3 depends on i and g only.)

5. Hence each term $\left(\hat{y}^{(-\kappa(i))}(x_i) - y_i\right)^2$ of Equation 52 is the same for every fold assignment. (Each term depends only on the fitted model in step 4, x_i and y_i.)

6. Therefore the sum over $i \in I$ in Equation 52, and its factor $1/n$, are the same for every fold assignment:

$$\widehat{\mathrm{Err}}_{\mathrm{CV}(n)} = \frac{1}{n} \sum_{i \in I} \left(\hat{y}^{(-\kappa(i))}(x_i) - y_i\right)^2.$$

Different assignments give the same n fits, with the folds numbered differently.

□

Definition 0.2 (Term vector)**.** Consider a model whose mean is a linear combination of p^* terms $t_1(x), \ldots, t_{p^*}(x)$ of the covariate value, such as 1, x and x^2. The **term vector** $\underset{\sim}{t}_i$ of observation i is the column of its terms:

$$\underset{\sim}{t}_i \stackrel{\text{def}}{=} \left(t_1(x_i), \ldots, t_{p^*}(x_i)\right)^\top. \tag{56}$$

ℹ Remark (The term vector of simple linear regression)

Remark 0.1 (The term vector of simple linear regression). In simple linear regression the terms are 1 and x, so the term vector $\underset{\sim}{t}_i$ is the covariate vector $\underset{\sim}{x}_i$.

Definition 0.3 (Cross-product matrix (sum-of-squares and cross-products matrix))**.** The **cross-product matrix** A of term vectors $\underset{\sim}{t}_i$, $i \in I$, of a training set indexed by I, is the sum of their outer products:

$$A \stackrel{\text{def}}{=} \sum_{i \in I} \underset{\sim}{t}_i \underset{\sim}{t}_i{}^\top. \tag{57}$$

ℹ Remark (Source: Cross-product matrix (sum-of-squares and cross-products matrix))

Remark 0.2 (Source: Cross-product matrix (sum-of-squares and cross-products matrix)). Also called a **sum-of-squares and cross-products matrix** [14], chap. "Matrix Cross-Products" (see the chapter online).

Remark (Cross-product matrix as a matrix product)

Remark 0.3 (Cross-product matrix as a matrix product). Let $\mathbf{D}$ be the matrix whose rows are the transposed term vectors $\underset{\sim}{t}_i^{\top}$, $i \in I$. The cross-product matrix then equals the matrix product

$$A = \mathbf{D}^{\top}\mathbf{D}.$$

Its diagonal entries are sums of squares of one term over the training set. Its off-diagonal entries are sums of cross-products of two different terms. Statisticians therefore call A the sum-of-squares and cross-products (SSCP) matrix of the terms. When every column of $\mathbf{D}$ has mean zero, $A/(|I|-1)$ is the sample covariance matrix of the terms.

Source

I restated the SSCP reading and the covariance special case from Sanchez [14], chap. "Matrix Cross-Products" (see the chapter online) in this page's notation. The book carries a non-commercial, share-alike license, so I paraphrased it and copied none of its text.

Definition 0.4 (Leverage)**.** Consider a linear model fitted by ordinary least squares, with term vectors $\underset{\sim}{t}_i$, $i \in I$, and an invertible cross-product matrix A. The **leverage** of observation i is [15]

$$h_i \overset{\text{def}}{=} \underset{\sim}{t}_i^{\top} A^{-1} \underset{\sim}{t}_i. \tag{58}$$

Source: Wikipedia contributors [15].

Remark (Source: Leverage)

Remark 0.4 (Source: Leverage). James et al. [6, Sec. 3.3.3], written here with term vectors and the matrix A, as in the vector form of the OLS estimates.

Theorem 0.1 (Leave-one-out cross-validation for least squares)**.** *For a linear model fitted by ordinary least squares with every leverage $h_i < 1$, the leave-one-out estimate can be computed from the single fit to all n observations, from its fitted values $\hat{y}_i$, their prediction errors $\varepsilon_{pred_i} = \hat{y}_i - y_i$, and the leverages:*

$$\widehat{\mathrm{Err}}_{CV(n)} = \frac{1}{n}\sum_{i \in I}\left(\frac{\varepsilon_{pred_i}}{1-h_i}\right)^2. \tag{59}$$

Proof

Proof. Not derived here; see the sources named under the theorem. The derivation needs a formula for how the inverse of the cross-product matrix A changes when one observation's outer product $\underset{\sim}{t}_i \underset{\sim}{t}_i^\top$ is removed from it (the Sherman–Morrison formula), which this page does not develop. □

Remark (Source: Leave-one-out cross-validation for least squares)

Remark 0.5 (Source: Leave-one-out cross-validation for least squares). James et al. [6, Sec. 5.1.2] states this formula for least squares linear or polynomial regression, and Hastie et al. [4, Sec. 7.10.1] for many linear fitting methods.

Example 0.1 (Leave-one-out cross-validation of the `mtcars` polynomials)**.** This example computes $\widehat{\mathrm{Err}}_{\mathrm{CV}(32)}$ for the four `mtcars` polynomials of Exercise 0.1 in two ways: by fitting each model 32 times, as in Equation 55, and by the single-fit formula of Theorem 0.1, with the leverages from `hatvalues()`.

```r
loocv_by_formula <- function(formula, data) {
  fit <- lm(formula, data = data)
  pred_error <- fitted(fit) - stats::model.response(model.frame(fit))
  mean((pred_error / (1 - hatvalues(fit)))^2)
}

mtcars_loocv <- tibble::tibble(degree = mtcars_degrees) |>
  dplyr::mutate(
    refitting = purrr::map_dbl(
      degree,
      \(d) cv_mse(mpg ~ poly(wt, d), mtcars, folds = seq_len(nrow(mtcars)))
    ),
    formula = purrr::map_dbl(
      degree,
      \(d) loocv_by_formula(mpg ~ poly(wt, d), mtcars)
    ),
    loocv_rmse = sqrt(formula)
  )

mtcars_loocv
#> # A tibble: 4 x 4
#>   degree refitting formula loocv_rmse
#>    <int>     <dbl>   <dbl>      <dbl>
#> 1      1     10.3    10.3        3.20
#> 2      2      7.79    7.79       2.79
#> 3      3      8.34    8.34       2.89
#> 4      4      8.71    8.71       2.95
```

```r
loocv_max_diff <- max(abs(mtcars_loocv$refitting - mtcars_loocv$formula))
mtcars_best_loocv <-
  mtcars_loocv$degree[which.min(mtcars_loocv$formula)]
c(max_difference = loocv_max_diff, best_degree = mtcars_best_loocv)
#> max_difference    best_degree
#>    8.88178e-15    2.00000e+00
```

The two columns differ by at most 8.9e-15, which is rounding error. The single-fit formula needs one fit per model instead of 32. LOOCV picks degree 2, and because LOOCV is not random (Lemma 0.1), rerunning it always gives the same choice.

Remark (When to use leave-one-out cross-validation)

Remark 0.6 (When to use leave-one-out cross-validation). LOOCV is the $k = n$ end of Remark 0.8. By Remark 0.7, $\widehat{\mathrm{Err}}_{\mathrm{CV}(n)}$ estimates $\mathrm{Err}^{g}(n-1)$ for the fitting procedure g: each fitted model is fitted to $n-1$ observations, so the bias of $\widehat{\mathrm{Err}}_{\mathrm{CV}(n)}$ as an estimate of $\mathrm{Err}^{g}(n)$ is the smallest of any k. But the n training sets are nearly identical, so the variance of $\widehat{\mathrm{Err}}_{\mathrm{CV}(n)}$ can be larger than that of 5- or 10-fold cross-validation [6, Sec. 5.1.4]. For least squares fits, Theorem 0.1 computes LOOCV from a single fit; for most other fitting methods, LOOCV requires n fits.

Choosing a model by cross-validation

Definition 0.1 (Cross-validation estimate of a named procedure)**.** Let a training set $\mathcal{D}_{\mathrm{train}}$, indexed by I, be divided into folds $\mathcal{F}_1, \ldots, \mathcal{F}_k$. The **cross-validation estimate of procedure** g on $\mathcal{D}_{\mathrm{train}}$, $\widehat{\mathrm{Err}}^{g}_{\mathrm{CV}(k)}(\mathcal{D}_{\mathrm{train}})$, is the cross-validation estimate computed with fold-out fitted models that g produces from $\mathcal{D}_{\mathrm{train}}$:

$$\widehat{\mathrm{Err}}^{g}_{\mathrm{CV}(k)}(\mathcal{D}_{\mathrm{train}}) \stackrel{\mathrm{def}}{=} \frac{1}{n} \sum_{i \in I} \left(\hat{y}^{(-\kappa(i))}(x_i) - y_i \right)^2. \tag{60}$$

Definition 0.2 (Cross-validation choice procedure)**.** Given candidate fitting procedures $g_1, \ldots, g_L$, the **cross-validation choice procedure** g^{CV} is the fitting procedure that, applied to a training set $\mathcal{D}_{\mathrm{train}}$, divides $\mathcal{D}_{\mathrm{train}}$ into k folds, computes each candidate's cross-validation estimate on $\mathcal{D}_{\mathrm{train}}$ with those folds, and applies to $\mathcal{D}_{\mathrm{train}}$ the candidate with the smallest estimate, taking the smallest l if several tie:

$$g^{\mathrm{CV}}(\mathcal{D}_{\mathrm{train}}) \stackrel{\mathrm{def}}{=} \hat{y}_{\min \arg\min_l \widehat{\mathrm{Err}}^{g_l}_{\mathrm{CV}(k)}(\mathcal{D}_{\mathrm{train}})}. \tag{61}$$

Example 0.1 (Inputs of the cross-validation choice procedure)**.** The cross-validation choice procedure g over polynomials of several degrees is specified by these inputs:

- model family $\mathcal{H}$: the polynomials of every candidate degree;
- objective function $\mathcal{J}$: the residual sum of squares within each candidate, and the cross-validation estimate between candidates;
- optimization algorithm $\mathcal{A}$: the normal equations for each candidate, then keeping the candidate with the smallest cross-validation estimate of its expected generalization error $\mathrm{Err}^{g_d}(n)$;
- random seed $\mathfrak{s}$: fixes the random fold assignment;

- tuning parameter λ: none, since cross-validation chooses the degree;
- preprocessing D: none.

Definition 0.3 (Nested cross-validation)**. Nested cross-validation** is k-fold cross-validation of the cross-validation choice procedure g^{CV}: its estimate on a training set $\mathcal{D}_{\text{train}}$ is

$$\widehat{\mathrm{Err}}^{g^{\mathrm{CV}}}_{\mathrm{CV}(k)}(\mathcal{D}_{\text{train}}). \tag{62}$$

Remark (Source: Nested cross-validation)

Remark 0.1 (Source: Nested cross-validation). Hastie et al. [4, Sec. 7.10.2] for cross-validating every step of a procedure, choices included.

Remark (What is nested)

Remark 0.2 (What is nested). Each fold-out fitted model of g^{CV} is computed from the observations outside one outer fold $\mathcal{F}_j$, so the whole choice, including the cross-validation that g^{CV} runs inside those observations, never sees $\mathcal{F}_j$. That inner cross-validation is the nested one.

Remark (Cross-validation replaces the validation set, not the test set)

Remark 0.3 (Cross-validation replaces the validation set, not the test set). Used to choose between models, cross-validation takes the place of the *validation* set of Definition 0.1, the one consulted again and again, and not of the test set, which stays unused until the end. The cross-validation estimate of the winning model is optimistic: that model won partly because the particular sample and split happened to favor it, so its estimate is the smallest of several noisy estimates. An honest estimate of the chosen model's error needs data that took no part in the choice [4, Sec. 7.2], such as a test set, or the outer folds of nested cross-validation.

Source

Based on a Spring 2025 lecture on generalization by Logan Sizemore.

Definition 0.4 (Fold-based standard error)**.** The **fold-based standard error** $\mathrm{SE}_{\mathrm{CV}(k)}$ of a k-fold cross-validation estimate is the sample standard deviation of the k fold sMSEs, divided by $\sqrt{k}$:

$$\mathrm{SE}_{\mathrm{CV}(k)} \stackrel{\text{def}}{=} \frac{\widehat{\mathrm{SD}}(\mathrm{MSE}_1, \dots, \mathrm{MSE}_k)}{\sqrt{k}}. \tag{63}$$

i Remark (Source: Fold-based standard error)

Remark 0.4 (Source: Fold-based standard error). Hastie et al. [4, Sec. 7.10.1], who call the result a standard error.

i Remark (The fold-based standard error is only a rough guide)

Remark 0.5 (The fold-based standard error is only a rough guide). Hastie et al. [4, Sec. 7.10.1] draw standard-error bars computed from the per-fold errors as in Equation 63. That formula treats the k fold errors as if they were independent, which they are not: every pair of folds shares $k-2$ folds of training data. So $\mathrm{SE}_{\mathrm{CV}(k)}$ is a rough indication of how much $\widehat{\mathrm{Err}}_{\mathrm{CV}(k)}$ might change with a different sample, not a standard error of $\widehat{\mathrm{Err}}_{\mathrm{CV}(k)}$.

Definition 0.5 (Fold mean squared error of a named procedure)**.** The **fold mean squared error of procedure** g for fold $\mathcal{F}_j$ of a training set $\mathcal{D}_{\text{train}}$, indexed by I, is the fold mean squared error computed with the fold-out fitted model that g produces:

$$\mathrm{MSE}_j^g(\mathcal{D}_{\text{train}}) \stackrel{\text{def}}{=} \frac{1}{n_j} \sum_{i \in \mathcal{F}_j} \left(\hat{y}^{(-j)}(x_i) - y_i\right)^2. \tag{64}$$

Definition 0.6 (Fold-based standard error of a named procedure)**.** The **fold-based standard error of procedure** g on a training set $\mathcal{D}_{\text{train}}$, $\mathrm{SE}^g_{\mathrm{CV}(k)}(\mathcal{D}_{\text{train}})$, is the fold-based standard error computed from the fold sMSEs of g:

$$\mathrm{SE}^g_{\mathrm{CV}(k)}(\mathcal{D}_{\text{train}}) \stackrel{\text{def}}{=} \frac{\widehat{\mathrm{SD}}(\mathrm{MSE}_1^g(\mathcal{D}_{\text{train}}), \dots, \mathrm{MSE}_k^g(\mathcal{D}_{\text{train}}))}{\sqrt{k}}. \tag{65}$$

Definition 0.7 (One-standard-error rule)**.** Let $g_1, \dots, g_L$ be candidate fitting procedures, each at least as flexible as the one before, cross-validated on a training set $\mathcal{D}_{\text{train}}$ with the same folds, and let g_{l^*} be the one with the smallest cross-validation estimate. The **one-standard-error rule** chooses the smallest l whose estimate is within one fold-based standard error of the smallest:

$$\widehat{\mathrm{Err}}^{g_l}_{\mathrm{CV}(k)}(\mathcal{D}_{\text{train}}) \le \widehat{\mathrm{Err}}^{g_{l^*}}_{\mathrm{CV}(k)}(\mathcal{D}_{\text{train}}) + \mathrm{SE}^{g_{l^*}}_{\mathrm{CV}(k)}(\mathcal{D}_{\text{train}}). \tag{66}$$

Source: [4, Sec. 7.10.1], [6, Sec. 6.1.3].

Example 0.2 (The one-standard-error rule for the simulated polynomials)**.** This example applies 10-fold cross-validation to the simulated training set of Example 0.12, records the sMSE in each fold, and applies Definition 0.7 to the ten degrees.

```r
set.seed(10)
sim_folds_ten <- assign_folds(nrow(sim_train), k = 10)

sim_cv_ten <- tibble::tibble(degree = sim_degrees) |>
  dplyr::mutate(
    fold_mse = purrr::map(
      degree,
      \(d) {
        sq_errors <- cv_squared_errors(y ~ poly(x, d), sim_train, sim_folds_ten)
        as.numeric(tapply(sq_errors, sim_folds_ten, mean))
      }
    ),
    cv = purrr::map_dbl(fold_mse, mean),
    se = purrr::map_dbl(fold_mse, \(v) sd(v) / sqrt(length(v)))
  ) |>
  dplyr::select(-fold_mse)

sim_min <- dplyr::slice_min(sim_cv_ten, cv, n = 1)
sim_one_se <-
  sim_cv_ten |>
  dplyr::filter(cv <= sim_min$cv + sim_min$se) |>
  dplyr::slice_min(degree, n = 1)

sim_cv_ten
#> # A tibble: 10 x 3
#>    degree     cv     se
#>     <int>  <dbl>  <dbl>
#>  1      1 0.366  0.0962
#>  2      2 0.447  0.127
#>  3      3 0.0714 0.0219
#>  4      4 0.0846 0.0235
#>  5      5 0.0919 0.0244
#>  6      6 0.0792 0.0214
#>  7      7 0.202  0.104
#>  8      8 0.0619 0.0161
#>  9      9 0.247  0.134
#> 10     10 0.150  0.0611
```

With 30 observations and 10 folds, each fold holds 3 observations, so all ten folds have equal size and `cv` is $\widehat{\mathrm{Err}}_{\mathrm{CV}(10)}$ (Remark 0.7).

```
sim_cv_ten |>
  ggplot2::ggplot(ggplot2::aes(degree, cv)) +
  ggplot2::geom_hline(
    yintercept = sim_min$cv + sim_min$se,
    linetype = "dashed"
  ) +
  ggplot2::geom_errorbar(
    ggplot2::aes(ymin = cv - se, ymax = cv + se),
    width = 0.2
  ) +
  ggplot2::geom_point(size = 2) +
  ggplot2::scale_x_continuous(breaks = sim_degrees) +
  ggplot2::scale_y_log10() +
  ggplot2::labs(
    x = "polynomial degree",
    y = "10-fold CV estimate (log scale)"
  ) +
  ggplot2::theme_bw()
```

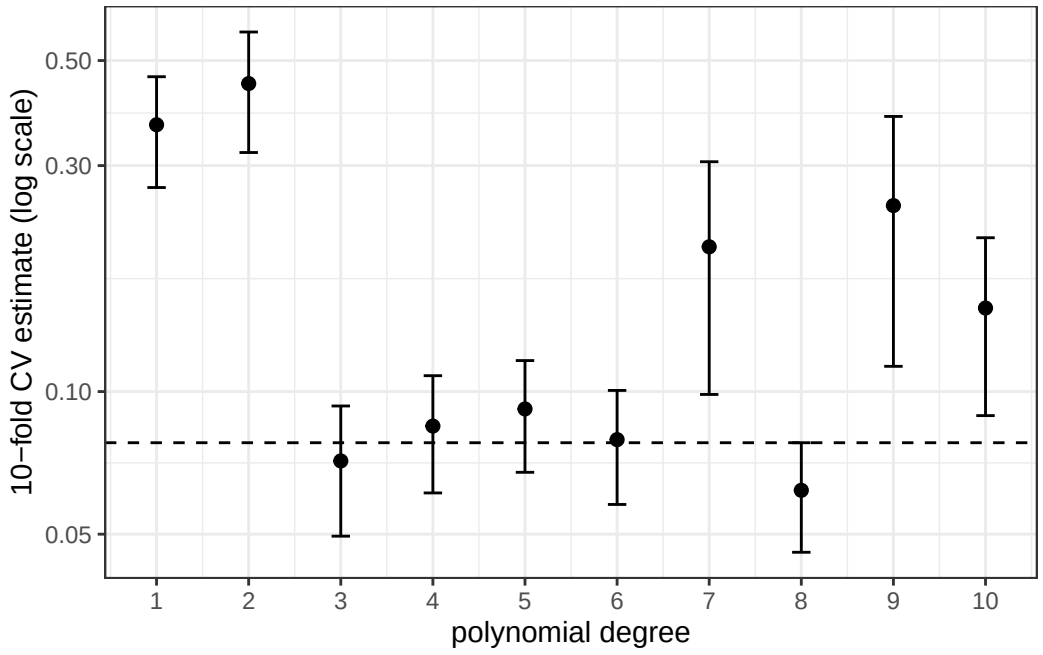


Figure 8: 10-fold cross-validation estimates, with bars of one fold-based standard error (Equation 63). The dashed line is one fold-based standard error above the smallest estimate.

The smallest estimate is at degree 8. The lowest degree within one fold-based standard error of it is degree 3, which is the one-standard-error rule's choice. The rule prefers the simpler model when the data cannot clearly tell it apart from the degree with the smallest estimate. Because the data are simulated, we can check both choices against the estimated generalization pRMSE of Example 0.12 for each: 0.335 for degree 8 and 0.315 for degree 3.

Criteria that adjust the training error

Definition 0.1 (Adjusted coefficient of determination)**.** Let a regression model with p predictors and a constant term be fitted to n observations, with $n > p+1$. The **adjusted coefficient of determination** of the fitted model is

$$R^2_{\text{adj}} \stackrel{\text{def}}{=} 1 - \frac{\text{RSS}(\widehat{\theta})/(n-p-1)}{\text{TSS}/(n-1)},$$

one minus the residual sum of squares per degree of freedom, divided by the total sum of squares per degree of freedom. A larger value is better.
Source: James et al. [6, Sec. 6.1.3].

Example 0.1 (Adjusted coefficient of determination of nested `mtcars` models)**.** The `mtcars` data set has the fuel economy `mpg` of 32 cars from the 1974 *Motor Trend* magazine. This example fits four nested linear models of `mpg`, each adding one predictor to the one before, and computes Definition 0.1 from its formula.

```r
mtcars_formulas <- list(
  mpg ~ wt,
  mpg ~ wt + hp,
  mpg ~ wt + hp + qsec,
  mpg ~ wt + hp + qsec + drat
)
mtcars_fits <- lapply(mtcars_formulas, lm, data = mtcars)
mtcars_full_sigma2 <- summary(mtcars_fits[[4]])$sigma^2

mtcars_tss <- sum((mtcars$mpg - mean(mtcars$mpg))^2)
mtcars_criteria <- tibble::tibble(
  predictors = 1:4,
  r_squared = vapply(mtcars_fits, \(f) summary(f)$r.squared, numeric(1)),
  adj_r_squared = vapply(
    mtcars_fits,
    \(f) {
      df <- nrow(mtcars) - length(coef(f))
      1 - (sum(resid(f)^2) / df) / (mtcars_tss / (nrow(mtcars) - 1))
    },
    numeric(1)
  )
)
mtcars_criteria
#> # A tibble: 4 x 3
#>   predictors r_squared adj_r_squared
#>        <int>     <dbl>         <dbl>
#> 1          1     0.753         0.745
#> 2          2     0.827         0.815
#> 3          3     0.835         0.817
#> 4          4     0.845         0.822
```

The coefficient of determination R^2 never falls as a predictor is added, and here it rises from 0.753 to 0.845. The adjusted coefficient of determination rises more slowly, from 0.745 to 0.822, because each added predictor costs a degree of freedom. It is largest for the model with 4 predictors. Its formula matches the `adj.r.squared` that `summary()` reports: TRUE.

Definition 0.2 (Mallows' C_p)**.** Let a least squares regression model with p predictors be fitted to n observations, and let $\hat{\sigma^2}$ be an estimate of the noise variance σ^2, usually the residual variance of the model with all the predictors under consideration. **Mallows'** C_p of the fitted model is

$$C_p \overset{\text{def}}{=} \frac{1}{n}\left(\text{RSS}(\hat{\underset{\sim}{\theta}}) + 2p\hat{\sigma^2}\right),$$

the average of the residual sum of squares plus a penalty of $2\hat{\sigma^2}$ for each predictor. A smaller value is better.
Source: Mallows [16]. This scaling is that of James et al. [6, Sec. 6.1.3].

Example 0.2 (Mallows' C_p of nested `mtcars` models)**.** This example computes Definition 0.2 for the four models of Example 0.1. The estimate $\hat{\sigma^2}$ of the noise variance is the residual variance of the model with all four predictors, 6.45.

```
mtcars_cp <- vapply(
  seq_along(mtcars_fits),
  \(k) {
    rss <- sum(resid(mtcars_fits[[k]])^2)
    (rss + 2 * k * mtcars_full_sigma2) / nrow(mtcars)
  },
  numeric(1)
)
tibble::tibble(predictors = 1:4, cp = mtcars_cp)
#> # A tibble: 4 x 2
#>   predictors    cp
#>        <int> <dbl>
#> 1          1  9.10
#> 2          2  6.90
#> 3          3  7.02
#> 4          4  7.05
```

The model with 2 predictors has the smallest C_p.

Definition 0.3 (Akaike information criterion (AIC))**.** Let a model with p^* estimated parameters be fitted by maximum likelihood to n observations, and let $\ell\left(\hat{\underset{\sim}{\theta}}\right)$ be the log-likelihood at the maximum likelihood estimate $\hat{\underset{\sim}{\theta}}$. The **Akaike information criterion** of the model is

$$\text{AIC} \overset{\text{def}}{=} -2\ell\left(\hat{\underset{\sim}{\theta}}\right) + 2p^*.$$

It is minus twice the maximized log-likelihood, plus two for each estimated parameter. For a Gaussian model, p^* includes the noise variance. A smaller AIC is better.
Source: Akaike [17].

Example 0.3 (AIC of nested `mtcars` models)**.** This example computes Definition 0.3 for the four models of Example 0.1, each fitted by maximum likelihood with Gaussian noise. The number of estimated parameters is the number of coefficients plus one for the noise variance.

```
mtcars_aic <- vapply(
  mtcars_fits,
  \(f) -2 * as.numeric(logLik(f)) + 2 * (length(coef(f)) + 1),
  numeric(1)
)
tibble::tibble(
  predictors = 1:4,
  aic = mtcars_aic,
  r_aic = vapply(mtcars_fits, AIC, numeric(1))
)
#> # A tibble: 4 x 3
#>   predictors   aic r_aic
#>        <int> <dbl> <dbl>
#> 1          1  166.  166.
#> 2          2  157.  157.
#> 3          3  157.  157.
#> 4          4  157.  157.
```

The `aic` column matches the `r_aic` column that `AIC()` returns: TRUE. The model with 2 predictors has the smallest AIC.

Definition 0.4 (Bayesian information criterion (BIC))**.** Let a model with p^* estimated parameters be fitted by maximum likelihood to n observations, and let $\ell\left(\underset{\sim}{\hat{\theta}}\right)$ be the log-likelihood at the maximum likelihood estimate $\underset{\sim}{\hat{\theta}}$. The **Bayesian information criterion** of the model is

$$\mathrm{BIC} \stackrel{\text{def}}{=} -2\ell\left(\underset{\sim}{\hat{\theta}}\right) + p^* \log\{n\}.$$

It is minus twice the maximized log-likelihood, plus $\log\{n\}$ for each estimated parameter. A smaller BIC is better.
Source: Schwarz [18].

Example 0.4 (BIC of nested `mtcars` models)**.** This example computes Definition 0.4 for the four models of Example 0.1, with p^* counted as in Example 0.3.

```r
mtcars_bic <- vapply(
  mtcars_fits,
  \(f) {
    -2 * as.numeric(logLik(f)) +
      (length(coef(f)) + 1) * log(nrow(mtcars))
  },
  numeric(1)
)
tibble::tibble(
  predictors = 1:4,
  bic = mtcars_bic,
  r_bic = vapply(mtcars_fits, BIC, numeric(1))
)
#> # A tibble: 4 x 3
#>   predictors   bic r_bic
#>        <int> <dbl> <dbl>
#> 1          1  170.  170.
#> 2          2  163.  163.
#> 3          3  164.  164.
#> 4          4  166.  166.
```

The `bic` column matches the `r_bic` column that `BIC()` returns: TRUE. The model with 2 predictors has the smallest BIC. With $n = 32$, each parameter costs $\log\{n\} = 3.47$ in BIC against 2 in AIC, so BIC penalizes each added parameter more heavily than AIC does.

Remark (Criteria that adjust the training error)

Remark 0.1 (Criteria that adjust the training error). Definition 0.1, Definition 0.2, Definition 0.3 and Definition 0.4 each correct a fit measure computed on the training set for the number of parameters. They need no held-out set, so they cost one fit per model, where cross-validation costs several.

For a least squares model with Gaussian noise of known variance, $-2\ell\big(\underset{\sim}{\hat{\theta}}\big)$ equals $\mathrm{RSS}(\underset{\sim}{\hat{\theta}})/\sigma^2$ plus a constant that does not depend on the model. AIC then equals nC_p/σ^2 plus a constant, so the two criteria rank models the same way.

Each criterion assumes a model family whose parameter count measures its flexibility. When the estimate of the noise variance comes from the fitted model itself, it can reach zero, and the criteria then fail. Cross-validation makes no such assumption.

Cross-validating the whole procedure

Exercise 0.1 (Screening predictors before cross-validating)**.** A data set has $n = 50$ observations of an outcome Y and of $p = 1000$ candidate predictors. In fact every variable is independent standard normal noise, so no predictor carries any information about Y.
An analyst

1. picks the 10 predictors most correlated with Y in all 50 observations, then
2. estimates the prediction error of a linear model in those 10 predictors by 5-fold cross-validation.

Answer the following:

1. What is the smallest possible generalization error (Equation 13) of any fitted model for Y here?
2. Simulate the analyst's procedure. What cross-validation estimate does it report?
3. Repeat the simulation, but carry out step 1 inside each fold, using only that fold's training observations. What estimate does this give?

Source

A regression version of the classification example in Hastie et al. [4, Sec. 7.10.2], "The Wrong and Right Way to Do Cross-validation".

Solution

Solution 0.1.

1. Y is independent of every predictor, with mean 0 and variance 1. The best prediction is the constant 0, and its generalization error is $\mathrm{Var}(Y) = 1$. Every other fitted model does worse, so any honest estimate should be near or above 1.

2. and 3. The function below runs one data set through both procedures. The only difference is whether the screening step sees the whole data set (`"outside"`) or only the training folds (`"inside"`).

```r
top_predictors <- function(x, y, n_keep) {
  order(abs(stats::cor(x, y)), decreasing = TRUE)[seq_len(n_keep)]
}

screened_cv <- function(x, y, folds, n_keep, screen) {
  if (screen == "outside") {
    keep <- top_predictors(x, y, n_keep)
  }
  sq_errors <- numeric(length(y))
  for (j in unique(folds)) {
    in_fold <- folds == j
    if (screen == "inside") {
      keep <- top_predictors(x[!in_fold, ], y[!in_fold], n_keep)
    }
    train <- tibble::tibble(
      y = y[!in_fold],
      tibble::as_tibble(x[!in_fold, keep])
    )
    fit <- lm(y ~ ., data = train)
    held_out <- tibble::as_tibble(x[in_fold, keep])
    sq_errors[in_fold] <- (predict(fit, newdata = held_out) - y[in_fold])^2
  }
  mean(sq_errors)
}

set.seed(7102)
n_obs <- 50
n_predictors <- 1000
n_sims <- 50

screening_results <- purrr::map(seq_len(n_sims), \(s) {
  x <- matrix(stats::rnorm(n_obs * n_predictors), n_obs, n_predictors)
  colnames(x) <- paste0("x", seq_len(n_predictors))
  y <- stats::rnorm(n_obs)
  folds <- assign_folds(n_obs, k = 5)
  tibble::tibble(
    sim = s,
    outside = screened_cv(x, y, folds, n_keep = 10, screen = "outside"),
    inside = screened_cv(x, y, folds, n_keep = 10, screen = "inside")
  )
}) |>
  purrr::list_rbind()

screening_summary <-
  screening_results |>
  dplyr::summarise(
    dplyr::across(c(outside, inside), mean)
  )

screening_summary
#> # A tibble: 1 x 2
#>   outside inside
```

Averaged over 50 simulated data sets:

- Screening outside the folds gives a mean CV estimate of 0.54, well below the smallest possible generalization error of 1. The procedure reports that pure noise predicts Y.
- Screening inside the folds gives 1.52, above 1, as an honest estimate for a model built from noise should be.

The 10 predictors chosen from all 50 observations were chosen *because* they happen to correlate with Y in the very observations that later serve as held-out folds. Those observations were not held out from the whole fitting procedure, so the independence that Theorem 0.5 needs fails.

Remark (Cross-validate the whole procedure)

Remark 0.1 (Cross-validate the whole procedure). Every step that uses the outcome is part of fitting, and must be repeated inside each fold using only that fold's training observations [4, Sec. 7.10.2]. Such steps include:

- screening or selecting predictors, as in Exercise 0.1;
- choosing transformations or a polynomial degree;
- choosing a penalty or another tuning parameter.

Hastie et al. [4, Sec. 7.10.2] note that steps using only the predictors, such as dropping predictors with almost no variation, can be done once on all the data before the folds are formed. Repeating such a step inside each fold is never wrong, though, and it scores each held-out fold the way new data would be scored. Examples are estimating the mean and standard deviation used for standardization, and the values used to fill in missing predictor values, when they are computed without the outcome.

Further reading

Sanchez [14] reviews the matrix algebra behind statistical learning, including the covariance and correlation matrices that cross-products produce.

For cross-validation, see James et al. [6, Ch. 5] and Hastie et al. [4, Sec. 7.10]. A PDF of each book can be downloaded at no cost from the authors' webpages: *An Introduction to Statistical Learning* and *The Elements of Statistical Learning*.

References

[1] Wikipedia contributors, "Random seed — Wikipedia, the free encyclopedia." 2026. Available: https://en.wikipedia.org/wiki/Random_seed

[2] Wikipedia contributors, "Hyperparameter (machine learning) — Wikipedia, the free encyclopedia." 2026. Available: https://en.wikipedia.org/wiki/Hyperparameter_%28machine_learning%29

[3] Wikipedia contributors, "Training, validation, and test data sets — Wikipedia, the free encyclopedia." 2026. Available: https://en.wikipedia.org/wiki/Training,_validation,_and_test_data_sets

[4] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: Data mining, inference, and prediction*, 2nd ed. Springer, 2009. doi: 10.1007/978-0-387-84858-7.

[5] Wikipedia contributors, "Generalization error — Wikipedia, the free encyclopedia." 2026. Available: https://en.wikipedia.org/wiki/Generalization_error

[6] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning: With applications in R*, 2nd ed. Springer, 2021. doi: 10.1007/978-1-0716-1418-1.

[7] D. W. Hosmer and S. Lemeshow, *Applied logistic regression*. New York: Wiley, 1989.

[8] W. N. Venables and B. D. Ripley, *Modern applied statistics with S*, 4th ed. New York: Springer, 2002. doi: 10.1007/978-0-387-21706-2.

[9] Wikipedia contributors, "Overfitting — Wikipedia, the free encyclopedia." 2026. Available: https://en.wikipedia.org/wiki/Overfitting

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

[11] Wikipedia contributors, "Bias–variance tradeoff — Wikipedia, the free encyclopedia." 2026. Available: https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff

[12] Wikipedia contributors, "Model selection — Wikipedia, the free encyclopedia." 2026. Available: https://en.wikipedia.org/wiki/Model_selection

[13] Wikipedia contributors, "Cross-validation (statistics) — Wikipedia, the free encyclopedia." 2026. Available: https://en.wikipedia.org/wiki/Cross-validation_%28statistics%29

[14] G. Sanchez, "A matrix algebra companion for statistical learning (matrix4sl)." Online book, work in progress. Licensed CC BY-NC-SA 4.0, 2020. Available: https://www.gastonsanchez.com/matrix4sl/

[15] Wikipedia contributors, "Leverage (statistics) — Wikipedia, the free encyclopedia." 2026. Available: https://en.wikipedia.org/wiki/Leverage_%28statistics%29

[16] C. L. Mallows, "Some comments on C_p," *Technometrics*, vol. 15, no. 4, pp. 661–675, 1973, doi: 10.1080/00401706.1973.10489103.

[17] H. Akaike, "A new look at the statistical model identification," *IEEE Transactions on Automatic Control*, vol. 19, no. 6, pp. 716–723, 1974, doi: 10.1109/TAC.1974.1100705.

[18] G. Schwarz, "Estimating the dimension of a model," *The Annals of Statistics*, vol. 6, no. 2, pp. 461–464, 1978, doi: 10.1214/aos/1176344136.