Generalization and Cross-Validation

Risk, held-out estimates, and cross-validation

Last modified: 2026-11-19

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

Source

Adapted from Brian Hutchinson's Fall 2025 lecture notes, which supplied the crime-and-income example and the generalization argument. Logan Sizemore's Spring 2025 lecture on generalization is the basis of the cross-validation section. The wider reading is Hastie et al. [1], Chapter 2, and James et al. [2], Chapter 2.

1 Generalization

Watch

Hutchinson covers generalization, bias, variance and overfitting, in the video Generalization[a] (31 min) [3].

[a] https://facultyweb.cs.wwu.edu/~hutchib2/video_lectures/data371/#generalization

Definition 1.1 (Objective function (objective))**.** An **objective function**, or **objective** for short [4, Sec. 1.1] and 4.1, is the quantity training optimizes as the model's parameters vary, by minimizing it or by maximizing it. It is a function of the parameters alone, with the data held fixed. For a model h_θ with parameters θ and a training set $\mathcal{D}_{\text{train}}$, the objective that is the empirical risk is

$$\mathcal{J}(\theta) \stackrel{\text{def}}{=} \hat{R}(h_\theta; \mathcal{D}_{\text{train}}), \tag{1}$$

and training minimizes $\mathcal{J}(\theta)$ over θ.

Example 1.1 (The objective as a function of the slope and intercept)**.**
For the crime data, the objective is the empirical risk as a function of the pair (m, b).

```r
# The five neighbourhoods of @tbl-crime-data, read once in @exm-crime-task.
x <- crime$x
y <- crime$y

objective <- function(m, b) mean((m * x + b - y)^2)

fit <- lm(y ~ x)
m_hat <- coef(fit)[["x"]]
b_hat <- coef(fit)[["(Intercept)"]]
print(round(c(objective(0, mean(y)), objective(m_hat, b_hat)), 6))
```

```
[1] 0.023056 0.002721
```

The flat line $(m, b) = (0, \bar{y})$ has objective value 0.023056, and the least-squares pair $(\hat{m}, \hat{b}) = (-1.2877, 0.7052)$ has the smaller value 0.002721. Training searches over (m, b) for the pair with the smallest value.

Remark (The empirical risk is one objective among many)

Remark 1.1 (The empirical risk is one objective among many). So far the only objective in these notes is the empirical risk, which the minimization problem minimizes over the slope and intercept. Later chapters change the objective without changing the loss: whatever is handed to the optimizer is the objective, and the empirical risk is only one thing it can be made from.

Definition 1.2 (Cost function)**.** A **cost function** is an objective function for which smaller values are better.

Example 1.2 (The empirical risk of the crime data as a cost function)**.** For the crime data, the objective of Example 1.1 is the empirical risk under squared-error loss, which is a cost function:

$$\mathcal{J}(m, b) = \frac{1}{n} \sum_{i=1}^{n} (m x_i + b - y_i)^2.$$

A line farther from the data has a larger value.

```r
# Raising the slope of the fitted line by 1 increases the squared errors.
cost_fitted <- objective(m_hat, b_hat)
cost_steeper <- objective(m_hat + 1, b_hat)
costs <- c(cost_fitted, cost_steeper)
round(costs, 6)
```

```
[1] 0.002721 0.119961
```

The pair $(\hat{m}, \hat{b})$ from `lm` has cost 0.002721, and raising the slope[a] by 1 raises the cost to 0.119961. Training moves toward smaller values, so it minimizes this cost. Figure 1 plots both against the data.

```r
crime_points <- tibble::tibble(x = x, y = y)
crime_points |>
  ggplot2::ggplot() +
  ggplot2::aes(x = x, y = y) +
  ggplot2::geom_point() +
  ggplot2::geom_abline(intercept = b_hat, slope = m_hat, colour = "#0072B2") +
  ggplot2::geom_abline(
    intercept = b_hat,
    slope = m_hat + 1,
    colour = "#D55E00"
  ) +
  ggplot2::expand_limits(y = 0.7) +
  ggplot2::labs(x = "Per-capita income", y = "Crime rate")
```

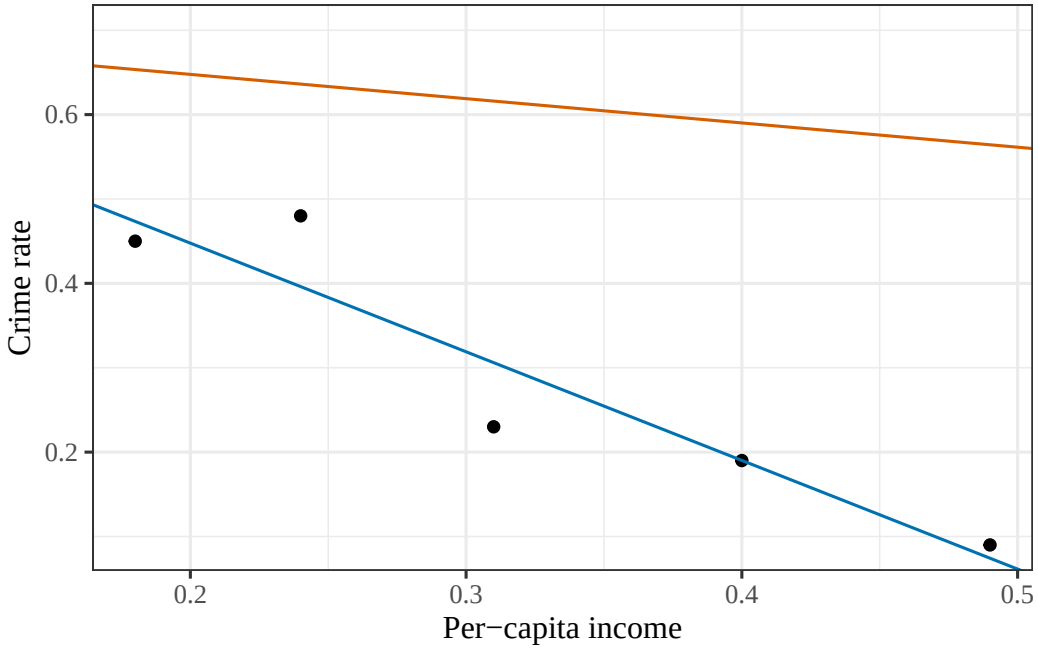


Figure 1: The fitted line (blue) and the line with its slope raised by 1 (orange), over the crime data.

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

Remark (How the loss, the empirical risk, the objective function and the cost function relate)

Remark 1.2 (How the loss, the empirical risk, the objective function and the cost function relate). Four related quantities answer different questions.

- The loss function assesses one prediction.
- The empirical risk is the loss averaged over the n examples in the training set, so it assesses a model on the whole training set.
- The objective function is whatever training optimizes as the parameters vary. So far it is the empirical risk. For a penalized fit such as the one in the ridge regression section, it combines the empirical risk, or a multiple of it, with a penalty.
- A cost function is an objective function for which smaller values are better, so training minimizes it. The log-likelihood[a] is an objective function that training maximizes, so minus the log-likelihood is a cost function. Other texts also use "cost function" for the loss function or for the empirical risk, but here it always refers to a minimized objective function.

Under squared-error loss, the empirical risk is the sample mean squared error, and the residual sum of squares[b] is n times that empirical risk. The two differ only by the factor $n \geq 1$, so they have the same minimizers[c].

[a] https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik
[b] https://morrison-lab.github.io/sds/correlation-regression.html#def-rss
[c] https://morrison-lab.github.io/mds/algebra-basics.html#def-global-minimizer

Exercise 1.1 (Estimate what cannot be computed)**.** The risk cannot be evaluated, and the empirical risk can. The solution to "Which model was actually better?" rules out reporting the second in place of the first.
Describe a procedure that uses only the available data and returns an honest estimate[a] of the risk, and say what the procedure costs.

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

Solution

Solution 1.1. Set some data aside. Split $\mathcal{D}_{\text{train}}$ before fitting anything; fit on one part; score on the other. The scored part played no role in choosing the model's parameters, so the sample mean[a] of its losses is an *unbiased*[b] estimate[c] of the risk: over repeated draws of the held-out points, it equals the risk on average, neither systematically above nor below.
The cost is the data itself. Points moved into the held-out set are points the model never learns from, so the model is worse than it needed to be, and the estimate depends on however few points were set aside and is noisy in proportion.
The usual arrangement is three sets rather than two:

1. A training set, which the fit uses.
2. A validation set (Definition 1.3; also called a development or dev set), scored repeatedly while choosing between models, tuning their settings, and deciding when to stop.
3. A test set (Definition 1.5), scored exactly once, at the end, on the model finally chosen. It stays unused until then.

The third set exists because of what the second one does. A validation set consulted a hundred times is a set a hundred decisions have been fitted to, and its score stops being honest about a model those decisions selected. Section 1.1 covers what to do when there is not enough data for all three.

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

Example 1.3 (Splitting the Auto data)**.**
Split the cars of the Auto data [2] at random[a], 60% for training, 20% for validation and the rest for testing.

```
# Draw the three index sets from one shuffled list of cars, so that they cannot overlap.
import numpy as np
import pandas as pd

auto_split = pd.read_csv("data/auto.csv").dropna(subset=["horsepower"])
n_cars = len(auto_split)
shuffled = np.random.default_rng(571).permutation(n_cars)
n_train = int(0.6 * n_cars)
n_val = int(0.2 * n_cars)
idx_train = shuffled[:n_train]
idx_val = shuffled[n_train:n_train + n_val]
idx_test = shuffled[n_train + n_val:]
print(n_train, n_val, len(idx_test))
```

```
235 78 79
```

Of the 392 cars with a recorded horsepower, 235 form the training set, which is all the fitting procedure sees, and the other 157 are held out of the fit.

[a] https://morrison-lab.github.io/pds/probability-basics.html#def-random-choice

Definition 1.3 (Validation set)**.** A **validation set** is a subset of the data set[a], the pairs indexed by a set I_{val} that shares no index with I_{train}, and that we use for model selection[b] among candidate models fitted to the training set [5]:

$$\mathcal{D}_{\text{val}} \overset{\text{def}}{=} \{(\underset{\sim}{x}_i, y_i) : i \in I_{\text{val}}\} \subseteq \mathcal{D}, \quad I_{\text{val}} \cap I_{\text{train}} = \emptyset. \tag{2}$$

The model with the smallest empirical risk on the validation set is the one we keep.

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

Definition 1.4 (Validation error)**.** The **validation error** of a model h is its empirical risk on the validation set, $\hat{R}(h; \mathcal{D}_{\text{val}})$.
See validation error[a] in the statistics notes.

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

Example 1.4 (Choosing a polynomial degree)**.** Fit polynomials of degree 1 to 9 to the 235 training cars of Example 1.3, predicting mpg from horsepower, and score each on the 78 validation cars.

```
# Scale horsepower to 0 to 1 so that high degrees are numerically stable.
hp = auto_split["horsepower"].to_numpy(dtype=float)
hp = (hp - hp.min()) / (hp.max() - hp.min())
mpg = auto_split["mpg"].to_numpy(dtype=float)

def mse(coef, idx):
    return np.mean((np.polyval(coef, hp[idx]) - mpg[idx]) ** 2)

degrees = range(1, 10)
fits = {d: np.polyfit(hp[idx_train], mpg[idx_train], d) for d in degrees}
val_mse = {d: mse(fits[d], idx_val) for d in degrees}
train_mse = {d: mse(fits[d], idx_train) for d in degrees}
best_degree = min(val_mse, key=val_mse.get)
train_mse_9 = train_mse[9]
val_mse_9 = val_mse[9]
val_mse_best = val_mse[best_degree]

def mse_ci(coef, idx):
    # Normal-approximation 95% interval for the mean of the squared errors.
    sq = (np.polyval(coef, hp[idx]) - mpg[idx]) ** 2
    se = sq.std(ddof=1) / np.sqrt(len(sq))
    return sq.mean() - 1.96 * se, sq.mean() + 1.96 * se

val_ci_best = mse_ci(fits[best_degree], idx_val)
val_ci_9 = mse_ci(fits[9], idx_val)
print(best_degree, round(val_mse[best_degree], 3), round(train_mse_9, 3), round(val_mse_9, 3))
```

```
2 18.006 16.78 19.559
```

```
print(np.round(val_ci_best, 3), np.round(val_ci_9, 3))
```

```
[12.012 23.999] [11.9   27.218]
```
The degree with the smallest validation error, 2, is the one we keep. The training cars could not make this choice, because the training error keeps falling as the degree grows: the degree-9 fit has training error 16.78 and validation error 19.56. The chosen degree's validation error, 18.01, has a 95% confidence interval of 12.01 to 24.00, and the degree-9 fit's has 11.90 to 27.22. Each interval comes from the 78 validation cars alone, and the chosen degree's is optimistic, because the same cars picked the degree.

Definition 1.5 (Test set)**.** A **test set** is a subset of the data set[a], the pairs indexed by a set I_{test} that shares no index with I_{train} or I_{val}, and that we use only once, after the model is chosen, to estimate[b] the risk of the chosen model h by its empirical risk on the test set, $\hat{R}(h; \mathcal{D}_{\text{test}})$:

$$\mathcal{D}_{\text{test}} \stackrel{\text{def}}{=} \{(\underset{\sim}{x}_i, y_i) : i \in I_{\text{test}}\} \subseteq \mathcal{D}, \quad I_{\text{test}} \cap (I_{\text{train}} \cup I_{\text{val}}) = \emptyset. \tag{3}$$

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

Definition 1.6 (Test error)**.** The **test error** of a model h is its empirical risk on the test set, $\hat{R}(h; \mathcal{D}_{\text{test}})$.
See test error[a] in the statistics notes.

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

Example 1.5 (Reporting the chosen degree's error)**.** Having chosen degree 2 on the validation cars of Example 1.4, score that one model on the 79 test cars.

```
# Refit the chosen degree on the training cars of @exm-training-set (scaled horsepower hp, mpg),
# so this chunk does not lean on the fits table of the validation example.
chosen_coef = np.polyfit(hp[idx_train], mpg[idx_train], best_degree)
# A normal-approximation 95% interval: the mean of the squared errors
# plus or minus 1.96 standard errors.
test_sq = (np.polyval(chosen_coef, hp[idx_test]) - mpg[idx_test]) ** 2
test_mse = np.mean(test_sq)
test_se = test_sq.std(ddof=1) / np.sqrt(len(test_sq))
test_lo, test_hi = test_mse - 1.96 * test_se, test_mse + 1.96 * test_se
print(round(test_mse, 3), round(test_lo, 3), round(test_hi, 3))
```

```
24.232 14.183 34.281
```
The estimate of the chosen model's risk is 24.23, with a 95% confidence interval of 14.18 to 34.28. The test cars played no part in the fit or the choice, so the choice did not tune this estimate to them. If it disappoints and we go back to choose again, the test cars have become a validation set.

Definition 1.7 (Generalization gap)**.** The **generalization gap** of a model h on a training set $\mathcal{D}_{\text{train}}$ is its risk minus its training error [6, Sec. 5.2]:

$$R(h) - \hat{R}(h; \mathcal{D}_{\text{train}}). \tag{4}$$

Example 1.6 (Estimating the chosen degree's gap)**.**
The risk in Equation 4 cannot be computed, so estimate[a] it by the test error of Example 1.5, and subtract the training error of the same degree-2 fit.

```
# train_mse holds each degree's training error from @exm-validation-set;
# test_mse is the chosen degree's test error from @exm-test-set.
train_mse_best = train_mse[best_degree]
gap_estimate = test_mse - train_mse_best
print(round(train_mse_best, 3), round(test_mse, 3), round(gap_estimate, 3))
```

```
17.693 24.232 6.539
```

The training error is 17.69 and the test error is 24.23, so the estimated[b] gap is 6.54. The estimate is noisy: the 95% interval for the test error in Example 1.5 runs from 14.18 to 34.28, and it contains the training error, so these cars alone do not show that the gap is positive.

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

Remark (The three sets together)

Remark 1.3 (The three sets together). The three index sets I_{train}, I_{val} and I_{test} are disjoint and together cover $1, \dots, n$, so the training, validation and test sets are disjoint subsets that together make up the data set $\mathcal{D}$, which is why Example 1.3 draws them from one shuffled list. The sets are not interchangeable: the first is for fitting, the second for choosing among fits, and the third for reporting. Data used for one job gives an optimistic answer to the next, as Figure 2 shows. Any pairs kept out of the fit are called *held-out*, so validation and test sets are both held-out sets.

The statistics notes define the same three sets formally: the training set[a], the validation set[b], and the test set[c]. They use a different symbol for the training set and a plain observation index, but the same three-way split.

[a] https://morrison-lab.github.io/sds/model-validation.html#def-training-set
[b] https://morrison-lab.github.io/sds/model-validation.html#def-validation-set
[c] https://morrison-lab.github.io/sds/model-validation.html#def-test-set

Remark (What the validation and test errors estimate)

Remark 1.4 (What the validation and test errors estimate). Like the training error, the validation error and the test error are empirical risks, and they differ only in the set that scores the model. The model was not fitted to the validation or test set, so each estimates[a] the risk $R(h)$ without the optimism of the training error. We choose the model with the smallest validation error, so the validation error of the chosen model is still optimistic. We compute the test error once, after the choice, so it is not.

Some textbooks use "test error" for the expected error of the fitted model on a new observation, which is the risk of the fitted model, not a number computed from a test set [1, Sec. 7.2]. Here the test error is the number computed from the test set, and the risk is what it estimates.

These ideas hold for any loss, not only squared error. Each error is the average loss on the set that scores the model: the training error[b], the validation error[c] and the test error[d]. The expected loss of the fitted model on a new observation is its generalization error[e], the risk of the fitted model. Averaging the generalization error over training sets gives the expected generalization error[f].

[a] https://morrison-lab.github.io/sds/estimation.html#def-estimate
[b] https://morrison-lab.github.io/sds/model-validation.html#def-training-mse
[c] https://morrison-lab.github.io/sds/model-validation.html#def-validation-error
[d] https://morrison-lab.github.io/sds/model-validation.html#def-test-error
[e] https://morrison-lab.github.io/sds/model-validation.html#def-generalization-error
[f] https://morrison-lab.github.io/sds/model-validation.html#def-expected-generalization-error

```
# The constants of the static figure, so its prose and plot agree.
tv_n <- 10       # training cars
tv_degree <- 3   # degree of the polynomial drawn on the left
tv_degrees <- 0:9
```

```r
library(ggplot2)
# The Auto cars with a horsepower, as in the interactive figure:
# the polynomials are fitted with horsepower rescaled to [0, 1] and mpg standardized,
# and the plots convert back, so every axis and number is in horsepower and mpg.
auto <- read.csv("data/auto.csv")
auto <- auto[!is.na(auto$horsepower), ]
hp_min <- min(auto$horsepower)
hp_range <- diff(range(auto$horsepower))
mpg_mean <- mean(auto$mpg)
mpg_sd <- sd(auto$mpg)
cars <- data.frame(
  x = (auto$horsepower - hp_min) / hp_range,
  y = (auto$mpg - mpg_mean) / mpg_sd
)
# Back to the original units, for plotting.
to_hp <- function(x) x * hp_range + hp_min
to_mpg <- function(y) y * mpg_sd + mpg_mean
set.seed(2100)
train_rows <- sample(nrow(cars), tv_n)
train <- cars[train_rows, ]
held <- cars[-train_rows, ]
# Least-squares polynomial of the given degree; degree 0 is the constant fit,
# which poly() does not accept. poly() stops when the degree is not below the number
# of distinct x values, so cars that share a horsepower lower the degree actually fitted.
fit_poly <- function(data, degree) {
  degree <- min(degree, length(unique(data$x)) - 1)
  if (degree == 0) lm(y ~ 1, data = data) else lm(y ~ poly(x, degree), data = data)
}
# Squared errors of a fit on a set of cars. The 95% confidence interval for the
# held-out RMSE is the mean squared error plus or minus 1.96 standard errors
# (central limit theorem), with its endpoints square-rooted.
# The RMSEs are multiplied by the sd of mpg, which puts them in mpg.
sq_errors <- function(fit, data) (predict(fit, data) - data$y)^2
scores <- do.call(rbind, lapply(tv_degrees, function(d) {
  fit <- fit_poly(train, d)
  e <- sq_errors(fit, held)
  half <- 1.96 * sd(e) / sqrt(length(e))
  data.frame(
    degree = d,
    set = c("training points", "held-out points"),
    # Clamped to the bottom of the log axis, so an RMSE of exactly 0 is drawn on the edge.
    rmse = pmax(mpg_sd * c(sqrt(mean(sq_errors(fit, train))), sqrt(mean(e))), 1e-1),
    lower = c(NA, mpg_sd * sqrt(max(mean(e) - half, 0))),
    upper = c(NA, mpg_sd * sqrt(mean(e) + half))
  )
}))
# Left panel: the cars and the fitted curve.
grid_x <- data.frame(x = seq(0, 1, length.out = 200))
grid_x$y <- predict(fit_poly(train, tv_degree), grid_x)
p_fit <- ggplot() +
  geom_point(data = held, aes(to_hp(x), to_mpg(y)), colour = "grey75", size = 0.8) +
  geom_line(data = grid_x, aes(to_hp(x), to_mpg(y)), colour = "#1f77b4", linewidth = 1) +
  geom_point(data = train, aes(to_hp(x), to_mpg(y)), size = 1.8) +
  coord_cartesian(ylim = to_mpg(c(-2, 3.5))) +
  labs(x = "horsepower", y = "mpg")
# Right panel: training and held-out RMSE against degree, on a log scale.
```

```r
p_err <- ggplot(scores, aes(degree, rmse, colour = set)) +
  geom_vline(xintercept = tv_degree, colour = "grey55") +
  geom_linerange(
    data = scores[scores$set == "held-out points", ],
    aes(ymin = pmax(lower, 1e-1), ymax = upper)
  ) +
  geom_line(linewidth = 0.8) +
  geom_point() +
  scale_x_continuous(breaks = tv_degrees) +
  scale_y_log10(
    limits = c(1e-1, 1e2), oob = scales::squish,
    breaks = 10^(-1:2), labels = c("0.1", "1", "10", "100")
  ) +
  scale_colour_manual(
    values = c("training points" = "#1f77b4", "held-out points" = "#ff7f0e")
  ) +
  labs(x = "degree", y = "RMSE (mpg, log scale)", colour = NULL) +
  theme(legend.position = "bottom")
```

```r
# Side by side, with base grid, since no layout package is in the render workflows.
grid::grid.newpage()
grid::pushViewport(grid::viewport(layout = grid::grid.layout(1, 2)))
print(p_fit, vp = grid::viewport(layout.pos.row = 1, layout.pos.col = 1))
print(p_err, vp = grid::viewport(layout.pos.row = 1, layout.pos.col = 2))
```

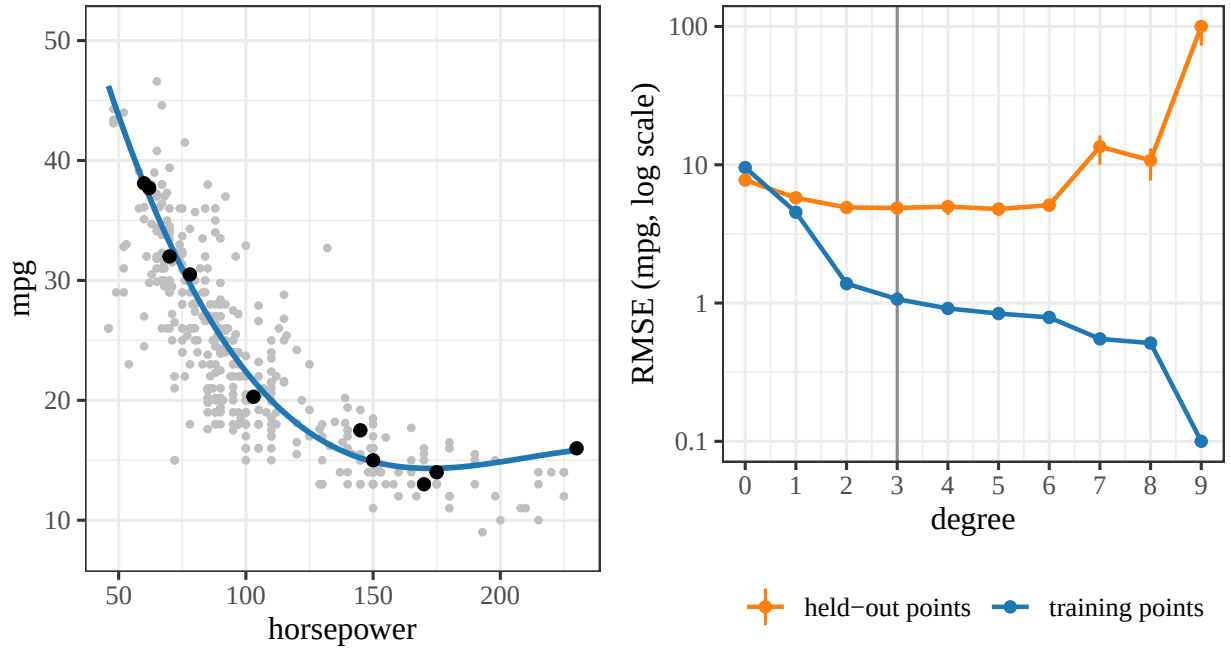


Figure 2: Static version of the interactive plot on the course website: polynomials fitted to a random sample of 10 of the cars in the Auto data [2], mpg against horsepower, scored on the cars in the sample (training) and on the rest (held out), with 95% confidence intervals on the held-out sRMSE. Left, the 10 training cars (black), the held-out cars (gray) and the degree-3 fit (blue). Right, the sRMSE at each degree from 0 to 9, with the degree-3 fit marked by a vertical line. The axes are in horsepower and miles per gallon, and the sRMSE is in miles per gallon.

Remark (Reading the figure on training and held-out error)

Remark 1.5 (Reading the figure on training and held-out error). The polynomials are fitted with horsepower scaled to run from 0 to 1 and mpg standardized, but every axis and number here is in the original units, horsepower and miles per gallon. The sRMSE is the square root of the sample mean squared error, so it is in the units of the response, here miles per gallon (why we report sRMSE). The vertical bars on the held-out curve are 95% confidence intervals for the held-out sRMSE: the mean of the squared errors plus or minus 1.96 standard errors, from the central limit theorem, with the endpoints square-rooted. It is rough when a high-degree fit makes a few squared errors enormous, and it describes the risk of this fitted polynomial, not of the fitting method. The training error has no interval: the fit was chosen to make those errors small, so they are not independent draws and their mean is biased low as an estimate of the risk, and an interval around it would be centered in the wrong place.

Exercise 1.2 (How far can a held-out score be trusted?)**.** A spam filter h was trained on one set of emails and then scored on a fresh held-out set $\mathcal{V}$ of n emails, drawn independently from the same distribution as the emails it will face. Its held-out score $\hat{R}(h;\mathcal{V})$ is the fraction of those emails it labels wrong, and its risk $R(h)$ is its error rate on the population.
Find an upper bound on the chance that the held-out score misses the risk by more than some $\varepsilon > 0$, using only the mean and variance of the held-out score.

Solution

Solution 1.2. Write Z_i for the loss on the ith held-out email: 1 if h labels it wrong, and 0 if right. The filter was fixed before $\mathcal{V}$ was drawn, so the Z_i are independent, and each is 1 with probability $R(h)$. The held-out score is their average, $\hat{R}(h;\mathcal{V}) = \frac{1}{n}\sum_{i=1}^{n} Z_i$, so its mean is $R(h)$.
Its variance is the variance of one Z_i divided by n. A 0/1 variable that is 1 with probability R has variance $R(1-R)$, and that is never more than $\frac{1}{4}$:

$$R(1-R) = \frac{1}{4} - \left(R - \frac{1}{2}\right)^2 \le \frac{1}{4}.$$

So $\mathrm{Var}\big(\hat{R}(h;\mathcal{V})\big) \le \frac{1}{4n}$, whatever the filter's error rate is. Chebyshev's inequality bounds the chance of a miss by the variance divided by the square of the miss:

$$P\Big(\big|\hat{R}(h;\mathcal{V}) - R(h)\big| > \varepsilon\Big) \le \frac{\mathrm{Var}\big(\hat{R}(h;\mathcal{V})\big)}{\varepsilon^2} \le \frac{1}{4n\varepsilon^2}.$$

Exercise 1.3 (How far can a held-out score be trusted? (part 2))**.** Use the setup of Exercise 1.2. How does your bound change as n grows?

Solution

Solution 1.3. The bound shrinks in proportion to $1/n$: ten times as many held-out emails make a miss of a given size at most a tenth as likely.

```
n_example <- 1000
eps_example <- 0.05
cheb_example <- 1 / (4 * n_example * eps_example^2)
```

For example, with $n = 1000$ and $\varepsilon = 0.05$, the chance of a miss is at most $1/(4n\varepsilon^2) = 0.1$.
Chebyshev uses only the variance[a], and the 0/1 range entered only through the cap of $\frac{1}{4}$ on it. Using the range itself, that each Z_i lies between 0 and 1, gives a much sharper bound, one that shrinks exponentially in n rather than in proportion to $1/n$.

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

Theorem 1.1 (Hoeffding's inequality for a held-out set)**.** *Score a classifier*[a] *h with the misclassification loss. Let $\mathcal{D}_{val}$ be a held-out set of n observations*[b] *drawn independently from the same distribution $P(x, y)$, and let $\hat{R}(h; \mathcal{D}_{val})$ be the fraction of them h gets wrong, as in the definition of the empirical risk. If h is fixed before $\mathcal{D}_{val}$ is drawn, then for every $\varepsilon > 0$ [7],*

$$P\Big(\Big|\hat{R}(h; \mathcal{D}_{val}) - R(h)\Big| > \varepsilon\Big) \leq 2e^{-2\varepsilon^2 n}. \tag{5}$$

[a] https://morrison-lab.github.io/sds/classification.html#def-classification-rule
[b] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-observation

Proof

Proof. The misclassification loss of h on the ith held-out point, $L\big(y_{(i)}, h\big(x_{(i)}\big)\big)$, is 1 when h gets the point wrong and 0 otherwise. Because h is fixed before $\mathcal{D}_{\text{val}}$ is drawn, each loss is the same function of one of n independent draws from $P(x, y)$. So the n losses are IID[a] random variables in $[0, 1]$. Their common population mean[b] is $R(h)$ by the definition of the risk, and their sample mean[c] is $\hat{R}(h; \mathcal{D}_{\text{val}})$ by the definition of the empirical risk. Hoeffding's inequality for an average of IID variables in $[0, 1]$[d] bounds the chance that the average misses the mean by at least ε by $2e^{-2\varepsilon^2 n}$. The event "misses by more than ε" is part of that event, so its chance is at most as large. □

[a] https://morrison-lab.github.io/pds/independence.html#def-iid
[b] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[c] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean
[d] https://morrison-lab.github.io/pds/limit-theorems.html#cor-hoeffding-two-sided

Remark (Reading Hoeffding's inequality)

Remark 1.6 (Reading Hoeffding's inequality). The misclassification loss is 1 when $h(x) \neq y$ and 0 otherwise, so $R(h)$, the risk, is the error rate of the classifier[a] on the population[b]. The probability in Equation 5 is over the draw of $\mathcal{V}$; $R(h)$ is unknown but not random.

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

Remark (Source: Hoeffding's inequality for a held-out set)

Remark 1.7 (Source: Hoeffding's inequality for a held-out set). Abu-Mostafa et al. [8, Sec. 1.3.2] states the inequality for a label that is a fixed function of x, and in section 1.4.2 extends its analysis to labels with noise, as here.

Exercise 1.4 (How many held-out emails are enough?)**.** A spam filter was trained on one set of emails, and you will estimate its error rate on a fresh held-out set. Say the estimate *misses* when it differs from the filter's true error rate by more than 0.05.
How many held-out emails make the chance of a miss at most 0.05?

Solution

Solution 1.4. In Exercise 1.4, the filter was fixed before the held-out set was drawn, so Equation 5 applies. Write ε for the allowed miss and δ for the allowed chance. The chance of a miss is at most δ whenever the bound is:

$$
\begin{aligned}
2\mathrm{e}^{-2\varepsilon^2 n} &\le \delta \\
\mathrm{e}^{-2\varepsilon^2 n} &\le \frac{\delta}{2} \\
-2\varepsilon^2 n &\le \ln\frac{\delta}{2} \\
2\varepsilon^2 n &\ge -\ln\frac{\delta}{2} = \ln\frac{2}{\delta} \\
n &\ge \frac{\ln(2/\delta)}{2\varepsilon^2}.
\end{aligned}
$$

The fourth line multiplies both sides by -1, which flips the inequality, and the fifth divides by $2\varepsilon^2$.

```
eps <- 0.05
delta <- 0.05
n_one <- ceiling(log(2 / delta) / (2 * eps^2))
```

With $\varepsilon = 0.05$ and $\delta = 0.05$, that is $n \ge 738$ emails. The bound is loose, so fewer emails may well do, but 738 are enough whatever the classifier[a]'s error rate is.

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

Exercise 1.5 (How many held-out emails are enough? (part 2))**.** Use the setup of Exercise 1.4. Suppose instead that 100 different filters, all trained before the held-out set was drawn, are scored on the same held-out set, and you keep the one with the lowest score. How many held-out emails make the chance that the kept filter's estimate[a] misses at most 0.05?

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

Solution

Solution 1.5. In Exercise 1.5, the kept filter was chosen *by* its held-out score, so it was not fixed before the held-out set was drawn, and Equation 5 does not apply to it directly.

```
n_models <- 100
```

It does apply to each of the $M = 100$ filters $h_1, \dots, h_M$, since each was trained before the held-out set was drawn. The kept filter's estimate[a] can miss only if at least one filter's estimate does, and the chance that at least one of several events happens is at most the sum of their chances. So

$$
\begin{aligned}
P(\text{the kept filter misses}) &\le P(\text{some } h_m \text{ misses}) \\
&\le \sum_{m=1}^{M} P\big(\big|\hat{R}(h_m; \mathcal{V}) - R(h_m)\big| > \varepsilon\big) \\
&\le \sum_{m=1}^{M} 2\mathrm{e}^{-2\varepsilon^2 n} \\
&= 2M\mathrm{e}^{-2\varepsilon^2 n},
\end{aligned}
$$

where the third line applies Equation 5 to each term.

```
n_many <- ceiling(log(2 * n_models / delta) / (2 * eps^2))
ratio_many <- round(n_many / n_one, 1)
```

Repeating the steps of Solution 1.4 with $2M$ in place of 2 gives $n \ge \ln(2M/\delta)/(2\varepsilon^2)$, which is 1659 emails, about 2.2 times as many as for one filter.

```
n_half <- ceiling(log(2 / delta) / (2 * (eps / 2)^2))
```

Both answers are sufficient sizes, not required ones, since they come from an upper bound. Two things still follow for Solution 1.1's three sets. The sufficient n is proportional to $1/\varepsilon^2$, so halving the allowed miss quadruples it: with $\varepsilon = 0.025$, Solution 1.4 needs 2952 emails rather than 738. And a

validation set on which a fixed list of M models is compared needs the size from Solution 1.5 for that M, $\ln(2M/\delta)/(2\varepsilon^2)$, while a test set scored once on one model needs only the size from Solution 1.4. The union bound covers only a finite list of models; tuning a setting that can take infinitely many values needs the further analysis that Abu-Mostafa et al. [8] defers to its chapter 2.

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

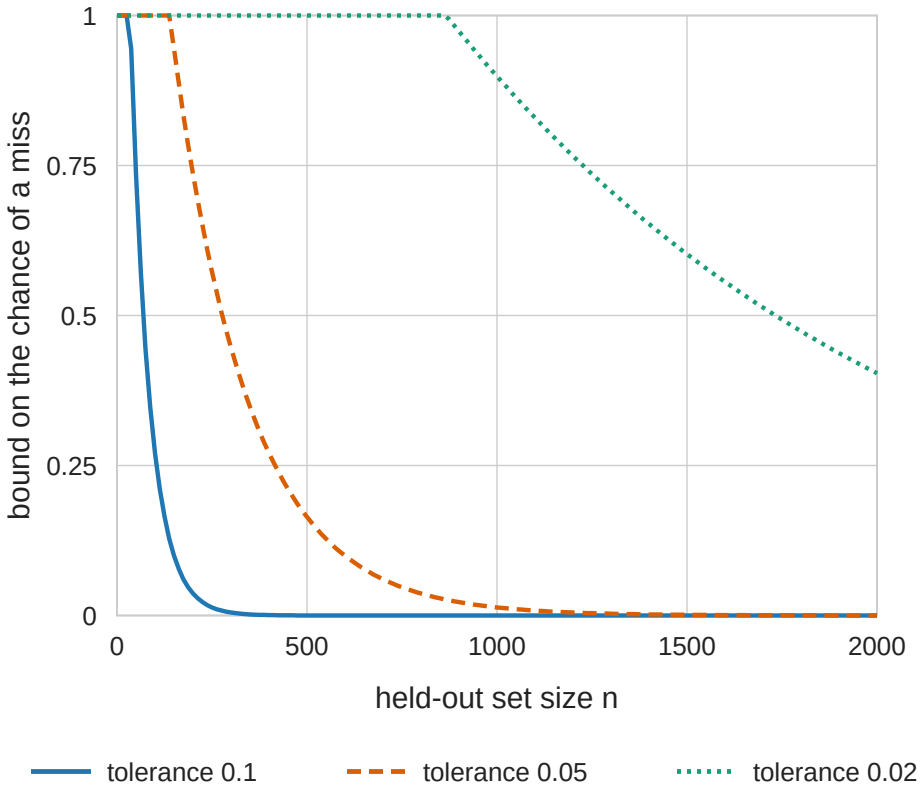


Figure 3: The bound of Equation 5 on the chance that a held-out estimate misses by more than the tolerance, as the held-out set grows, for three tolerances. The bound is capped at 1, since a probability cannot exceed it. A smaller tolerance needs a much larger held-out set before the bound says anything.

1.1 Cross-validation

Exercise 1.6 (Use every point twice)**.** The three sets of Solution 1.1 are practical only when we have a lot of data. Setting aside a tenth of the data for validation and a tenth for testing leaves the model fitting on the remaining eighty percent, and on a small dataset that costs twice: the model is worse for having seen less, and the held-out estimate is noisy because it is based on so few points.
Design a procedure in which every point is scored by a model that was not fitted to it, and every point also helps fit a model.

Solution

Solution 1.6. Divide the n points at random into k groups of roughly equal size. For each group in turn, fit the model on the other $k-1$ groups and score it on the group left out. Average the k scores. This is k-fold cross-validation (Definition 1.9), and the groups are its folds.

Exercise 1.7 (Use every point twice (part 2))**.** Use the setup of Exercise 1.6.
In your procedure, how many times is each point scored?

Solution

Solution 1.7. Each point belongs to exactly one group, so it is scored exactly once, by the one model that did not see it.

Exercise 1.8 (Use every point twice (part 3))**.** Use the setup of Exercise 1.6.
In your procedure, how many times is each point trained on?

Solution

Solution 1.8. Each point is trained on by the other $k-1$ models. The estimate[a] therefore rests on all n points rather than on a held-out fraction of them.

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

Exercise 1.9 (Use every point twice (part 4))**.** Use the setup of Exercise 1.6.
What does the procedure cost?

Solution

Solution 1.9. The cost is computation: the model is fitted k times instead of once. Five and ten are the usual values of k. So use a train-validation-test split when there is plenty of data, and this procedure when there is not.

Definition 1.8 (Folds)**.** Given a data set of n pairs, k **folds** $I_1, \ldots, I_k$ are a random partition of the indices $\{1, \ldots, n\}$ into k index sets of roughly equal size.
See folds[a] in the statistics notes.

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

Definition 1.9 (k-fold cross-validation)**.** Given a data set of n pairs and a learning algorithm, **k-fold cross-validation** splits the pairs into k folds $I_1, \ldots, I_k$ [9]. For each fold I_j, the algorithm is fit to the pairs outside that fold, which gives the model $\hat{y}_{-j}$.

Definition 1.10 (k-fold cross-validation estimate)**.** The **k-fold cross-validation estimate** of the risk scores each pair by the model $\hat{y}_{-j}$ that did not see it (the model fit outside fold I_j in k-fold cross-validation), and averages those scores over all n pairs:

$$\hat{R}_{k\text{-fold}} \stackrel{\text{def}}{=} \frac{1}{n} \sum_{j=1}^{k} \sum_{i \in I_j} L\Big(y_{(i)}, \hat{y}_{-j}\big(x_{(i)}\big)\Big). \tag{6}$$

Definition 1.11 (Leave-one-out cross-validation)**. Leave-one-out cross-validation** is k-fold cross-validation with $k = n$ [9]: each pair is a fold of its own, and the model $\hat{y}_{-i}$ is fitted to the other $n-1$ pairs. Its estimate[a] of the risk is

$$\hat{R}_{\text{LOO}} \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^{n} L\Big(y_{(i)}, \hat{y}_{-i}\big(x_{(i)}\big)\Big). \tag{7}$$

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

Example 1.7 (Leave-one-out on the five neighborhoods)**.** With $n = 5$ neighborhoods, k-fold cross-validation with $k = n$ puts each neighborhood in a fold of its own. This is leave-one-out cross-validation (Definition 1.11): five fits, each on four neighborhoods and scored on the fifth. Fit the least-squares line to each training set of four, and score it by its squared error on the neighborhood left out:

1.1.1 Python

```
import numpy as np

# The five neighbourhoods of the crime table, read once in the first example.
x_loo = crime["x"].to_numpy()
y_loo = crime["y"].to_numpy()

loo_py = []
for i_loo in range(5):
    # Train on every neighbourhood except i_loo, so the fit never sees the one it is scored on.
    keep_loo = np.arange(5) != i_loo
    m_loo, b_loo = np.polyfit(x_loo[keep_loo], y_loo[keep_loo], 1)
    # Score the line on the held-out neighbourhood alone.
    loo_py.append((m_loo * x_loo[i_loo] + b_loo - y_loo[i_loo]) ** 2)

loo_py = np.array(loo_py)
# A 95% interval for the mean of the five errors; 2.776 is the t quantile for 4 degrees of freedom,
# which replaces 1.96 because five errors are too few for the normal approximation.
loo_se_py = loo_py.std(ddof=1) / np.sqrt(5)
print(loo_py.round(4), loo_py.mean().round(4), (2.776 * loo_se_py).round(4))
```

```
[0.002  0.0026 0.0091 0.     0.015 ] 0.0057 0.0077
```

1.1.2 R

```
# The five neighbourhoods of the crime table, read once in the first example.
crime_loo <- crime
x_loo <- crime$x
y_loo <- crime$y

loo_r <- sapply(1:5, function(i) {
  # Train on every neighbourhood except i, so the fit never sees the one it is scored on.
  fit <- lm(y ~ x, data = crime_loo[-i, ])
  # Score the line on the held-out neighbourhood alone.
  (predict(fit, crime_loo[i, ]) - crime_loo$y[i])^2
})

# The same interval, with the t quantile for 4 degrees of freedom.
loo_se_r <- sd(loo_r) / sqrt(5)
loo_half_r <- qt(0.975, df = 4) * loo_se_r

print(round(c(loo_r, mean = mean(loo_r), half_width = loo_half_r), 4))
```

```
         1          2          3          4          5       mean half_width
    0.0020     0.0026     0.0091     0.0000     0.0150     0.0057     0.0077
```

Figure 4 shows the five splits. Each panel lists the neighborhoods in a small table, with the one held out shaded, and plots the same neighborhoods, with that one in a different color and shape. The line

is fitted to the four blue circles alone.

```
import matplotlib.pyplot as plt

# Okabe-Ito colours, which stay distinct under the common colour-vision deficiencies;
# the circle against the diamond keeps the two groups apart in greyscale too.
train_col, held_col = "#0072B2", "#D55E00"
grid_loo = np.linspace(0.1, 0.55, 50)

fig, axes = plt.subplots(2, 5, figsize=(10, 4.2), layout="constrained",
                         height_ratios=[1.1, 1.4])
for i_loo in range(5):
    keep_loo = np.arange(5) != i_loo
    m_loo, b_loo = np.polyfit(x_loo[keep_loo], y_loo[keep_loo], 1)

    # Top row: the data as a table, with the held-out row shaded and labelled.
    top = axes[0, i_loo]
    top.axis("off");
    top.set_xlim(0, 1);
    top.set_ylim(0, 6.4);
    top.set_title(f"Split {i_loo + 1}", fontsize=10);
    top.text(0.05, 5.7, "x", fontsize=8, fontweight="bold");
    top.text(0.40, 5.7, "y", fontsize=8, fontweight="bold");
    for j in range(5):
        row_y = 4.7 - j
        if j == i_loo:
            top.fill_between([0, 1], row_y - 0.35, row_y + 0.85, color=held_col, alpha=0.25);
        top.text(0.05, row_y, f"{x_loo[j]:.2f}", fontsize=8);
        top.text(0.40, row_y, f"{y_loo[j]:.2f}", fontsize=8);
        top.text(0.72, row_y, "held out" if j == i_loo else "train", fontsize=7,
                 fontweight="bold" if j == i_loo else "normal");

    # Bottom row: the same neighbourhoods as a scatterplot, with the fitted line.
    bot = axes[1, i_loo]
    bot.plot(grid_loo, m_loo * grid_loo + b_loo, color="grey", linewidth=1.5);
    bot.scatter(x_loo[keep_loo], y_loo[keep_loo], color=train_col, marker="o", s=30,
                label="train");
    bot.scatter(x_loo[~keep_loo], y_loo[~keep_loo], color=held_col, marker="D", s=50,
                label="held out");
    bot.set_xlim(0.1, 0.55);
    bot.set_ylim(0, 0.6);
    bot.set_xlabel("per-capita income $x$", fontsize=8);
    if i_loo == 0:
        bot.set_ylabel("crime rate $y$", fontsize=8);
    bot.text(0.12, 0.03, f"error {loo_py[i_loo]:.3f}", fontsize=8);
    bot.tick_params(labelsize=7);
# One key for all five panels, in a row along the bottom of the figure.
fig.legend(*axes[1, 0].get_legend_handles_labels(), fontsize=8,
           loc="outside lower center", ncol=2);
plt.show()
```

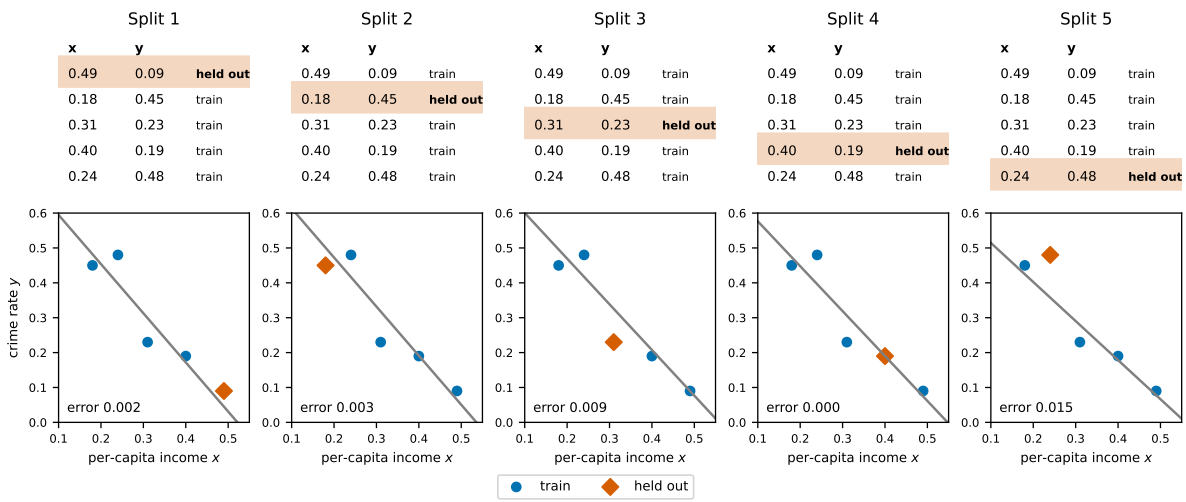


Figure 4: The five leave-one-out splits of the crime data. In each panel the table lists the five neighbourhoods and shades the one held out; the scatterplot draws the held-out neighbourhood as an orange diamond, the four training neighbourhoods as blue circles, and the line fitted to the four, with the squared error on the held-out one.

The five squared errors are 0.002, 0.003, 0.009, 0.000, 0.015, and their mean, the leave-one-out estimate of the risk, is 0.006, with a 95% confidence interval of −0.002 to 0.013. The interval is wide because five errors say little about their mean, and it is not a valid interval: the five errors share training points, so they are not independent, and squared errors are skewed, so a t interval fits them poorly. It reaches below zero, which a mean of squared errors cannot, so treat it as a rough guide to the uncertainty. Every neighborhood is scored once, by a line that never saw it, and every neighborhood helps fit four of the five lines. The largest error, 0.015, is in split 5, and the next largest, 0.009, is in split 3. Those two neighborhoods have incomes 0.24 and 0.31, on either side of the steep drop in crime rate that a straight line fitted without one of them smooths over.

Watch

Hutchinson covers train, dev and test sets and cross-validation, Generalization, part 2[a] (11 min) [3]. The chapter Model Selection puts its cross-validation videos beside each method.

[a] https://facultyweb.cs.wwu.edu/~hutchib2/video_lectures/data371/#generalization_pt2

Watch

Starmer explains cross-validation[a] [10].

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

Read

GeeksforGeeks explains cross-validation[a] [11].

[a] https://www.geeksforgeeks.org/machine-learning/cross-validation-machine-learning/

```
# The constants of this figure live here, so the widget, the static plot and the prose agree.
cv_n <- 40          # points drawn, and so held-out squared errors
cv_k <- 5           # folds of the static plot
cv_degree <- 3      # degree of the static plot
cv_fresh_n <- 2000  # fresh points the static plot scores
```

```r
cv_sd <- 0.1        # standard deviation of the noise in the static plot
```

```r
# The simulated world of the bias-variance figure: this true mean function plus
# Gaussian noise with standard deviation 0.1, on inputs in [0.05, 0.95].
truth <- function(x) 0.55 - 0.9 * x + 0.6 * sin(6 * x)
draw <- function(n) {
  x <- runif(n, 0.05, 0.95)
  data.frame(x = x, y = truth(x) + rnorm(n, sd = cv_sd))
}
set.seed(3100)
train <- draw(cv_n)
fresh <- draw(cv_fresh_n)
# Deal the points into k folds at random: shuffle, then fold = position mod k.
k <- cv_k
fold <- sample(rep_len(seq_len(k), nrow(train)))
# Least-squares polynomial of the given degree; degree 0 is the constant fit,
# which poly() does not accept.
fit_poly <- function(data, degree) {
  if (degree == 0) lm(y ~ 1, data = data) else lm(y ~ poly(x, degree), data = data)
}
# Held-out root mean squared error of a polynomial fit of the given degree on each fold,
# which is fit on the other k - 1 folds.
fold_rmse <- function(degree) {
  vapply(seq_len(k), function(f) {
    fit <- fit_poly(train[fold != f, ], degree)
    sqrt(mean((predict(fit, train[fold == f, ]) - train$y[fold == f])^2))
  }, numeric(1))
}
# Error of the fit to all the points on the fresh points.
fresh_rmse <- function(degree) {
  fit <- fit_poly(train, degree)
  sqrt(mean((predict(fit, fresh) - fresh$y)^2))
}
# The squared error of every point under the fit that left it out, so each point is scored once.
oof_sq <- function(degree) {
  sq <- numeric(nrow(train))
  for (f in seq_len(k)) {
    fit <- fit_poly(train[fold != f, ], degree)
    sq[fold == f] <- (predict(fit, train[fold == f, ]) - train$y[fold == f])^2
  }
  sq
}
# An approximate 95% confidence interval for the k-fold estimate: the percentile bootstrap
# interval for the mean of the held-out squared errors, with its endpoints square-rooted.
# It runs too narrow, because the errors are not independent (Bates, Hastie and Tibshirani 2024).
cv_ci <- function(degree) {
  sq <- oof_sq(degree)
  roots <- replicate(1000, sqrt(mean(sample(sq, replace = TRUE))))
  unname(quantile(roots, c(0.025, 0.975)))
}
degree <- cv_degree
scores <- fold_rmse(degree)
# The k-fold estimate is the root of the average of the fold MSEs.
kfold <- function(s) sqrt(mean(s^2))
degrees <- 0:8
cv_curve <- vapply(degrees, function(d) kfold(fold_rmse(d)), numeric(1))
```

```
cv_bands <- vapply(degrees, cv_ci, numeric(2))
fresh_curve <- vapply(degrees, fresh_rmse, numeric(1))
cv_now <- cv_ci(degree)
library(ggplot2)
orange <- "#ff7f0e"
est <- sprintf("%d-fold estimate", k)
left <- "Held-out fold"
right <- "Polynomial degree"
panels <- c(left, right)
panel_of <- function(p) factor(p, levels = panels)
# Left panel: each fold's RMSE as a dot, with the k-fold estimate
# and the fresh-point error as lines.
fold_dots <- data.frame(panel = panel_of(left), x = seq_len(k), y = scores)
levels_hline <- data.frame(
  panel = panel_of(left), series = c(est, "fresh points"),
  y = c(kfold(scores), fresh_rmse(degree))
)
# The approximate 95% confidence interval of the k-fold estimate, a pale band.
left_band <- data.frame(
  panel = panel_of(left), xmin = 0.5, xmax = k + 0.5, ymin = cv_now[1], ymax = cv_now[2]
)
# Right panel: the same two quantities at every degree,
# with the same band around the estimate.
curves <- data.frame(
  panel = panel_of(right), x = rep(degrees, 2),
  y = c(cv_curve, fresh_curve),
  series = rep(c(est, "fresh points"), each = length(degrees))
)
right_band <- data.frame(
  panel = panel_of(right), x = degrees, ymin = cv_bands[1, ], ymax = cv_bands[2, ]
)
chosen <- data.frame(panel = panel_of(right), x = degree)
series_colors <- setNames(c(orange, "black"), c(est, "fresh points"))
series_lines <- setNames(c("solid", "dashed"), c(est, "fresh points"))
```

```
ggplot() +
  geom_rect(
    data = left_band, aes(xmin = xmin, xmax = xmax, ymin = ymin, ymax = ymax),
    fill = orange, alpha = 0.2
  ) +
  geom_ribbon(
    data = right_band, aes(x = x, ymin = ymin, ymax = ymax),
    fill = orange, alpha = 0.2
  ) +
  geom_vline(data = chosen, aes(xintercept = x), color = "grey55") +
  geom_hline(
    data = levels_hline, aes(yintercept = y, color = series, linetype = series),
    linewidth = 0.8
  ) +
  geom_line(data = curves, aes(x = x, y = y, color = series), linewidth = 0.8) +
  geom_point(data = curves, aes(x = x, y = y, color = series)) +
  geom_point(data = fold_dots, aes(x = x, y = y), color = orange, size = 2.5) +
  facet_wrap(~panel, scales = "free_x", strip.position = "bottom") +
  scale_x_continuous(breaks = function(lim) seq(ceiling(lim[1]), floor(lim[2]))) +
  scale_y_log10() +
```

```
  scale_color_manual(values = series_colors) +
  scale_linetype_manual(values = series_lines) +
  labs(x = NULL, y = "RMSE (log scale)", color = NULL, linetype = NULL) +
  theme(strip.placement = "outside", legend.position = "bottom")
```

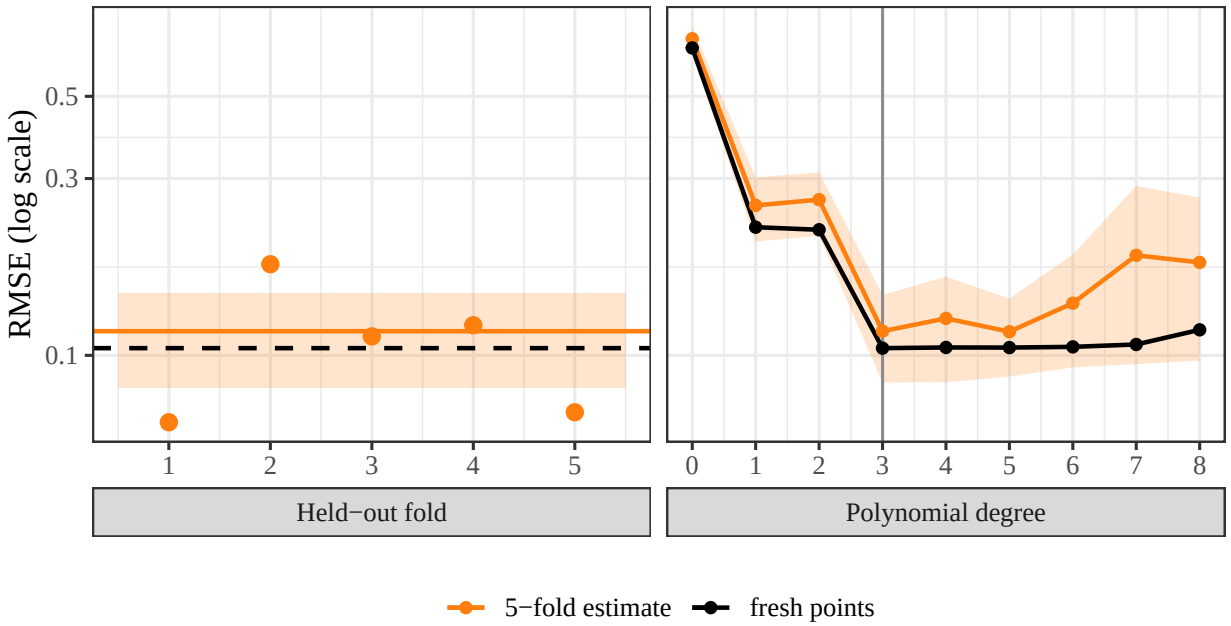


Figure 5: Static version of the interactive plot on the course website: 5-fold cross-validation of a degree-3 polynomial fit to 40 points from a simulated world, a smooth curve plus Gaussian noise with standard deviation 0.1. Left, the sRMSE on each held-out fold (dots), their root-mean-square, the 5-fold estimate (solid line) with its approximate 95% confidence interval (pale band), and the error on 2,000 fresh points (dashed line). Right, the same two quantities for degrees 0 to 8. The band is a bootstrap interval for the mean of the 40 held-out squared errors, square-rooted; it runs too narrow because the errors are not independent.

> **Remark** (Cross-validation replaces the validation set)
>
> *Remark* 1.8 (Cross-validation replaces the validation set). Used to choose between models, cross-validation takes the place of the **validation** set, the one used again and again, and not of the final test set, which stays unused. Whichever scheme picked the model, the data used to pick it cannot also be the data used to report how good it is. If cross-validation picks the polynomial degree in Figure 5, for example, the cross-validation error of the winning degree is optimistic, because that degree won partly by fitting this sample's noise well; an honest estimate needs points that took no part in the choice.

Reading

- Starmer [12] covers the fundamental concepts of machine learning and cross-validation with pictures and little algebra.
- Stanford University [13], Lectures 3–6, are handwritten notes from a Stanford course on probability[1] and inference[2]. They cover concentration inequalities (Hoeffding's inequality in Lecture 3), empirical risk minimization and overfitting (Lecture 5), and uniform convergence with generalization bounds (Lecture 6), for readers who want the mathematics behind Hoeffding's inequality in more detail. The textbook

[1] https://morrison-lab.github.io/pds/probability-basics.html#def-probability
[2] https://morrison-lab.github.io/sds/inference.html#def-inference

these notes recommend for this material is Shalev-Shwartz and Ben-David [14], Sections 2.1–2.3 and 4.1–4.2, which is free online.

> **Watch**
>
> Andrew Ng's short lecture for DeepLearning.AI explains how to split data into a training set, a validation set and a test set [15]. The lecture calls the validation set the "dev set".

References

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

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

[3] B. Hutchinson, "DATA 471/571 (machine learning) and CSCI 481/581 (deep learning) video lectures." Western Washington University. Accessed: Sep. 28, 2026. [Online]. Available: https://facultyweb.cs.wwu.edu/~hutchib2/video_lectures/data371/

[4] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge University Press, 2004. doi: 10.1017/CBO9780511804441[3].

[5] 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

[6] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT Press, 2016. Available: https://www.deeplearningbook.org/

[7] Wikipedia contributors, "Hoeffding's inequality — Wikipedia, the free encyclopedia." 2026. Available: https://en.wikipedia.org/wiki/Hoeffding%27s_inequality

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

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

[10] J. Starmer, "Machine learning fundamentals: Cross validation." StatQuest, video on YouTube, 2018. Available: https://www.youtube.com/watch?v=fSytzGwwBVw

[11] GeeksforGeeks, "Cross validation in machine learning." GeeksforGeeks, tutorial article, 2026. Available: https://www.geeksforgeeks.org/machine-learning/cross-validation-machine-learning/

[12] J. Starmer, *The StatQuest illustrated guide to machine learning*. StatQuest Publications, 2022. Available: https://statquest.org/statquest-store/

[13] Stanford University, "EE 278, probability and statistical inference: Fall 2024 lecture notes." Handwritten lecture notes, Lectures 1–17, 2024. Available: http://web.stanford.edu/class/ee278/lectures.html

[14] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 2014. doi: 10.1017/CBO9781107298019[4].

[15] A. Ng, "Train/dev/test sets." DeepLearning.AI, Deep Learning Specialization, Course 2; video on YouTube, 2017. Available: https://www.youtube.com/watch?v=1waHlpKiNyY

[3] https://doi.org/10.1017/CBO9780511804441
[4] https://doi.org/10.1017/CBO9781107298019