Model selection and validation 1: Cross-validation

Ryan Tibshirani
Data Mining: 36-462/36-662

March 26 2013

Optional reading: ISL 2.2, 5.1, ESL 7.4, 7.10

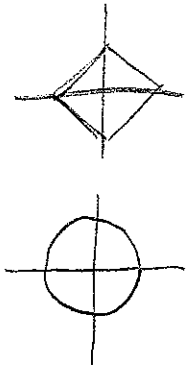
Reminder: modern regression techniques

Over the last two lectures we've investigated modern regression techniques. We saw that linear regression has generally low bias (zero bias, when the true model is linear) but high variance, leading to poor predictions. These modern methods introduce some bias but significantly reduce the variance, leading to better predictive accuracy

Given a response $y \in \mathbb{R}^n$ and predictors $X \in \mathbb{R}^{n \times p}$, we can think of these modern methods as constrained least squares:

$$\hat{\beta} = \underset{\beta \in \mathbb{R}^p}{\operatorname{argmin}} \ \|y - X\beta\|_2^2 \quad \text{subject to } \|\beta\|_q \leq t,$$

- $q = 1$ gives the lasso (and $\|\beta\|_1 = \sum_{j=1}^p |\beta_j|$)
- $q = 2$ gives ridge regression (and $\|\beta\|_2 = \sqrt{\sum_{j=1}^p \beta_j^2}$)



Regularization at large

Most other modern methods for regression can be expressed as

$$\hat{\beta} = \underset{\beta \in \mathbb{R}^p}{\operatorname{argmin}} \ \|y - X\beta\|_2^2 \quad \text{subject to } R(\beta) \leq t,$$

or equivalently,

$$\hat{\beta} = \underset{\beta \in \mathbb{R}^p}{\operatorname{argmin}} \ \|y - X\beta\|_2^2 + \lambda \cdot R(\beta).$$

The term R is called a penalty or regularizer, and modifying the regression problem in this way is called applying regularization

- Regularization can be applied beyond regression: e.g., it can be applied to classification, clustering, principal component analysis
- Regularization goes beyond sparsity: e.g., design R to induce smoothness or structure, instead of pure sparsity

Example: smoothing splines

Smoothning splines use a form of regularization:

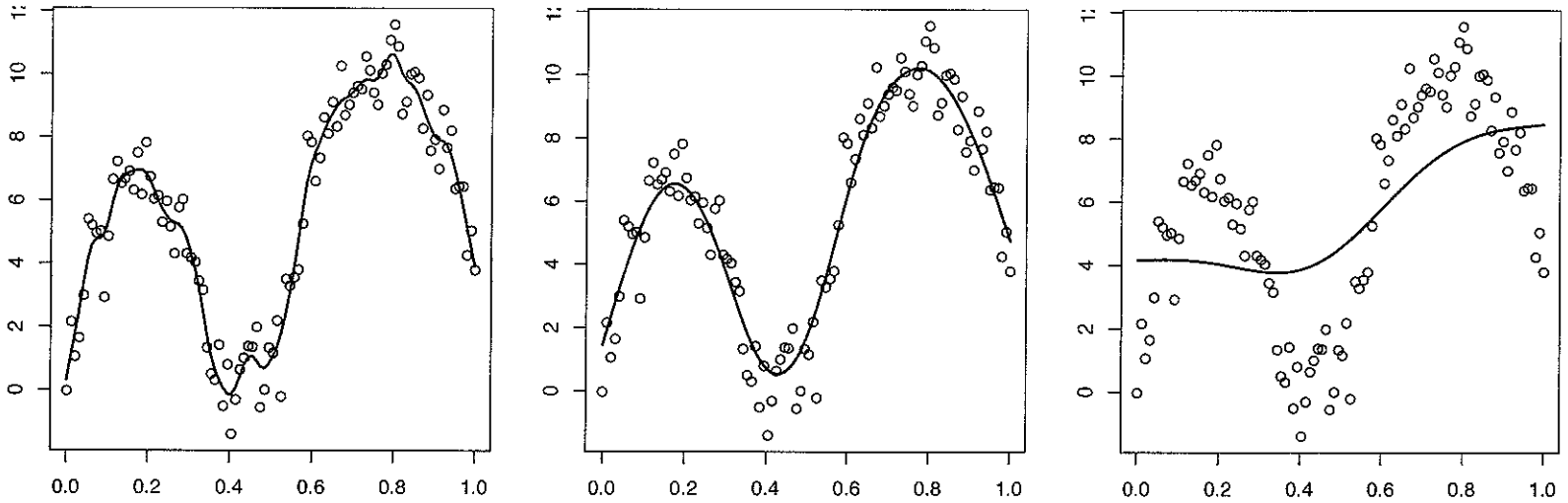
Regularization

$$\hat{f} = \underset{f}{\operatorname{argmin}} \sum_{i=1}^{n} \left(y_i - f(x_i)\right)^2 + \lambda \cdot \underbrace{\int \left(f''(x)\right)^2 dx}_{R(f)}$$

Loss

Penalty

Example with $n = 100$ points:



λ too small λ just right λ too big

Choosing a value of the tuning parameter

Each regularization method has an associated tuning parameter: e.g., this was λ in the smoothing spline problem, and λ for lasso and ridge regression in the penalized forms (or t in the constrained forms)

The tuning parameter controls the amount of regularization, so choosing a good value of the tuning parameter is crucial. Because each tuning parameter value corresponds to a fitted model, we also refer to this task as model selection

What we might consider a good choice of tuning parameter, however, depends on whether our goal is prediction accuracy or recovering the right model for interpretation purposes. We'll cover choosing the tuning parameter for the purposes of prediction; choosing the tuning parameter for the latter purpose is a harder problem

Prediction error and test error

Our usual setup: we observe

$$y_i = f(x_i) + \epsilon_i, \quad i = 1, \ldots n$$

where $x_i \in \mathbb{R}^p$ are fixed (nonrandom) predictor measurements, f is the true function we are trying to predict (think $f(x_i) = x_i^T \beta^*$ for a linear model), and ϵ_i are random errors

We call (x_i, y_i), $i = 1, \ldots n$ the training data. Given an estimator $\hat{f}$ built on the training data, consider the average prediction error over all inputs

$$\mathrm{PE}(\hat{f}) = \mathrm{E}\left[\frac{1}{n}\sum_{i=1}^{n}\left(y_i' - \hat{f}(x_i)\right)^2\right]$$

where $y_i' = f(x_i) + \epsilon_i'$, $i = 1, \ldots n$ are another set of observations, independent of $y_1, \ldots y_n$

Suppose that $\hat{f} = \hat{f}_\theta$ depends on a tuning parameter θ, and we want to choose θ to minimize $\mathrm{PE}(\hat{f}_\theta)$

If we actually had training data $y_1, \ldots y_n$ and test data $y'_1, \ldots y'_n$ (meaning that we don't use this to build $\hat{f}_\theta$) in practice, then we could simply use

$$\mathrm{TestErr}(\hat{f}_\theta) = \frac{1}{n} \sum_{i=1}^{n} \left(y'_i - \hat{f}_\theta(x_i)\right)^2$$

called the test error, as an estimate for $\mathrm{PE}(\hat{f}_\theta)$. The larger that n is, the better this estimate

We usually don't have test data. So what to do instead?

What's wrong with training error?

It may seem like

$$\mathrm{TestErr}(\hat{f}_\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i' - \hat{f}_\theta(x_i)\right)^2 \quad \text{and}$$

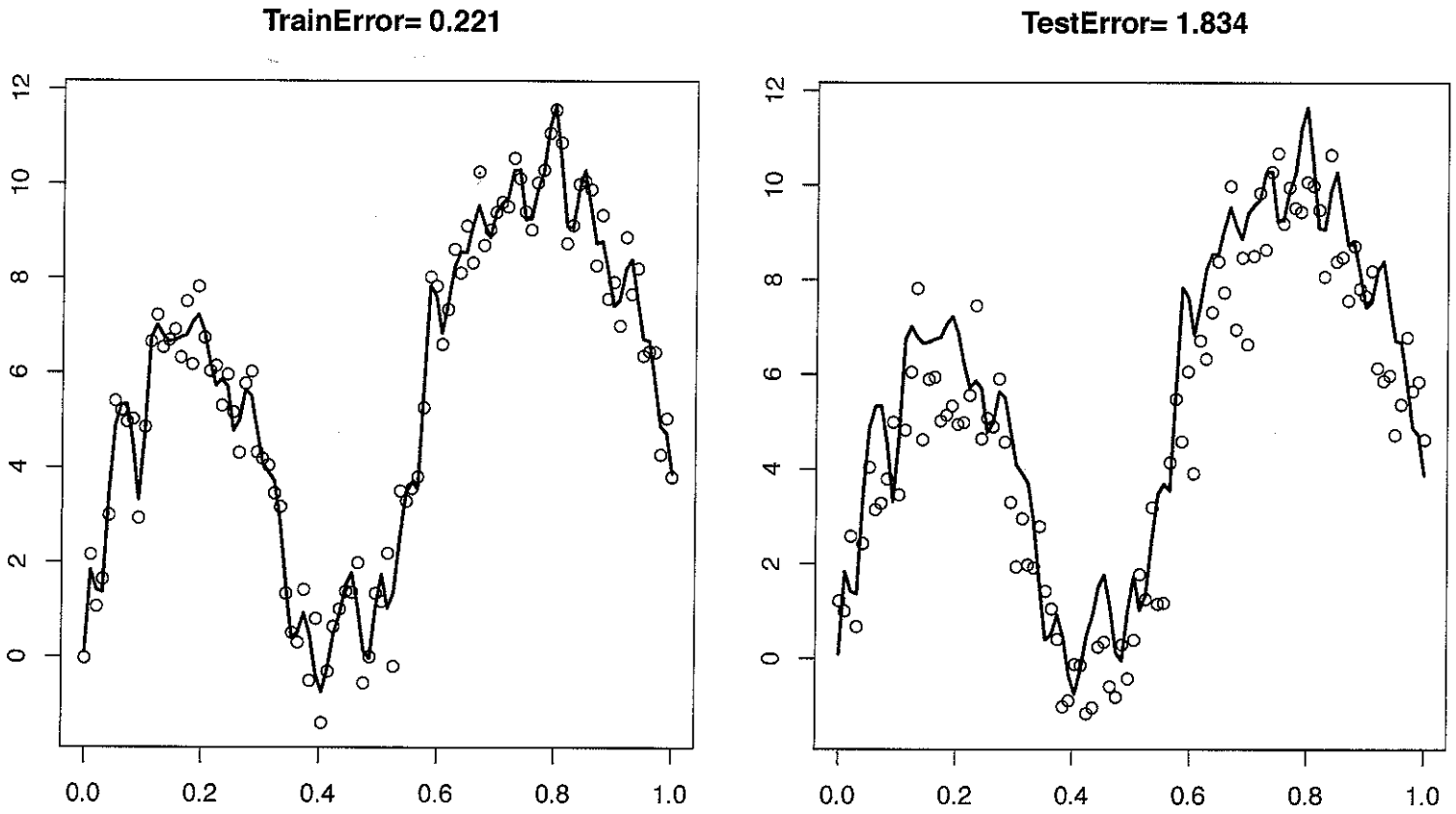
$$\mathrm{TrainErr}(\hat{f}_\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{f}_\theta(x_i)\right)^2$$

shouldn't be too different—after all, y_i and y_i' are independent copies of each other. The second quantity is called the training error: this is the error of $\hat{f}$ as measured by the data we used to build it

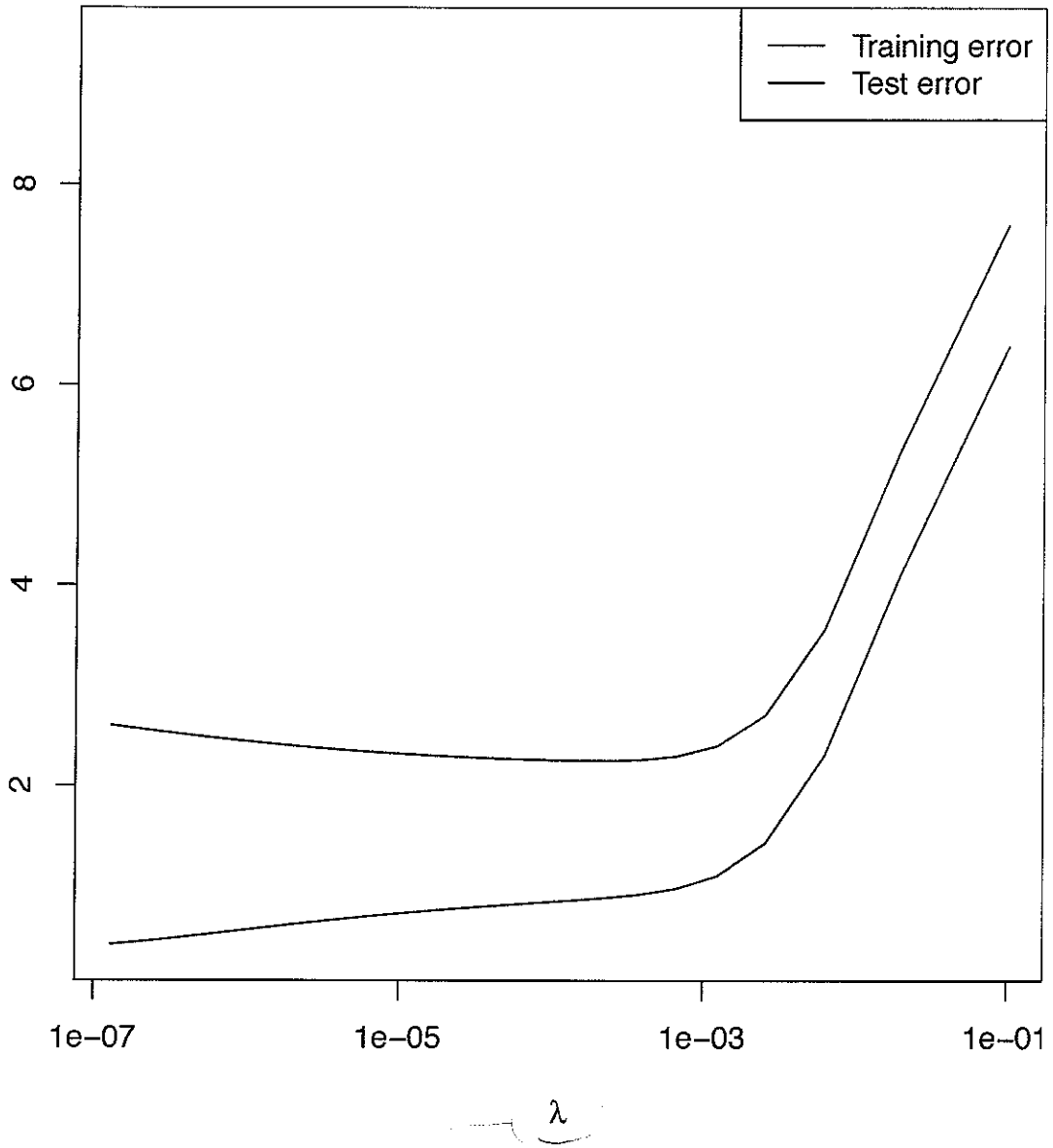
But actually, the training and test error curves are fundamentally different. Why?

Example: smoothing splines (continued)

Back to our smoothing spline example: for a small value of the tuning parameter λ, the training and test errors are very different



Training and test error curves, averaged over 100 simulations:

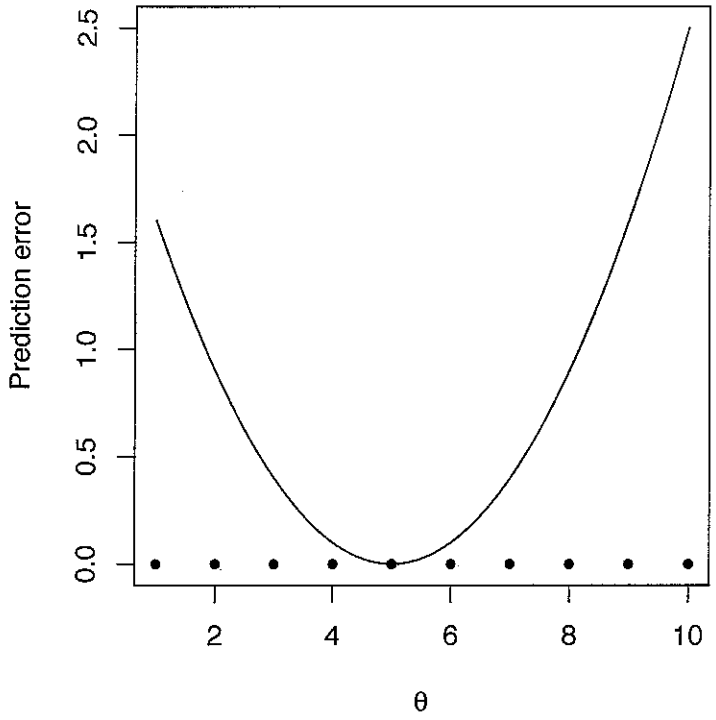


Cross-validation

Cross-validation is a simple, intuitive way to estimate prediction error

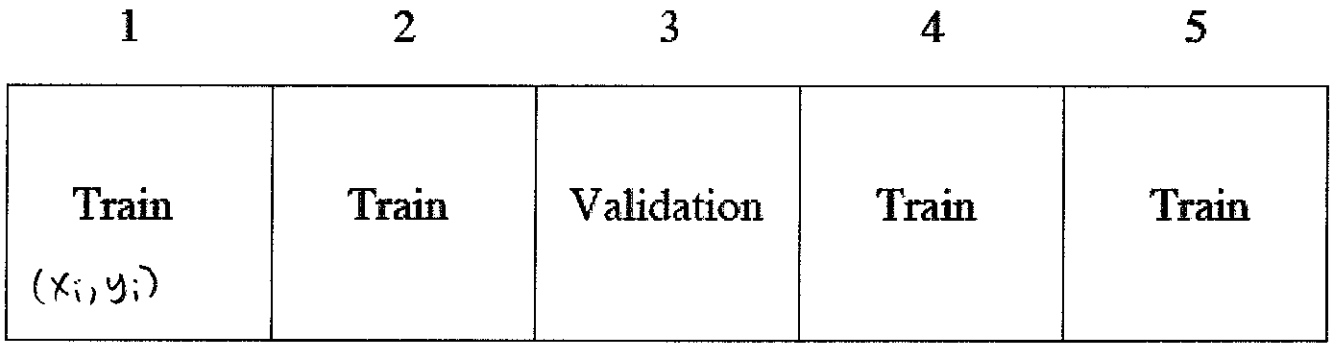
Given training data (x_i, y_i), $i = 1, \ldots n$ and an estimator $\hat{f}_\theta$, depending on a tuning parameter θ

Even if θ is a continuous parameter, it's usually not practically feasible to consider all possible values of θ, so we discretize the range and consider choosing θ over some discrete set $\{\theta_1, \ldots \theta_m\}$



For a number K, we split the training pairs into K parts or "folds" (commonly $K = 5$ or $K = 10$)

10-fold CV



K-fold cross validation considers training on all but the kth part, and then validating on the kth part, iterating over $k = 1, \ldots K$

(When $K = n$, we call this leave-one-out cross-validation, because we leave out one data point at a time)

K-fold cross validation procedure:

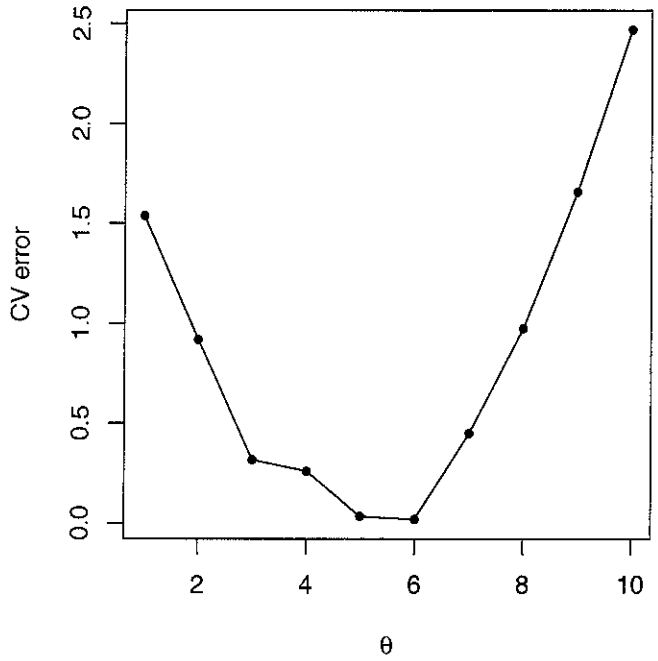
- Divide the set $\{1, \ldots n\}$ into K subsets (i.e., folds) of roughly equal size, $F_1, \ldots F_K$ randomly.
- For $k = 1, \ldots K$:
  - Consider training on (x_i, y_i), $i \notin F_k$, and validating on (x_i, y_i), $i \in F_k$
  - For each value of the tuning parameter $\theta \in \{\theta_1, \ldots \theta_m\}$, compute the estimate $\hat{f}_\theta^{-k}$ on the training set, and record the total error on the validation set:

$$e_k(\theta) = \sum_{i \in F_k} \left(y_i - \hat{f}_\theta^{-k}(x_i)\right)^2$$

- For each tuning parameter value θ, compute the average error over all folds,

$$\mathrm{CV}(\theta) = \frac{1}{n} \sum_{k=1}^{K} e_k(\theta) = \frac{1}{n} \sum_{k=1}^{K} \sum_{i \in F_k} \left(y_i - \hat{f}_\theta^{-k}(x_i)\right)^2$$

Having done this, we get a cross-validation error curve $\mathrm{CV}(\theta)$ (this curve is a function of θ), e.g.,



and we choose the value of tuning parameter that minimizes this curve,

$$\hat{\theta} = \underset{\theta \in \{\theta_1, \ldots \theta_m\}}{\operatorname{argmin}} \ \mathrm{CV}(\theta)$$

set.seed

Example: choosing λ for the lasso

Recall our running example from last time: $n = 50$, $p = 30$, and the true model is linear with 10 nonzero coefficients. We consider

$$\hat{\beta}^{\text{lasso}} = \underset{\beta \in \mathbb{R}^p}{\operatorname{argmin}} \|y - X\beta\|_2^2 + \lambda \|\beta\|_1$$

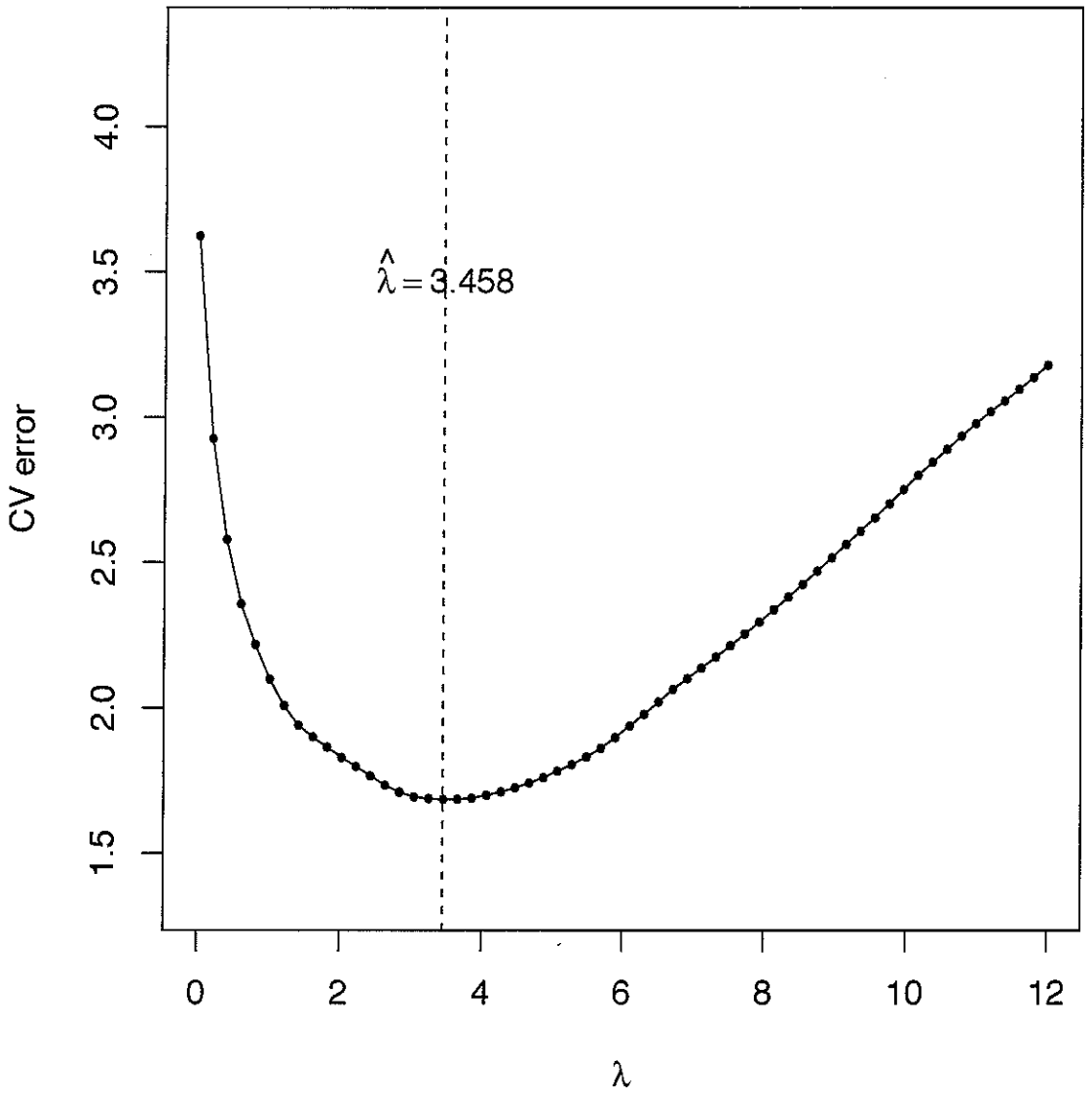
and use 5-fold cross-validation to choose λ. Sample code for making the folds:

```
K = 5
i.mix = sample(1:n)
folds = vector(mode="list",length=K)
# divide i.mix into 5 chunks of size n/K=10, and
# store each chunk in an element of the folds list
> folds
[[1]]
 [1] 30 29  4  2 50 42 27 25 23 41

[[2]]
 [1] 21 44 45  9 10 26 16  6 24 48
...
```

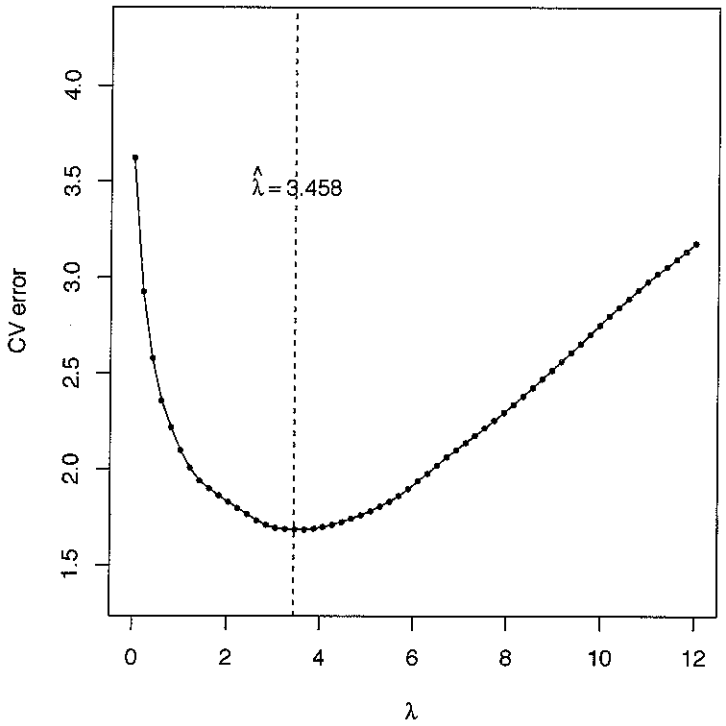
set.seed(0)

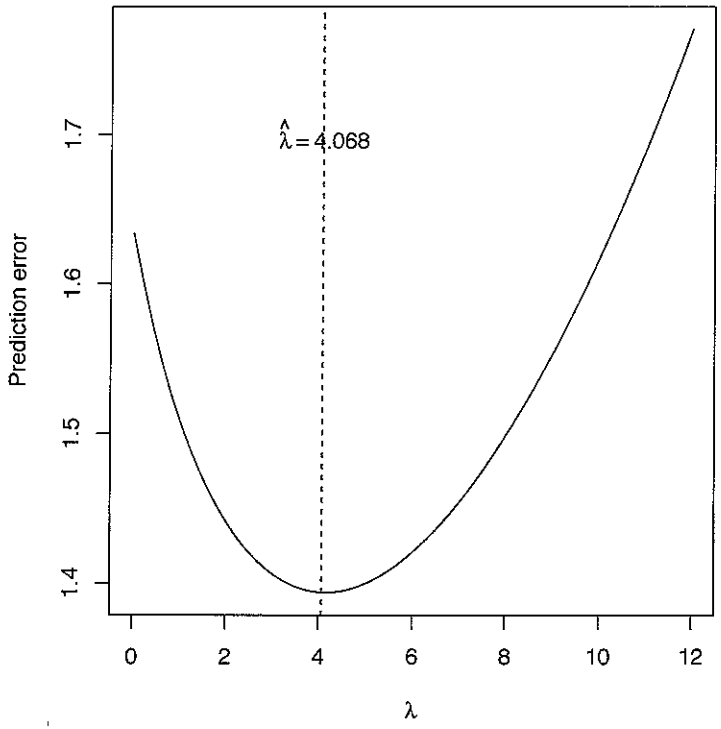
We consider the values: `lam = seq(0,12,length=60)`. The resulting cross-validation error curve:



$\hat{\beta}^{lasso}$ at $\lambda = 3.458$

How does this compare to the prediction error curve? Because this is a simulation, we can answer this question.





Standard errors for cross-validation

One nice thing about K-fold cross-validation (for a small $K \ll n$, e.g., $K = 5$) is that we can estimate the standard deviation of $\mathrm{CV}(\theta)$, at each $\theta \in \{\theta_1, \ldots \theta_m\}$

First, we just average the validation errors in each fold:

$$\mathrm{CV}_k(\theta) = \frac{1}{n_k} e_k(\theta) = \frac{1}{n_k} \sum_{i \in F_k} \left(y_i - \hat{f}_\theta^{-k}(x_i)\right)^2$$

where n_k is the number of points in the kth fold

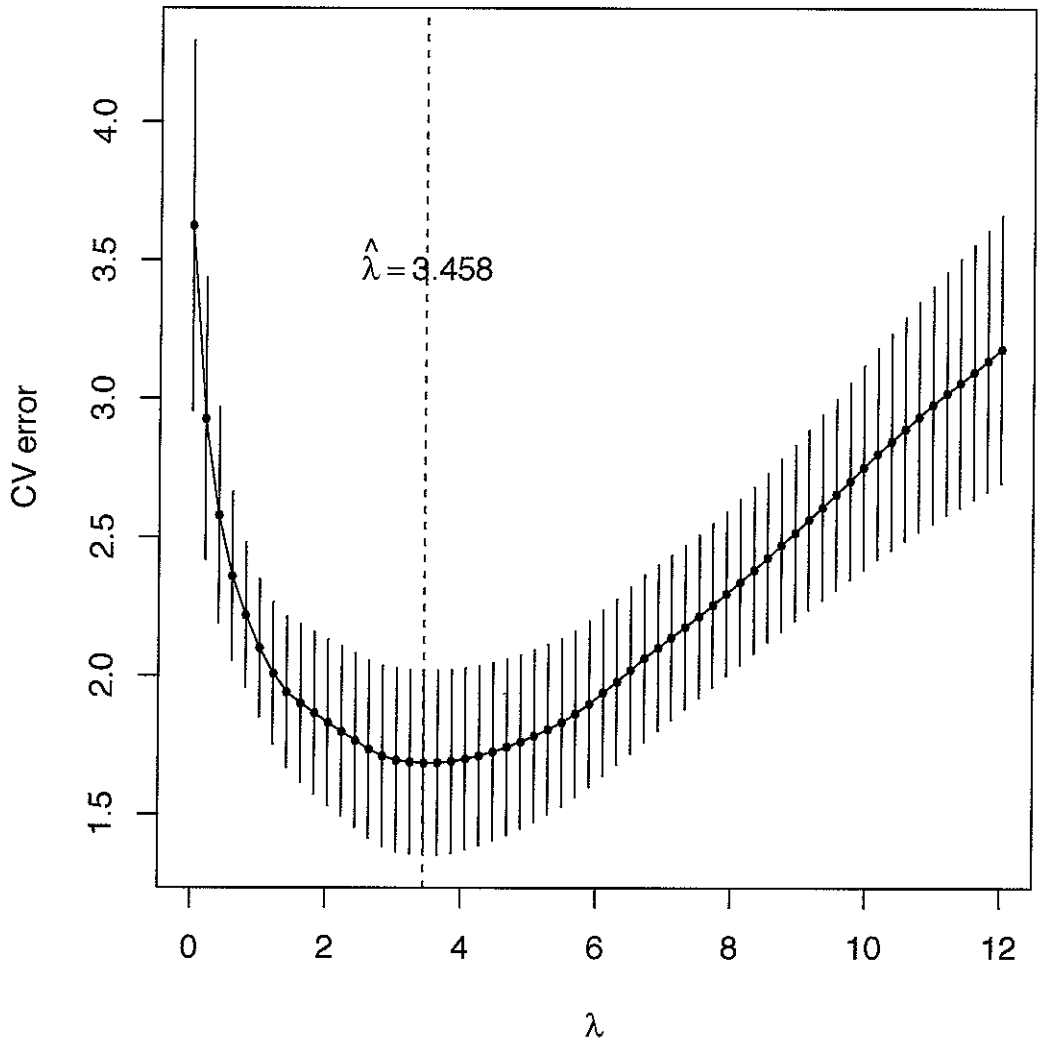
We then compute the sample standard deviation of $\mathrm{CV}_1(\theta), \ldots$ $\mathrm{CV}_K(\theta)$,

$$\mathrm{SD}(\theta) = \sqrt{\mathrm{var}\big(\mathrm{CV}_1(\theta), \ldots \mathrm{CV}_K(\theta)\big)}$$

Finally we estimate the standard deviation of $\mathrm{CV}(\theta)$—called the standard error of $\mathrm{CV}(\theta)$—by $\mathrm{SE}(\theta) = \mathrm{SD}(\theta)/\sqrt{K}$

Example: choosing λ for the lasso (continued)

The cross-validation error curve from our lasso example, with $\pm$ standard errors:



The one standard error rule

The one standard error rule is an alternative rule for choosing the value of the tuning parameter, as opposed to the usual rule

$$\hat{\theta} = \underset{\theta \in \{\theta_1, \ldots \theta_m\}}{\operatorname{argmin}} \ \mathrm{CV}(\theta)$$

We first find the usual minimizer $\hat{\theta}$ as above, and then move θ in the direction of increasing regularization (this may be increasing or decreasing θ, depending on the parametrization) as much as we can, such that the cross-validation error curve is still within one standard error of $\mathrm{CV}(\hat{\theta})$. In other words, we maintain

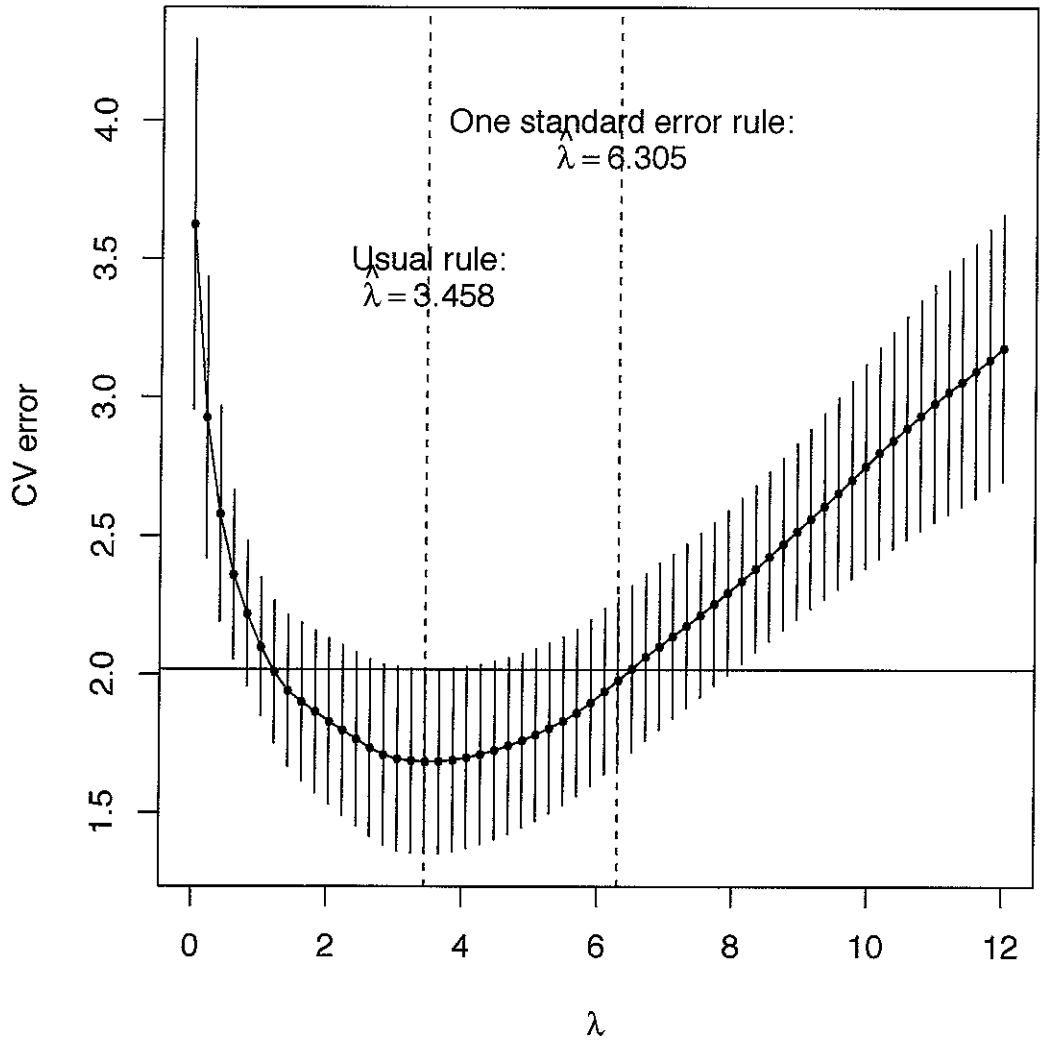
$$\mathrm{CV}(\theta) \leq \mathrm{CV}(\hat{\theta}) + \mathrm{SE}(\hat{\theta})$$

Idea: "All else equal (up to one standard error), go for the simpler (more regularized) model"

parametrize $\quad \|\beta\|_1 \leq t$

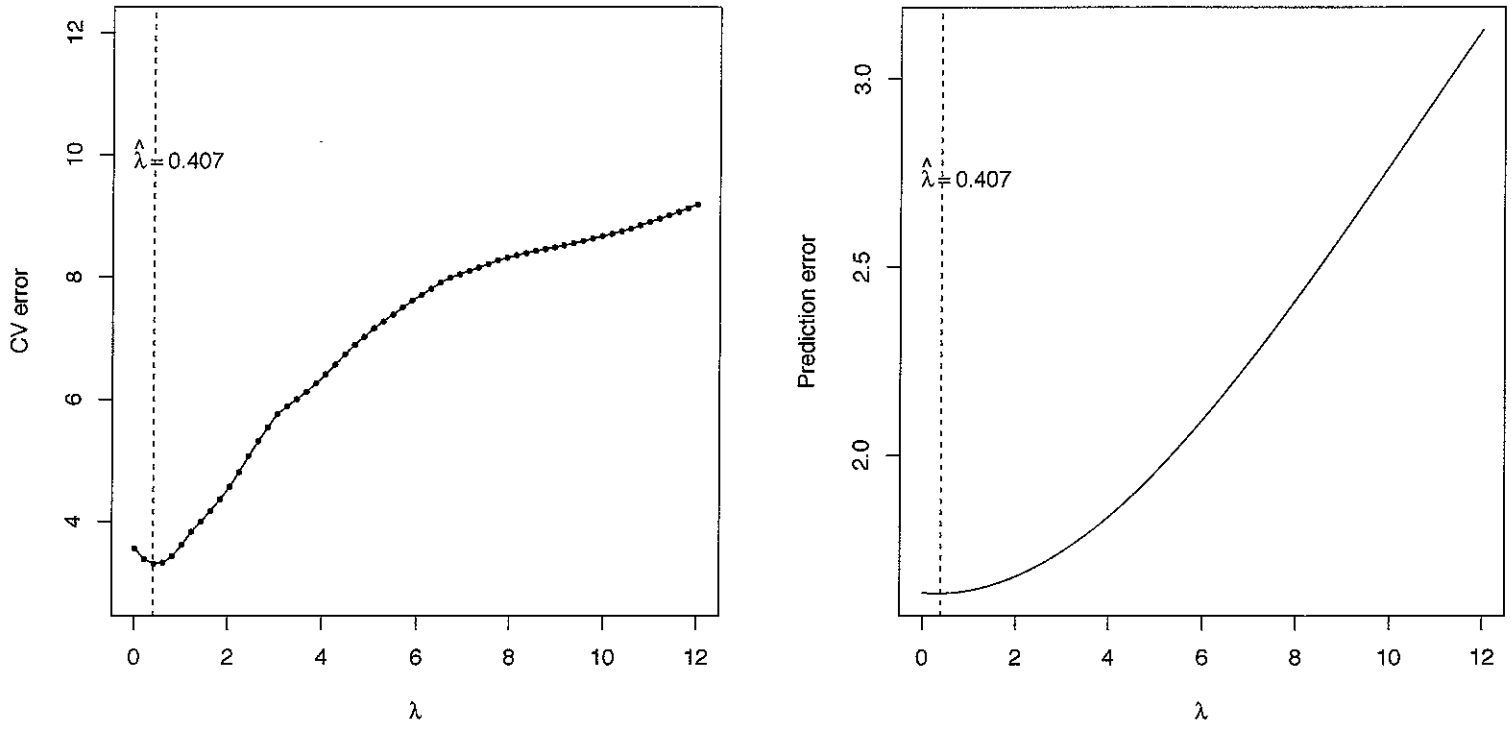
Example: choosing λ for the lasso (continued)

Back to our lasso example: the one standard error rule chooses a larger value of λ (more regularization):

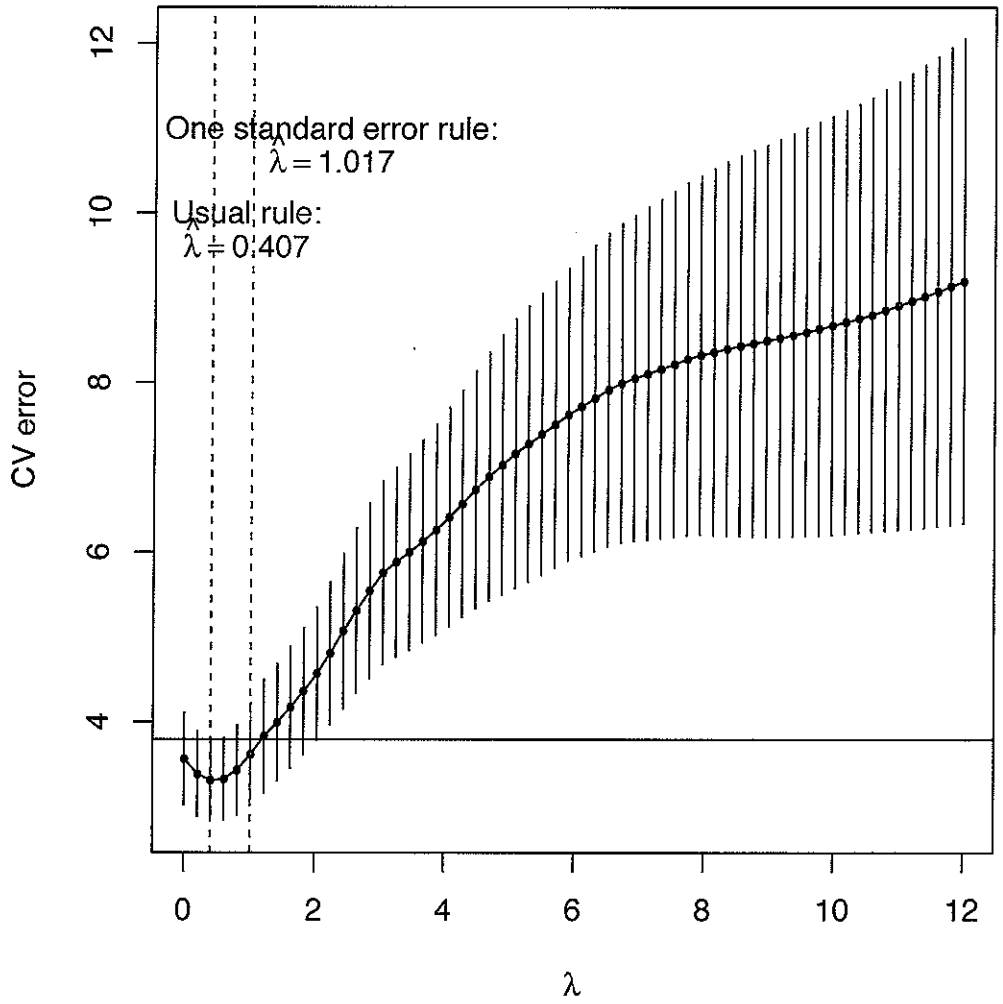


What happens if we really shouldn't be shrinking in the first place? We'd like cross-validation, our automated tuning parameter selection procedure, to choose a small value of λ

Recall our other example from last time: $n = 50$, $p = 30$, and the true model is linear with all moderately large coefficients



The standard errors here reflect uncertainty about the shape of the CV curve for large λ



The one standard error rule doesn't move much from the minimum

Prediction versus interpretation

Remember: cross-validation error estimates prediction error at any fixed value of the tuning parameter, and so by using it, we are implicitly assuming that achieving minimal prediction error is our goal

Choosing λ for the goal of recovering the true model, for the sake of interpretation, is somewhat of a different task. The value of the parameter that achieves the smallest cross-validation error often corresponds to not enough regularization for these purposes. But the one standard error rule is a step in the right direction

There are other (often more complicated) schemes for selecting a value of the tuning parameter for the purposes of true model recovery

Recap: cross-validation

Training error, the error of an estimator as measured by the data used to fit it, is not a good surrogate for prediction error. It just keeps decreasing with increasing model complexity

Cross-validation, on the other hand, much more accurately reflects prediction error. If we want to choose a value for the tuning parameter of a generic estimator (and minimizing prediction error is our goal), then cross-validation is the standard tool

We usually pick the tuning parameter that minimizes the cross-validation error curve, but we can also employ the one standard error rule, which helps us pick a more regularized model

Next time: model assessment and more cross-validation

How would we do cross-validation for a structured problem like that solved by smoothing splines?