Validation and Cross-Validation

Using a test sample for both validation and testing is a bad idea.

Any method is likely to work better on data used previously (even if just to choose a tuning parameter) than on virgin data.

Reserving part of the data as a **validation sample**, or **validation set**, or **hold-out sample** is a way to estimate tuning parameters cleanly.

If we divide the sample into training and validation samples, it is important to do so randomly.

Unless we know the data are ordered randomly, we cannot just put the first half of the dataset into the training sample and the second half into the validation sample.

We also cannot generate n random variables η_i from $\mathrm{U}(0,1)$ and assign observation i to training if $\eta_i < 0.5$ and to validation otherwise, because the two samples will be different sizes.

We could generate n values of η_i, sort them, and assign observation i to the training set if $i \leq n/2$ and to the validation set otherwise.

Of course, this assumes the data are independent across observations, which is false for time-series data and clustered data.

- It is not obvious what to do in such cases.
- For clustered data, it seems natural to put half the clusters (randomly) into the T sample and half into the V sample.
- But the two samples will almost certainly be different sizes!
- Dealing with time-series data is even harder.
- We do not want to put the first half of the dataset into one sample and the second half into the other.
- Perhaps we could divide it into blocks and randomly put half the blocks into each sample. But how long should the blocks be?
- There will inevitably be dependence between observations at the end of one block and the start of the next.

The validation sample approach is far from perfect.

- Because both training and validation samples have (roughly) $n/2$ observations, the estimates are less efficient and more biased.
- The right tuning parameter for a sample of size $n/2$ may not be the right one for a sample of size n.
- Both the tuning parameter and the estimates will depend on how the sample happens to be split.

Figure 5.2 illustrates this for the polynomial regression of MPG on horsepower. The object is to choose d, the degree of the polynomial.

Different splits give very different MSE estimates, and different choices for d. [But this is a stupid model.]

The choice of d is not at all clear. Of course, it was also not clear when we used t-tests and the full sample.

The only certain thing is that $d > 1$: The model is not linear.

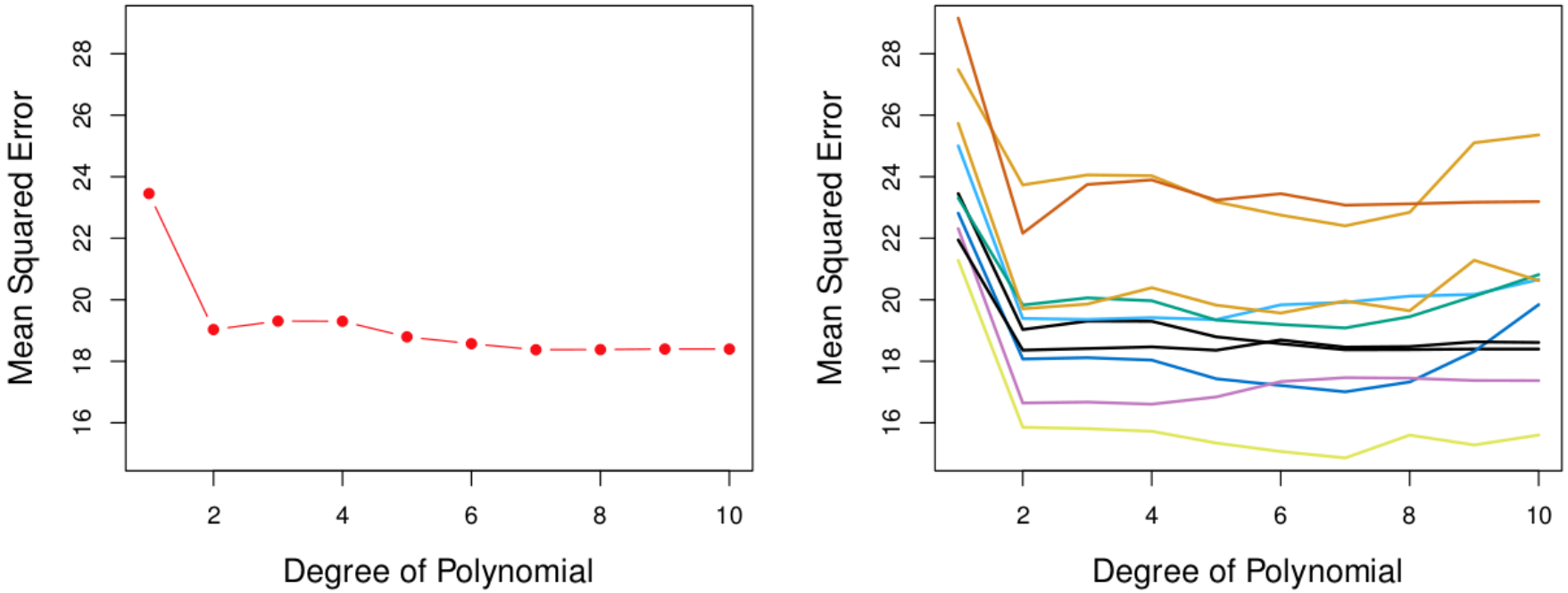

FIGURE 5.2. *The validation set approach was used on the* `Auto` *data set in order to estimate the test error that results from predicting* `mpg` *using polynomial functions of* `horsepower`. Left: *Validation error estimates for a single split into training and validation data sets.* Right: *The validation method was repeated ten times, each time using a different random split of the observations into a training set and a validation set. This illustrates the variability in the estimated test MSE that results from this approach.*

Leave-One-Out Cross-Validation

With LOO cross-validation, we use the training sample both for estimation and for validation.

Suppose we estimate $f(x_1)$ using every observation except number 1 and call the resulting estimate $\hat{f}_{-1}(x_1)$, *not* (as the book does) $\hat{y}_1$.

Inevitably, $\hat{f}_{-1}(x_1)$ will fit worse than $\hat{f}(x_1)$, because we did not use information about y_1.

If we applied this LOO estimator to some unobserved x_0, it ought to work just about the same as the actual estimator, because $n - 1 \approx n$, and the two samples have $n - 1$ observations in common.

But the validation sample is of size 1, which is insanely small.

We therefore run through the entire sample n times, omitting one observation each time, and computing the fits for each of the omitted observations.

Let $\hat{f}_{-i}(\boldsymbol{x}_i)$ denote the estimate of $f(\boldsymbol{x}_i)$ based on all observations except observation i.

Then we define the **leave-one-out cross-validation**, or **LOOCV**, estimate of test MSE as

$$\text{CV}_{(n)} = \frac{1}{n}\sum_{i=1}^{n}\text{SE}_i, \quad \text{where} \quad \text{SE}_i = \left(y_i - \hat{f}_{-i}(\boldsymbol{x}_i)\right)^2. \tag{1}$$

Here the subscript "(n)" can be read as "n-fold." Remember that this is just an estimate. The "true" test MSE would require us to have an infinitely large test dataset!

LOOCV has two major advantages:

- The LOO estimates usually have almost the same bias and variance as the actual ones.
- Conditional on the training sample, the LOOCV estimate of MSE and choice of tuning parameter are not random.

In general, LOOCV can be insanely expensive. But there are important cases where it is remarkably cheap.

Consider KNN. The usual estimate of $f(\boldsymbol{x}_i)$ is

$$\hat{f}_K(\boldsymbol{x}_i) = \frac{1}{K} \sum_{j \in \mathcal{N}_i} y_j, \tag{2}$$

where $\mathcal{N}_i$ contains the K observations nearest to $\boldsymbol{x}_i$, including observation i itself.

For the LOO estimate, we just have to modify $\mathcal{N}_i$ slightly. We omit observation i and add the observation that is $(K+1)^{\text{st}}$ nearest:

$$\hat{f}_{K,-i}(\boldsymbol{x}_i) = \hat{f}_K(\boldsymbol{x}_i) + \frac{1}{K}(y_{i,K+1} - y_i), \tag{3}$$

where $y_{i,K+1}$ denotes the $(K+1)^{\text{st}}$ nearest observation.

Suppose we are interested in values of K from $K_{\min}$ to $K_{\max}$.

We could start by finding an ordered list of the nearest observations to each observed $\boldsymbol{x}_i$ for $k = 1, \ldots, K_{\max} + 1$.

For each i, the KNN estimator itself uses the first K observations in the sorted list for observation i, which starts with the observation itself.

The LOO estimator then uses observations 2 to $K + 1$.

We can rapidly compute $\hat{f}_{K,-i}(\boldsymbol{x}_i)$ for $K = K_{\min}, \ldots, K_{\max}$, all at the same time.

Then we can compute the $\mathrm{CV}_{(n)}$ for each value of K and use them to choose the value that seems to work best.

Of course, replacing $\boldsymbol{x}_i$ with number $K + 1$ in the sorted list will increase the bias. The increase may be noticeable if the data points are sparse. Thus even LOO cross-validation is not perfect.

Similar computational tricks can be used for kernel regression and smoothing splines, making LOO cross-validation attractive.

The `knn.reg` function reports predictive R^2 based on LOO cross-validation. Here are results for the regression of `gpm` on horsepower:

$$K = 3 \;\; 0.7413301, \quad K = 5 \;\; 0.7733892, \quad \boldsymbol{K = 7 \;\; 0.7775314}, \quad K = 9 \;\; 0.7740142, \quad K = 11 \;\; 0.7675287, \quad K = 13 \;\; 0.7684997$$

The `knn.reg` function is part of the `FNN` package. Here "F" stands for "fast", which suggests that it will work for large n.

This package also provides the functions `knn` for classification and `knn.cv` for classification with LOO cross-validation.

LOO Cross-Validation for Linear Regression

For OLS regression, the effect of omitting observation i on the fitted value for that observation is just

$$\hat{f}(\boldsymbol{x}_i) - \hat{f}_{-i}(\boldsymbol{x}_i) = \hat{u}_i^{(-i)} - \hat{u}_i = \hat{u}_i \frac{h_i}{1 - h_i}, \tag{4}$$

where $\hat{u}_i = y_i - \hat{y}_i$ is the OLS residual for observation i, and h_i is the i^{th} diagonal element of the **hat matrix**

$$\boldsymbol{P}_X = \boldsymbol{X}(\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top. \tag{5}$$

We made use of (4) when we discussed leverage and influence.

The result (4) makes it very easy to compute omit-1 residuals and fitted values.

Since $\hat{u}_i^{(-i)} = \hat{u}_i/(1-h_i)$,

$$\text{CV}_{(n)} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{\hat{u}_i}{1-h_i}\right)^2. \tag{6}$$

Thus, once we have calculated the diagonals of the hat matrix (perhaps using the `hatvalues` function in R), it is trivial to compute the LOO estimate of test MSE.

As mentioned earlier, LOO cross-validation does not make sense for time-series data and clustered data.

Both LOO cross-validation and k-fold cross-validation (to be discussed next) are used only to estimate the best value of a tuning parameter.

The final estimates are then obtained using the entire training sample. Optionally, they may then be evaluated using the test sample.

There is another approach, called **cross-fitting**, which takes averages of estimates computed using different folds (subsets) of the sample.

k-Fold Cross-Validation

When LOO cross-validation is infeasible, we can do something similar, but using k **folds** of size n/k.

Ideally, n/k is an integer. Popular values are 5 and 10. If n happens to divide evenly by 6, 7, 8, or 9, we might use one of those values.

Unlike LOO cross-validation, k-fold cross-validation involves random splitting of the sample. We need to be careful how we do this.

Every fold should be a random subsample of size n/k.

If we change the random numbers used to split the sample into folds, our results will change. But there is much less variation than with the validation set approach; compare Figures 5.2 and 5.4.

As with LOO, we estimate the model k times, each time using a subsample of $k-1$ folds for estimation and the remaining fold to compute the MSE.

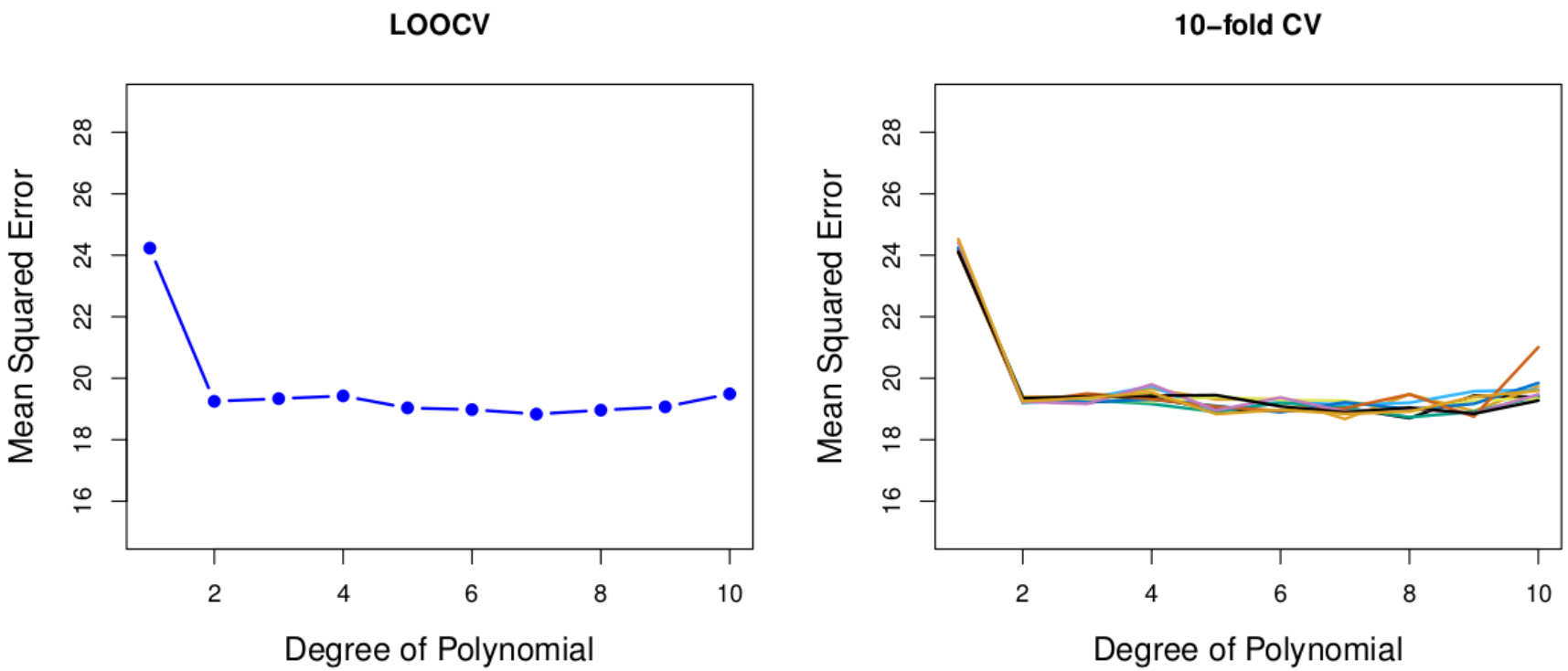

FIGURE 5.4. *Cross-validation was used on the* `Auto` *data set in order to estimate the test error that results from predicting* `mpg` *using polynomial functions of* `horsepower`. Left: *The LOOCV error curve.* Right: 10*-fold CV was run nine separate times, each with a different random split of the data into ten parts. The figure shows the nine slightly different CV error curves.*

For computation, we want k to be small. Nobody performs 47-fold cross-validation (unless $n = 47$, when it becomes LOOCV).

In most cases, the main purpose of cross-validation is to estimate a tuning parameter.

Thus we care about the shape of the function

$$\mathrm{CV}_{(k)}(\lambda) = \frac{1}{k}\sum_{j=1}^{k}\mathrm{MSE}_j(\lambda), \tag{7}$$

where $\mathrm{MSE}_j(\lambda)$ is the test MSE for fold j, based on estimates from the other $k-1$ folds, for tuning parameter λ.

We want (7) to be similar to the shape of the unknown function $\mathrm{MSE}(\lambda)$, especially near the minima of both functions.

In (7), λ denotes a generic tuning parameter. For obvious reasons, I did not use K or $1/K$ (as in KNN) here.

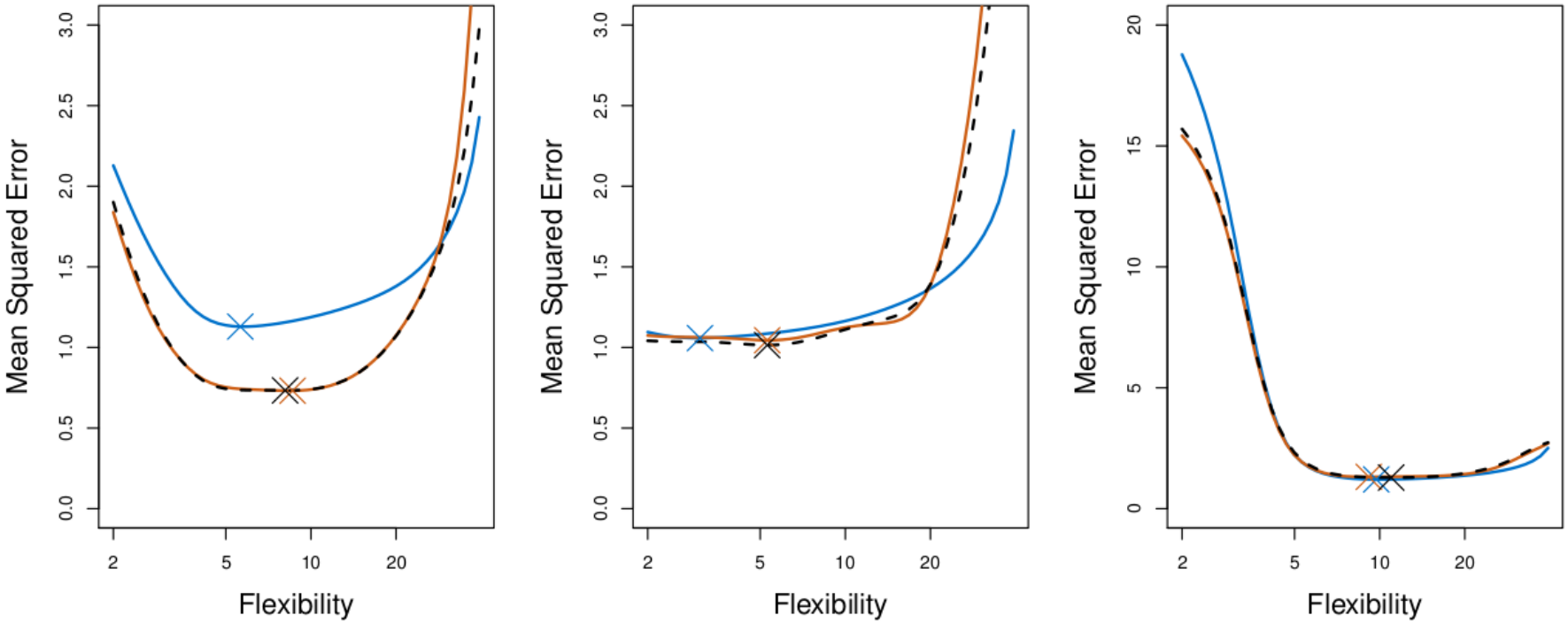

FIGURE 5.6. *True and estimated test MSE for the simulated data sets in Figures 2.9 (*left*), 2.10 (*center*), and 2.11 (*right*). The true test MSE is shown in blue, the LOOCV estimate is shown as a black dashed line, and the* 10*-fold CV estimate is shown in orange. The crosses indicate the minimum of each of the MSE curves.*

LOO versus k-fold Cross-Validation

LOO cross-validation estimates the MSE with little bias, because $n - 1 \approx n$.

k-fold cross-validation sometimes estimates the MSE with considerable bias, because $n(k-1)/k < n$.

However, LOO cross-validation has more variance than k-fold cross-validation.

- The n samples of size $n - 1$ differ only slightly from each other, so that their estimates are very highly correlated.
- The k samples of size $n(k-1)/k$ also yield correlated estimates, but the correlation is not as strong.

The optimal choice of k depends on n (ESL, Chapter 7).

For many estimators, performance initially improves rapidly with n, but the rate of improvement diminishes as n increases.

This happens because the bias of $\hat{f}(x_0)$ depends nonlinearly on how many observations there are near x_0.

For large n, samples of size $4n/5$ will perform almost as well as samples of size n. 5-fold CV will provide excellent estimates of MSE.

For small n, even samples of size $9n/10$ may have significantly more bias than ones of size n. So 10-fold CV may not work that well.

We probably want k to get smaller as n gets larger. We might even use $k = 3$ or $k = 4$ when n is very large.

- For large n, computational cost matters, so try to keep k small.
- In this case, $\text{MSE}(n(k-1)/k) \approx \text{MSE}(n)$ even when k is small.
- For small n, $\text{MSE}(n(k-1)/k)$ is likely to exceed $\text{MSE}(n)$, and the chosen tuning parameter may call for too much flexibility.
- Make k large, perhaps even 12 or 15, when n is small. Luckily, this is when using large k is cheap.
- We can try more than one value of k and several random splits.

The Wrong Way to Do Cross-Validation

It is often tempting to do cross-validation incorrectly, perhaps leading to severely misleading results.

For each validation subsample, we need to do exactly the same things that we did with the full training sample.

But when there are many predictors, it is very common to start with some sort of pre-screening procedure, perhaps dropping predictors that have little correlation with the output.

After that, we employ an ML estimation method (polynomial regression, KNN, many more coming soon) that requires a tuning parameter.

If we use CV in such a case, we need to perform both the pre-screening procedure and the ML estimation method for every validation sample.

If we just use CV for the second step, ignoring the pre-screening procedure, we can get very misleading results; see ESL, Section 7.10.2.

Essentially the same advice applies whenever we use bootstrap methods (to be discussed fairly soon).

- The CV subsamples should look as much as possible like the training sample, and we want to treat them in the same way.
- For k-fold cross-validation, it is good to try more than one k and, if possible, more than one set of random splits for each k.
- We could then average the estimates of $\mathrm{CV}_{(k)}(\lambda)$ across random splits for each k and then plot each of the averages against λ (the tuning parameter).
- Ideally, all k and all random splits will yield roughly the same λ.

When n is not large enough for bias to be nearly invariant to k, we would expect smaller values of k to suggest that bias is larger than it is. Thus they would call for more flexibility (less smoothness).