Resampling Methods

Resampling methods are necessary tools in modern statistics. They involve repeatedly drawing samples from a training set and refitting a model of interest on each sample in order to obtain additional information about the fitted model.

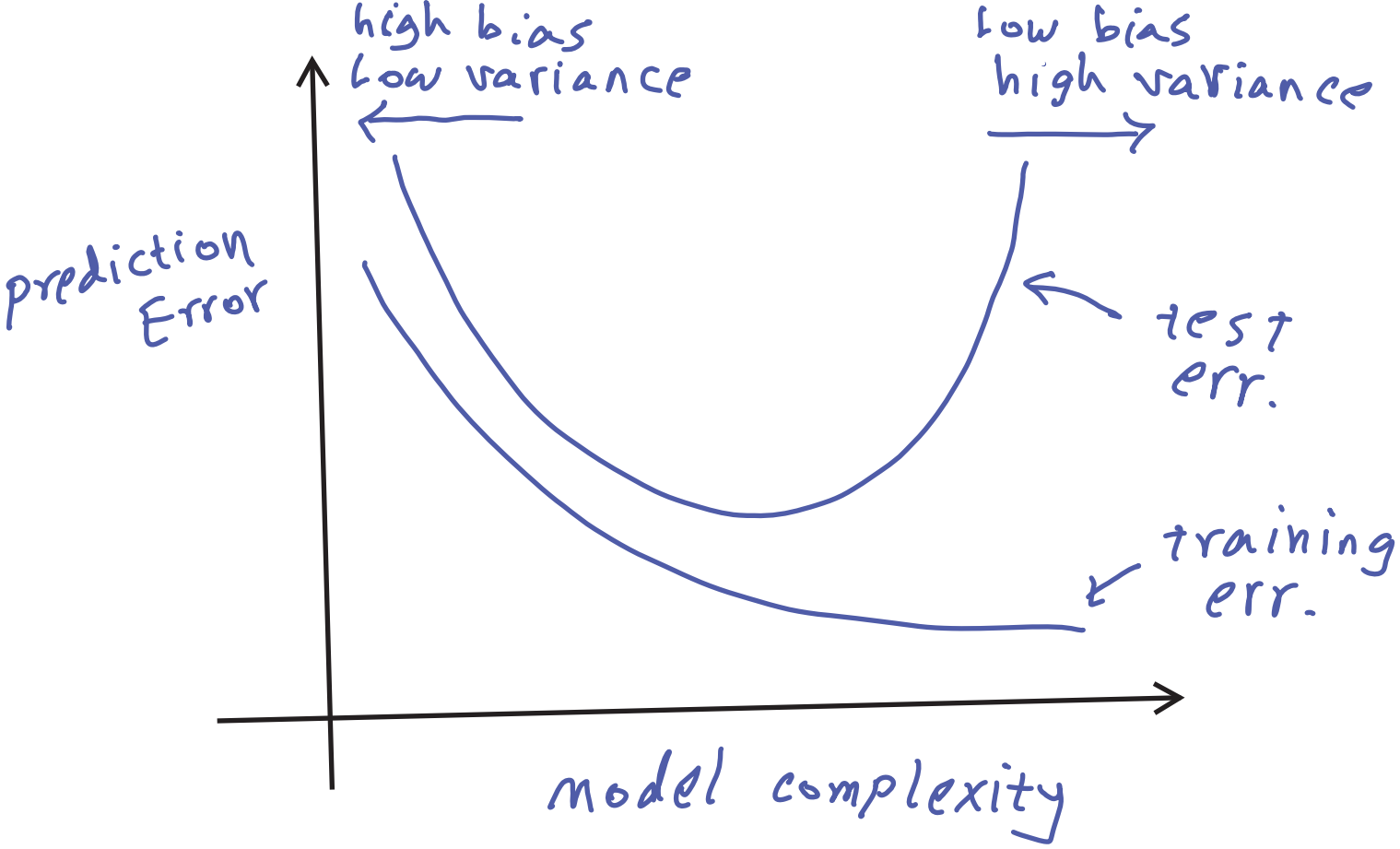
For example, they provide estimates of test-set prediction error, and the standard deviation and bias of our parameter estimates.

Here, we discuss two of the most commonly used resampling methods, cross-validation and the bootstrap.

Cross-Validation

- Recall the distinction between the test error and the training error:
- The test error is the average error that results from using a statistical learning method to predict the response on a new observation, one that was not used in training the method.
- In contrast, the training error can be easily calculated by applying the statistical learning method to the observations used in its training.
- But the training error rate often is quite different from the test error rate, and in particular the former can dramatically underestimate the latter.

In order to directly estimate the test error rate, we need a large designated test set which is often not available.

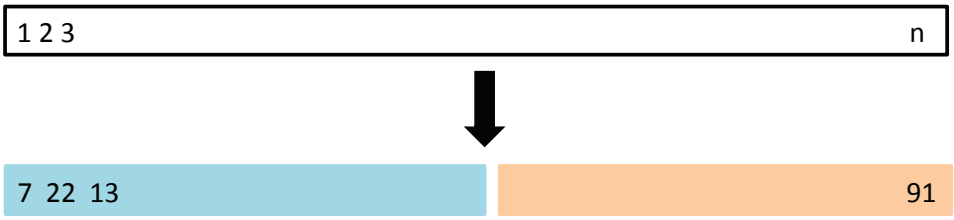


- Some methods make a mathematical adjustment to the training error rate in order to estimate the test error rate.
- Here we instead consider a class of methods that estimate the test error by holding out a subset of the training observations from the fitting process, and then applying the statistical learning method to those held out observations.

The Validation Set Approach

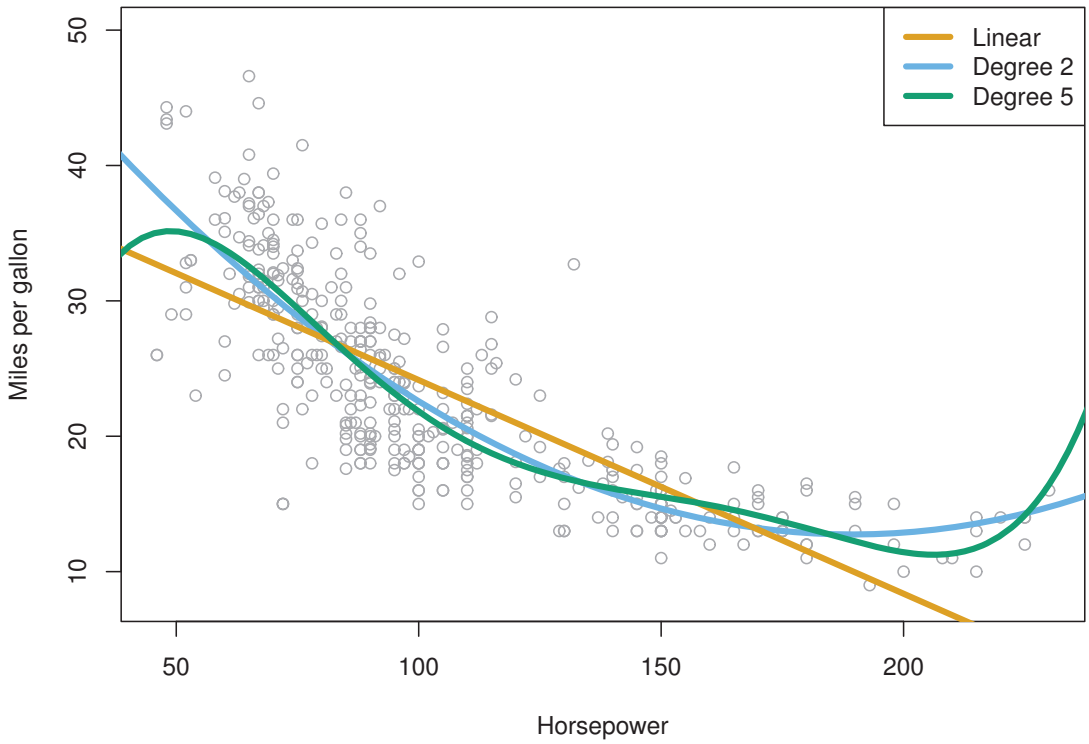
- Here we randomly divide the available set of samples into two parts: a training set and a validation or hold-out set.
- The model is fit on the training set, and the fitted model is used to predict the responses for the observations in the validation set.

- The resulting validation-set error provides an estimate of the test error. This is typically assessed using MSE in the case of a quantitative response and misclassification rate in the case of a qualitative (discrete) response.



A schematic display of the validation set approach. A set of n observations are randomly split into a training set (shown in blue, containing observations 7, 22, and 13, among others) and a validation set (shown in beige, and containing observation 91, among others). The statistical learning method is fit on the training set, and its performance is evaluated on the validation set.

We illustrate the validation set approach on the Auto data set.



The Auto data set. For a number of cars, mpg and horsepower are shown. The linear regression fit is shown in orange. The linear regression fit for a model that includes horsepower^2 is shown as a blue curve. The linear regression fit for a model that includes all polynomials of horsepower up to fifth-degree is shown in green.

Here we want to use the validation method to see whether a model that predicts mpg using horsepower and horsepower^2 or higher orders of horsepower gives better results than a model that uses only a linear term.

We begin by using the sample() function to split the set of observations into two halves, by selecting a random subset of 196 observations out of the original 392 observations. We then fit a linear regression using only the observations corresponding to the training set.

```
> library(ISLR2)
> set.seed(1)
```

```
> train <- sample(392, 196)
> lm_fit_auto_1 <- lm(mpg ~ horsepower, data = Auto, subset = train)
```

We now use the predict() function to estimate the response for all 392 observations, and we use the mean() function to calculate the MSE of the 196 observations in the validation set.

```
> mean((mpg - predict(lm_fit_auto_1, Auto))[-train]^2)
[1] 26.14142
```

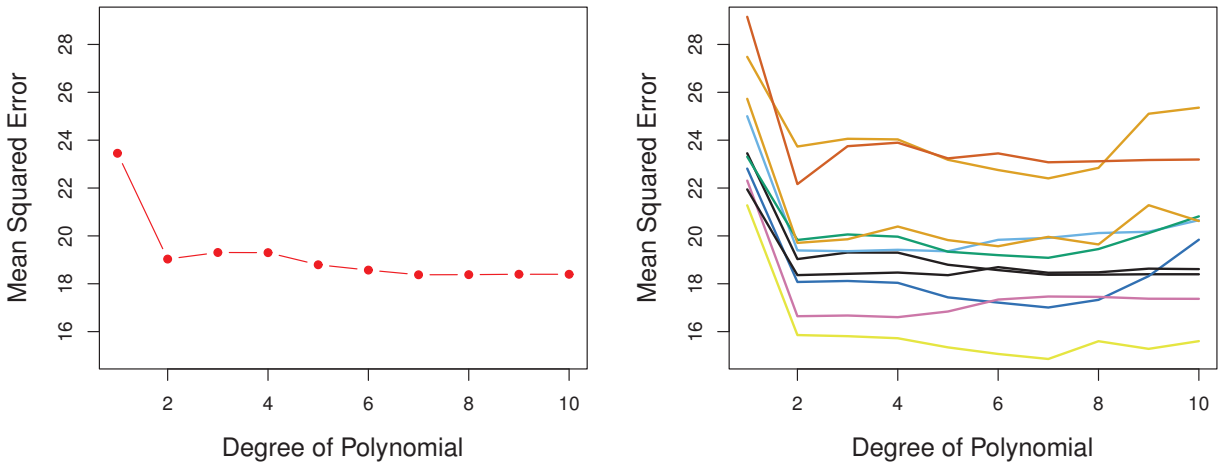
Therefore, the estimated test MSE for the linear regression fit is 26.14. We can use the poly() function to estimate the test error for the quadratic and cubic regressions.

```
> lm_fit_auto_2 <- lm(mpg ~ poly(horsepower, 2, raw = TRUE), data = Auto,
   subset = train)
> mean((mpg - predict(lm_fit_auto_2, Auto))[-train]^2)
[1] 19.82259
> lm_fit_auto_3 <- lm(mpg ~ poly(horsepower, 3, raw = TRUE), data = Auto,
   subset = train)
> mean((mpg - predict(lm_fit_auto_3, Auto))[-train]^2)
[1] 19.78252
```

Note: If we choose a different training set instead, then we will obtain somewhat different errors on the validation set.

```
> set.seed(2)
> train <- sample(392, 196)
> lm_fit_auto_1 <- lm(mpg ~ horsepower, data = Auto, subset = train)
> mean((mpg - predict(lm_fit_auto_1, Auto))[-train]^2)
[1] 23.29559
> lm_fit_auto_2 <- lm(mpg ~ poly(horsepower, 2, raw = TRUE), data = Auto,
   subset = train)
```

```
> mean((mpg - predict(lm_fit_auto_2, Auto))[-train]^2)
[1] 18.90124
> lm_fit_auto_3 <- lm(mpg ~ poly(horsepower, 3, raw = TRUE), data = Auto,
   subset = train)
> mean((mpg - predict(lm_fit_auto_3, Auto))[-train]^2)
[1] 19.2574
```



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.

Based on the variability among these curves, all that we can conclude with any confidence is that the linear fit is not adequate for this data.

The validation set approach is conceptually simple and is easy to implement. But it has two potential drawbacks:

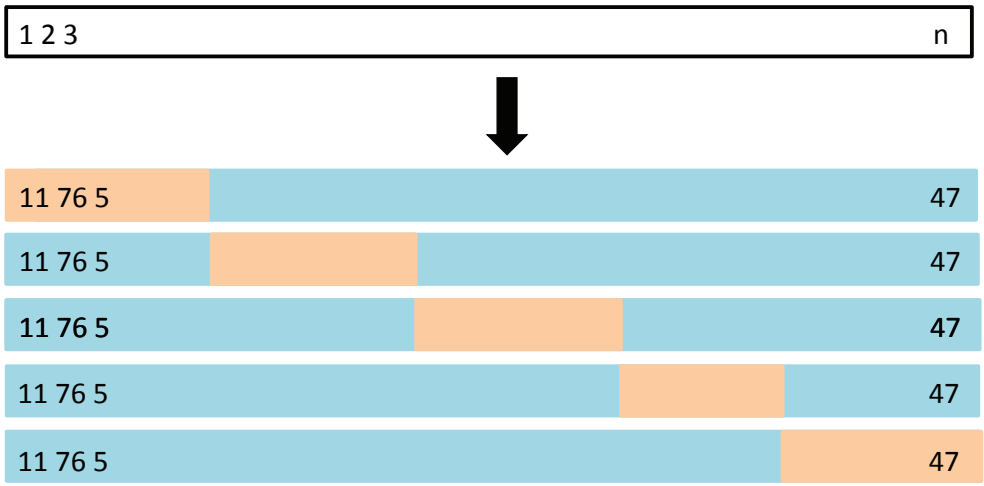
1. The validation estimate of the test error rate can be highly variable, depending on precisely which observations are included in the training set and which observations are included in the validation set.
2. In the validation approach, only a subset of the observations—those that are included in the training set rather than in the validation set—are used to fit the model. Since statistical methods tend to perform worse when trained on fewer observations, this suggests that the validation set error rate may tend to overestimate the test error rate for the model fit on the entire data set.

k-Fold Cross-Validation

k-fold CV is a widely used approach for estimating test error.

- This approach involves randomly dividing the set of observations into k groups, or folds, of approximately equal size.
- The first fold is treated as a validation set, and the method is fit on the remaining $k - 1$ folds.
- The mean squared error, MSE_1, is then computed on the observations in the held-out fold.
- This procedure is repeated k times; each time, a different group of observations is treated as a validation set.
- This process results in k estimates of the test error, $\text{MSE}_1, \text{MSE}_2, \ldots, \text{MSE}_k$. The k-fold CV estimate is computed by averaging these values,

$$CV_{(k)} = \frac{1}{k} \sum_{i=1}^{k} \text{MSE}_i$$



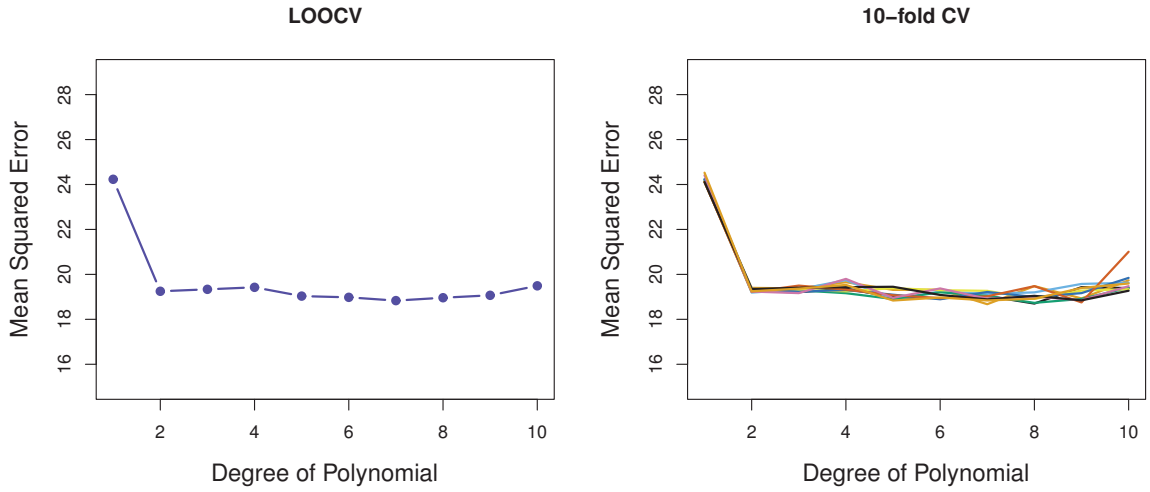
A schematic display of 5-fold CV. A set of n observations is randomly split into five non-overlapping groups. Each of these fifths acts as a validation set (shown in beige), and the remainder as a training set (shown in blue). The test error is estimated by averaging the five resulting MSE estimates.

- Setting $k = n$ yields n-fold or leave-one out cross-validation (LOOCV).
- LOOCV is sometimes useful. It will give approximately unbiased estimates of the test error, since each training set contains $n-1$ observations, which is almost as many as the number of observations in the full data set. But LOOCV has some disadvantages.

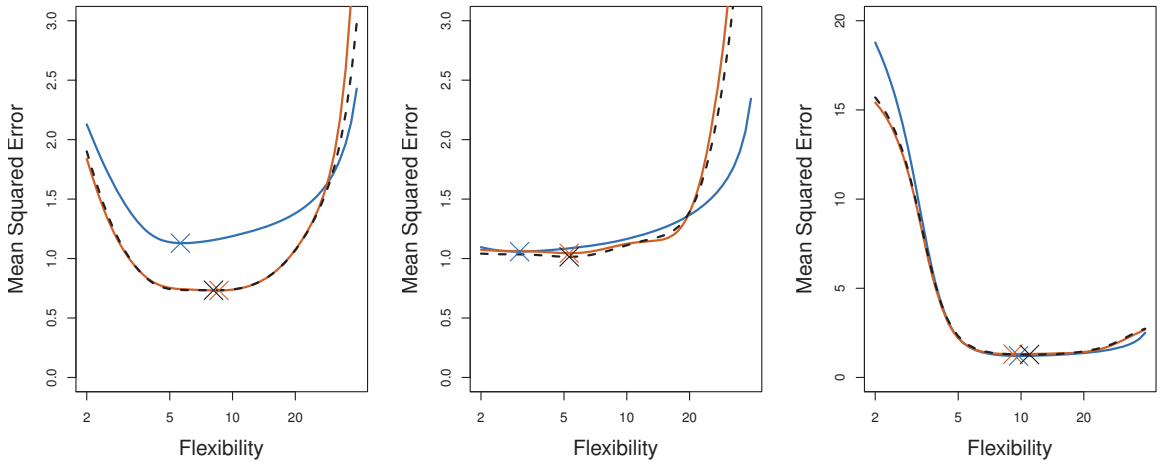
(i) It requires fitting the statistical learning method n times. This has the potential to be computationally expensive.

(ii) The estimates from each fold are highly correlated and hence their average can have high variance.

- In k-fold CV, since each training set is only $(k-1)/k$ as big as the original training set, the estimates of prediction error will typically be biased upward.
- $k = 5$ or 10 provides a good compromise for this bias-variance trade-off.



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.



True and estimated test MSE for three different simulated data sets. 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.

Now we perform LOOCV and k-fold CV on the Auto data set.

The LOOCV estimate can be automatically computed for any generalized linear model using the glm() and cv.glm() functions. The cv.glm() function is part of the boot library. If we use glm() to fit a model without passing in the family argument, then it performs linear regression, just like the lm() function.

```
> library(boot)
> glm_fit <- glm(mpg ~ horsepower, data = Auto)
> loocv_err <- cv.glm(Auto, glm_fit)
> loocv_err$delta[1]
[1] 24.23151
```

The cv.glm() function produces a list with several components. The first number in the delta vector is the cross-validation result.