Resampling Methods

Szu-Chi Chung

Department of Applied Mathematics, National Sun Yat-sen University

1. Why resampling?

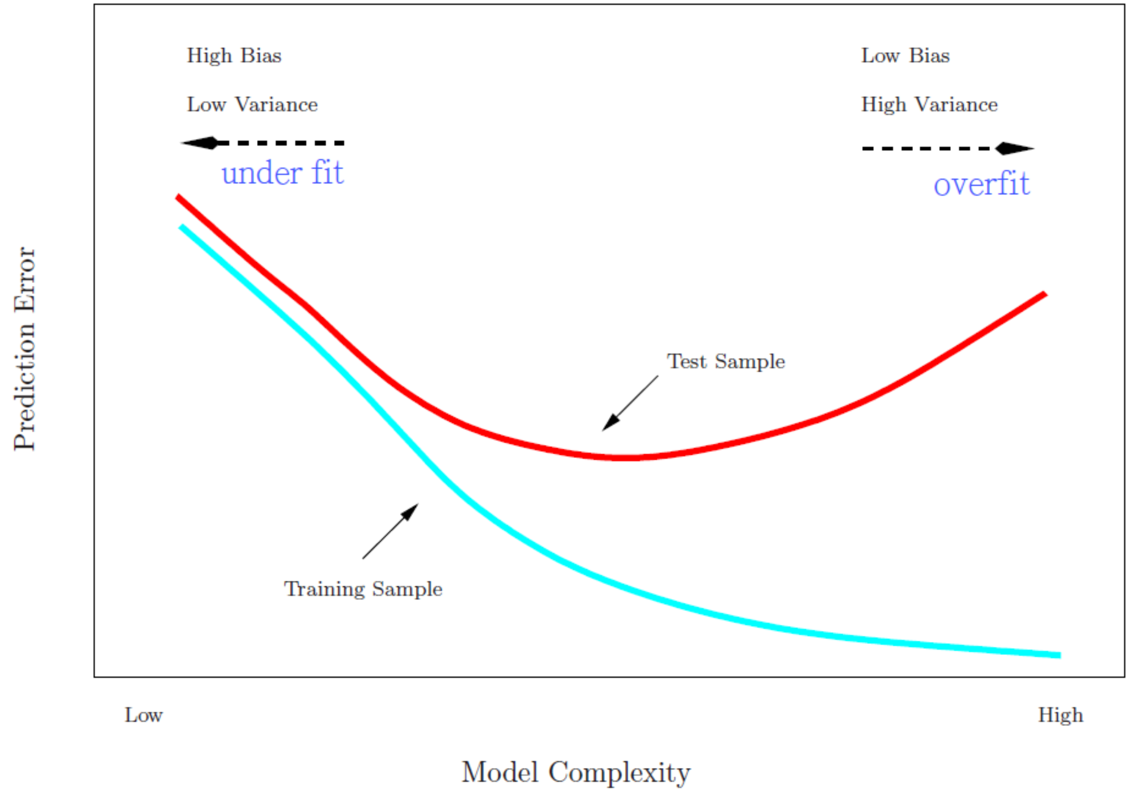
- These methods refit a model of interest by sampling from the training set, in order to obtain additional information about the fitted model
- Resampling approaches can be computationally expensive because they involve fitting the same statistical method multiple times using different subsets of the training data
  - However, due to recent advances in computing power, the computational requirements of resampling methods generally are not prohibitive

Cross-validation and the bootstrap

- We discuss two resampling methods: cross-validation and the bootstrap
  - They provide estimates of test-set prediction error and select the appropriate level of flexibility (cross-validation). It also helps us to obtain the standard deviation of our parameter estimates or conduct ensemble learning (bootstrap)
  - A *hyperparameter* is a parameter whose value is used to control the learning process and cannot be inferred while fitting the model to the training set
    - Model hyperparameters
    - Algorithm hyperparameters

Training versus testing-set performance

- 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

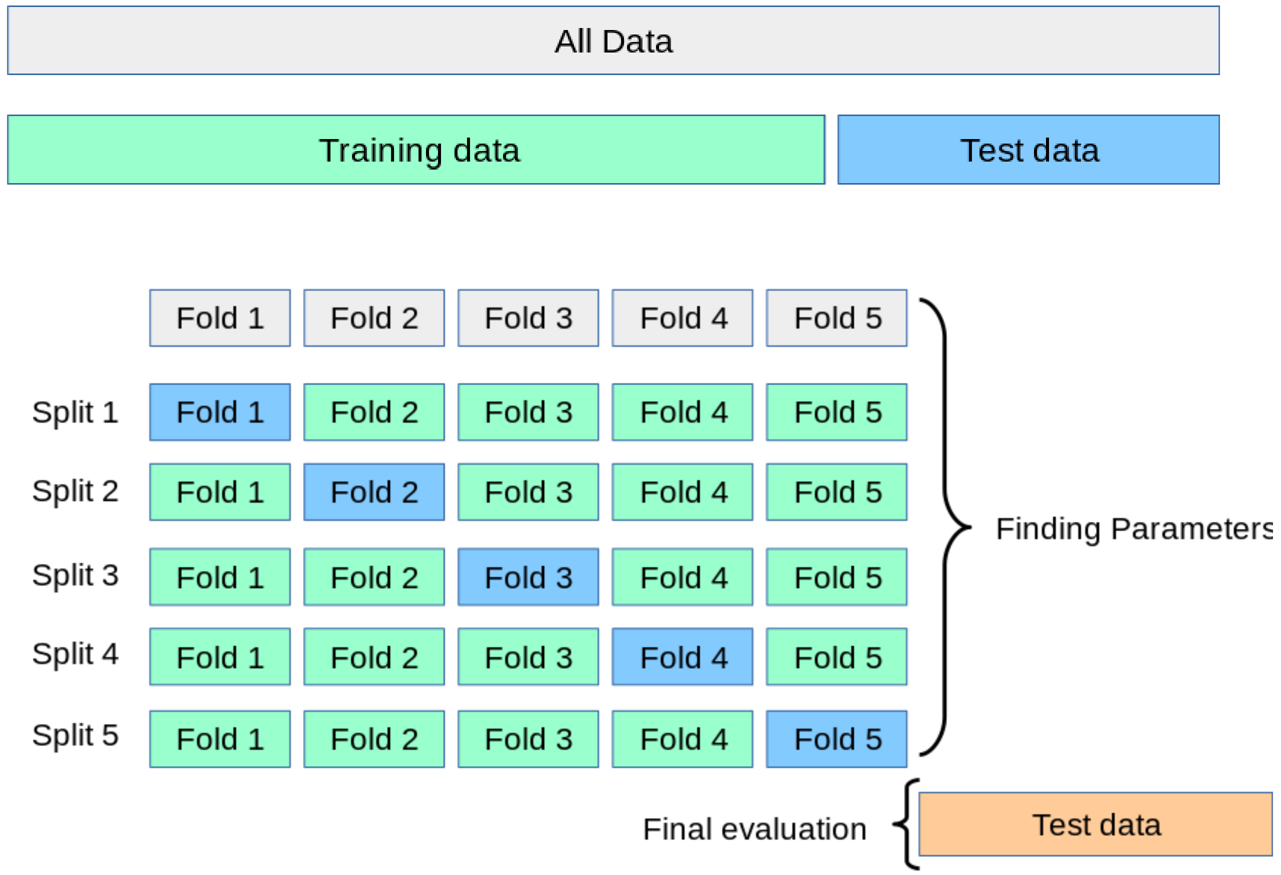


More on prediction-error estimates

- The best solution: a large designated test set. Often not available

1. Some methods make a mathematical adjustment to the training error rate in order to estimate the test error rate
   - These include the C_p statistic, AIC and BIC. They are discussed in Chapter 6
2. 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

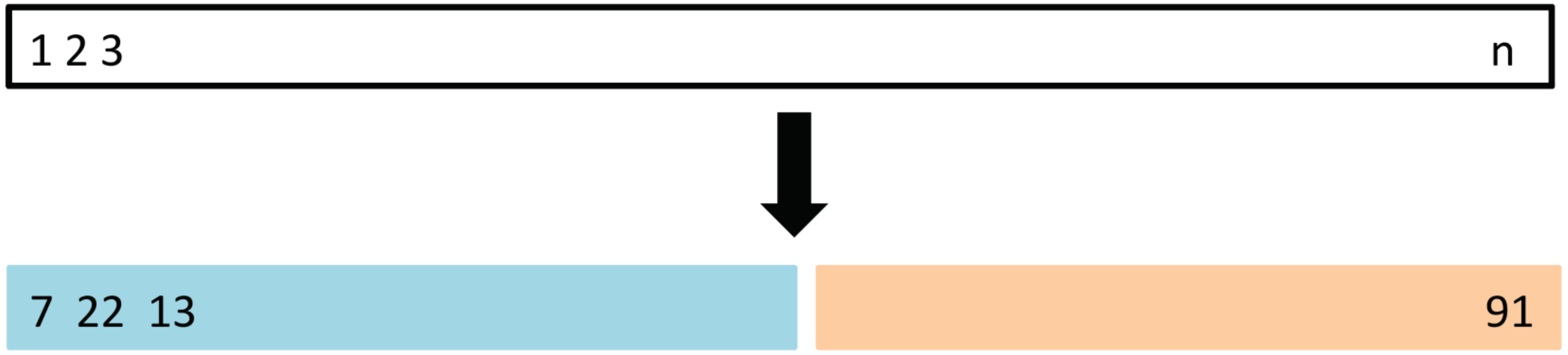
More on prediction-error estimates



https://scikit-learn.org/stable/modules/cross_validation.html

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. Estimates can be used to select the best model and to give an idea of the test error of the final chosen model



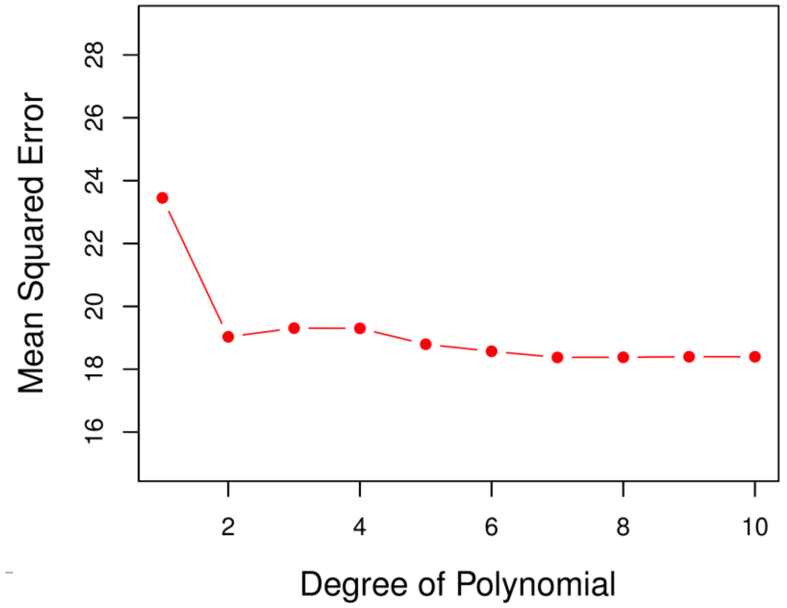
Example: automobile data

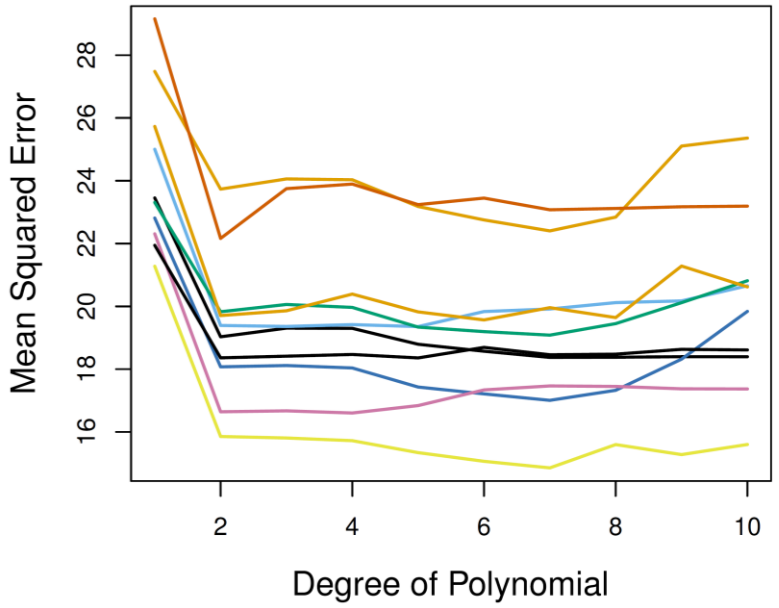
- Want to compare linear vs higher-order polynomial terms in a linear regression

$$\underset{\text{mpg}}{y} = \sum_{i=0}^{p} \beta_i \underset{\text{horsepower}}{x^i},\ p = 1,2\ \ldots$$

  - We randomly split the 392 observations into two sets, a training set containing 196 of the data points, and a validation set containing the remaining 196 observations

A single run of the validation set MSE ($\frac{RSS}{n}$)





Ten runs of validation set MSE

Drawbacks of validation set approach

1. The validation estimate of the test error 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 - are used to fit the model
    - This suggests that the validation set error may overestimate the test error for the model fit on the entire data set. Since statistical methods tend to perform worse when trained on fewer observations

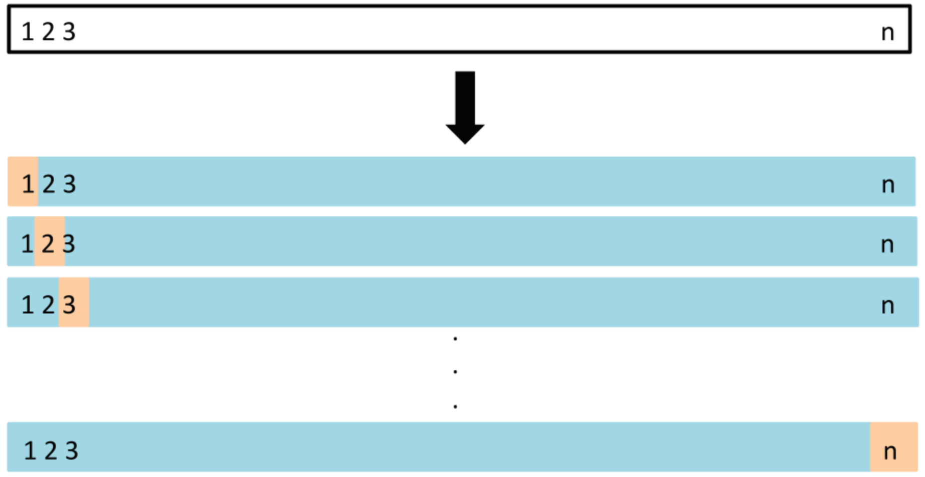
2. Leave-One-Out Cross-Validation (LOOCV)

- Instead of creating two subsets of comparable size, a single observation (x_1, y_1) is used for the validation set

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} MSE_i$$

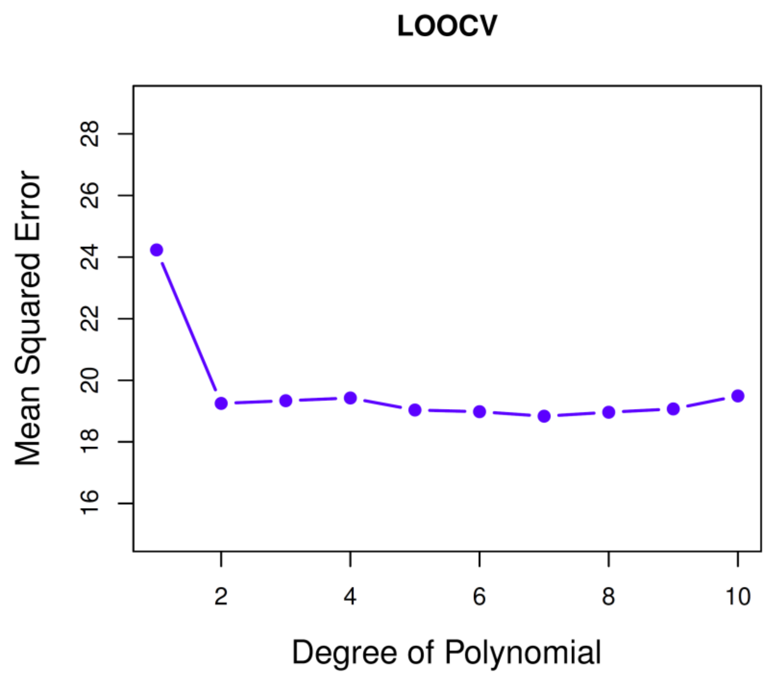
$$\text{Where } MSE_i = (y_i - \hat{y}_i)^2$$

- It has less bias and performing LOOCV multiple times will always yield the same results!



Auto data revisited

- LOOCV on the auto dataset
  - However, LOOCV has the potential to be expensive to implement, since the model has to be fit n times!



A nice special case!

- With least-squares linear or polynomial regression, a shortcut makes the cost of LOOCV the same as that of a single model! The following formula holds:

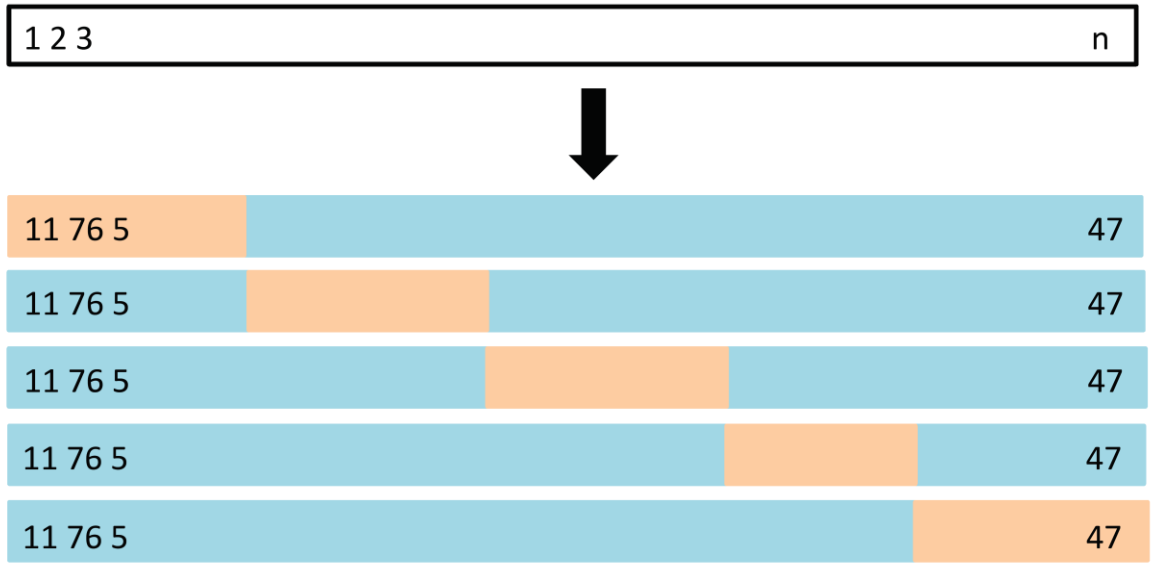
$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n}(\frac{y_i - \hat{y}_i}{1 - h_i})^2$$

  - Where $\hat{y}_i$ is the ith fitted value from the original least squares fit, and h_i is the leverage This is like the ordinary MSE, except the ith residual is divided by $1 - h_i$ (ESL, 5.5,7.10)
- LOOCV is sometimes useful but typically doesn't shake up the data enough
  - The estimates from each fold are highly correlated and hence their average can have a high variance
  - It is still time-consuming for most of the learning methods

K-fold cross-validation

- Widely used approach for estimating test error
  - The idea is to randomly divide the data into k equal-sized parts. We leave out part k, fit the model to the other $k-1$ parts, and then obtain predictions for the left-out ith part. This is done in turn for each part $i = 1,2, \ldots k$, and then the results are combined

$k = 5$ here



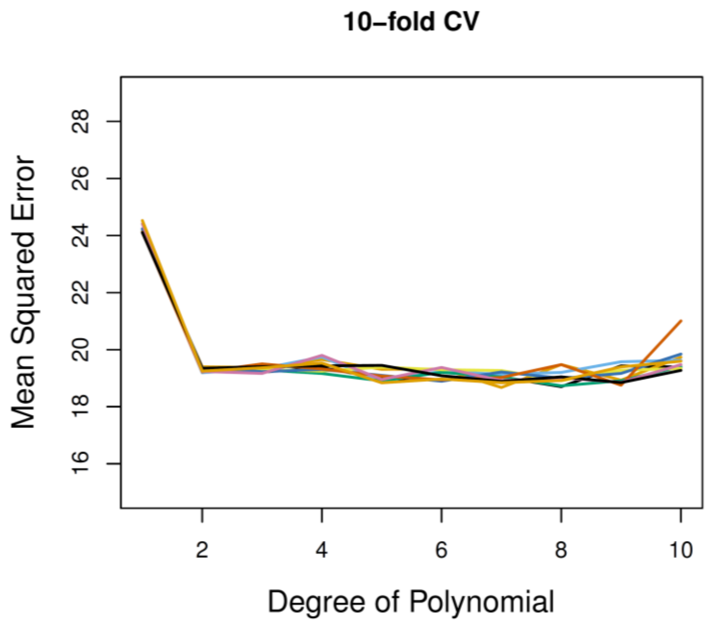
The details

- Let the k parts be $C_1, C_2, \ldots C_k$, where C_i denotes the indices of the observations in part i. There are n_i observations in part i. if n is a multiple of k, $n_i = n/k$

$$CV_{(k)} = \sum_{i=1}^{k} \frac{n_i}{n} MSE_i$$

$MSE_i = \frac{\sum_{j \in C_i} (y_j - \hat{y}_j)^2}{n_i}$, $\hat{y}_j$ is the fit for observation j, obtained with part i removed

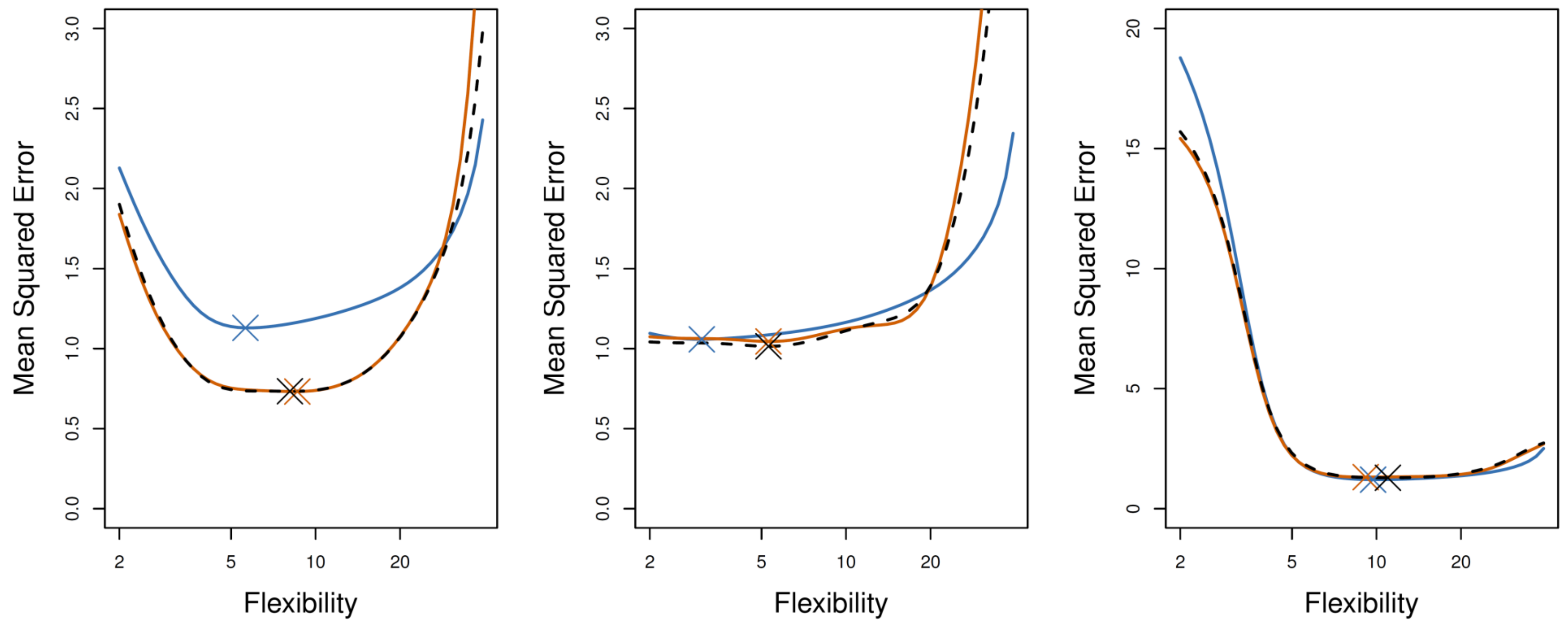
- Setting $k = n$ yields n-fold or leave-one out cross-validation (LOOCV)
  - In practice, one typically performs k-fold CV using $k = 5$ or $k = 10$



True and estimated test MSE for the simulated data

- For the simulation dataset in Chapter 2 and use linear regression

$f(x)$ is a low order polynomial — $f(x)$ is close to linear — $f(x)$ is highly nonlinear



Blue: True test MSE
Black: LOOCV's MSE
Orange: 10-Fold CV's MSE

Bias-variance trade-off for cross-validation

- 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
- This bias is minimized when $k = n$ (LOOCV), but this estimate has a high variance, as noted earlier. $k = 5$ or 10 provides a good compromise for this bias-variance tradeoff (empirically)
  - When we perform LOOCV, we are in effect averaging the outputs of n fitted models, each of which is trained on an almost identical set of observations; therefore, these outputs are highly (positively) correlated with each other
  - Since the mean of many highly correlated quantities has higher variance than does the mean of many quantities that are not as highly correlated, the test error estimate resulting from LOOCV tends to have a higher variance than does the test error estimate resulting from k-fold CV

Cross-validation for classification problems

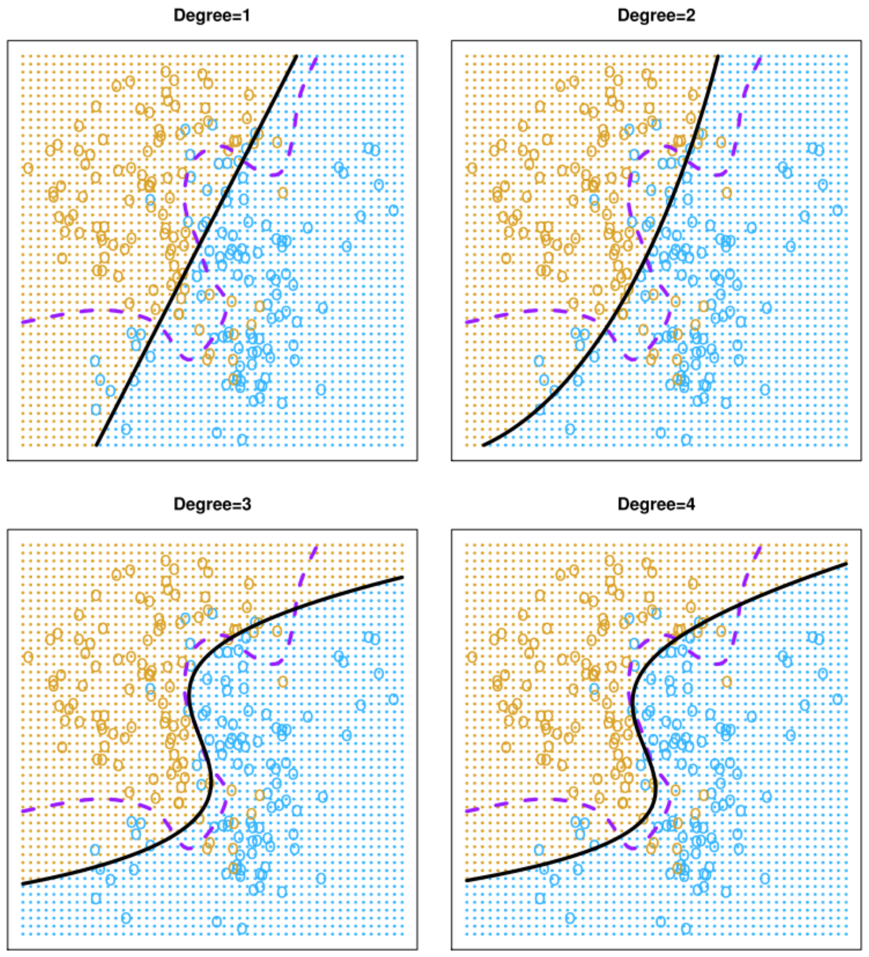
- We divide the data into k roughly equal-sized parts $C_1, C_2, \dots C_k$, where C_i denotes the indices of the observations in part i. There are n_i observations in part i
- Compute

$$CV_{(k)} = \sum_{i=1}^{k} \frac{n_i}{n} Err_i$$

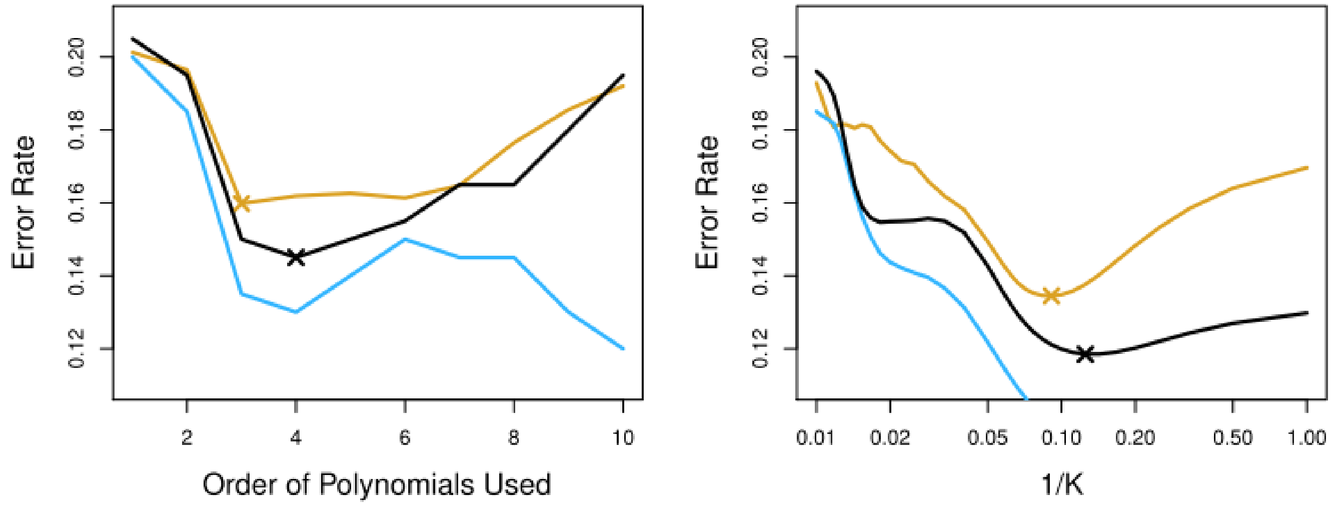
where $Err_i = \sum_{j \in C_i} I(y_j \neq \hat{y}_j)/n_i$

Cross-validation for classification problems

- For the simulation dataset in Chapter 2 and use logistic regression



Brown: True test error
Blue: Training error
Black: 10-Fold CV's error

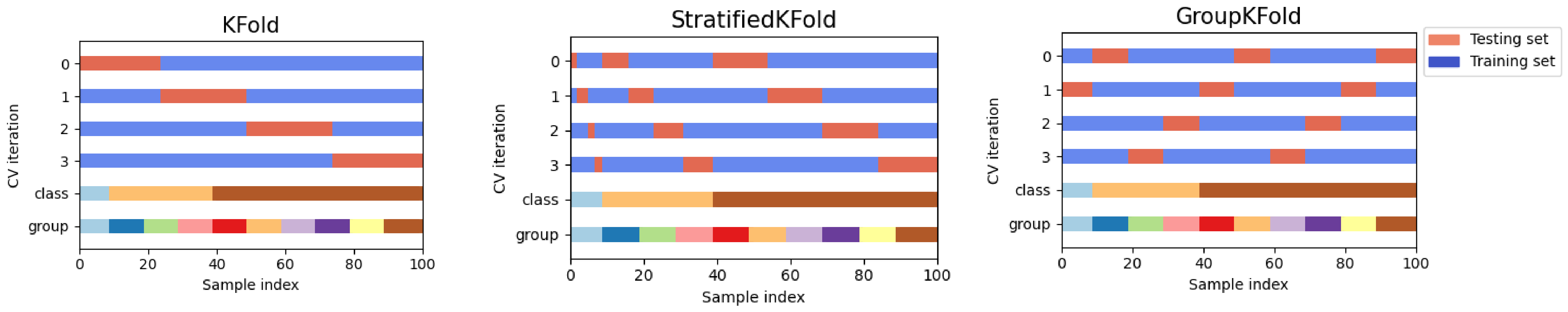


Cross-validation: right and wrong

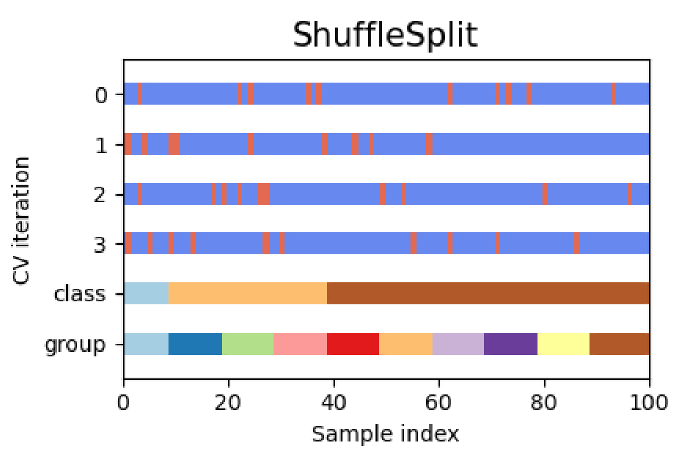
- Consider a simple classifier applied to some two-class data:
  1. Starting with 5,000 predictors and 50 samples, find the 100 predictors having the largest correlation with the class labels
  2. We then apply a classifier, such as logistic regression, using only these 100 predictors
  - How do we estimate the test set performance of this classifier?
  - Can we apply cross-validation in step 2, forgetting about step 1?
  - This would ignore the fact that in Step 1, the procedure has already seen all the labels of the data and made use of them. <u>This is a form of training and must be included in the validation process</u>
- If that preprocessing depends on the data (e.g. standardization, one-hot encoding), then you should <u>calculate it on your training data</u> and then use the parameters from that calculation to apply it to your validation and test data

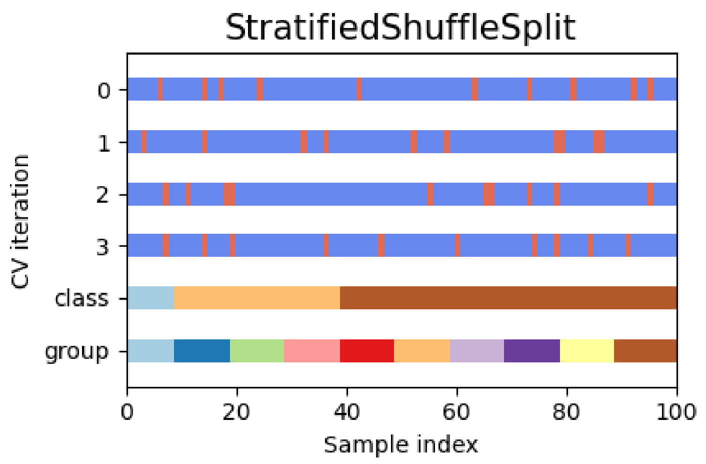
More about cross-validation

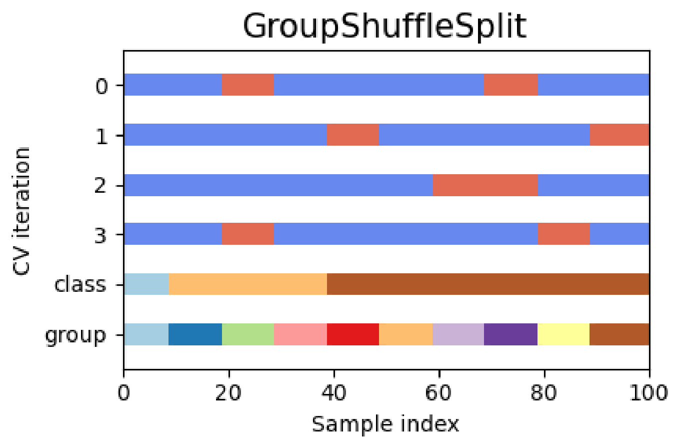
- A note on shuffling



More about cross-validation

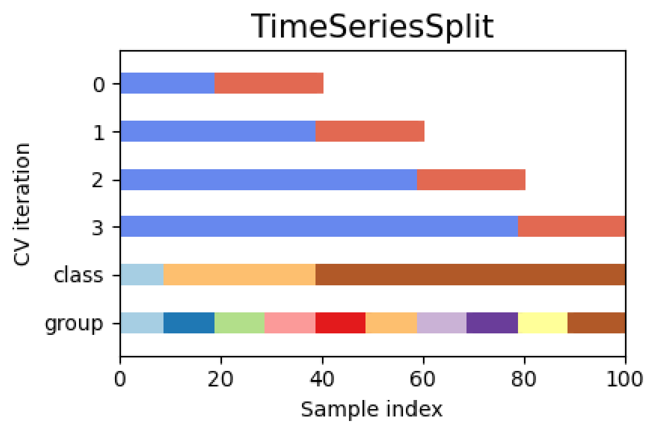






Finer control on validation set size

- Some discussion about time series data

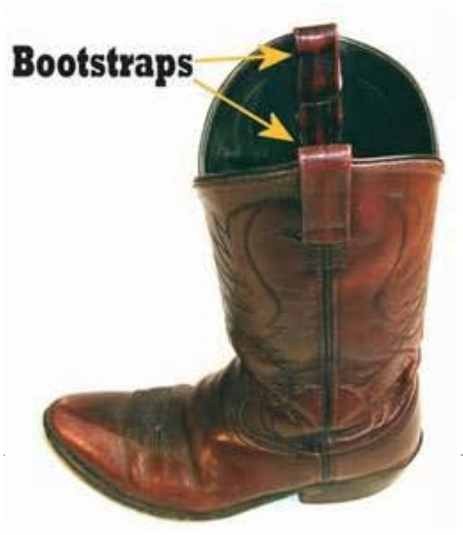


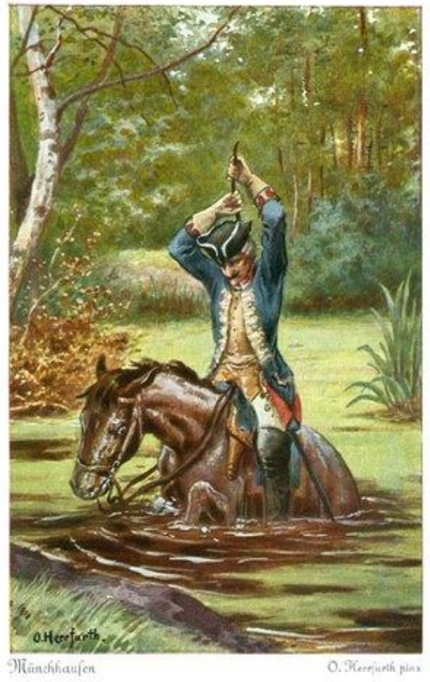
3. The Bootstrap

- The bootstrap is a flexible and powerful statistical tool that can be used to quantify the uncertainty associated with a given estimator or statistical learning method
  - For example, it can provide an estimate of the standard error of a coefficient, or a confidence interval for that coefficient

Where does the name came from?

- The use of the term bootstrap derives from the phrase to pull oneself up by one's bootstraps, widely thought to be based on one of the eighteenth century "*The Surprising Adventures of Baron Munchausen*" by Rudolph Erich Raspe:
  - *The Baron had fallen to the bottom of a deep lake. Just when it looked like all was lost, he thought to pick himself up by his own bootstraps*
- Bootstrapping usually refers to the starting of a self-starting process that is supposed to proceed without external input





https://mentallyagile.com/blog/2022/5/23/bootstrap-absurdity

A simple example

- Suppose that we wish to invest a fixed sum of money in two financial assets that yield returns of X and Y, respectively
  - We will invest a fraction α of our money in X, and will invest the remaining $1-\alpha$ in Y
  - We wish to choose α to minimize the total risk, or variance, of our investment. In other words, we want to minimize $Var(\alpha X + (1-\alpha)Y)$
- One can show that the value that minimizes the risk is given by (Exercise 1)

$$\alpha = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}}$$

where $\sigma_X^2 = Var(X), \sigma_Y^2 = Var(Y)$, and $\sigma_{XY} = Cov(X,Y)$

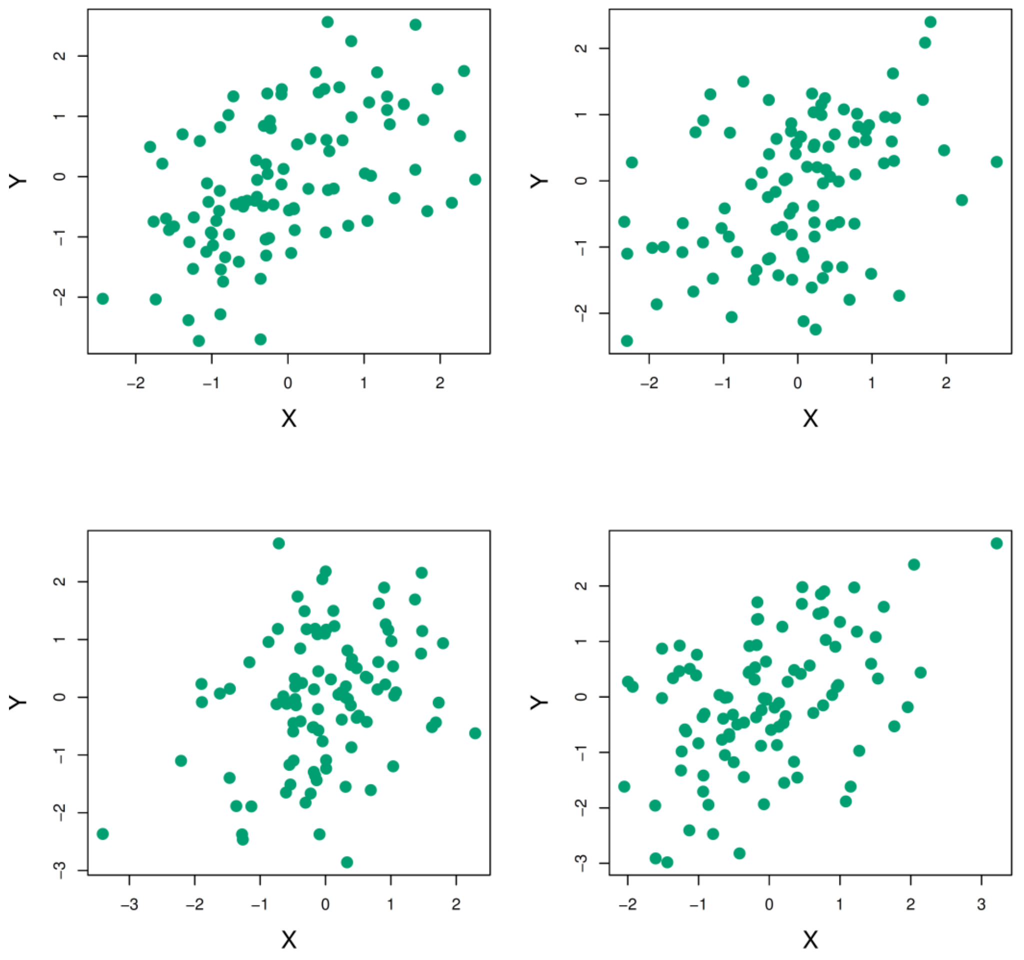
Example continued

- But the values of σ_X^2, σ_Y^2 and σ_{XY} are unknown
- We can compute estimates for these quantities, $\hat{\sigma}_X^2, \hat{\sigma}_Y^2, \hat{\sigma}_{XY}$ using a data set that contains measurements for X and Y
- We can then estimate the value of α that minimizes the variance of our investment using

$$\hat{\alpha} = \frac{\hat{\sigma}_Y^2 - \hat{\sigma}_{XY}}{\hat{\sigma}_X^2 + \hat{\sigma}_Y^2 - 2\hat{\sigma}_{XY}}$$

Example continued

- Each panel displays 100 simulated returns for investments X and Y
  - To estimate the standard deviation of $\hat{\alpha}$, we repeated the process of simulating 100 paired observations of X and Y, and estimating α 1,000 times
  - From left to right and top to bottom, the resulting estimates for α are $0.576, 0.532, 0.657$, and 0.651
  - We thereby obtained 1,000 estimates for α, which we can call $\hat{\alpha}_1, \hat{\alpha}_2, \dots, \hat{\alpha}_{1000}$



Example continued

- For these simulations, the parameters were set to $\sigma_X^2 = 1,\ \sigma_Y^2 = 1.25$ and $\sigma_{XY} = 0.5$, and so we know that the true value of α is 0.6
  - The mean over all 1,000 estimates for α is

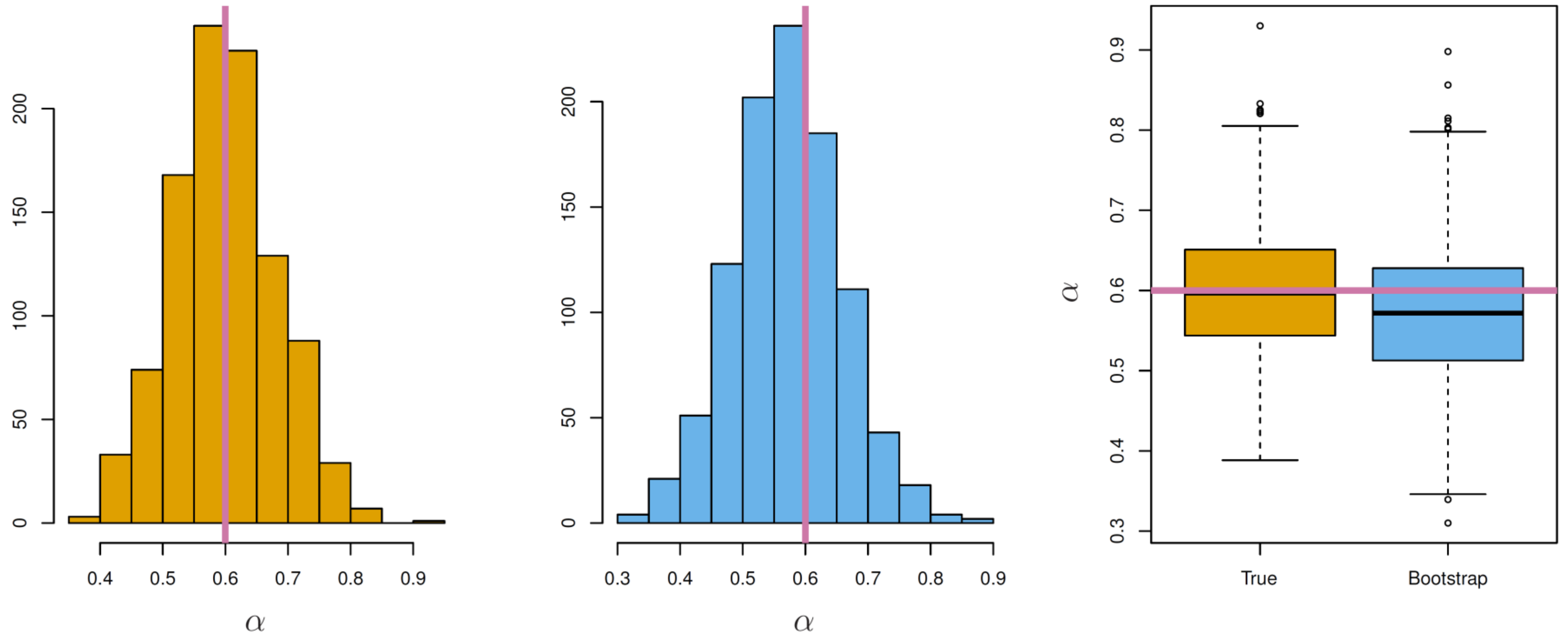
$$\bar{\alpha} = \frac{1}{1000}\sum_{r=1}^{1000} \hat{\alpha}_r = 0.5996$$

very close to $\alpha = 0.6$, and the standard deviation of the estimates is

$$\sqrt{\frac{1}{1000-1}\sum_{r=1}^{1000} (\hat{\alpha}_r - \bar{\alpha})^2} = 0.083$$

This gives us a very good idea of the accuracy of $\hat{\alpha}$: $SE(\hat{\alpha}) \approx 0.083$

Results

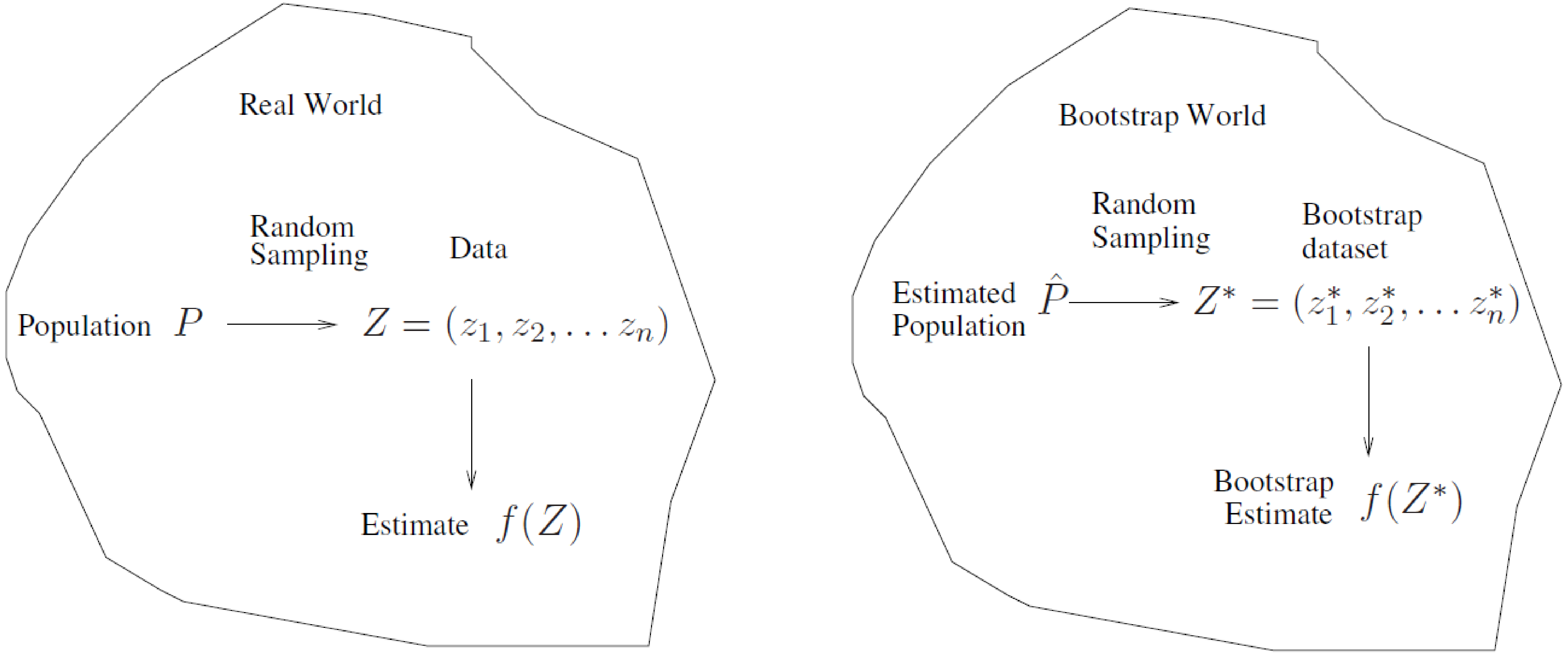


- Left: A histogram of the estimates of α obtained by generating 1,000 simulated data sets from the true population. Center: A histogram of the estimates of α obtained from 1,000 bootstrap samples from a single data set. Right: The estimates α of displayed in the left and center panels are shown as boxplots. In each panel, the pink line indicates the true value of α

Now back to the real world

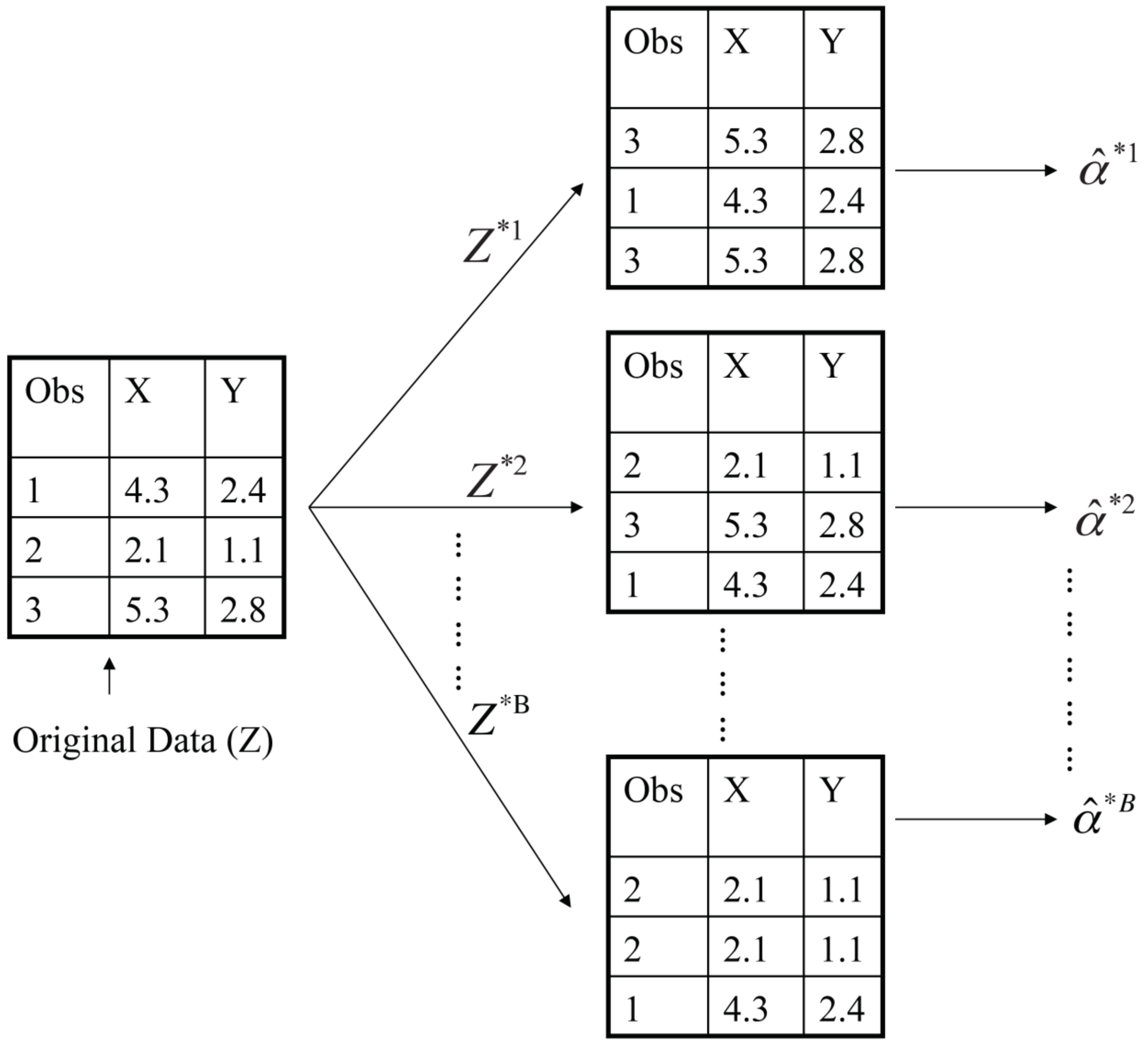
- The procedure outlined above cannot be applied, because for real data we cannot generate new samples from the original population!
  - However, the bootstrap approach allows us to use a computer to mimic the process of obtaining new data sets, so that we can estimate the variability of our estimate without generating additional samples
  - Rather than repeatedly obtaining independent data sets from the population, we instead obtain distinct data sets by repeatedly sampling observations from the original data set with replacement
  - Each of these "bootstrap datasets" is the same size as our original dataset. As a result, some observations may appear more than once in a given bootstrap data set and some not at all

A general picture for the bootstrap


Real World
Random Sampling
Data
Population P
$Z = (z_1, z_2, \ldots z_n)$
Estimate $f(Z)$
Bootstrap World
Random Sampling
Bootstrap dataset
Estimated Population $\hat{P}$
$Z^* = (z_1^*, z_2^*, \ldots z_n^*)$
Bootstrap Estimate $f(Z^*)$

Example with just 3 observations

- A graphical illustration of the bootstrap approach on a small sample containing $n = 3$ observations. Each bootstrap data set contains n observations, *sampled with replacement* from the original data set. Each bootstrap data set is used to obtain an estimate of α



Obs	X	Y
1	4.3	2.4
2	2.1	1.1
3	5.3	2.8

Obs	X	Y
3	5.3	2.8
1	4.3	2.4
3	5.3	2.8

Obs	X	Y
2	2.1	1.1
3	5.3	2.8
1	4.3	2.4

Obs	X	Y
2	2.1	1.1
2	2.1	1.1
1	4.3	2.4

Bootstrap of α

- Denoting the first bootstrap data set by Z^{*1}, we use Z^{*1} to produce a new bootstrap estimate for α, which we call $\hat{\alpha}^{*1}$
  - This procedure is repeated B times for some large value of B (say 100 or 1000), in order to produce B different bootstrap data sets, $Z^{*1}, Z^{*2}, \ldots, Z^{*B}$ and B corresponding estimates, $\hat{\alpha}^{*1}, \hat{\alpha}^{*2}, \ldots, \hat{\alpha}^{*B}$
  - We estimate the standard error of these bootstrap estimates using the formula

$$SE_B(\hat{\alpha}) = \sqrt{\frac{1}{B-1}\sum_{r=1}^{B}\left(\hat{\alpha}^{*r} - \frac{1}{B}\sum_{r'=1}^{B}\hat{\alpha}^{*r'}\right)^2}$$

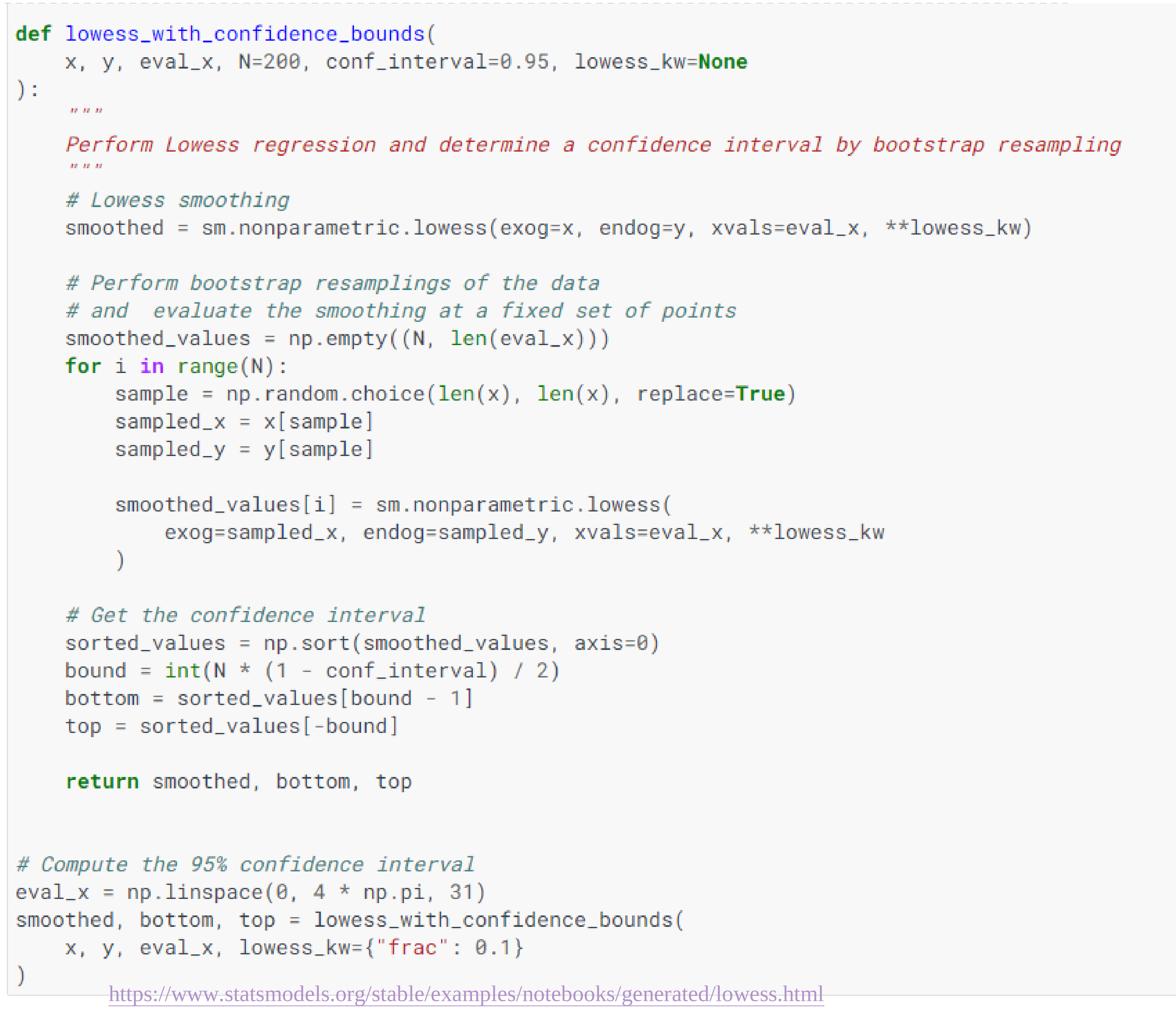
- This serves as an estimate of the standard error of $\hat{\alpha}$ estimated from the original data set. See center and right panels of Figure in the previous slides. Bootstrap results are in blue. For this example $SE_B(\hat{\alpha}) = 0.087$

Other uses of the bootstrap

- Primarily used to obtain standard errors of an estimate
  - Also provides approximate confidence intervals for a population parameter. For example, looking at the histogram in the middle panel of the Figure on slide 28, the 5% and 95% quantiles of the 1,000 values are (0.43; 0.72)
  - This represents an approximate 90% confidence interval for the true α. This interval is called a *Bootstrap Percentile confidence interval*. It is the simplest method (among many approaches) for obtaining a confidence interval from the bootstrap

Bootstrap for confidence interval

1. Generate n “bootstrap sample” data points x_i^*, y_i^*
2. Fit linear regression using x_i^*, y_i^*
3. Evaluate the regression line on fix x-grid
4. Repeat step 1-3 for B times and collect the values in step 3
5. For each point in the x-grid, calculate the confidence interval using collected values

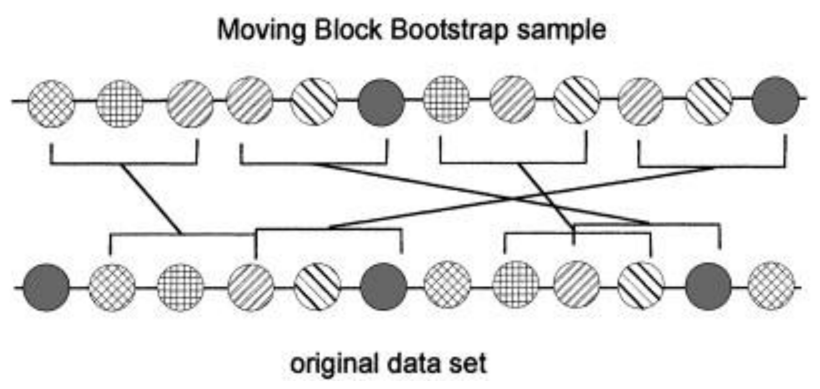


```python
def lowess_with_confidence_bounds(
    x, y, eval_x, N=200, conf_interval=0.95, lowess_kw=None
):
    """
    Perform Lowess regression and determine a confidence interval by bootstrap resampling
    """
    # Lowess smoothing
    smoothed = sm.nonparametric.lowess(exog=x, endog=y, xvals=eval_x, **lowess_kw)

    # Perform bootstrap resamplings of the data
    # and  evaluate the smoothing at a fixed set of points
    smoothed_values = np.empty((N, len(eval_x)))
    for i in range(N):
        sample = np.random.choice(len(x), len(x), replace=True)
        sampled_x = x[sample]
        sampled_y = y[sample]

        smoothed_values[i] = sm.nonparametric.lowess(
            exog=sampled_x, endog=sampled_y, xvals=eval_x, **lowess_kw
        )

    # Get the confidence interval
    sorted_values = np.sort(smoothed_values, axis=0)
    bound = int(N * (1 - conf_interval) / 2)
    bottom = sorted_values[bound - 1]
    top = sorted_values[-bound]

    return smoothed, bottom, top


# Compute the 95% confidence interval
eval_x = np.linspace(0, 4 * np.pi, 31)
smoothed, bottom, top = lowess_with_confidence_bounds(
    x, y, eval_x, lowess_kw={"frac": 0.1}
)
```

https://www.statsmodels.org/stable/examples/notebooks/generated/lowess.html

The bootstrap in general

- In more complex data situations, figuring out the appropriate way to generate bootstrap samples can require some thought
  - For example, if the data is a time series, we can't simply sample the observations with replacement (Not i.i.d)
  - We can instead create blocks of consecutive observations, and sample those with replacements. Then we paste together sampled blocks to obtain a bootstrap dataset



https://www.sciencedirect.com/science/article/pii/S0003267000008503

Can the bootstrap estimate prediction error?

- In cross-validation, each of the k validation folds is distinct from the other $k-1$ folds used for training: there is no overlap. This is crucial for its success
  - To estimate prediction error using the bootstrap, we could think about using each bootstrap dataset as our training sample and the original sample as our validation sample
  - But each bootstrap sample has a significant overlap with the original data. About two-thirds of the original data points appear in each bootstrap sample (Exercise 2)
    - This will cause the bootstrap to seriously underestimate the true prediction error
- Can partly fix this problem by only using predictions for those observations that did not (by chance) occur in the current bootstrap sample
  - But the method gets complicated, and in the end, cross-validation provides a simpler, more attractive approach for estimating prediction error

Appendix

Other resampling methods

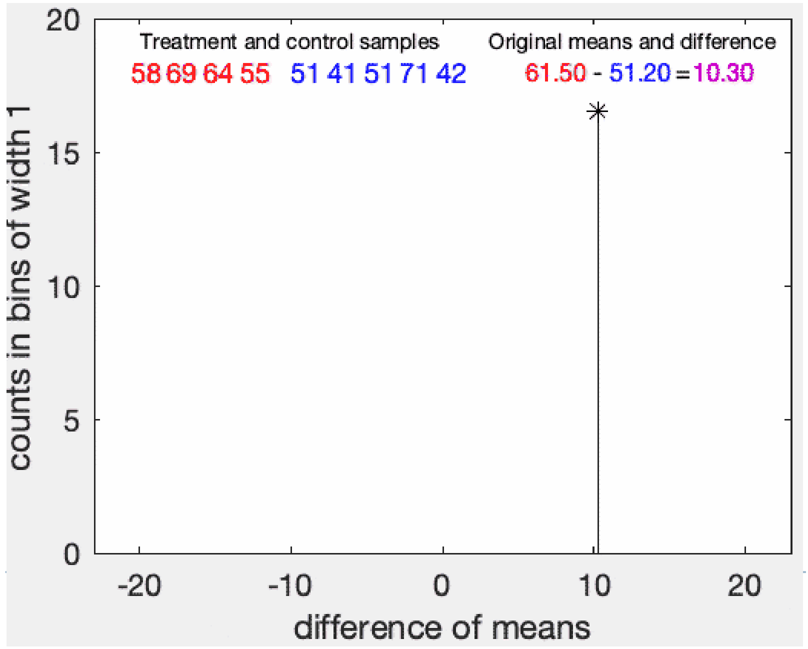
- Bootstrap for prediction interval
  - https://www.saattrupdan.com/2020-03-01-bootstrap-prediction
  - https://stats.stackexchange.com/questions/226565/bootstrap-prediction-interval
- Bootstrap-based test vs permutation test
  - https://en.wikipedia.org/wiki/Permutation_test
- Jackknife
  - https://en.wikipedia.org/wiki/Jackknife_resampling
  - https://stats.stackexchange.com/questions/249333/comparison-of-the-jacknife-vs-the-bootstrap

The Bootstrap versus Permutation tests

- The bootstrap samples from the estimated population, and uses the results to estimate standard errors and confidence intervals
  - The bootstrap can be used to test a null hypothesis in simple situations. Eg if $\theta = 0$ is the null hypothesis, we check whether the confidence interval for θ contains zero.
  - There's no real advantage over permutations. The primary difference is that while bootstrap analyses typically seek to quantify the sampling distribution of some statistic computed from the data, permutation analyses typically seek to *quantify the null distribution*
- Permutation methods sample from an estimated null distribution for the data, and use this to estimate p-values for hypothesis tests

Permutation Test

1. Define test statistics T
2. Calculate the alternative hypothesis T_a using original observation data
3. Permute the observation data N times, Compute $T_1, T_2, \dots, T_N$ for each permutation. This forms the distribution under null hypothesis
4. Calculate the percentage of $T_i \geq |T_a|$ to obtain the p-value



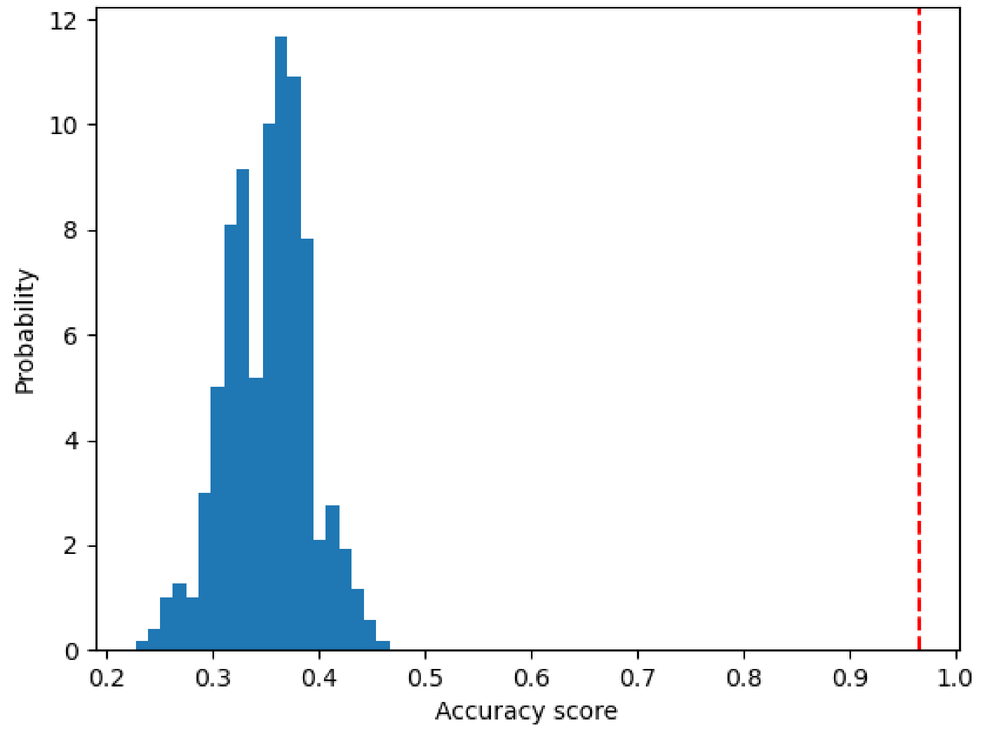
https://en.wikipedia.org/wiki/Permutation_test

Permutation Test - Classifier

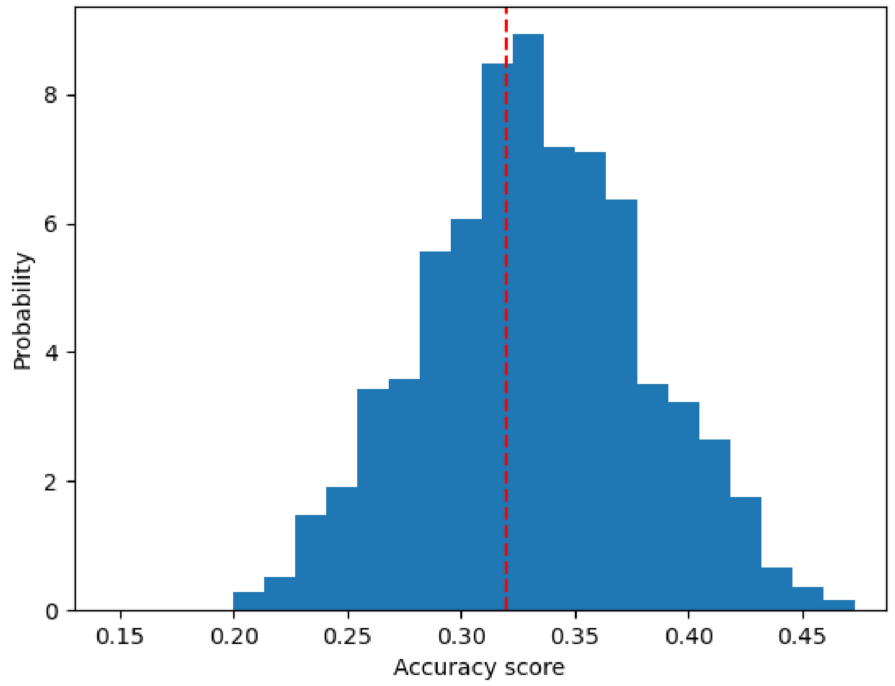
- The null hypothesis in this test is that the classifier fails to leverage any statistical dependency between the features and the labels to make correct predictions on left-out data
  1. Generates a null distribution by calculating $n_{permutations}$ different permutations of the data. In each permutation, the labels are randomly shuffled, thereby removing any dependency between the features and the labels
  2. The p-value output is the fraction of permutations for which the average cross-validation score obtained by the model is better than the cross-validation score obtained by the model using the original data
  3. A low p-value provides evidence that the dataset contains real dependency between features and labels and the classifier was able to utilize this to obtain good results. A high p-value could be due to a lack of dependency between features and labels (there is no difference in feature values between the classes) or because the classifier was not able to use the dependency in the data

Permutation Test - Classifier

Original data – Low *p*-value



Random data– High *p*-value


Probability
Accuracy score