Resampling Methods

Business Analytics Practice
Winter Term 2015/16
Stefan Feuerriegel

Today's Lecture

Objectives

1. Distinguishing between explanatory and predictive power
2. Learning the reasoning behind the validation set approach
3. Understanding cross-validation and the bootstrap for resampling
4. Tuning models to improve the predictive performance

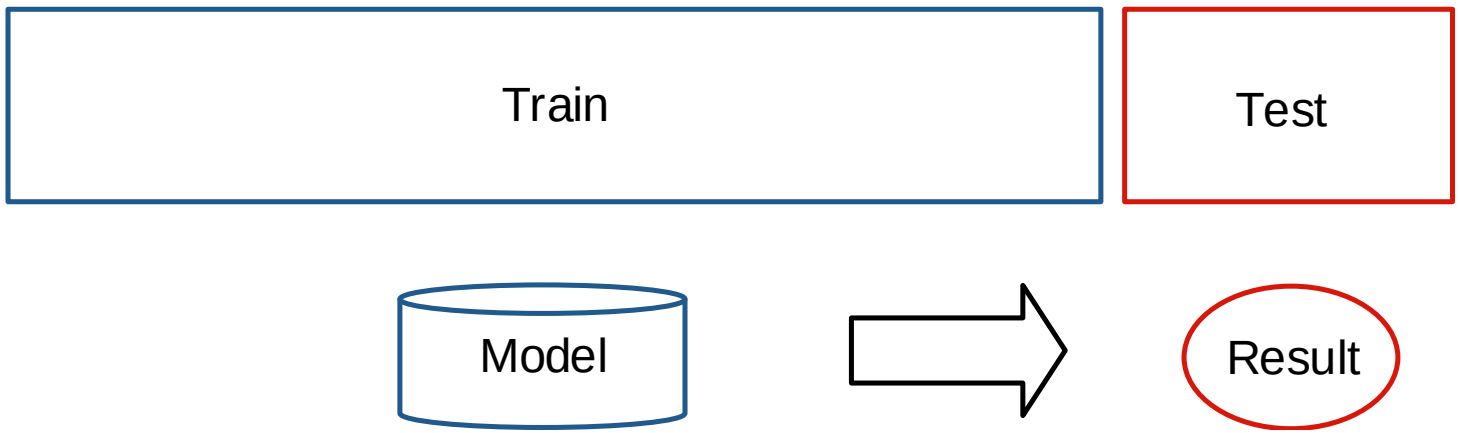
Outline

1. Validation Set Approach
2. Cross-Validation
3. Model Tuning
4. Bootstrapping
5. Wrap-Up

Outline

Training and Test Set

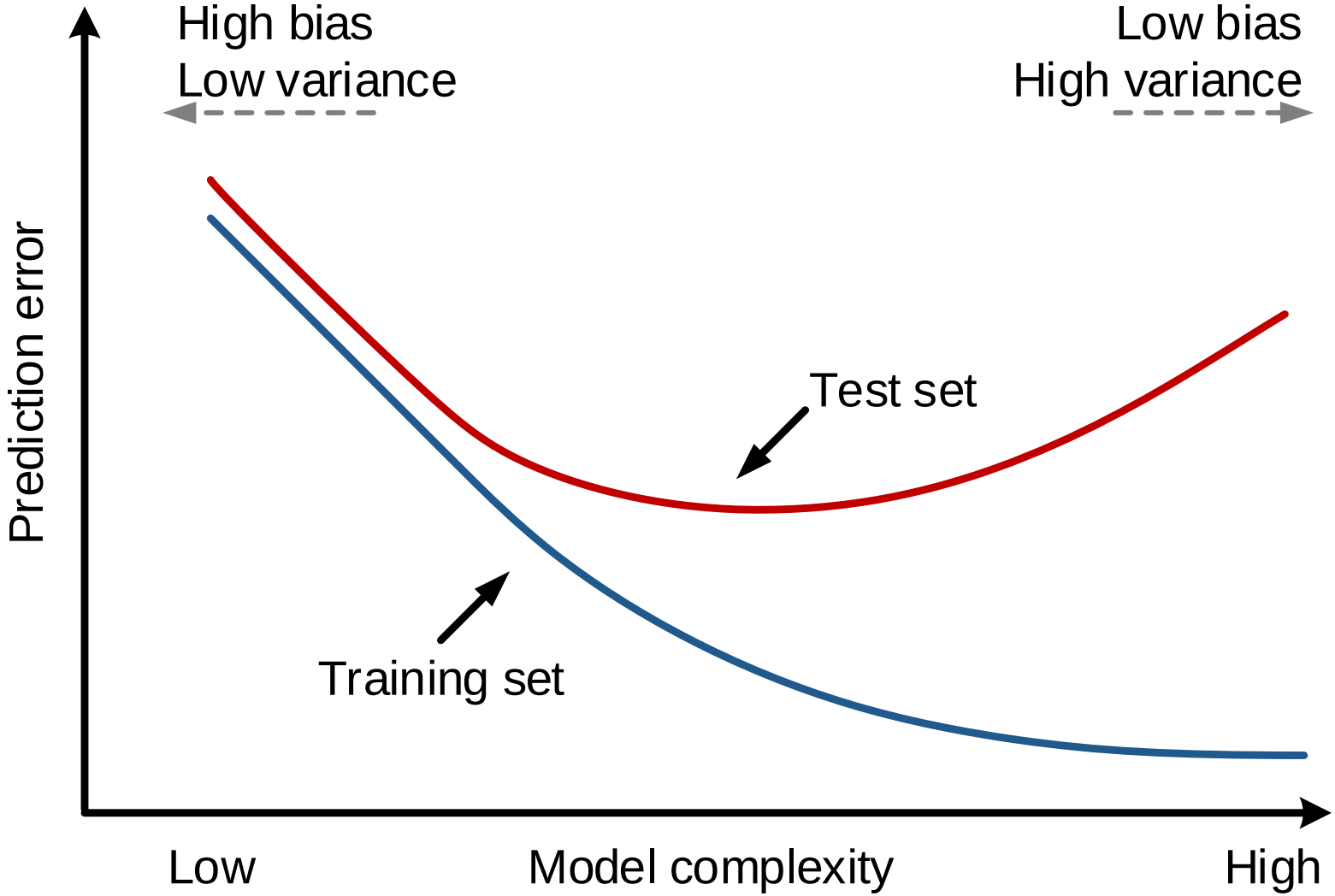
- Datasets in machine learning are usually split into disjunct sets for training and testing
  1. **Training set** is used to fit and calibrate the model parameters
  2. **Test set** is used to measure the predictive performance on unseen data
- Each measures a different error, i. e. the training and test error
- Rule-of-thumb: 80 % for training and 20 % for testing (or 90 % vs. 10 %)



Training vs. Test Error

- Training error results from applying the model to the training data
- Test error is the average error when predicting on unseen observations
- Alternative terms refer to in-sample and out-of-sample performance
- The training error underestimates the test error, since it is usually substantially smaller

Training vs. Test Error



Remedies to Overfitting

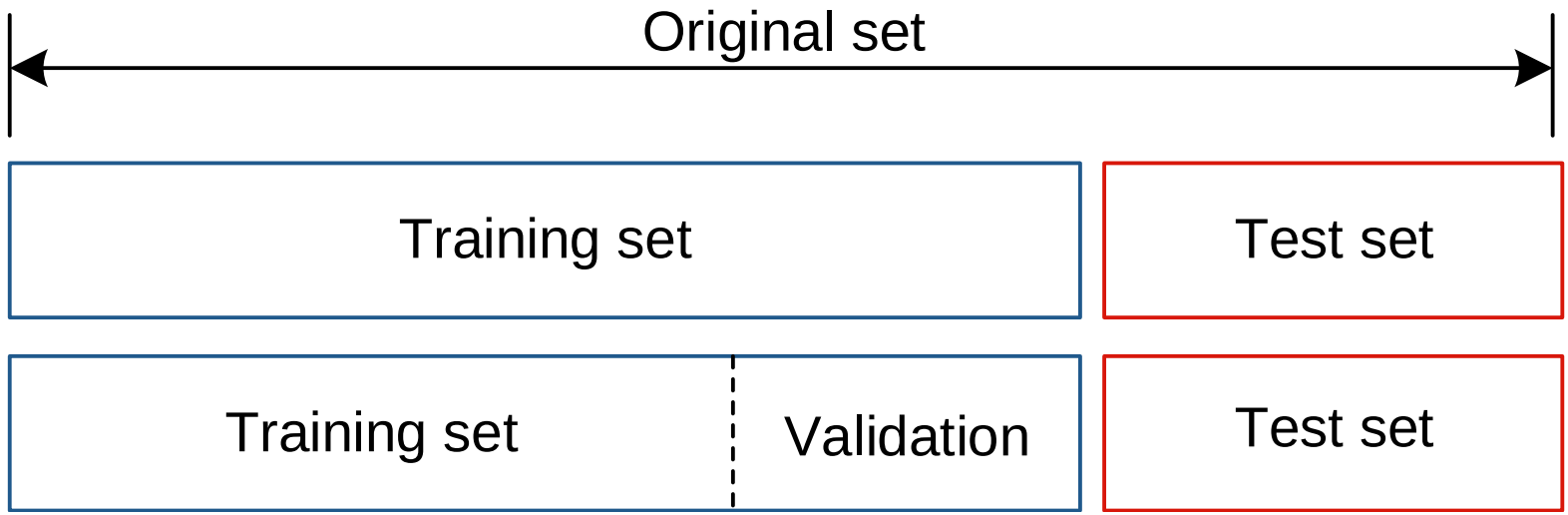
1. Using a large training set
   - Easiest solution
   - However, available is often limited
2. Mathematical penalties to prefer simpler models
   - Information criteria (e. g. AIC, BIC) find a trade-off between fit and model complexity
   - For linear models, regularization shrinks coefficients towards zero
   - Pruning of decision trees limits their size
3. Common alternative: use a third set, the hold-out or validation set

Validation Set

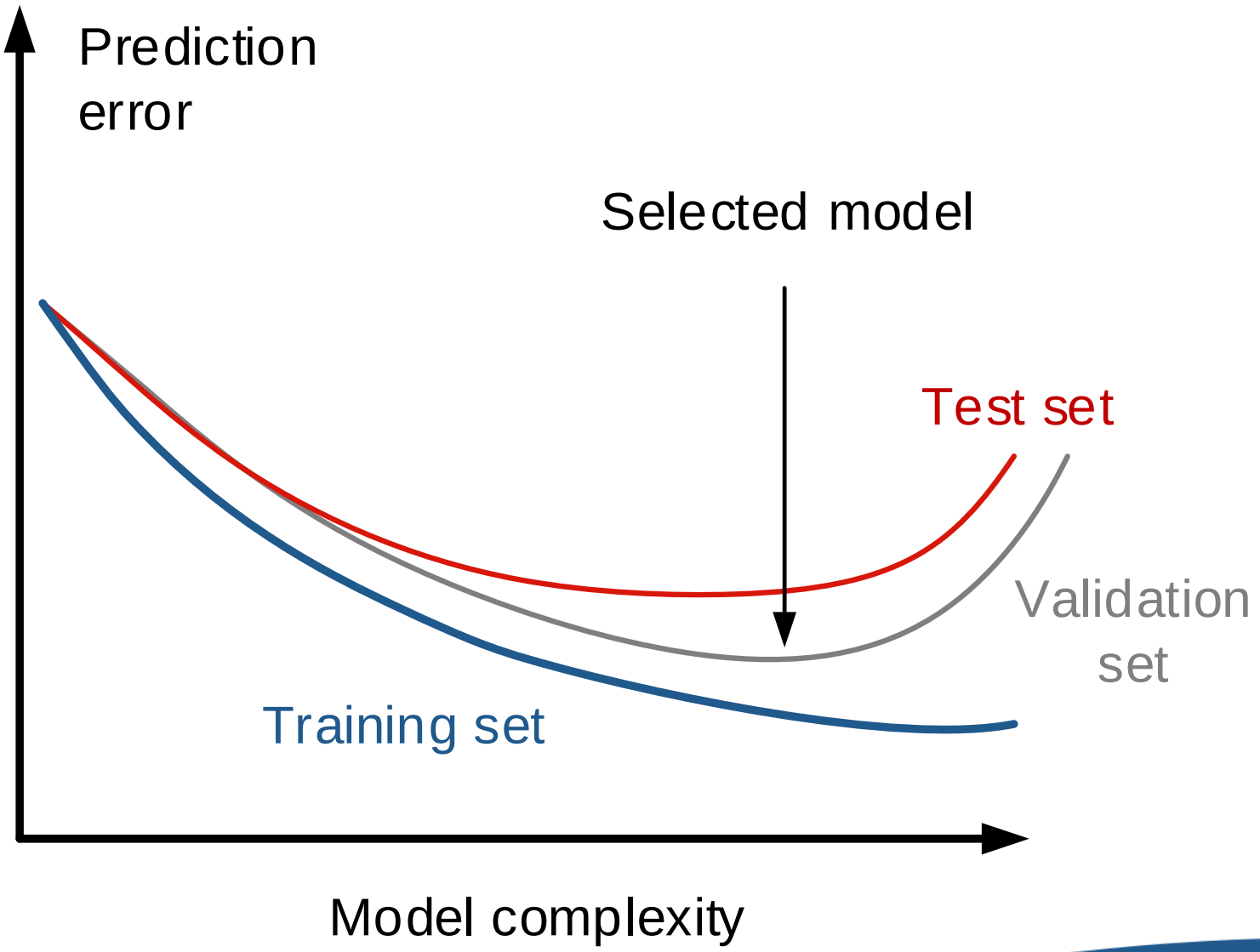
Three-fold split into training, validation and test set

1. Fit model of different complexity to training data
2. Select model based on performance on unseen data from the validation set
3. Measure predictive performance based on the test set

Rule-of-thumb: 60 % for training, 20 % for validation and 20 % for testing



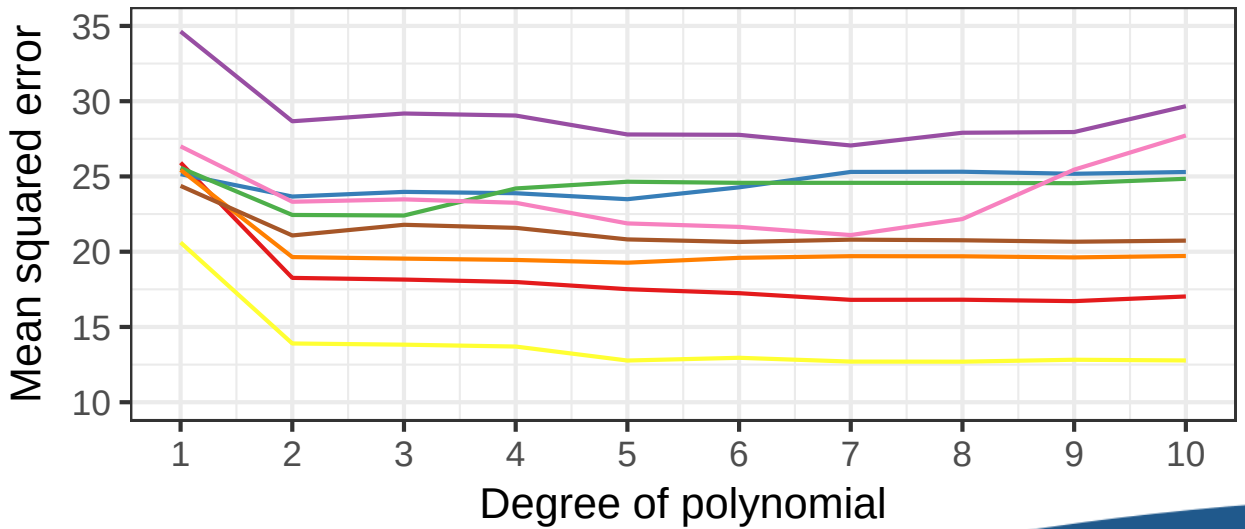
Validation Set



Motivation for Resampling

- Often only limited data is available for measuring performance
- Sometimes performance is subject to the (random) split
  - If splitting is repeated randomly, there might be a high variability across the results
  - Especially relevant for time-dependent or ordered data
  - Making splits random can be of importance here
- Model performance is often inferior the less data is used

Example: Test error across different train/test splits



Resampling

Idea

- Repeatedly draw sub-samples from the given data set
- Then use these splits to fit and assess the model

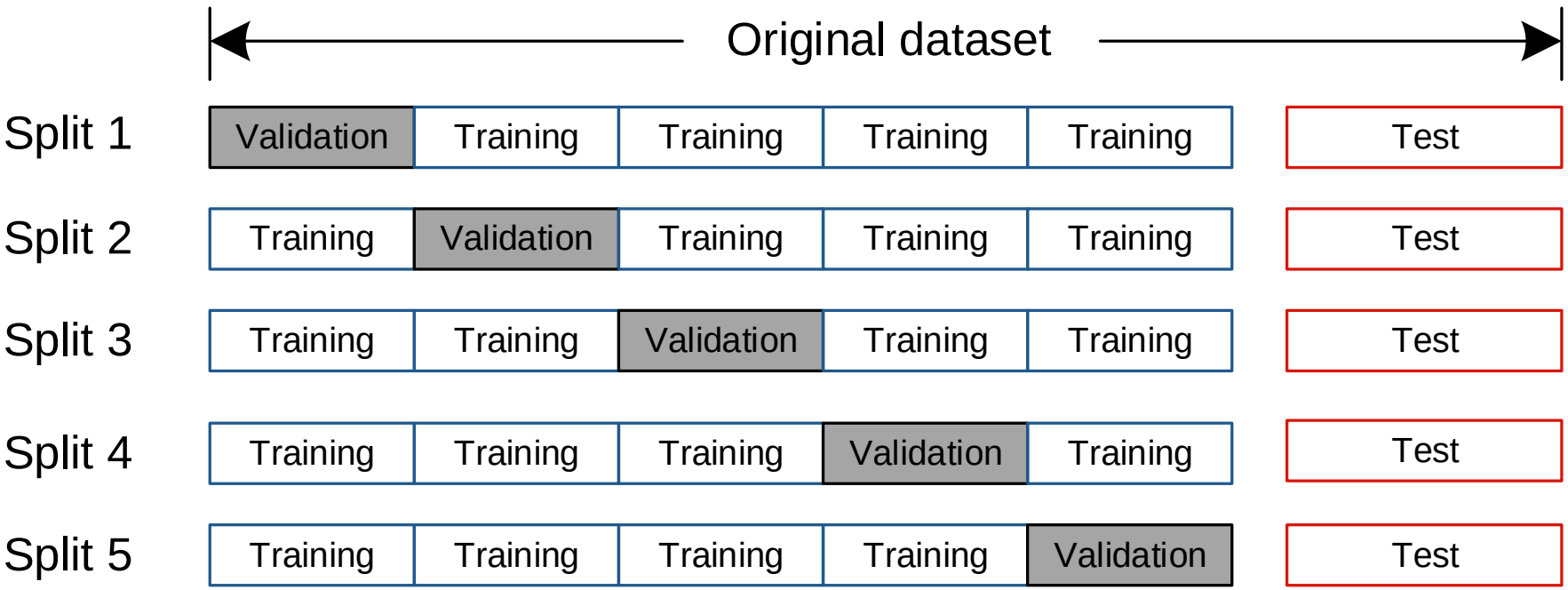
Common methods

1. Cross-validation (CV)
   - Improved approach for estimating the test error
   - k-fold cross-validation
   - Special case: leave-one-out cross-validation (LOOCV)
2. Bootstrap
   - Quantify the uncertainty of an estimator or method
   - Returns standard errors or confidence intervals for a coefficient

Outline

k-Fold Cross-Validation

- Randomly divide dataset into k equal-sized subsamples (i. e.folds)
  1. Fit data using $k - 1$ folds
  2. Make predictions with the left-out k-th fold and measure the performance
  3. Repeat this k times such that each fold becomes a validation set
- Typical choice is $k = 5$ or $k = 10$



Example with $k = 5$

k-Fold Cross-Validation

Algorithm

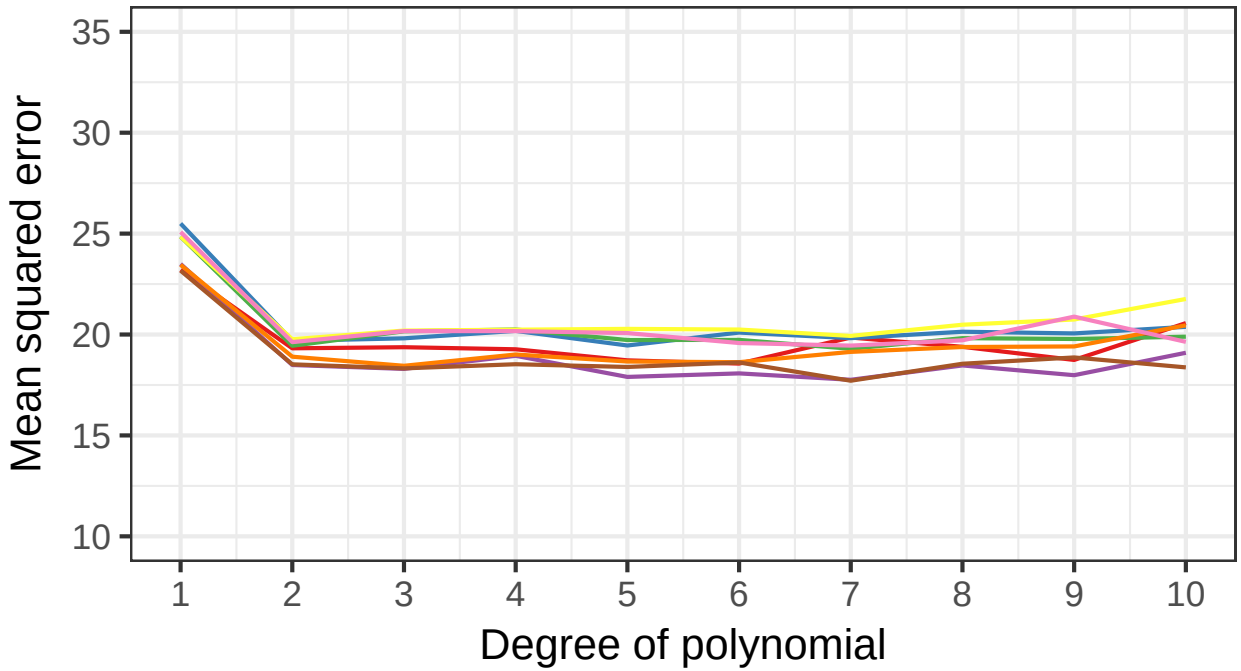
Given pre-defined k and a dataset with n observations

1. Randomly divide data into k-folds $C_1, \ldots, C_k$
2. Let n_i be the number of observations in C_i
   $\Rightarrow n_i \approx \frac{n}{k}$ (or equal if n is multiple of k)
3. For all $i = 1, \ldots, k$, compute the predictive performance $perf_i$ on fold C_i and using the remaining folds for fitting
4. Compute the average performance

$$CV_k = \sum_{i=1}^{k} \frac{n_i}{n} perf_i$$

Cross-Validation

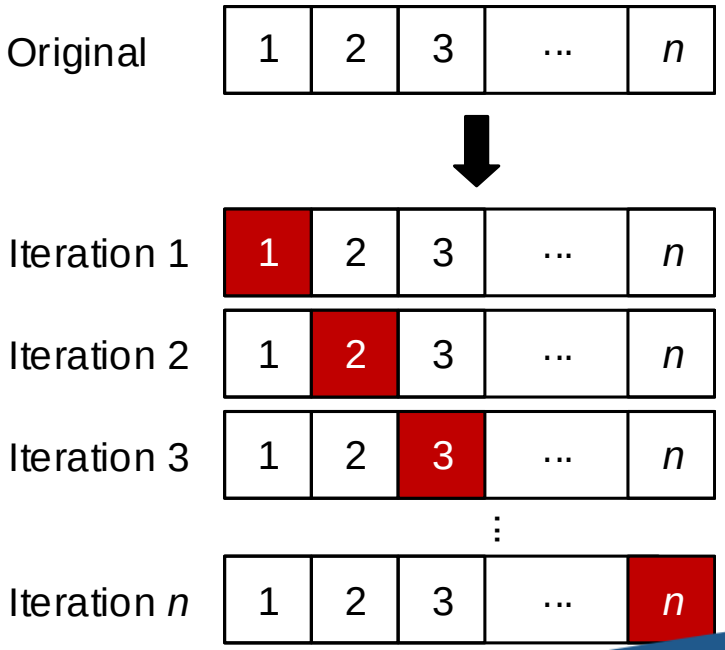
Example: average error on validation sets from 10-fold cross-validation across different random splits



$\rightarrow$ Variance of root mean squared errors on validation set decreased

Leave-One-Out Cross-Validation

- Leave-one-out cross-validation (LOOCV) is a special case with $k = n$
  1. $n - 1$ variables are used for training
  2. The n-th variable is used as a validation set
- Has a low bias since $n - 1$ observations are used for training
- Avoids randomness, but with high computational costs
- Estimated models are highly correlated $\Rightarrow$ average has a high variance
- In practice, better choice is $k = 5$ or $k = 10$



Outline

Model tuning

Problem statement

- Most classifiers have parameters that influence their performance
- Variable selection can additionally overfitting
- How to choose the best parameters and variables?

Examples

Classifier	Parameters
k-nearest neighbor	Neighbors k
Polynomial regression	Order of polynomial
Ridge regression, LASSO	Parameter λ
Support vector machine	Cost parameter C, choice of kernel, ...
Random forest	Number of trees, depth, ...

Solution: use cross-validation during training to tune parameters

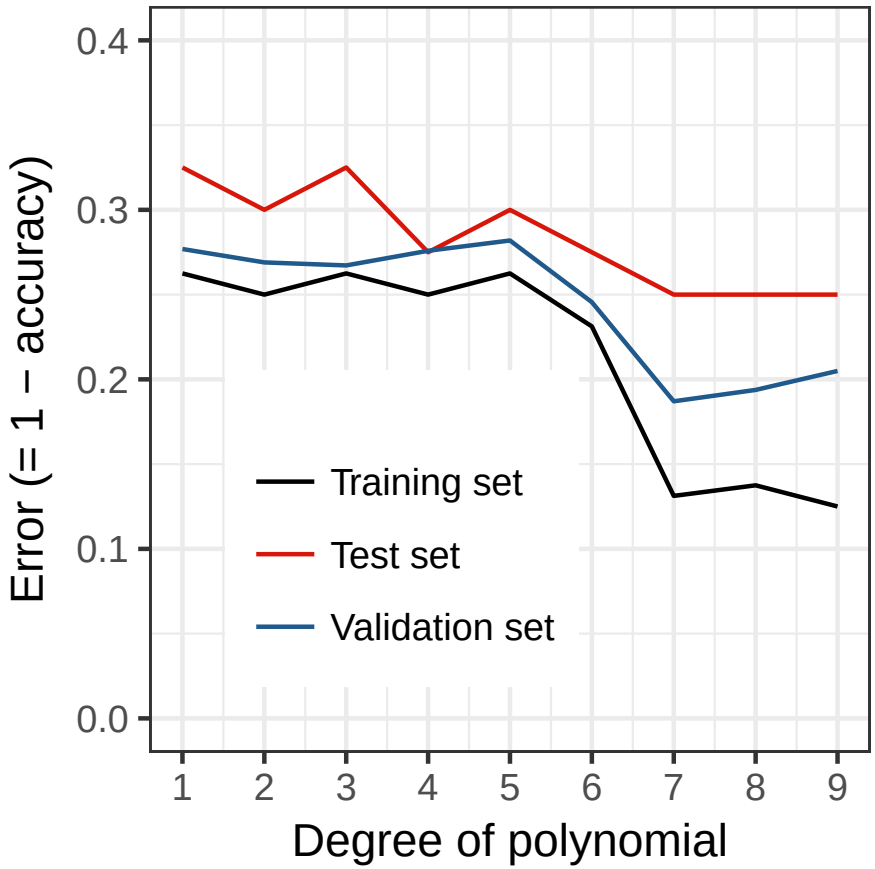
Model Tuning

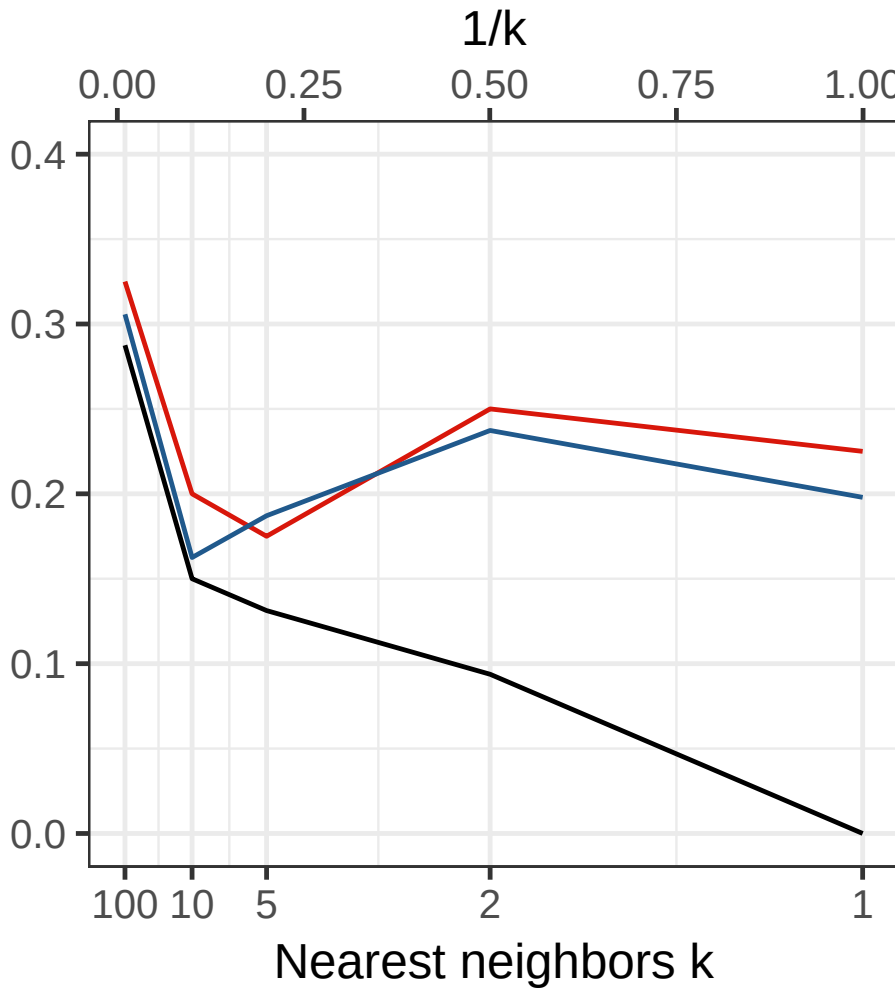
Algorithm

Split dataset into training (incl. validation) and test set
Define sets of model parameters to test
for *each parameter set p* **do**
    for *each cross-validation split i* **do**
        Fit model on the remaining splits
        Make prediction on this split *i*
        Measure performance
    end
    Calculate the average performance across all splits
end
Determine the optimal parameter set p^*
Fit the final with p^* to all training (incl. validation) samples
Measure performance on test set

Model Tuning

Example: predictive performance of polynomial model and *k*-nearest neighbors





Model Tuning with caret

Example with package `caret`

- Load package `caret`

```
library(caret)
```

- Split dataset into training and testing

```
set.seed(0)
data(GermanCredit)
inTrain <- createDataPartition(GermanCredit$Class, p=0.8,
                               list=FALSE)
training <- GermanCredit[inTrain, ]
testing <- GermanCredit[-inTrain, ]
```

- Define configuration for 10-fold cross-validation

```
fitControl <- trainControl("cv", number=10)
```

Model Tuning with caret

- Run random forest with default parameter tuning

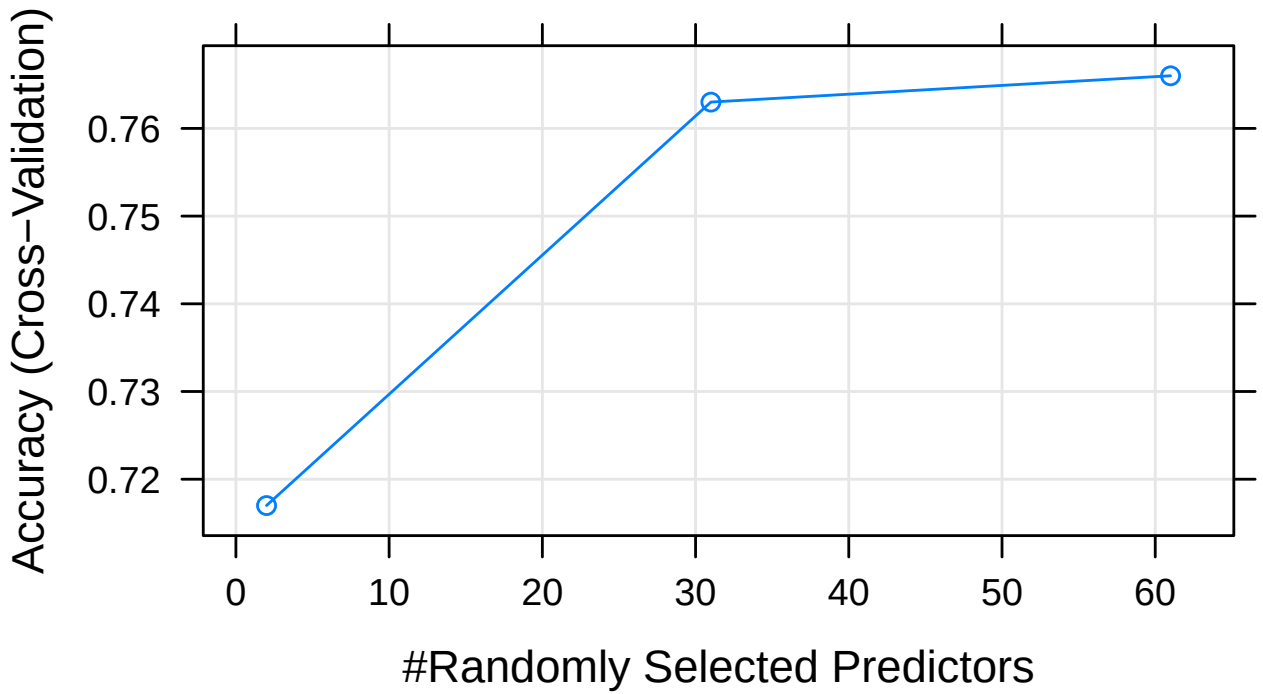
```
set.seed(0)
rf_tuned <- train(Class ~ ., data=GermanCredit,
                  method="rf", trControl=fitControl)
rf_tuned

## Random Forest
##
## 1000 samples
##   61 predictor
##    2 classes: 'Bad', 'Good'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 900, 900, 900, 900, 900, 900, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa
##    2    0.717     0.07864541
##   31    0.763     0.39202731
##   61    0.766     0.40236369
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 61.
```

Note: `mtry` is the number of variables randomly sampled at each split in the tree

Model Tuning with caret

- Plot performance across model parameters

```
plot(rf_tuned)
```



Model Tuning with caret

- Predictive performance on test set

```
pred <- predict(rf_tuned, newdata=testing)
confusionMatrix(pred, testing$Class)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction Bad Good
##       Bad   60    0
##       Good   0  140
##
##                Accuracy : 1
##                  95% CI : (0.9817, 1)
##     No Information Rate : 0.7
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0
##             Specificity : 1.0
##          Pos Pred Value : 1.0
##          Neg Pred Value : 1.0
##              Prevalence : 0.3
##          Detection Rate : 0.3
##    Detection Prevalence : 0.3
##       Balanced Accuracy : 1.0
##
##        'Positive' Class : Bad
##
```

Practice Recommendations for caret

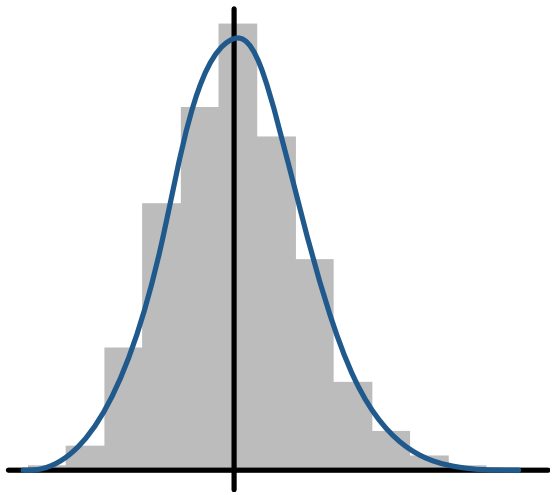
- Instead of `"cv"`, one often uses `"repeatedcv"` which repeats the training procedure several times to avoid non-beneficial splits
- `set.seed(...)` is called prior to the `train(...)` function to make results reproducible
- By default, `caret` tests three values for each parameter
- Alternative searches for parameters can be inserted via argument `tuneGrid`
- See `caret` for details
  - Vignette: `https://cran.r-project.org/web/packages/caret/vignettes/caret.pdf`
  - Manual: `https://topepo.github.io/caret/`

Outline

The Bootstrap

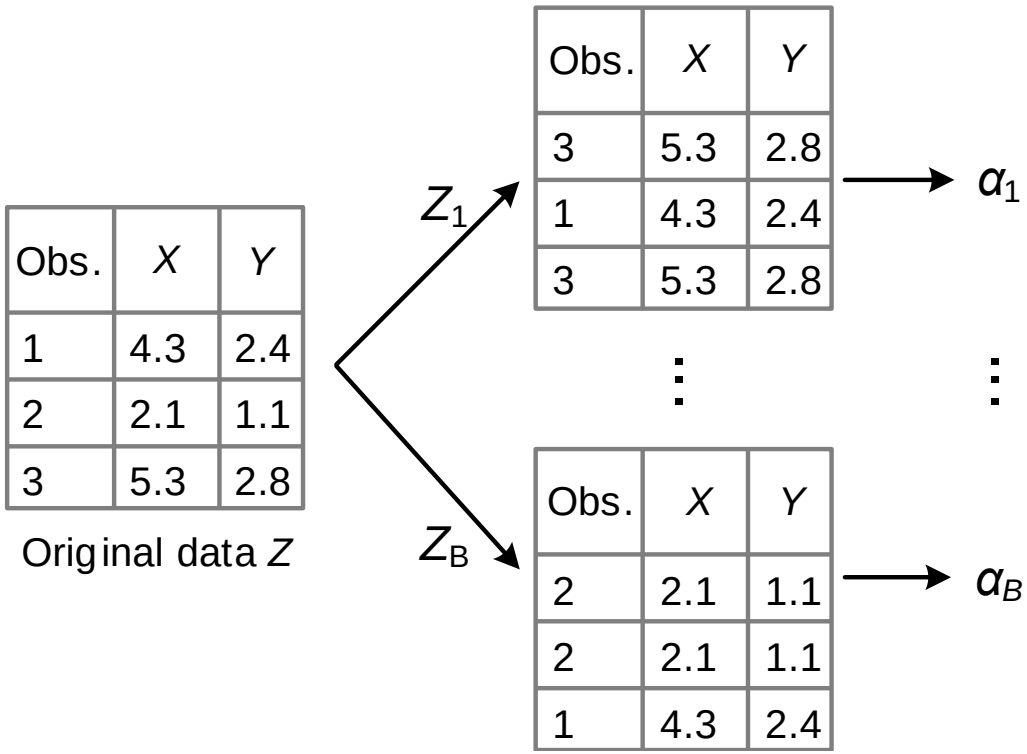
- The bootstrap quantifies the uncertainty of an estimator or a machine learning method
  1. Generate new samples and thus augment the dataset
  2. In practice not possible, but one can mimic this process by constructing artifical data
  3. Approximate the distribution of a desired statistics
     $\rightarrow$ e. g. of coefficient in least squares

- In practice, it returns standard errors or confidence intervals
- Easily applicable, as the model or estimation is not changed but only repeated multiple times



Bootstrapping

The bootstrap creates bootstrap samples by randomly collecting observations from the original dataset with replacement



The overlap between the original dataset and a bootstrap sample will be around two-thirds, one-third are duplicates

Bootstrapping

Algorithm

- Repeat the following steps for $i = 1, \ldots, B$
  1. Randomly sample n observations with replacements from the original dataset in order to produce a bootstrap dataset Z_i
     $\rightarrow$ Implies that the same observation can occur more than once
  2. Estimate statistic α_i with new the bootstrap sample Z_i
- Calculate standard error of the bootstrap estimate

$$\text{SE}_B = \sqrt{\frac{1}{B-1} \sum_{i=1}^{B} (\alpha_i - \overline{\alpha})^2}$$

Bootstrap in R

Implementation

1. Load package `boot`

```r
library(boot)
```

2. Implement a function `f` that computes the statistic of interest

```r
f <- function(data, indices) {
  # select subset with bootstrap samples
  bootstrap_sample <- data[indices, ]

  # estimate model
  model <- estimate(boostrap_sample)

  # extract statistic
  out <- ...

  return(out)
}
```

3. Call the function `boot(data, f, R)` to bootstrap `R` replicates

Bootstrap in R

Example: standard errors of median

- Create data

```
set.seed(0) # initialize seed for random number generator
data <- round(rnorm(100, mean=3, sd=5))
head(data)

## [1]  9  1 10  9  5 -5
```

- Create function to extract statistic

```
f <- function(data, indices) {
  return(median(data[indices]))
}
```

- Bootstrapping with $B = 100$ replicates

```
b <- boot(data, f, 100)
```

```
b
```

```
## Bootstrap Statistics :
##     original  bias    std. error
## t1*        3    -0.1    0.662868
```

Bootstrap Percentiles

Computed confidence intervals are named bootstrap percentiles

- Calculated via `boot.ci(...)`

```
boot.ci(b, type="basic")

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 100 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = b, type = "basic")
##
## Intervals :
## Level      Basic
## 95%   ( 2,  4 )
## Calculations and Intervals on Original Scale
## Some basic intervals may be unstable
```

Bootstrap in R

Example: standard errors in least squares

- Create function to extract statistic

```
f <- function(data, indices) {
  bootstrap_sample <- data[indices, ]
  m <- lm(mpg ~ horsepower, data=bootstrap_sample)
  return(coef(m))
}
```

- Bootstrapping with $B = 100$ replicates

```
library(ISLR)
```

```
data(Auto)
b <- boot(Auto, f, 100)
```

```
b
```

```
## Bootstrap Statistics :
##       original        bias    std. error
## t1* 39.9358610 4.214671e-03 0.797487230
## t2* -0.1578447 1.184515e-05 0.00606
```

Bootstrap in R

- Bootstrap percentiles (argument `index` picks a variable of interest)

```
boot.ci(b, type="basic", index=2)
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 100 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = b, type = "basic", index = 2)
##
## Intervals :
## Level      Basic
## 95%   (-0.1726, -0.1449 )
## Calculations and Intervals on Original Scale
## Some basic intervals may be unstable
```

- Comparison to least squares

```
coef(summary(lm(mpg ~ horsepower, data=Auto)))
```

```
##                Estimate  Std. Error   t value      Pr(>|t|)
## (Intercept) 39.9358610 0.717498656  55.65984 1.220362e-187
## horsepower  -0.1578447 0.006445501 -24.48914  7.031989e-81
```

Bootstrap in R

Example: risk minimization of portfolio

- Given a fixed amount of money and two assets with returns X and Y
- Invest a share α in X and $1-\alpha$ in Y
- Aim: find the allocation that minimizes the total risk

$$\min_{\alpha} \operatorname{Var}(\alpha X + (1-\alpha) Y)$$

- Mathematical solution is

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

for which one can estimate σ_X^2, σ_Y^2 and σ_{XY}^2 from historic returns

Bootstrap in R

- Historic returns of X and Y in dataset `Portfolio`

```
data(Portfolio)
Portfolio[1:4, ]
```

```
##             X          Y
## 1 -0.8952509 -0.2349235
## 2 -1.5624543 -0.8851760
## 3 -0.4170899  0.2718880
## 4  1.0443557 -0.7341975
```

- Define function for extraction

```
f_alpha <- function(data, index) {
  X <- data$X[index]
  Y <- data$Y[index]
  return((var(Y)-cov(X,Y)) / (var(X)+var(Y)-2*cov(X,Y)))
}
```

- Compute optimal α from historic returns

```
f_alpha(Portfolio, 1:nrow(Portfolio))
```

```
## [1] 0.5758321
```

Bootstrap in R

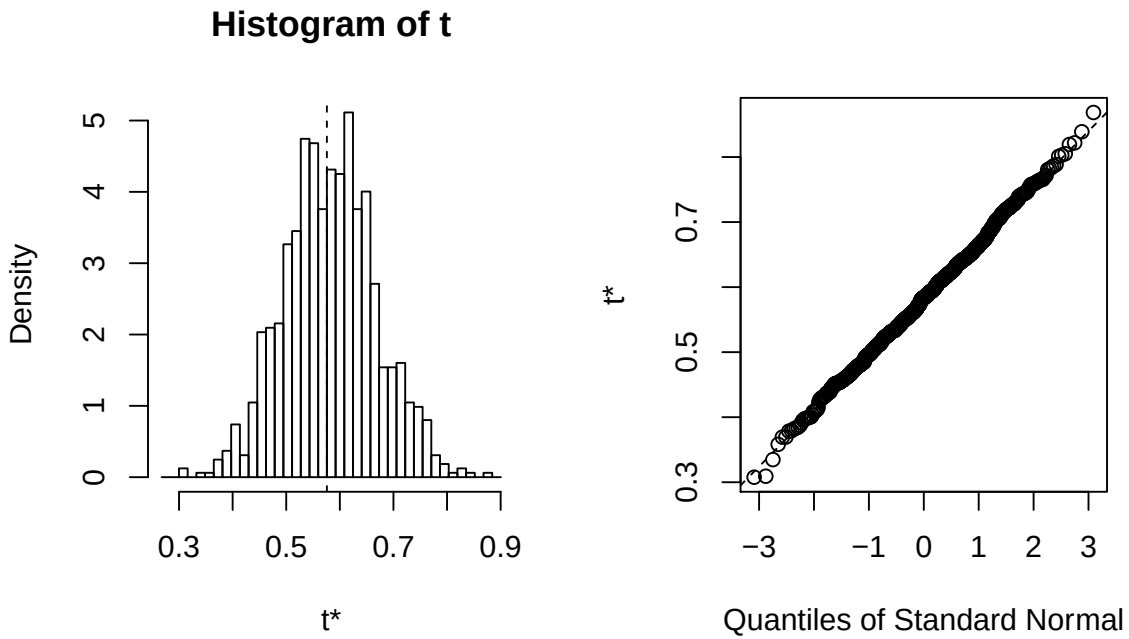
- Run bootstrap (note: true value is $\alpha = 0.6$)

```
b <- boot(Portfolio, f_alpha, R=1000)
```

```
## Bootstrap Statistics :
##      original       bias    std. error
## t1* 0.5758321 0.006197081  0.08594182
```

- Visualize distribution of α as histogram

```
plot(b)
```



Advanced Use of Boostrapping

- **Stratified bootstrap** controls how to pick observations during resampling
  - Ensures certain relationships or group memberships
  - For instance, time series can be split in different chunks of consecutive observations which are then sampled
- **Bayesian bootstrap**
  - Produces similar results
  - But makes different/explicit assumptions regarding distributions
  - Package `bayesboot`
- **Random forest** is a bootstrap of individual decision trees

Outline

Summary

- Resampling methods facilitate statistical inferences based on drawing new observations from an initial sample
  - **Cross-validation:** is an improved strategy to estimate the test error
  - **Bootstrap** quantifies the uncertainty of model parameters
- Especially useful if only a few observations are available
- Disadvantage: high computation time
- Some disciplines even use two-stage cross-validation such that each observation contributes to the test set