Cross-Validation and Model Selection

SYS 6018 | Spring 2023

crossval.pdf

Contents

1 Predictive Performance

1.1 Model Complexity

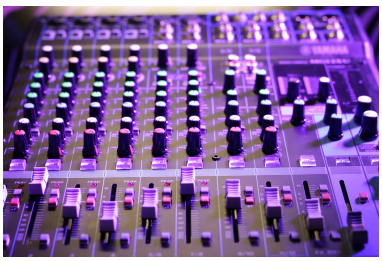
Most model families can be tuned to have a complexity (or edf):

- number of parameters in a linear model (polynomials, interactions, subset selection)
- neighborhood size (the k in knn models)
- penalty λ or constraint (t) for regularized models
- number of trees, tree depth (classification and regression trees, random forest, boosting)
- We will cover many more models with tuning parameters later in the course

Highly adaptable model families can accommodate complex relationships, but easily overemphasize patterns that are not reproducible (e.g. noise or statistical fluctuation)

> Goal is to *tune* the model structure so that it has just enough flexibility (complexity) to capture the reproducible (true) structure, but not too much that it overfits.
>
> - Note: the optimal complexity for a given training data set depends on the sample size.

1.2 Tuning parameters



Tuning parameters are the "control knobs" of a model.

- Some tuning parameters directly control the complexity/flexibility of a model (e.g., the k in knn).
- Other tuning parameters impact the structure or algorithm of a model (e.g., using Euclidean or Manhattan distance in knn)
- Other ancillary aspects like pre-processing, feature engineering, and imputation approaches are not really tuning parameters, per se, but can be treated as such for determining if they help with prediction. E.g.,
  - Some implementations can scale data internally before estimating parameters (e.g., `glmnet()`)
  - Dropping observations with missing data or using median imputation

Given the value of the *tuning parameters*, it is often straightforward to estimate the *model parameters* from the data (e.g., $\hat{\beta}$)

This discussion will focus on the *complexity parameters*, but the same framework will help you choose the best values of the other configurable aspects of a model. We represent the tuning parameters with ω:

- For knn regression, $\omega = k$
- For polynomial regression, $\omega = J$ in the model $\hat{f}(x) = \sum_{j=0}^{J} \beta_j x^j$
- In best subsets regression, $\omega = p$, the total number of features allowed in the model
- In ridge regression, $\omega = \lambda$ in the ridge penalty $\text{Pen}(\beta) = \lambda \sum_{\text{j}=1}^{\text{p}} \beta_{\text{j}}^2$

1.3 Predictive Performance

Because our focus is on predictive performance (not interpretation or inference), the optimal tuning parameter(s), given training data $\mathcal{D}_{\text{train}}$, is the value(s) that minimizes the expected prediction error (PE):

$$\text{EPE}(\omega) = \text{E}[\text{PE}(\omega)] = \text{E}_{\tilde{X},\tilde{Y}}[\ell(\tilde{Y}, \hat{f}_\omega(\tilde{X}))]$$

where:

- $\ell()$ is the loss function (e.g., RSS/MSE, negative logL)
- $\tilde{X}, \tilde{Y}$ are the *test data* drawn from the same distribution (hopefully) as the training data
- $\hat{f}_\omega(x) = f_\omega(x; \hat{\theta}_\omega(\mathcal{D}_{\text{train}}))$ is the prediction function which has been estimated under the tuning parameter ω
  - $\hat{\theta}_\omega(\mathcal{D}_{\text{train}})$ are the **model parameters** estimated from the training data $\mathcal{D}_{\text{train}}$ *given* the tuning value ω.

Ideally, we want to pick tuning parameter(s) ω to minimize EPE

$$\omega^{opt} = \arg\min_{\omega} \ \text{EPE}_{\mathcal{D}_{\text{train}}}(\omega)$$

1.4 Training Error vs. Testing Error

There are a few ways to estimate EPE. A *not* very good way is to use the average training error (TE):

$$\text{TE}(\omega, \mathcal{D}_{\text{train}}) = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, \hat{f}_\omega(x_i))$$

where $\mathcal{D}_{\text{train}} = \{(x_i, y_i)\}_{i=1}^n$ is the training data.

Using the ω that minimizes the training error:

$$\omega^* = \arg\min_{\omega} \ \text{TE}(\omega, \mathcal{D}_{\text{train}})$$

will always lead to a model that overfits (as long as f is flexible enough) Why?

Figure Taken from: ESL Fig 7.1

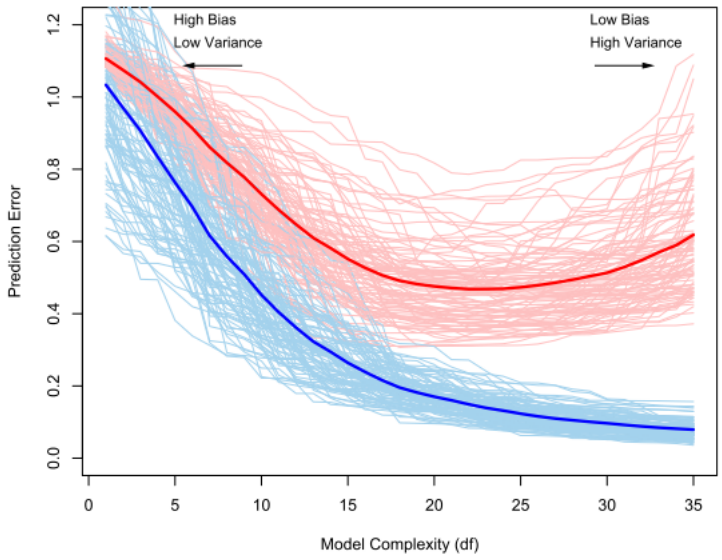


FIGURE 7.1. *Behavior of test sample and training sample error as the model complexity is varied. The light blue curves show the training error* $\overline{\text{err}}$*, while the light red curves show the conditional test error* $\text{Err}_{\mathcal{T}}$ *for* 100 *training sets of size* 50 *each, as the model complexity is increased. The solid curves show the expected test error* Err *and the expected training error* $\text{E}[\overline{\text{err}}]$*.*

2 Model Selection

The goal of model selection is to pick the model that will provide the best *predictive performance* on test data (i.e, model with smallest EPE).

- Note: "model" in this context means model family plus tuning parameter (e.g., polynomial with degree = 3 or knn with k = 12)

There are two main approaches to estimating the predictive performance (EPE) of a model:

1. Predict on hold-out data
2. Make a mathematical adjustment to the training error that better estimates the test error

2.1 Hold out set

An obvious way to assess how well a model will perform on test data is to evaluate it on hold-out data.

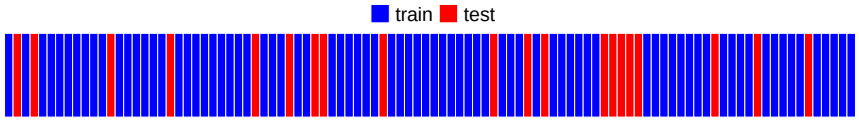
Split the data into a training and test set: $\mathcal{D} = (\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}})$.

- Fit models with $\mathcal{D}_{\text{train}}$ to get estimates $\hat{\theta}_\omega = \hat{\theta}_\omega(\mathcal{D}_{\text{train}})$ and $\hat{f}_\omega(\cdot) = f(\cdot; \hat{\theta}_\omega)$.
- Use $\mathcal{D}_{\text{test}}$ to calculate:

$$\text{PE}(\omega, \mathcal{D}_{\text{test}}) = \frac{1}{m} \sum_{j=1}^{m} \ell(\tilde{y}_j, \hat{f}_\omega(\tilde{x}_j))$$

  where $(\tilde{x}_j, \tilde{y}_j) \in \mathcal{D}_{\text{test}}$ for $j = 1, 2, \ldots, m$.

- Note: in practice, the observations should be assigned **randomly** to the train/test sets (not sequentially like in the above figure)
  - This will reduce the influence of potential correlation in the data



Your Turn #1

1. How large should the training set be?
2. What estimates suffers where there is not enough data in Train Set? Test Set?

2.1.1 Problems with using a single Hold-Out set

1. Need to decide on how to split the data.
   - Too little in training and poor parameter estimates
   - Too little in test and poor performance estimates and model selection
2. We may happen to choose a split that uses a train or test set that poorly represents the data generating process.

- The chance of *bad* split is reduced when $n - m$ and m are both large.

2.2 Adjustments to Training Error (AIC/BIC)

Another approach is to use all the data for training, but adjust the training error to account for potential over-fitting

- The adjustment is usually a function of model complexity (e.g., edf)

Examples:

- AIC/BIC
- Adjusted R^2
- Mallow's Cp

2.2.1 AIC/BIC

- Let $\mathcal{M}(\theta)$ be a model (i.e., model family and tuning parameters) with model parameters θ
- Let $\hat{\mathcal{M}} = \arg\max_\theta \ \log L(\mathcal{M}(\theta))$ be the parameters that maximize the log-likelihood (or penalized log-likelihood).
- $L(M)$ is the likelihood of the model
  - note: to use AIC/BIC a distributional form must be specified
- Let $d(\hat{\mathcal{M}})$ be the *effective degrees of freedom (edf)* (e.g., number of estimated model parameters) of the model

Akaike information criterion (AIC)

$$\text{AIC}(\mathcal{M}) = -2 \log L(\hat{\mathcal{M}}) + 2d(\hat{\mathcal{M}})$$

Bayesian information criterion (BIC)

$$\begin{aligned}\text{BIC}(\mathcal{M}) &= -2 \log L(\hat{\mathcal{M}}) + d(\hat{\mathcal{M}}) \log n \\ &= \text{AIC}(\hat{\mathcal{M}}) + d(\hat{\mathcal{M}}) \left(\log(n) - 2\right)\end{aligned}$$

Information Criterion (generic)

$$\begin{aligned}\text{IC}(\mathcal{M}) &= \ell(\hat{\mathcal{M}}) + P(\hat{\mathcal{M}}) \\ &= \text{Training Loss} + \text{Complexity Penalty}\end{aligned}$$

Example

For a linear regression model with $d = p + 1$ estimated coefficients, $\mathcal{M}_d = f(x; \beta) = \sum_{j=0}^{p} \beta_j x_j$,

- d is the tuning parameter, β are the model parameters, Gaussian distribution:

$$\log L(\hat{\mathcal{M}}_d) = -\frac{n}{2} \log(\text{MSE}(\hat{\beta})) + C$$

where C is a constant that doesn't depend on any model parameters or tuning parameters

$$\text{AIC}(\hat{\mathcal{M}}_{\text{d}}) = \text{n}\log(\text{MSE}(\hat{\beta})) + 2\text{d}$$

- Choose the model that *minimizes* the AIC/BIC
- The AIC/BIC can be used for any likelihood (e.g., Bernoulli for logistic regression)

2.2.2 Adjusted R^2

$$R^2_{\text{adj}} = 1 - \frac{\text{RSS}(\hat{\beta})}{\text{RSS}(\bar{y})}\frac{n-1}{n-d}$$

where d is the number of estimated parameters in the model.

- R^2_{adj} is only appropriate under a squared error loss function.
- Choose the model with the *largest* R^2_{adj}

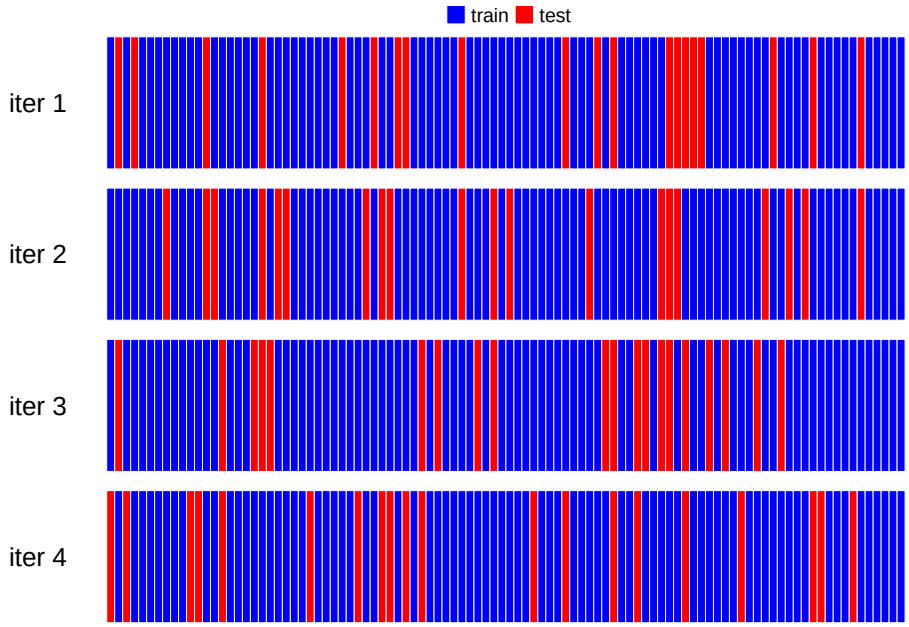
3 Cross-Validation and other Resampling based model selection procedures

3.1 Monte Carlo Cross-Validation (Repeated Hold-Outs)

A *single* train/test split is not ideal due to too much variability:

- Too few observations in training set and poor parameter estimates
- Too few observations in test set and poor performance estimates and model selection
- Increased likelihood of choosing a "bad" split.
  - The chance of *bad* split is reduced when $n - m$ and m are both large.

But we can reduce the variability by **repeating the hold-out analysis multiple times**.



Thoughts:

- Can control exactly how much data used for estimating model parameters and how much for model evaluation.
- Can use lots of iterations to reduce variability
- Notice that some observations are used for training (test) multiple times, while other observations are never used.

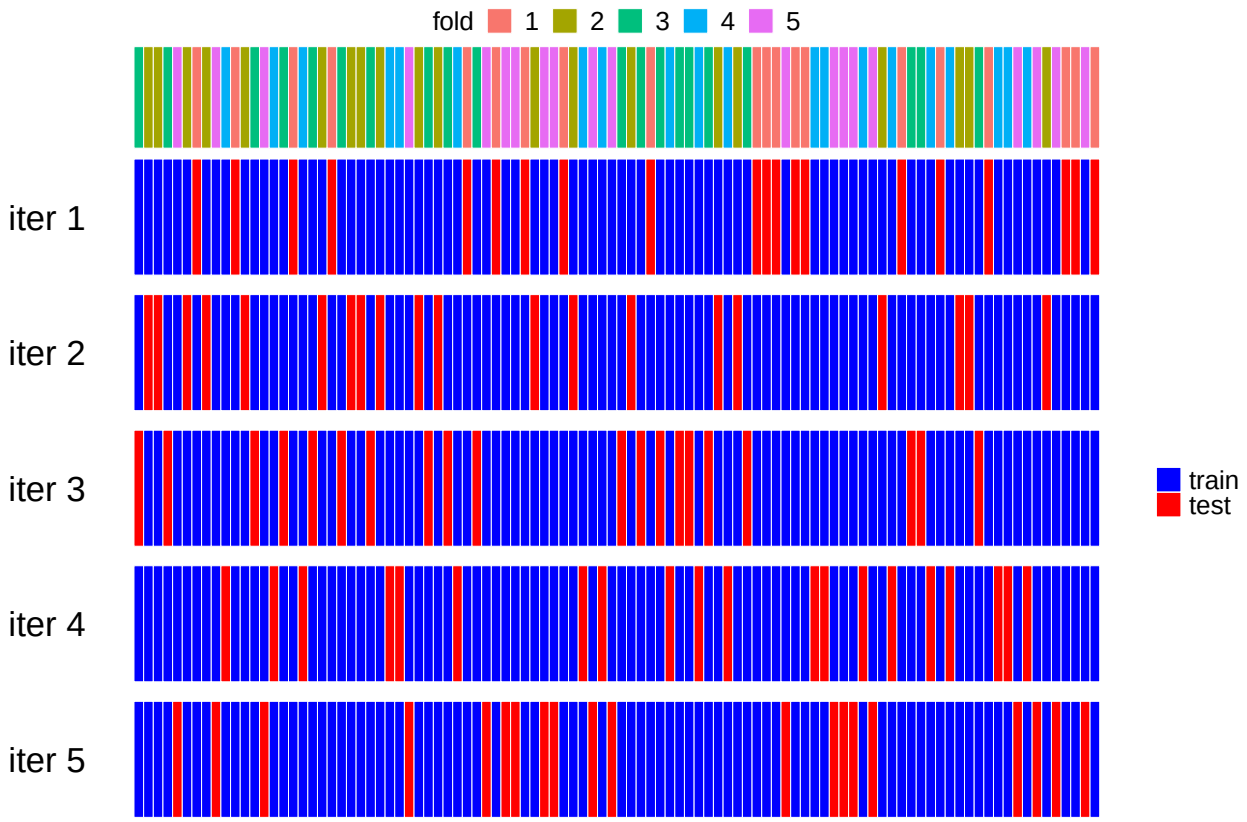
3.2 K-fold Cross-Validation

K-fold (or V-fold) cross-validation is a special implementation of repeated hold-outs.

- Randomly divide the set of observations into K groups, or folds, of approximately equal size.
- One fold is treated as a validation/test set, and the model parameters are estimated from the remaining $K - 1$ folds. And predictions are made on the hold-out set.
- The performance on each fold is combined to get a more accurate assessment of model performance on future data.
- Every observation is used exactly $K - 1$ times for training and once for evaluation.

- $(K-1)/K$ proportion of data used for training
- $1/K$ proportion of data used for evaluation

3.2.1 5-fold cross-validation



3.2.2 Cross-Validation Algorithm

Algorithm: Cross-Validation #1

1. Split data into K folds (of roughly equal size)
   - $\mathcal{F}_1, \ldots, \mathcal{F}_K$
2. For $k = 1, \ldots, K$ and for all models (e.g., for all ω) :
   a. Use the data in $\mathcal{D} \setminus \mathcal{F}_k$ to estimate the model $\hat{f}_\omega^{-k}$
   b. Predict for data in $\mathcal{F}_k$ and calculate the average loss

$$L_k(\omega) = \frac{1}{n_k} \sum_{i \in \mathcal{F}_k} \ell(y_i, \hat{f}_\omega^{-k}(x_i))$$

   where n_k is the number of observations in fold k
3. Choose tuning parameters (model selection) that minimize cross-validation loss

$$\hat{\omega} = \arg\min_\omega \text{CV}(\omega)$$

   where $\text{CV}(\omega) = \frac{1}{n} \sum_{k=1}^K n_k L_k(\omega)$
4. Refit all data using $\hat{\omega}$ to get the final model ($\hat{\theta}$).
   - An alternative is to use an *ensemble* model by combining the models from all folds with weights $1/K$

Note: in step 3, do not blindly choose the tuning parameter without looking at the $\{L_k(\omega)\}$!

- What is the variability across folds?
- Does performance differ over folds?
- Do some folds cause outliers?

3.2.3 Alternative CV error approach

Instead of summarizing the performance in each fold, we can predict the response and summarize after all folds are predicted.

truth	fold	Model 1	Model 2
y_1	5	$\hat{y}_1^1$	$\hat{y}_1^2$
y_2	2	$\hat{y}_2^1$	$\hat{y}_2^2$
y_3	2	$\hat{y}_3^1$	$\hat{y}_3^2$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
y_n	9	$\hat{y}_n^1$	$\hat{y}_n^2$

Then we can calculate the loss for every observation, e.g., $L_i(\omega) = (y_i - \hat{y}_i(\omega))^2$ and overall cross-validation error is $\text{CV}(\omega) = \frac{1}{n} \sum_{i=1}^n L_i$.

Algorithm: Cross-Validation #2

1. Split data into K folds (of roughly equal size)
   - $\mathcal{F}_1, \ldots, \mathcal{F}_K$
2. For $k = 1, \ldots, K$ and for all models (e.g., for all ω) :
   a. Use the data in $\mathcal{D} \setminus \mathcal{F}_k$ to estimate the model $\hat{f}_\omega^{-k}$
   b. Predict for data in $\mathcal{F}_k$:
   $$\hat{y}_i(\omega) = \hat{f}_\omega^{-k}(x_i) \quad \text{for all } i \in \mathcal{F}_k$$
3. Choose tuning parameters (model selection) that minimize cross-validation loss
   $$\hat{\omega} = \arg\min_\omega \text{CV}(\omega)$$
   where $\text{CV}(\omega) = \frac{1}{n}\sum_{i=1}^n \ell(y_i, \hat{y}_i(\omega))$
4. Refit all data using $\hat{\omega}$ to get the final model ($\hat{\theta}$).
   - An alternative is to use an *ensemble* model by combining the models from all folds with weights $1/K$

3.3 1 SE Rule

One Standard Error Rule suggests that instead of using $\hat{\omega} = \arg\min_\omega \text{CV}(\omega)$, we should use the least complex model that is within one standard error of $\text{CV}(\hat{\omega})$.

- To adjust for the uncertainty in the evaluation results. "Given everything equal, choose the least complex model" (*parsimony*)

The standard error of the average cross-validation error can be estimated by:

$$\text{SE}(\omega) = \frac{\text{standard deviation of } \{L_k\}}{\sqrt{K}}$$

Using the alternative approach, the standard error is estimated as

$$\text{SE}(\omega) = \frac{\text{standard deviation of } \{L_i\}}{\sqrt{n}}$$

The two estimates will not be equal, but should be close.

3.4 Choice of K

- $K = 5, 10, n$ are common choices
- $K = n$ is called *leave-one-out (LOOCV)*
  - There exists a closed form solution for models that are linear in y
- Note: These options are probably not sufficient. Consider *repeated* cross-validation instead.

3.4.1 Performance Bias/Variance Trade-off

- Note: around $(K-1)/K$ of the data is used for training and $1/K$ for testing
- if K is too small, then not enough training data and poor cv error estimate (**bias** in prediction error)
- if K is too large, then the f_ω^{-k} are correlated and variance is not reduced (**variance** in prediction error)

  - because the training data is similar across folds

- Computational: need to fit each model K times. Note special exceptions possible for $K = n$ with some models (linear predictors).

3.5 Balanced Data Splitting for Cross-Validation

- The most basic approach is to use random sampling to assign observations to folds
- But this can be problematic if there are outliers or classes with small frequency
  - Especially a problem for categorical data. Some levels are not included in training set.
  - So how to predict when they show up in test set?
- Stratified sampling can be used to ensure similar distributions in each fold
  - e.g., equal number of observations in each fold from same quartile of y
  - e.g., equal number of observations in each fold from same race/ethnicity
- Can stratify on outcome or predictors
  - e.g., cluster the training data and assign observations into folds such that an equal number of observations from each cluster are in each fold

3.6 Other Considerations

3.6.1 Different Model Families

Most of the discussion has been on finding the optimal complexity/tuning parameter for a given *model family*.

But cross-validation (and AIC/BIC) can be used to compare across model families.

3.6.2 Prevent data leakage

Note

Be careful to ensure that all aspects of estimation (e.g., variable selection, tuning parameter selection) are **inside** the cross-validation.

3.7 Repeated Cross-Validation

If you have the patience (or computing resources), you can be more certain about the model performance if you repeat cross-validation several times.

- if you repeat K-fold cross-validation 5 times, then you will need to fit each model $5K$ times
- Take the average of the cross-validation score as the performance measure.

3.8 Monte Carlo CV (Repeated Train/Test splits)

But why even bother about the added code complexity involved in making the folds?

- Just repeatedly split the data into a training and test set
- This will be similar to the repeated cross-validation, but can be simpler to code

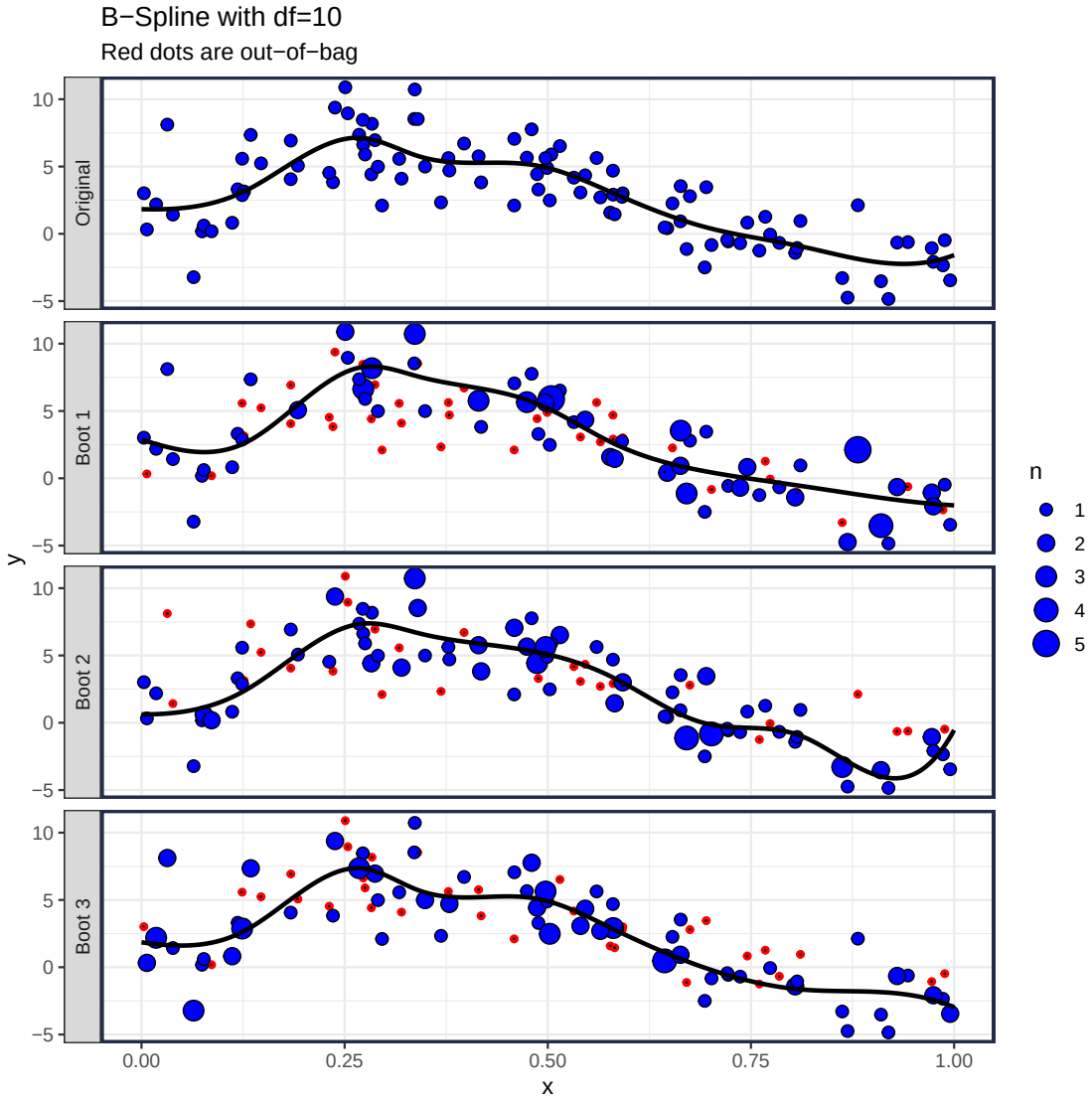
  - holding out 20% of the data and repeating 5 times is comparable to $K = 5$ fold cross-validation.

3.9 Bootstrap Out-of-Bag

Because about 37% of the bootstrap data will be used for model assessment (testing data), it is *similar* to $K = 3$ cross-validation (if repeated 3 times). Some differences:

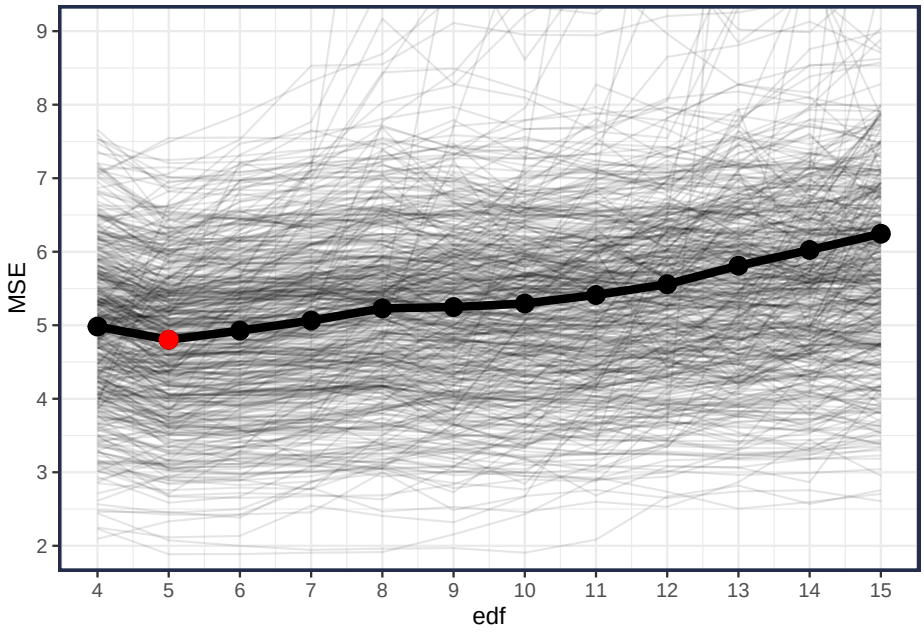
- Use n observations for model fitting (instead of $2/3 \cdot n$)
- Can repeat the bootstrap more than 3 times to reduce variability.

Let's look at a few bootstrap fits:

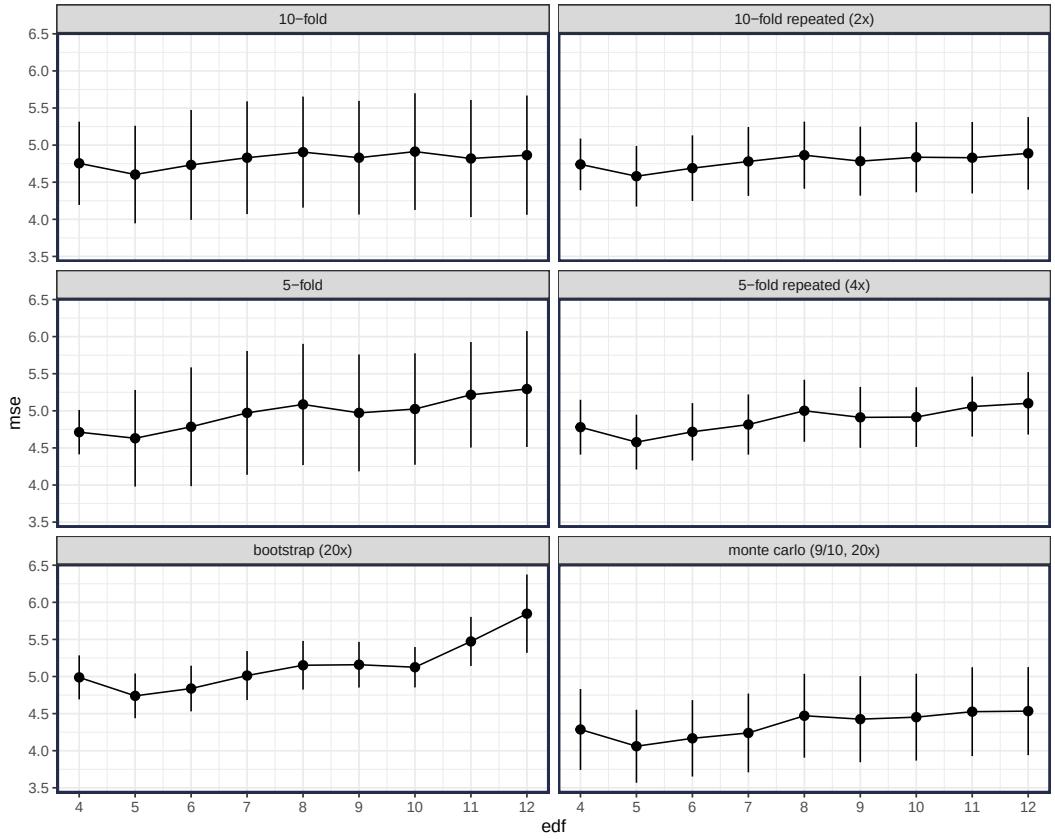


- Notice that each bootstrap sample does not include about 37% of the original observations.
- These are called *out-of-bag* samples and can be used to assess model fit
  - The out-of-bag observations were not used to estimate the model parameters, so will be sensitive to over/under fitting

- Below, we evaluate the oob error over the spline complexity (df = number of estimated coefficients)



3.10 Resampling Comparison



The bars are 1 standard error.

3.11 Linear Smoothers and LOOCV

A *linear smoother* is a model that the fitted training data can be represented by the equation

$$\hat{Y} = HY$$

- For example, in linear regression $\hat{Y} = X\hat{\beta}$ and therefore, $H = X(X^\mathsf{T}X)^{-1}X^\mathsf{T}$.
  - H is called the *hat matrix* or *projection matrix*
  - The diagonal elements of H are called the *leverages* of the observations
  - High leverage points has a large impact on the model fit. Thus, the LOOCV will be especially sensitive to observations with high leverage.

It turns out that for linear smoothers, the LOOCV error (under squared error loss) is

$$\text{LOOCV}(\omega) = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{f}_\omega(x_i)}{1 - h_{ii}}\right)^2$$

where h_{ii} is the i^{th} diagonal element of H.

- This takes away the computational burden of n model fits! The model is fit once to the full data and *corrected* for in the above equation.
- A similar, but even faster to compute, version is the *Generalized Cross-Validation*

$$\begin{aligned}\text{GCV}(\omega) &= \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{f}_\omega(x_i)}{1 - tr(H)/n}\right)^2 \\ &= \frac{\text{MSE}_\omega}{(1 - tr(H)/n)^2}\end{aligned}$$

  where $tr(H) = \sum_{i=1}^{n} h_{ii}$ is the *trace* of H.
  - In unpenalized linear regression $tr(H) = d$, the number of estimated parameters.

4 Model Assessment

We have been concerned with **model selection**, which means choosing the best model (or tuning parameter(s)). But let's turn our attention to **model assessment**, which involves understanding how well our chosen model(s) will perform on new, unseen data (i.e., estimating the EPE).

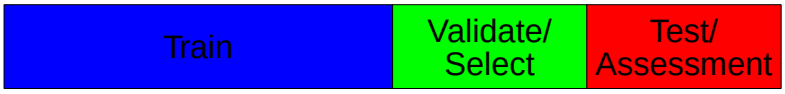
While the results from cross-validation (K-fold and Monte Carlo) do give some sense of performance it suffers from some issues.

1. It could be biased
   - usually too optimistic if using CV to pick best tuning parameter or model
2. There is still uncertainty/variance (especially for single cross-validation).
3. The new data is not from the same distribution as the training data.

One good way to check how well the selected model(s) perform on new data is to use new data to evaluate it. Consider the original version of this idea:

4.1 Train/Validate/Test

If it were possible to have lots of data (all from the same distribution), then we could split up the data into three pieces:



- **Train:** Estimate model parameters θ (for given tuning ω) for many models ($\omega_1, \omega_2, \ldots$)
  - Use the training dataset to estimate the *model parameters* (e.g., β) for a set of *tuning parameters* (e.g., ω) (and possibly different model families)
  - The output from this step will be a set of fitted models, $\hat{f}_1, \hat{f}_2, \ldots$
- **Validate/Select:** Choose optimal tuning parameters ω (i.e., model selection step)
  - Evaluate the performance of each fitted model on the Validate/Select data
  - Choose the best/final model based on the performance
- **Test/Assessment:** Estimate EPE (i.e., final model assessment)
  - Never use the Test/Assessment data until **all** model fitting, tuning, selection is finished.
  - Then the performance of the best/final model can be assessed (without bias)

4.2 Resampling Based Model Assessment

Here is a resampling based improvement on this concept:

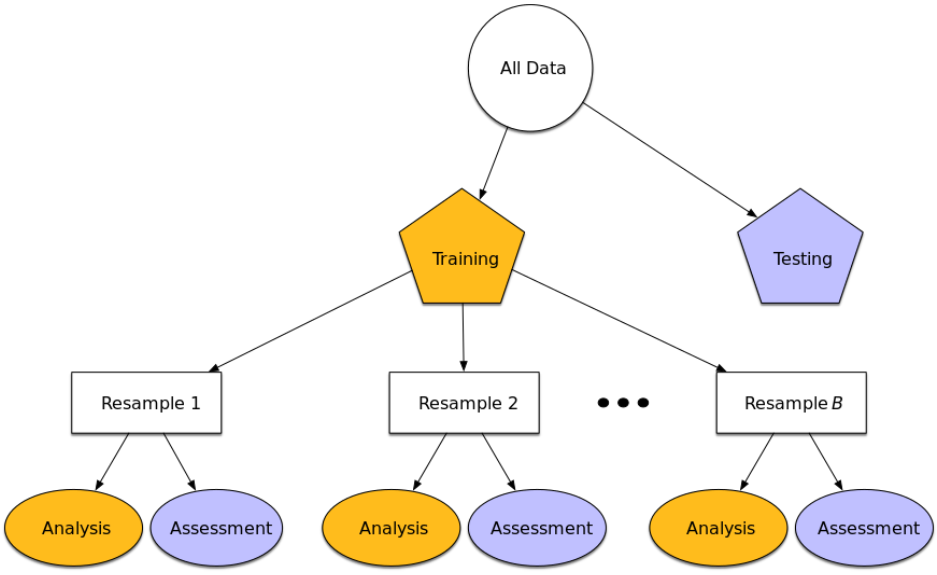
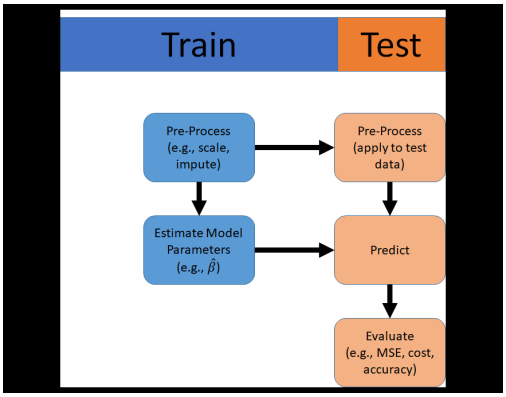
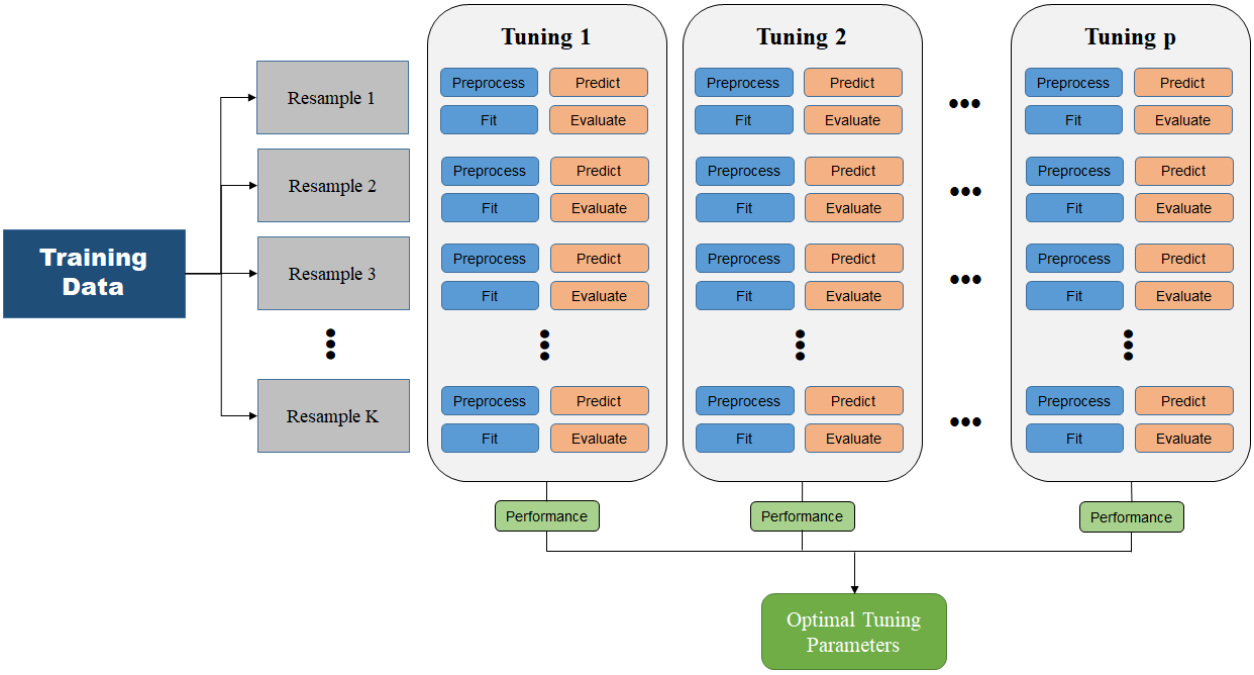


Figure taken from: **Feature Engineering and Selection: A Practical Approach for Predictive Models** by Max Kuhn and Kjell Johnson (2019-06-21)

And if you have the patience, or enough computing resources, you could create a *nested* cross-validation: an outer loop over the initial train/test split to evaluate final performance; with an inner cross-validation loop to select the best model for each training set.





5 Appendix: R Code

```
library(tidymodels) # for optional tidymodels approaches
```

5.1 Simulate Data

```
#-- Data Generation
generate_x <- function(n) runif(n)        # U[0,1]
f <- function(x) 1 + 2*x + 5*sin(5*x)     # true mean function
sd = 2                                    # stdev for error

#- Training data
```

```
n = 208                                 # number of observations
set.seed(825)                           # set seed for reproducibility
x = generate_x(n)                       # get x values
y = f(x) + rnorm(n, sd=sd)              # get y values
data_train = tibble(x,y)                # training data

#- Test data
n = 50                                  # number of observations
set.seed(825)                           # set seed for reproducibility
x = generate_x(n)                       # get x values
y = f(x) + rnorm(n, sd=sd)              # get y values
data_test = tibble(x,y)                 # test data
```

5.2 Model Set-up

We are going to evaluate the choice of B-spline tuning parameter $df \in \{3, 4, \ldots\}$ using a variety of resampling methods. The `df` parameter in `bs()` controls the number of B-splines (basis functions) and hence the number of estimated parameters (edf). In the formulation below, I'll use cubic B-splines (degree = 3) where the knots are selected from the quantiles of the training data.

We will use the `bs()` function from the `splines` package for the model. I'll make a function `sp_eval()` that will fit a set of B-spline models (of varying complexity) to the training data, make predictions on the test data, and calculate the MSE.

```
library(splines)  # for the bs() function
library(dplyr)    # for tibble() and bind_rows() functions

# sp_eval(): fit set of B-spline models and evaluate on test data
#-------------------------------------------------------------------------
# data_fit, data_eval: training and test data (requires column names x,y)
# df: set of spline degrees (tuning parameters)
# kts.bdry: boundary knots for the splines (to help extrapolate)
# output: tibble with df and associated mean squared error (MSE) on test data
sp_eval <- function(data_fit, data_eval, df = seq(3, 15, by=1), kts_bdry = c(-.1, 1.1)) {

  MSE = numeric(length(df)) # initialize

  for(i in 1:length(df)) {
    # set tuning parameter value
    df_i = df[i]
    # fit with training data (no intercept)
    fmla = formula(y~bs(x, df = df_i, degree = 3, Boundary.knots = kts_bdry) - 1)
    fit = lm(fmla, data = data_fit)
    # predict on test data
    yhat = predict(fit, data_eval)
    # get errors / loss
    MSE[i] = mean( (data_eval$y - yhat)^2 )
  }
tibble(df, mse=MSE) # output
}
```

5.3 Single train/test split

Set the number of hold-out observations and random seed.

```
n.holdout = 20                      # size of hold-out set
set.seed(2021)                      # set seed for reproducibility
```

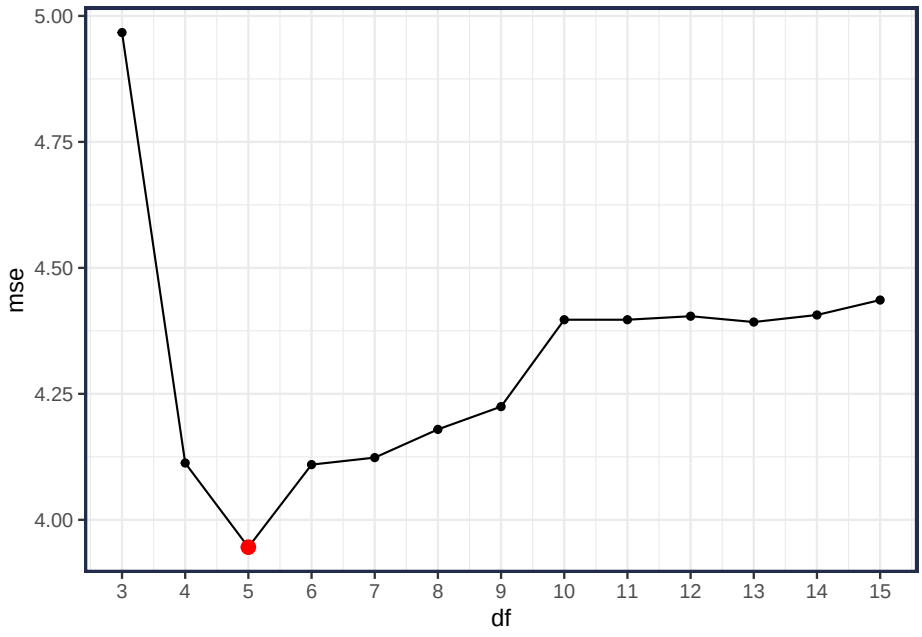
```
holdout = sample(n, size=n.holdout) # indices to include in holdout set
```

Evaluate the performance over each tuning parameter (`DF`)

```
DF = seq(3, 15, by=1)      # set complexity (edf) values
results = sp_eval(
        data_fit = slice(data_train, -holdout),
        data_eval = slice(data_train, holdout),
        df = DF
        )
```

This visualizes the results showing that $\hat{df} = 5$ is the optimal value for this train/test split.

```
ggplot(results, aes(df, mse)) +
  geom_point() + geom_line() +
  geom_point(data = . %>% slice_min(mse, n=1), color="red", size=3) +
  scale_x_continuous(breaks = seq(0, 20, by=1))
```



5.3.1 Using `rsample`

The `mc_cv` function from the `rsample` package will generate a monte carlo cross-validation. For a single holdout:

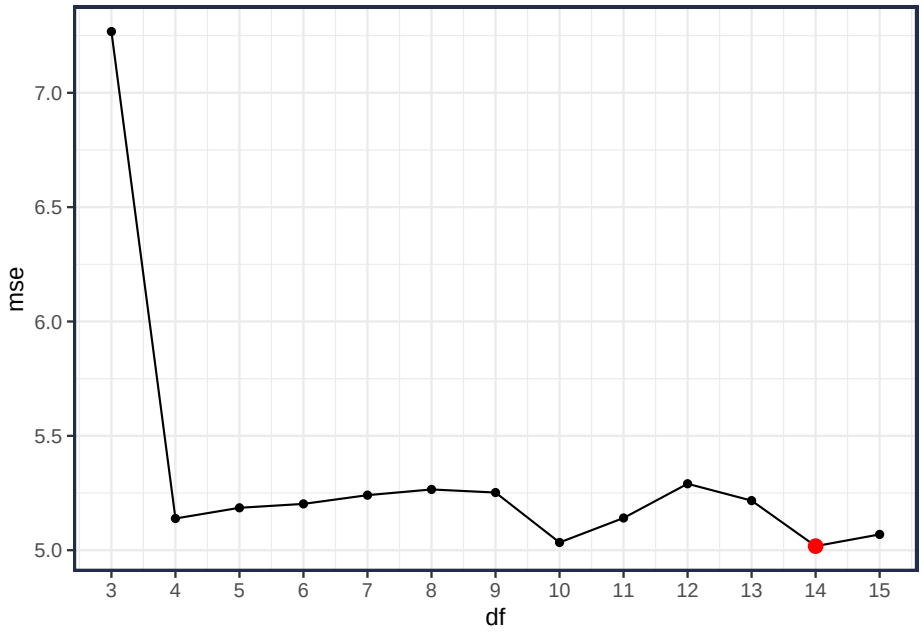
```
n.holdout = 20                      # size of hold-out set
set.seed(2021)                      # set seed for reproducibility
data_mc = mc_cv(data_train, prop = 1- n.holdout/nrow(data_train), times=1)
```

Evaluate the performance over each tuning parameter (`DF`)

```
DF = seq(3, 15, by=1)      # set complexity (edf) values
results2 = sp_eval(
        data_fit = training(data_mc$splits[[1]]),
        data_eval = testing(data_mc$splits[[1]]),
        df = DF
        )
```

This visualizes the results showing that $\hat{df} = 14$ is the optimal value for this train/test split.

```
ggplot(results2, aes(df, mse)) +
  geom_point() + geom_line() +
  geom_point(data = . %>% slice_min(mse, n=1), color="red", size=3) +
  scale_x_continuous(breaks = seq(0, 20, by=1))
```



Note that we have a different result, even with same seed, because the `mc_cv()` function uses a different method to select holdout data.

5.4 Repeated train/test splits (Monte Carlo Cross-Validation)

Now, let's repeat this procedure $M = 10$ times:

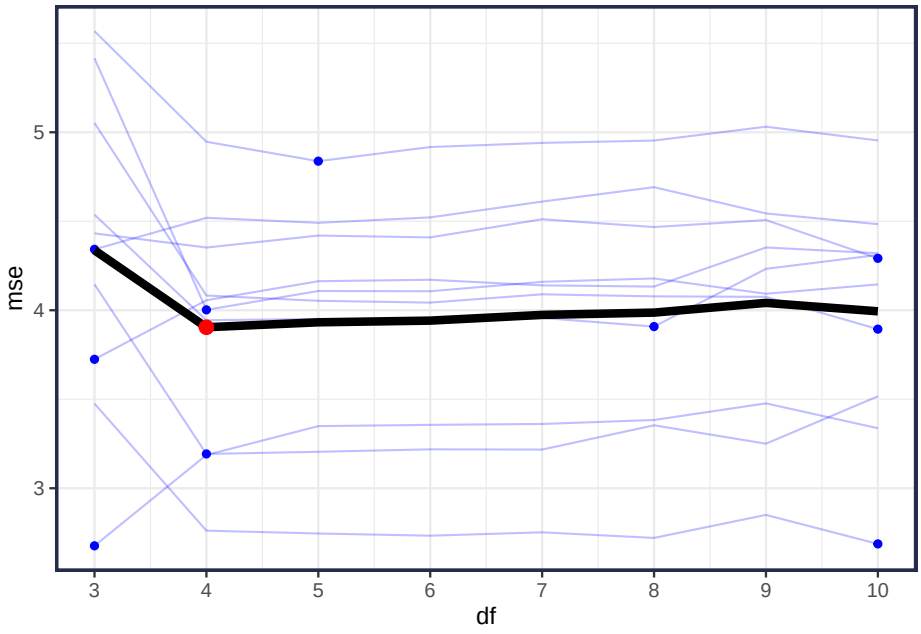
```
n = nrow(data_train) # number of training observations
n.holdout = 20       # size of hold-out set
M = 10               # number of repeats
set.seed(2021)       # set seed for reproducibility

#-- Repeat the train/test split M times
results = vector("list", M)

for(m in 1:M) {
  #- select hold out observations
  holdout = sample(n, size=n.holdout) # indices to include in holdout set
  #- fit and evaluate models
  results[[m]] = sp_eval(
        data_fit = slice(data_train, -holdout),
        data_eval = slice(data_train, holdout),
        df = seq(3, 10, by=1)
        ) %>%
    mutate(iter = m)  # add iteration number
}
RESULTS = bind_rows(results)
```

Here is the visualization of the results. There is one line for each repetition (colored blue). The solid black line is the mean.

```
ggplot(RESULTS, aes(df, mse)) +
  geom_line(aes(group = iter), color = "blue", alpha=.25) +
  geom_point(data=. %>% group_by(iter) %>% slice_min(mse, n=1), color="blue") +
  geom_line(data = . %>% group_by(df) %>% summarize(mse = mean(mse)), size=2) +
  geom_point(data = . %>% group_by(df) %>% summarize(mse = mean(mse)) %>%
               slice_min(mse, n=1), size=3, color="red") +
  scale_x_continuous(breaks = seq(0, 20, by=1))
```

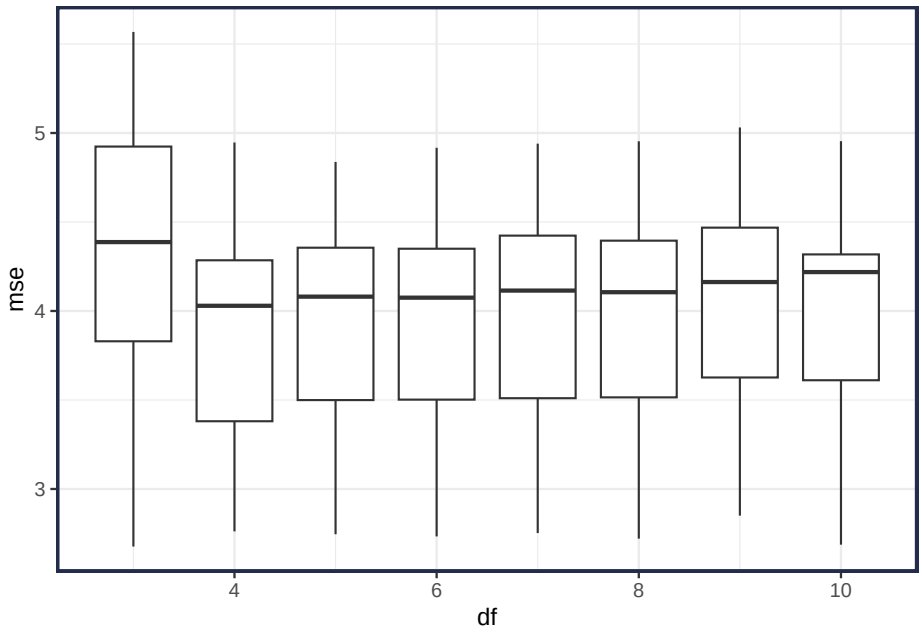


This suggests that $\hat{df} = 4$.

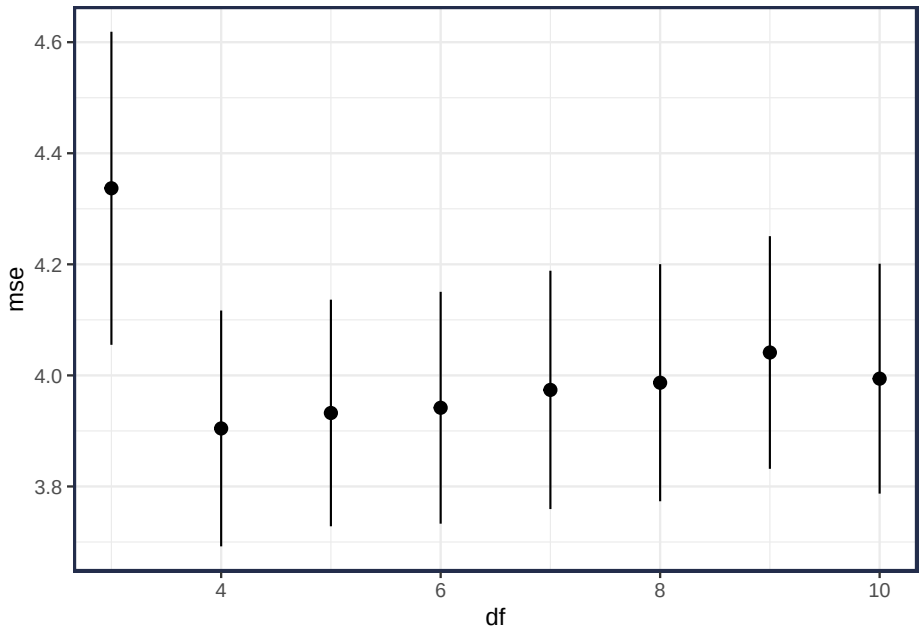
Notice that the iterations have different optimal df. Usually the average MSE is used to determine the optimal tuning parameters, but analysis of the individual optimal may be valuable at times.

You'll often see this information represented with boxplots or standard errorbar plots:

```
#- Boxplots
RESULTS %>%
  ggplot(aes(df, mse, group=df)) + geom_boxplot()
```




```
#-- 1 standard error errorbars
RESULTS %>%
  group_by(df) %>%
  summarize(sd = sd(mse), mse = mean(mse), n = n(), se = sd/sqrt(n)) %>%
  ggplot(aes(df, mse)) + geom_pointrange(aes(ymin=mse-se, ymax=mse+se))
```



5.4.1 Using `rsample`

The `mc_cv` function from the `rsample` package will generate a monte carlo cross-validation.

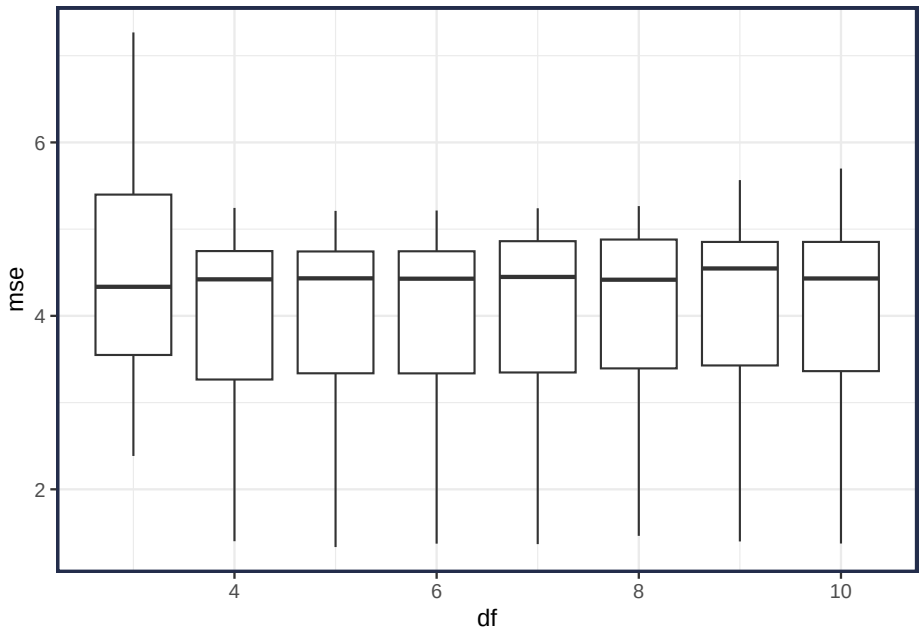
```
n.holdout = 20                          # size of hold-out set
M = 10                                  # number of repeats
set.seed(2021)                          # set seed for reproducibility
```

```
data_mc = mc_cv(data_train,
                prop = 1- n.holdout/nrow(data_train),
                times=M)
```

Evaluate the performance over each tuning parameter (`DF`)

```
DF = seq(3, 10, by=1)      # set complexity (edf) values
RESULTS2 = map_df(data_mc$splits, ~sp_eval(
        data_fit = training(.x),
        data_eval = testing(.x),
        df = DF), .id = "iter"
        )
```

```
#- Boxplots
RESULTS2 %>%
  ggplot(aes(df, mse, group=df)) + geom_boxplot()
```



5.5 K-Fold Cross-Validation

K-fold (or sometime called V-fold) cross-validation is just a special case of Monte Carlo CV where each observation is used for training K-1 times and in evaluation exactly once. To get folds of almost equal size I use the function `rep(1:n.folds, length=n)` and then shuffle (using `sample()`) to further limit potential correlation effects.

```
#- Get K-fold partition
n = nrow(data_train) # number of training observations
n.folds = 10         # number of folds for cross-validation
set.seed(2021)       # set seed for reproducibility
fold = sample(rep(1:n.folds, length=n))  # vector of fold labels
# notice how this is different than:  sample(1:K,n,replace=TRUE),
#  which won't necessarily give almost equal group sizes

results = vector("list", n.folds)
#- Iterate over folds
for(j in 1:n.folds){
```

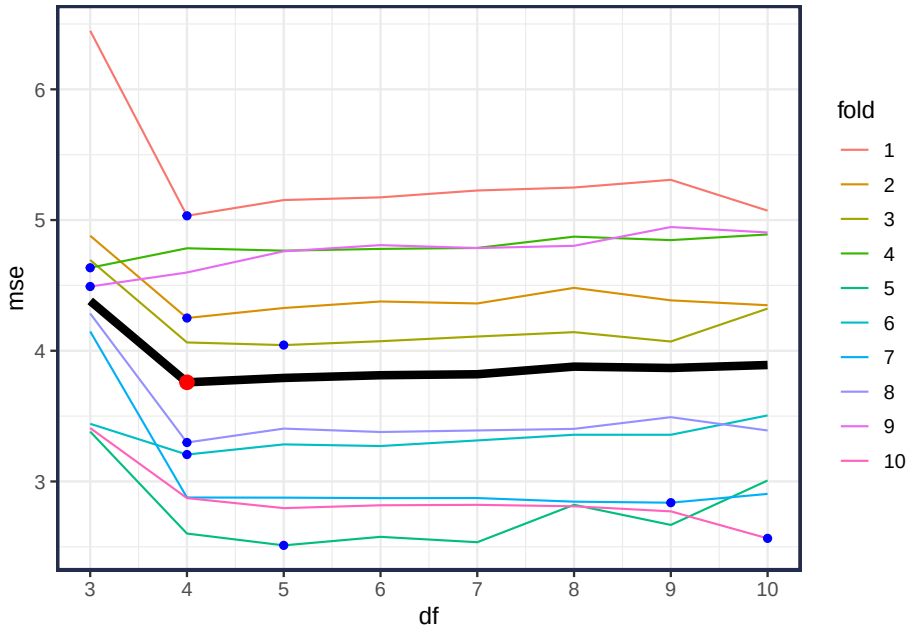
```
  #-- Set training/val data
  val = which(fold == j)     # indices of holdout/validation data
  train = which(fold != j)   # indices of fitting/training data
  n.val = length(val)        # number of observations in validation

  #- fit and evaluate models
  results[[j]] = sp_eval(
        data_fit = slice(data_train, train),
        data_eval = slice(data_train, val),
        df = seq(3, 10, by=1)
        ) %>%
    mutate(fold = j, n.val)  # add fold number and number in validation
}
RESULTS = bind_rows(results)
```

Notice that this is almost identical to the repeated hold-out process. The only difference is in how the train and hold-out sets are determined.

Here is the visualization of the results. There is one line for each repetition (colored blue). The solid black line is the mean.

```
RESULTS %>% mutate(fold = factor(fold)) %>%
  ggplot(aes(df, mse)) +
  geom_line(aes(color=fold)) +
  geom_point(data=. %>% group_by(fold) %>% slice_min(mse, n=1), color="blue") +
  geom_line(data = . %>% group_by(df) %>% summarize(mse = mean(mse)), size=2) +
  geom_point(data = . %>% group_by(df) %>% summarize(mse = mean(mse)) %>%
               slice_min(mse, n=1), size=3, color="red") +
  scale_x_continuous(breaks = seq(0, 20, by=1))
```



This suggests that $\hat{df} = 4$.

There is still much variability in the mean mse. This could be reduced by repeated cross-validation.

5.5.1 Using `rsample`

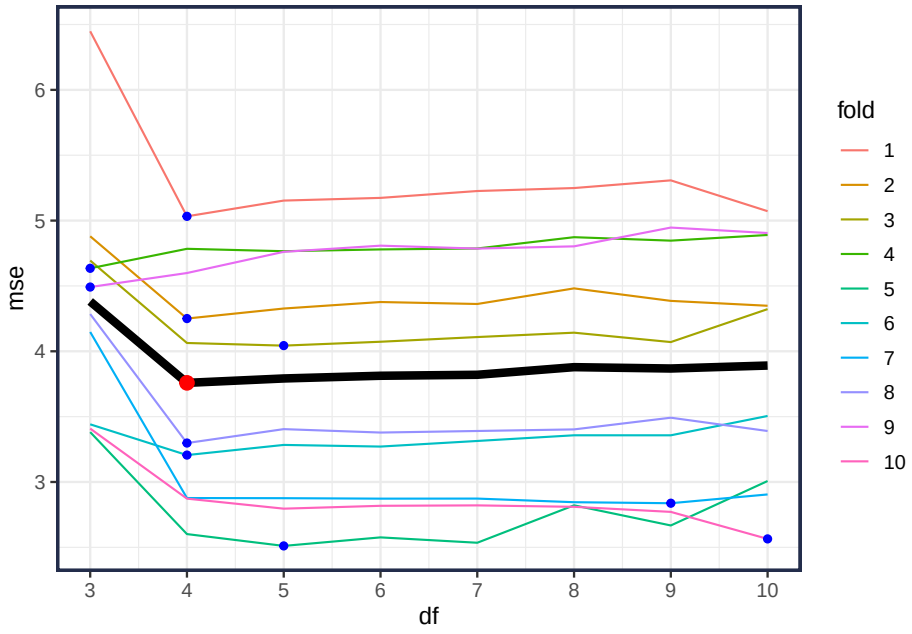
The `vfold_cv` function from the `rsample` package will generate v-fold cross-validation.

```
n_folds = 10         # number of folds for cross-validation
set.seed(2021)                      # set seed for reproducibility
data_cv = vfold_cv(data_train, v = n_folds)
```

Evaluate the performance over each tuning parameter (`DF`)

```
DF = seq(3, 10, by=1)      # set complexity (edf) values
RESULTS2 = map_df(data_cv$splits, ~sp_eval(
        data_fit = training(.x),
        data_eval = testing(.x),
        df = DF), .id = "fold"
        )
```

```
RESULTS2 %>%
  mutate(fold = factor(fold, 1:n_folds)) %>%
  ggplot(aes(df, mse)) +
  geom_line(aes(color=fold)) +
  geom_point(data=. %>% group_by(fold) %>% slice_min(mse, n=1), color="blue") +
  geom_line(data = . %>% group_by(df) %>% summarize(mse = mean(mse)), size=2) +
  geom_point(data = . %>% group_by(df) %>% summarize(mse = mean(mse)) %>%
               slice_min(mse, n=1), size=3, color="red") +
  scale_x_continuous(breaks = seq(0, 20, by=1))
```



5.6 Out-of-Bag

We'll use $M = 20$ bootstraps:

```
n = nrow(data_train) # number of training observations
M = 20               # number of bootstraps
set.seed(2021)       # set seed for reproducibility

#-- Repeat the train/test split M times
RESULTS = vector("list", M)
```

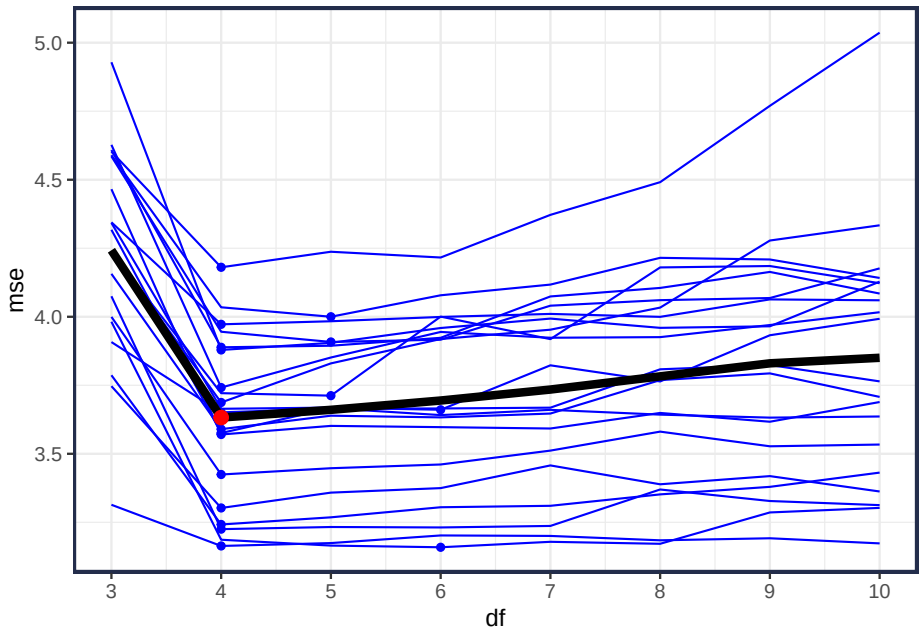
```
for(m in 1:M) {

  #- bootstrap sampling
  boot = sample(n, size=n, replace=TRUE) # indices in bootstrap sample
  oob = setdiff(1:n, boot)               # out-of-bag indices

  #- fit and evaluate models
  results[[m]] = sp_eval(
        data_fit = slice(data_train, boot),
        data_eval = slice(data_train, oob),
        df = seq(3, 10, by=1)
        ) %>%
    mutate(iter = m, n.oob = length(oob))  # add fold number and number in oob
}
RESULTS = bind_rows(results)
```

Here is the visualization of the results. There is one line for each repetition (colored blue). The solid black line is the mean.

```
RESULTS %>%
  ggplot(aes(df, mse)) +
  geom_line(aes(group = iter), color="blue") +
  geom_point(data=. %>% group_by(iter) %>% slice_min(mse, n=1), color="blue") +
  geom_line(data = . %>% group_by(df) %>% summarize(mse = mean(mse)), size=2) +
  geom_point(data = . %>% group_by(df) %>% summarize(mse = mean(mse)) %>%
               slice_min(mse, n=1), size=3, color="red") +
  scale_x_continuous(breaks = seq(0, 20, by=1))
```

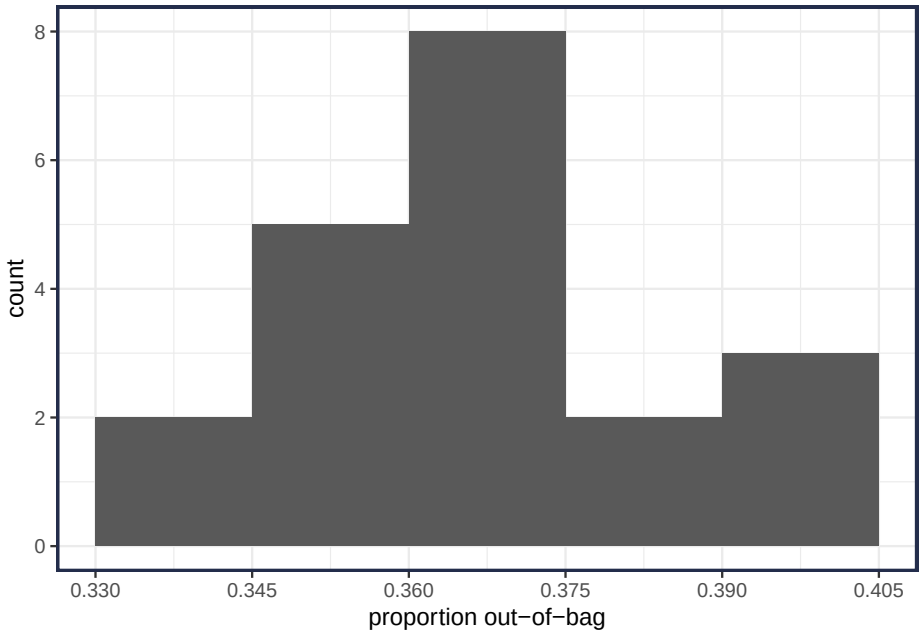


This suggests that $\hat{df} = 4$.

Notice that the bootstrap procedure uses a sample size of n to estimate the model parameters. The other approaches use a smaller sample size.

This procedure also uses a hold-out size of around 37%. Compare this with 10-fold cross-validation which only used 10% of the data for evaluation.

```
RESULTS %>% distinct(iter, n.oob) %>% mutate(p.oob = n.oob/n) %>%
  ggplot(aes(p.oob)) + geom_histogram(binwidth = .015, boundary=0) +
  scale_x_continuous(breaks = seq(0, 1, by=.015)) +
  labs(x= "proportion out-of-bag")
```



5.6.1 Using `rsample`

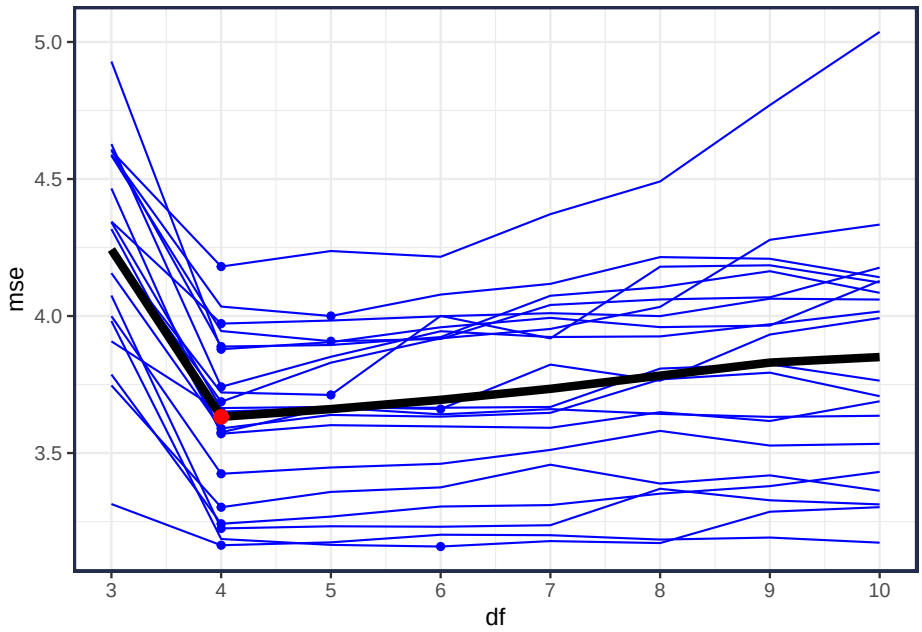
The `bootstraps()` function from the `rsample` package will generate bootstrap samples.

```
M = 20                # number of bootstraps
set.seed(2021)                        # set seed for reproducibility
data_boot = bootstraps(data_train, times = M)
```

Evaluate the performance over each tuning parameter (`DF`)

```
DF = seq(3, 10, by=1)      # set complexity (edf) values
RESULTS2 = map_df(data_boot$splits, ~sp_eval(
        data_fit = training(.x),
        data_eval = testing(.x),
        df = DF), .id = "iter"
        )
```

```
RESULTS2 %>%
  ggplot(aes(df, mse)) +
  geom_line(aes(group = iter), color="blue") +
  geom_point(data=. %>% group_by(iter) %>% slice_min(mse, n=1), color="blue") +
  geom_line(data = . %>% group_by(df) %>% summarize(mse = mean(mse)), size=2) +
  geom_point(data = . %>% group_by(df) %>% summarize(mse = mean(mse)) %>%
               slice_min(mse, n=1), size=3, color="red") +
  scale_x_continuous(breaks = seq(0, 20, by=1))
```



5.7 Evaluation on the test data

There are 208 training observations. For a model with edf = 7 there are only 29.7 observations per model parameter - which is not a lot. This suggests to me, and the plots confirm, that there is still much uncertainty in both the optimal tuning parameter (*df*) and the expected prediction error (EPE). For estimating *df*, I would repeat one of the approaches (cross-validation, monte carlo cross validation, out of bag) many more times to reduce the variability in the results.

This would also give an estimate of the EPE, but with bias. Once the optimal *df* is selected (`df_opt`), then a final model is fit, using all the data and predictions for the *test data* are made.

```
df_opt = 4  # best tuning parameter from cross-validation

sp_eval(data_fit = data_train,    # use all training data
        data_eval = data_test,    # predict on test data
        df = df_opt)              # use optimal tuning parameter(s)
#> # A tibble: 1 x 2
#>      df   mse
#>   <dbl> <dbl>
#> 1     4  4.54
```

The error on the test data, which hasn't been used in any way yet, will provide an unbiased estimate of performance (EPE).