Cross-Validation and Model Selection

EN5422/EV4238 | Fall 2023

w03_cross_validation_model_selection.pdf

(Week 3 – 2/2)

Contents

1 INTRODUCTION TO THE BOOTSTRAP 2

1.1 UNCERTAINTY IN A TEST STATISTIC 2

2 BOOTSTRAPPING REGRESSION PARAMETERS 6

2.1 BOOTSTRAPPING REGRESSION PARAMETERS 7

3 NON-LINEAR MODELING VIA BASIS EXPANSION 7

3.1 PIECEWISE POLYNOMIALS 10

3.2 B-SPLINES 10

3.3 BOOTSTRAP CONFIDENCE INTERVAL FOR $F(X)$ 13

4 MORE BAGGING 14

4.1 OUT-OF-BAG SAMPLES 14

4.2 NUMBER OF BOOTSTRAP SIMULATIONS 16

5 MORE RESOURCES 17

5.1 VARIATIONS OF THE BOOTSTRAP 17

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 *k*-NN 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 is 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 *k*-NN).
- Other tuning parameters impact the structure or algorithm of a model (e.g., using Euclidean or Manhattan distance in *k*-NN).
    - Note: Euclidean distance is the straight-line distance between points in Euclidean space (typically in two or three dimensions).
    - Manhattan distance, also known as city block distance or L1 distance, is the distance between two points measured along the axes of the coordinate system.

- Other ancillary aspects like pre-processing, feature engineering, and imputation approaches are not really tuning parameters, per se, but can treated as such for determining if they help with predictions.
    - E.g.,
        - Some implementations can scale data internally before estimating parameters.
        - 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 *k*-NN regression, $\omega = k$.
- For polynomial regression, $\omega = J$ in the model $\hat{f} = \sum_{j=0}^{J} \beta_j x^j$
- In best subsets regression, $\omega = p$, the total number of features allpwed in the model
- In ridge regression, $\omega = \lambda$ in the ridge penalty $P\text{en}(\beta) = \lambda \sum_{j=1}^{P} \beta_j^2$
    - Note: The main goal of ridge regression is to prevent overfitting, which happens when a model is too complex and fits the noise in the training data rather than the underlying relationship.
    - The key idea behind ridge regression is to introduce a penalty on the size of the coefficients in the linear regression model. This penalty discourages large coefficients which can lead to overfitting.

1.3 Predictive Performance

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

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

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.
- $\hat{f}_\omega\left(\tilde{X}\right) = f_\omega(x;\ \hat{\theta}_\omega(\mathcal{D}_{train}))$ is the prediction function which has been estimated under the tuning parameter ω.
    - $\hat{\theta}_\omega(\mathcal{D}_{train})$ are the **model parameters** estimated from the training data $\mathcal{D}_{train}$ *given* the tuning value ω.

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

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

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):

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

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

Using the ω that minimizes the training error:

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

will always lead to a model that overfits (as long as f is flexible enough) *Why?*
(Figure Taken from: ESL Fig 7.1)

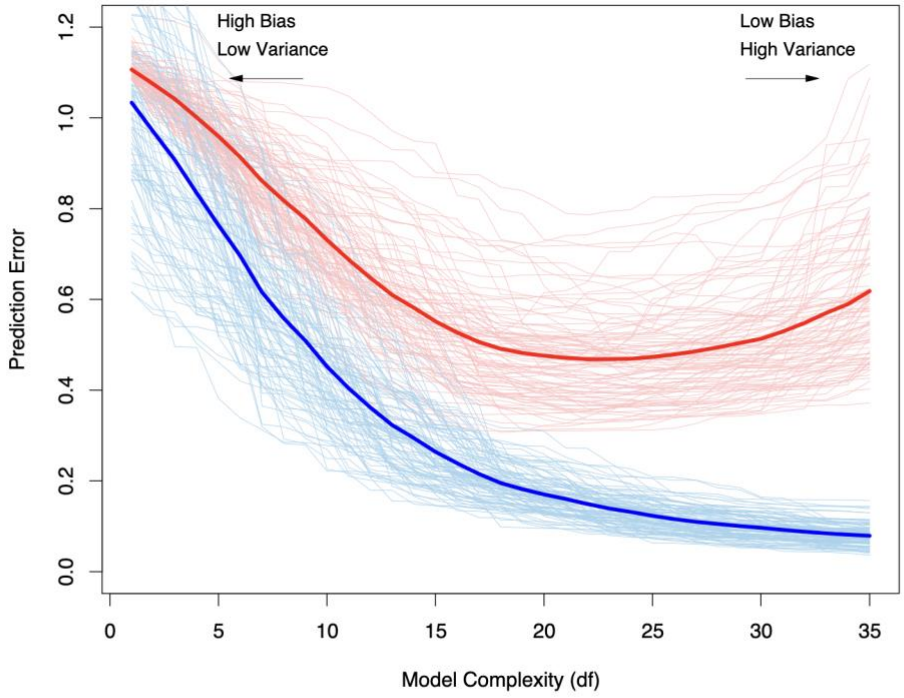


FIGURE 7.1. *Behavior of test sample and training sample error as the model complexity is varied. The light blue curves show the training error* $\overline{\mathrm{err}}$*, while the light red curves show the conditional test error* $\mathrm{Err}_{\mathcal{T}}$ *for* 100 *training sets of size* 50 *each,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

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 *k*-NN 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

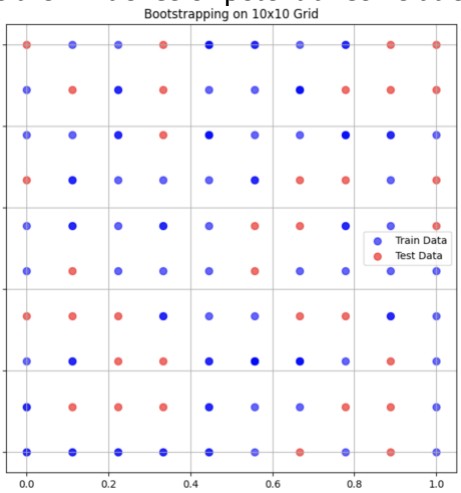
2.1 Hold-Out set

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: $\mathcal{D} = (\mathcal{D}_{train}, \mathcal{D}_{test})$.

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

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

  where $(\tilde{x}_j, \tilde{y}_i) \in \mathcal{D}_{test}$ 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 Problems with using a single Hold-Out set

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 Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC)

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

Akaike Information Criterion (AIC)

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

Bayesian Information Criterion (BIC)

$$\text{BIC}(\mathcal{M}) = -2\log L\left(\widehat{\mathcal{M}}\right) + d\left(\widehat{\mathcal{M}}\right)\log n = AIC\left(\widehat{\mathcal{M}}\right) + d(\widehat{\mathcal{M}})(\log(n) - 2)$$

Information Criterion (generic)

$$\mathrm{IC}(\mathcal{M}) = \ell(\widehat{\mathcal{M}}) + P(\widehat{\mathcal{M}}) = \text{Training Loss} + \text{Complexity Penalty}$$

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:

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

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

$$\mathrm{AIC}(\widehat{\mathcal{M}}_d) = n\log\left(MSE(\hat{\beta})\right) + C$$

- Choose the model that *minimizes* the AIC/BIC
- The AIC/BIC can be used for any likelihood (e.g., Bernoulli for logistic regression)
  - Note: The Bernoulli distribution is a discrete probability distribution for a random variable which can take on one of two possible outcomes, typically labeled 0 and 1.
  - $f(x) = \begin{cases} p, & k < 0 \\ 1-p, & k \geq 0 \end{cases}$, $P(X = k) = p^k(1-p)^{1-k}$ for $k \in \{0,1\}$.

2.2.2 Adjusted R^2

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

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}.
  - Note: The motivation behind this adjustment is to prevent the R^2 value from artificially inflating as more predictors are added to the model, regardless of their true contribution to explaining the variance.

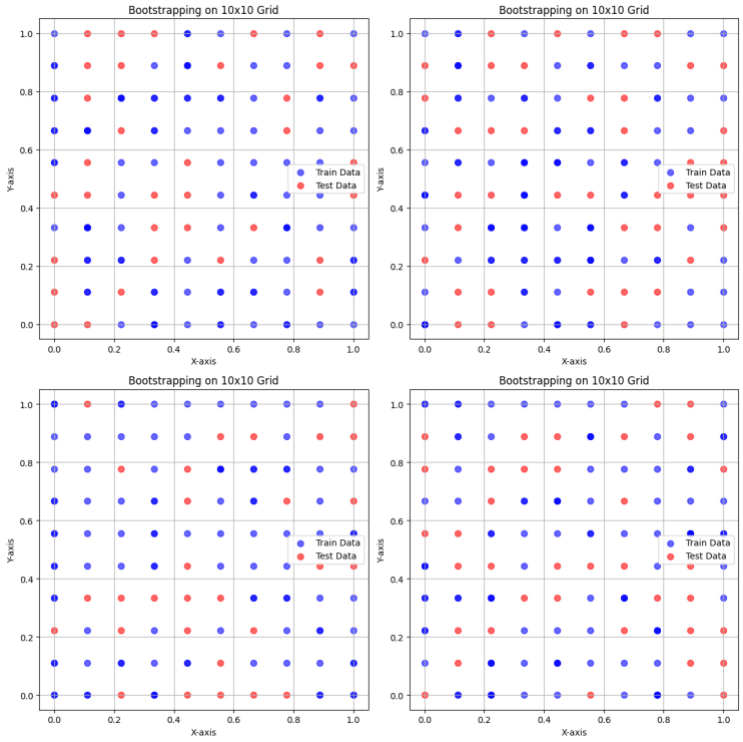
3 Cross-Validation and other Resampling based model selection procedures

3.1 Cross-Validation and other Resampling based model selection procedures

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:

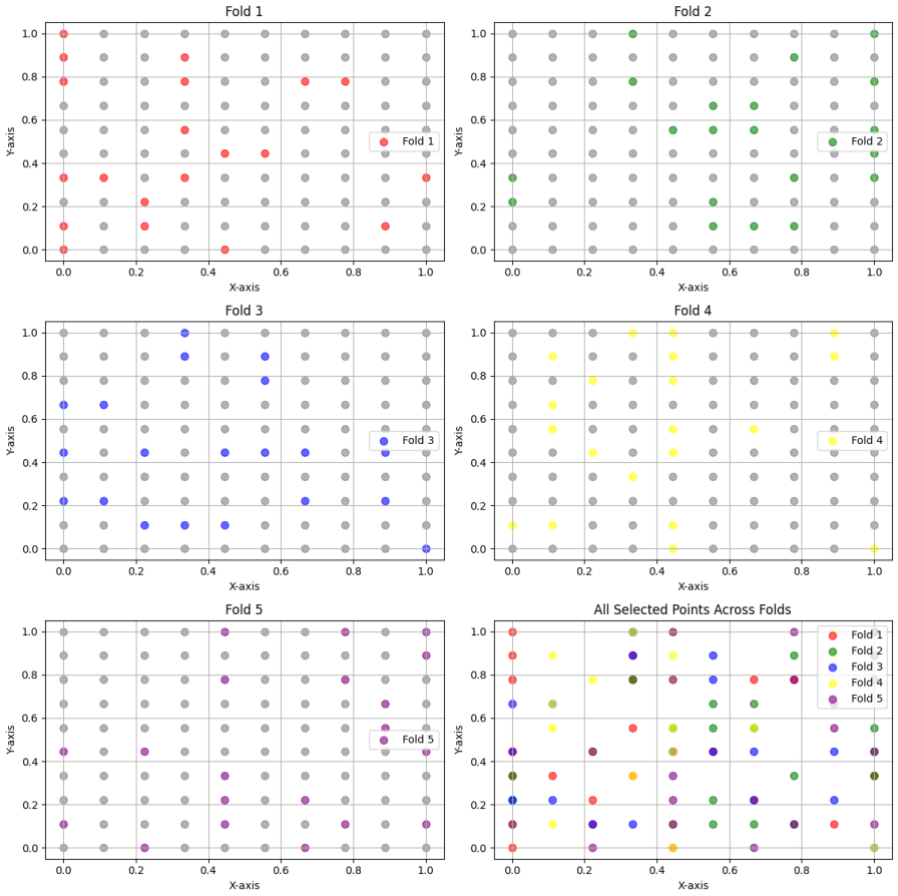
- Can control exactly how much data used for estimating model parameters and how much for model evaluation.
- Can use lots of iterations to reduce variability.
- Notice that some observations are used for training (test) multiple times, while other observations are never used.

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):
   • Use the data in $\mathcal{D} \backslash \mathcal{F}_K$ to estimate the model $\hat{f}_\omega^{-k}$
   • Predict for data in $\mathcal{F}_K$ and calculate the average loss:

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

   where n_k is the number of observations in fold *k*.

3. Choose tuning parameters (model selection) that minimize cross-validation loss.

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

   where $\mathrm{CV}(\omega) = \frac{1}{n} \sum_{k=1}^{K} n_k L_k(\omega)$

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., $L_i(\omega) = \left(y_i - \hat{y}_i(\omega)\right)^2$ and overall cross-validation error is $\mathrm{CV}(\omega) = \frac{1}{n}\sum_{i=1}^{n} L_i$.

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):
   • Use the data in $\mathcal{D}\backslash\mathcal{F}_K$ to estimate the model $\hat{f}_{\omega}^{-k}$
   • Predict for data in $\mathcal{F}_K$:

$$\hat{y}_i(\omega) = \hat{f}_{\omega}^{-k}(x_i) \text{ for all } i \in \mathcal{F}_K$$

3. Choose tuning parameters (model selection) that minimize cross-validation loss.

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

   where $\mathrm{CV}(\omega) = \frac{1}{n}\sum_{i=1}^{n} \ell\left(y_i, \hat{y}_i(\omega)\right)$

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 $\hat{\omega} = arg\min_{\omega} \mathrm{CV}(\omega)$, we should use the least complex model that is within one standard error of $\mathrm{CV}(\hat{\omega})$.

- 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:

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

Using the alternative approach, the standard error is estimated as

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

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.

Example:

Imagine you have a small dataset of weights (in kg) and heights (in cm) of 5 people:

Person	Weight (kg)	Height (cm)
A	50	160
B	60	170
C	70	180
D	80	190
E	90	200

You want to predict a person's weight from their height using a simple linear regression model. Mathematically, this relationship is represented as:

$$Weight = a \times Height + b$$

This model is "linear in y" because the dependent variable (Weight) is a linear function of the parameters a and b.

Now, consider using Leave-One-Out (LOO) cross-validation. You'd do the following:

1. **Leave out A**: Train on B, C, D, E. Test on A.
2. **Leave out B**: Train on A, C, D, E. Test on B. ... and so on, until each person has been left out once.

The "closed form solution" part means that instead of actually doing all 5 separate trainings, there's a faster mathematical shortcut that gets you the same answer as if you'd done all those trainings. This is especially helpful in larger datasets.

$$y_i, LOO = \hat{y}_i + \frac{Residual_i}{1 - h_i}$$

$$Residual_i = y_i - \hat{y}_i$$

Where h_i is the "leverage" of the i^{th} data point, a value that ranges between 0 and 1. The leverage tells us how much the i^{th} data point influences its own prediction. Higher leverage means that the data point has more influence.

$$h_i = \frac{1}{n} + \frac{(x_i - \bar{x})^2}{\sum_{j=1}^{n} (x_j - \bar{x})^2} \text{ or } H = X(X^T X)^{-1} X^T$$

- 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.

Example:

Imagine you have a dataset of 12 students and their scores on a test:

Student	Score
A	10
B	20
C	30
D	40
E	50
F	60
G	70
H	80
I	90
J	100
K	110
L	120

If we divide the scores into quartiles:

1. Q1 (0 to 30): A, B, C
2. Q2 (31 to 60): D, E, F
3. Q3 (61 to 90): G, H, I
4. Q4 (91 to 120): J, K, L

Now, let's say you want to divide the dataset into 3 folds using stratified sampling.

A straightforward division (without stratification) might look like:

- Fold 1: A, B, C, D
- Fold 2: E, F, G, H
- Fold 3: I, J, K, L

Notice that Fold 1 only contains the lower scores, while Fold 3 only contains the higher scores. This isn't a fair representation.

With stratified sampling, we'd aim to have one student from each quartile in each fold:

- Fold 1: A (from Q1), D (from Q2), G (from Q3), J (from Q4)
- Fold 2: B (from Q1), E (from Q2), H (from Q3), K (from Q4)
- Fold 3: C (from Q1), F (from Q2), I (from Q3), L (from Q4)

This way, each fold has a representative distribution of the entire range of scores. This is especially beneficial when evaluating a predictive model because each fold is a mini-representation of the whole dataset, leading to more reliable and generalizable results.

- 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.

Code Example:

```
import numpy as np
from sklearn.model_selection import KFold, StratifiedKFold

# Sample data: 12 students with their scores
students = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L']
scores = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120]

# Convert scores to quartiles
quartiles = np.digitize(scores, bins=[30, 60, 90])

# Non-stratified 3-fold
print("Non-Stratified:")
```

```
kf = KFold(n_splits=3, shuffle=True, random_state=42)
for train, test in kf.split(students):
    print("Train:", [students[i] for i in train], "Test:", [students[i] for
i in test])

print("\nStratified:")
# Stratified 3-fold
skf = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)
for train, test in skf.split(students, quartiles):
    print("Train:", [students[i] for i in train], "Test:", [students[i] for
i in test])
```

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.

Example:

Imagine you're building a machine learning model and using 5-fold cross-validation. For each fold, you'll have a training set and a validation set.

Incorrect (Outside Cross-Validation):

1. Use the **entire dataset** to select important features.
2. Use the **entire dataset** to choose the best hyperparameters (using some criterion).
3. Perform 5-fold cross-validation to assess model performance.

In this approach, the feature selection and hyperparameter tuning have "seen" the entire dataset, including what will later be treated as validation data in each fold. This can result in an inflated, overly optimistic estimate of the model's performance.

Correct (Inside Cross-Validation):

1. For each fold in the 5-fold cross-validation:
    - Use only the **training data** of that fold to select important features.
    - Use only the **training data** of that fold to decide on the best hyperparameters.
    - Train your model on the training data of that fold.
    - Validate the model on the validation set of that fold.

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 5*K* 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?

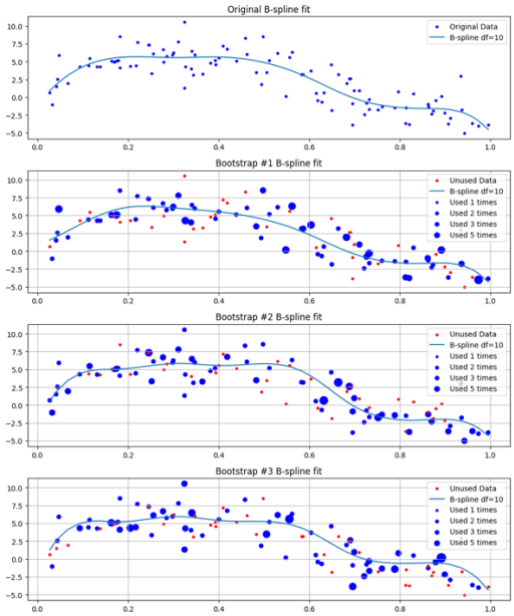
- Just repeatedly split the data into a training and test set.
- This will be similar to the repeated cross-validation, but can be simpler to code.
- Hold out 20% of the data and repeating 5 times is compared to $K = 5$ fold cross-validation.

3.9 Bootstrap Out-of-Bag

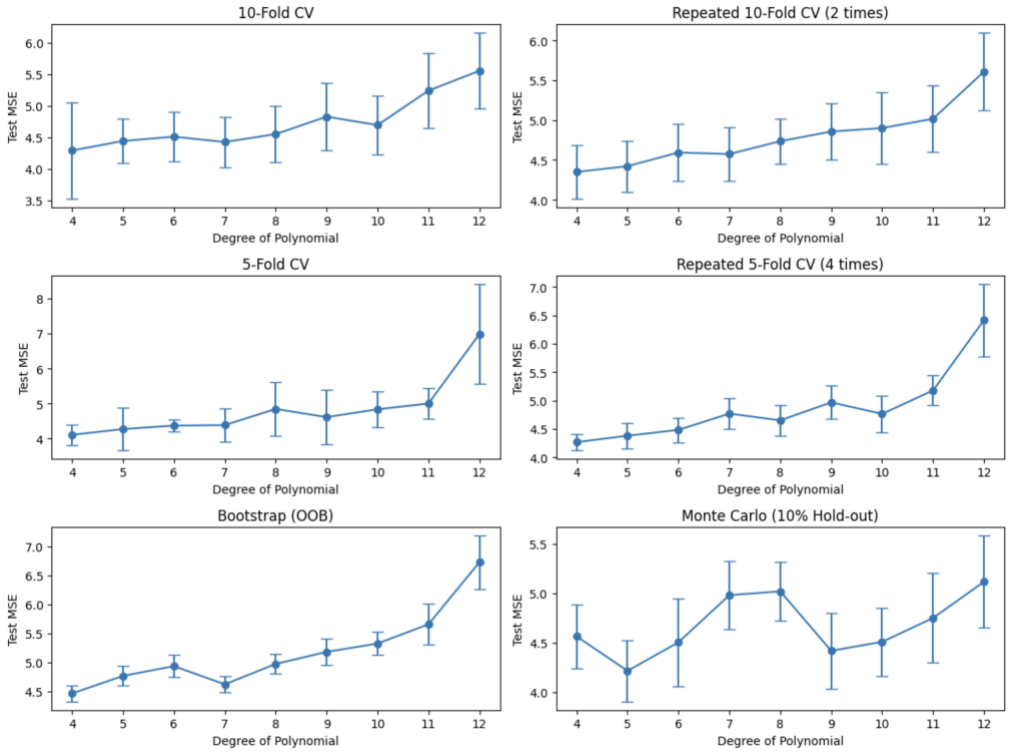
Because about 37% of the bootstrap data will be used for model assessment (testing data), it is *similar* to $K = 3$ cross-validation (if repeated 3 times). Some differences:

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

Let's look at a few bootstrap fits:



3.10 Resampling Comparison



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, $H = X(X^T X)^{-1}X^T$.
  - 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.

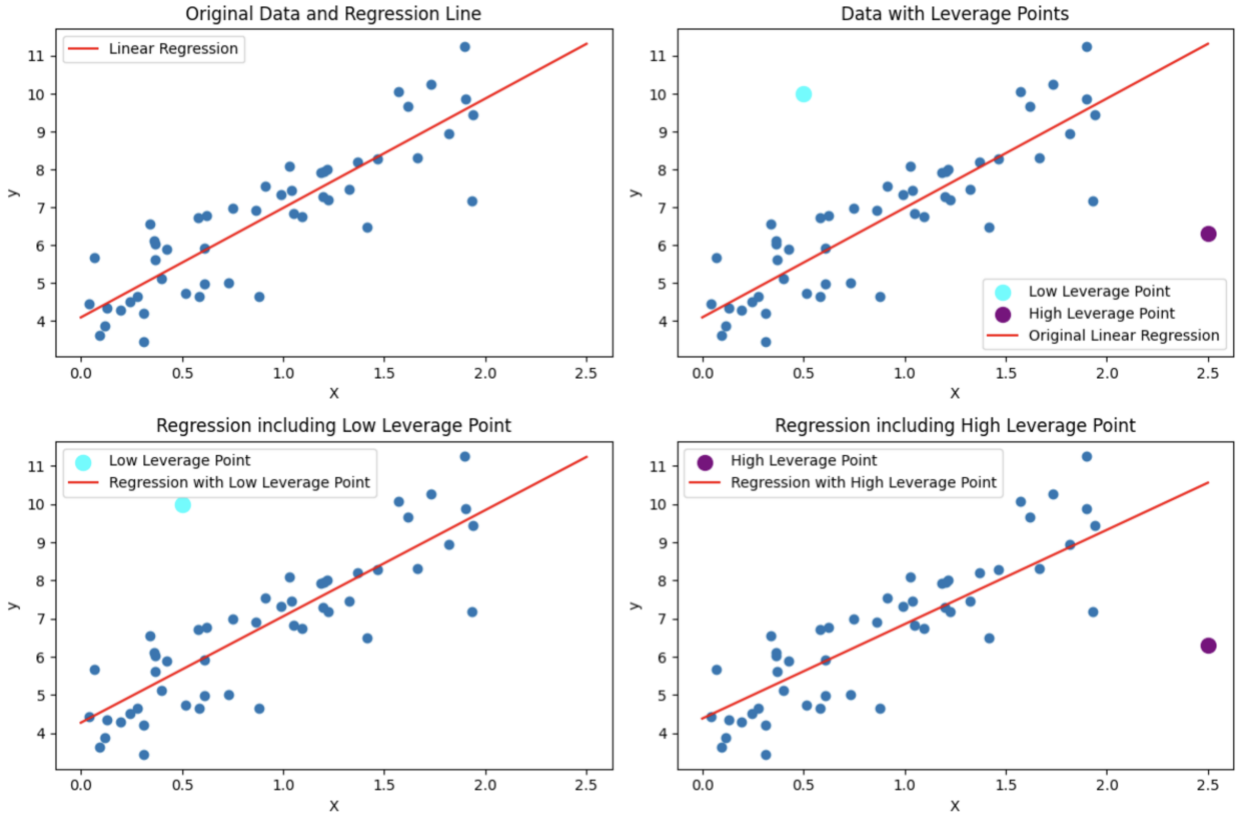
- This takes away the computational burden of n model fits! The model is fit once to the full data and *corrected* for in the above equation.

- A similar, but even faster to compute, version is the *Generalized Cross-Validation*

$$\mathrm{GCV}(\omega) = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{f}_\omega(x_i)}{1 - \frac{tr(H)}{n}}\right)^2 = \frac{\mathrm{MSE}_\omega}{\left(1 - \frac{tr(H)}{n}\right)^2}$$

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

   a. 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 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 Model Assessment

If it was 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 (ω_1, ω_2, …)
  - 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, $\hat{f}_1$, $\hat{f}_2$, …
- **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:

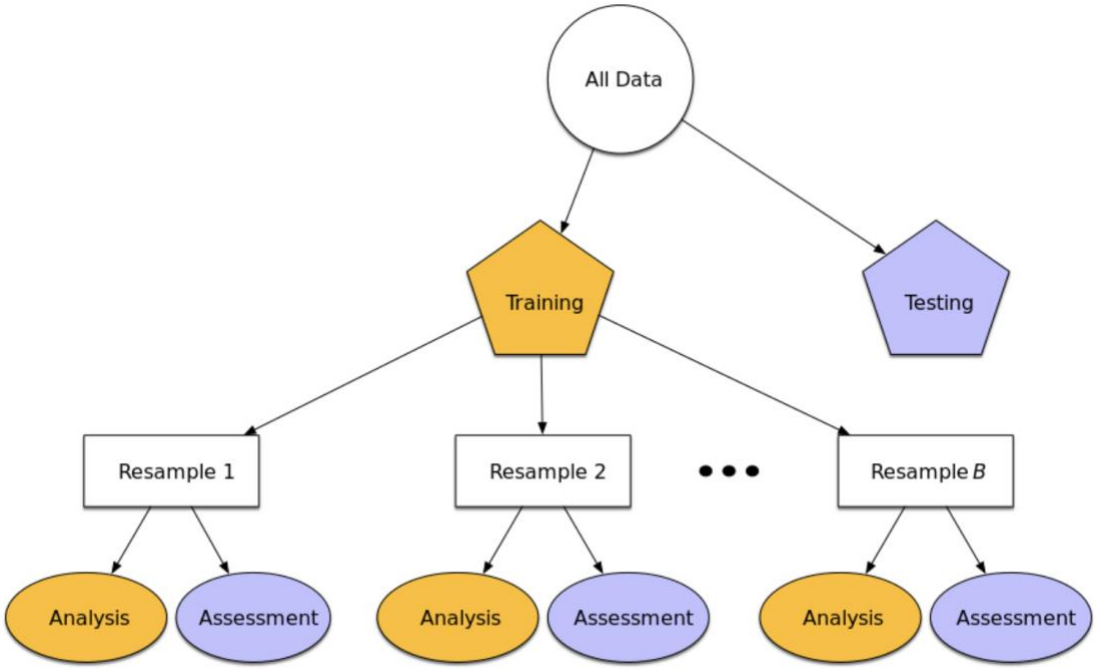
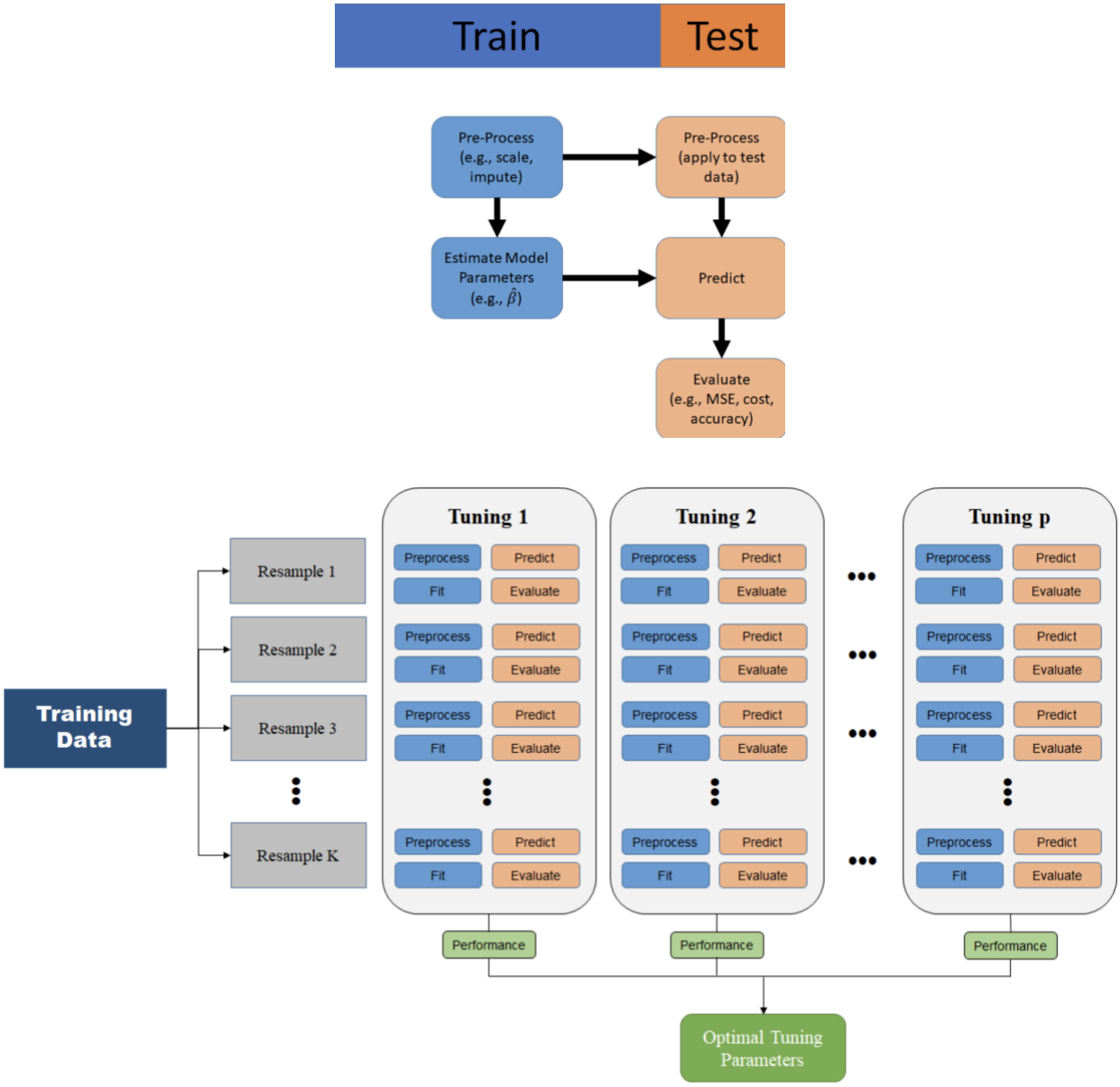


Figure taken from: **Feature Engineering and Selection**: **A Practical Approach for Predictive Models** by Max Kuhn and Kjell Johnson (2019-06-21)

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.


Train
Test
Pre-Process (e.g., scale, impute)
Pre-Process (apply to test data)
Estimate Model Parameters (e.g., $\hat{\beta}$)
Predict
Evaluate (e.g., MSE, cost, accuracy)
Training Data
Resample 1
Resample 2
Resample 3
Resample K
Tuning 1
Tuning 2
Tuning p
Preprocess
Predict
Fit
Evaluate
Performance
Optimal Tuning Parameters