Generalization: Why Training Error Lies

Introduction

Every supervised method in this course – trees, k-NN, regression, neural networks – is judged by one question: **how well does it predict on data it has never seen?**

Today we make that question precise, and see why the most natural answer (pick the model with the lowest training error) is wrong.

The Supervised Learning Problem

- You are given some example data, which we'll think of abstractly as tuples $\{(\mathbf{x}_i, y_i) \mid i = 1, \dots, N\}$.
  - $\mathbf{x}_i$ are the inputs or independent variables. Its components are called **features.**
  - y_i are the outputs or dependent variables (a.k.a. labels, targets or ground truth).
- Your goal is to learn a rule that allows you to predict y_j for some $\mathbf{x}_j$ that is **not** in the example data you were given.
- The collection $\{(\mathbf{x}_i, y_i) \mid i = 1, \dots, N\}$ is called the **training data.**
- The collection $\{(\mathbf{x}_j, y_j) \mid j = 1, \dots, M\}$ is called the **test data.**

The Supervised Contract

What do we have to assume to make this problem tractable?

We assume two things:

1. There is a set of functions that could be used to predict y_i from $\mathbf{x}_i$.
    - This allows us to turn the learning problem into one that searches through this set for the "right" function.
    - However, this set is probably **very** large!
2. The rule for predicting y_i from $\mathbf{x}_i$ is the same as the rule for predicting y_j from the new item $\mathbf{x}_j$.
    - Speaking probabilistically, we say that $(\mathbf{x}_i, y_i)$ and $(\mathbf{x}_j, y_j)$ are drawn **independently from the same distribution** (i.i.d.).

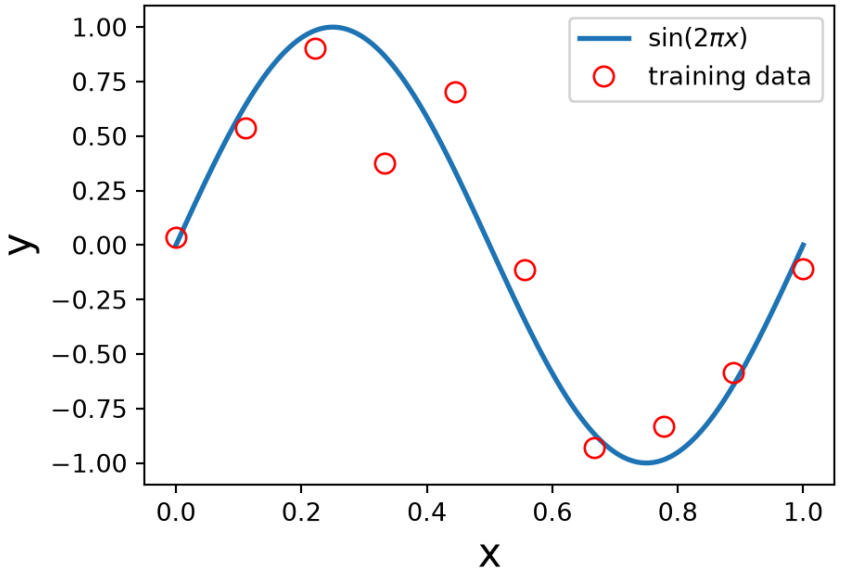
A Toy Example: Polynomial Curve Fitting

> **Note**
>
> Based on *Pattern Recognition and Machine Learning,* Christopher Bishop (2006), Section 1.1.

We generate $N = 10$ points x_i equally spaced in $[0, 1]$, and set

$$y_i = \sin(2\pi x_i) + \epsilon_i,$$

where ϵ_i is Gaussian noise. Many data sets are like this: some component of y depends on x, and some component we treat as random – "noise" – because it depends on features we cannot see.

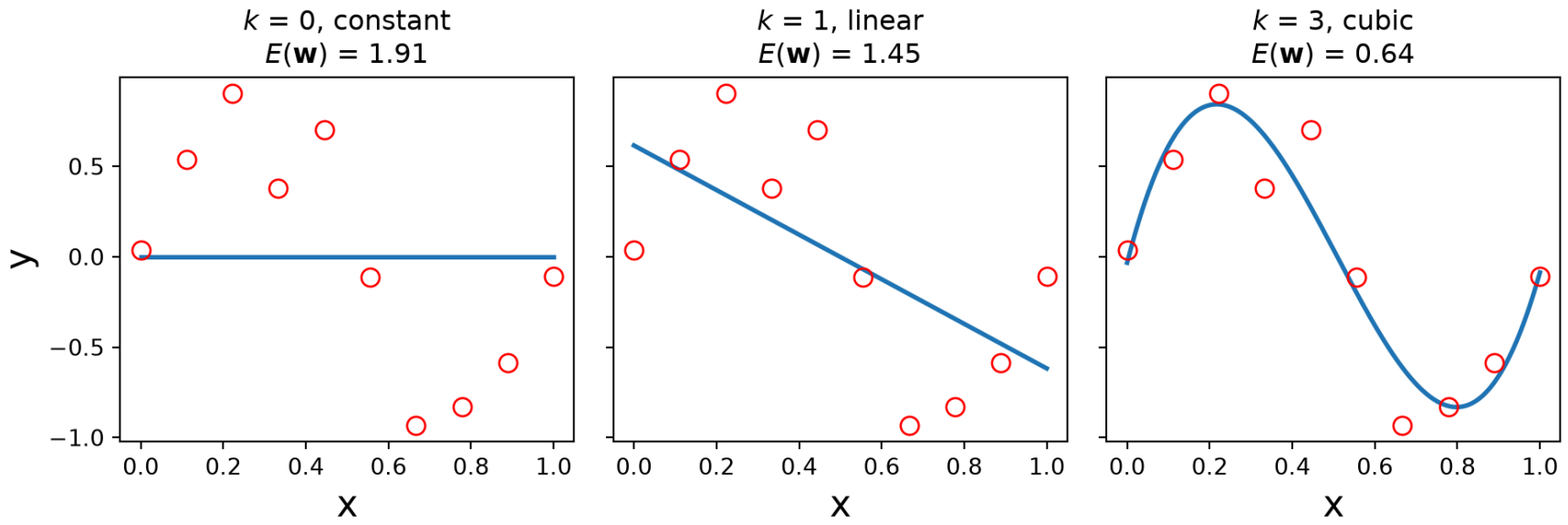


Model Selection

- $\mathbf{w} = (w_0, \ldots, w_k)$ are the **parameters** of the model; least squares finds the $\mathbf{w}^*$ that **minimizes the error on the training data.**
- But what about choosing k, the order of the polynomial?
- A cubic ($k = 3$) is a **different model** from a quadratic ($k = 2$). The problem of choosing k is called **model selection.**

Model Selection, cont.

Let's look at constant (order 0), linear (order 1), and cubic (order 3) models, each fit using the least squares criterion:



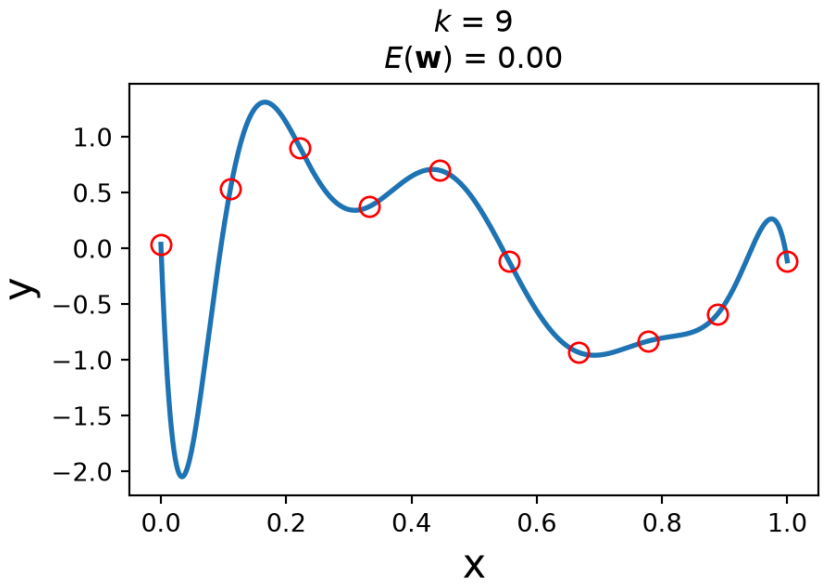
So it looks like a third-order polynomial ($k = 3$) is a good fit!

How do we know it's good? Well, the training error $E(\mathbf{w})$ is small.

Make training error smaller?

Yes, we can, if we increase the order of the polynomial.

We can reduce the error to zero by setting $k = 9$, we get the following polynomial fit to the data:



So … is the 9th order polynomial a “better” model for this dataset?

Absolutely not!

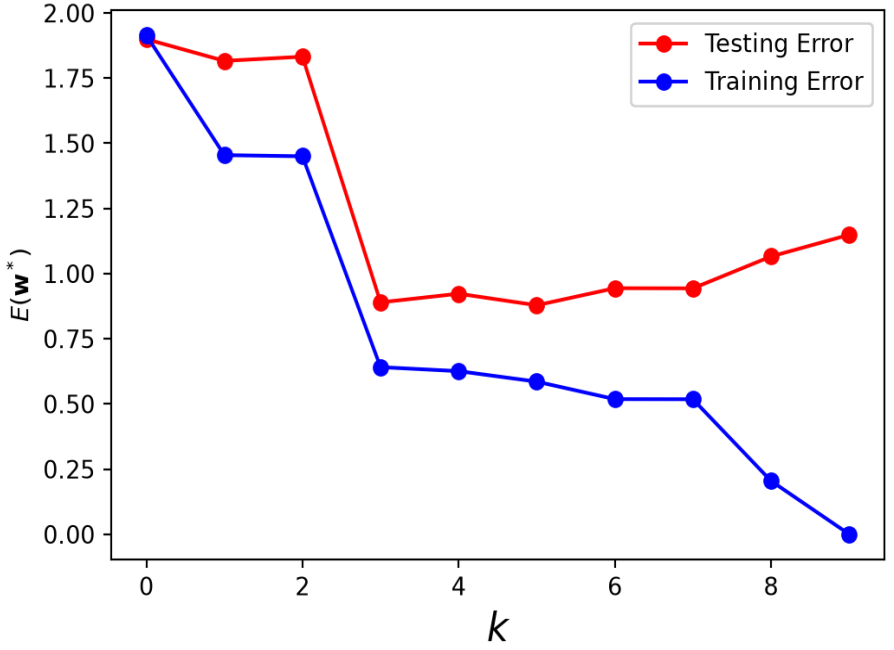
Why?

- Informally, the model is very “wiggly”. It seems unlikely that the real data generation process is governed by this curve.
- In other words, we don’t expect that, if we had **more** data from the same source, that this model would do a good job of fitting the additional data.
- We want the model to do a good job of predicting on **future** data.
- This is called the model’s **generalization** ability.

The 9th degree polynomial would seem to have poor generalization ability.

Generalization Error

- To assess generalization, we evaluate each polynomial on new **test** data – not part of the training set. (We know how the data is generated, so we can easily make more.)
- As we increase the order of the polynomial, the *training error* always declines.
- Eventually, the training error reaches zero.
- However, the *test error* does not – it reaches its smallest value at $k = 3$, a cubic polynomial.
- The phenomenon in which *training error* declines, but *testing error* does not, is called **overfitting.**
- In a sense we are fitting the training data



Overfitting

There are two ways to think about overfitting:

1. The number of parameters in the model is too large, compared to the size of the training data. We can see this in the fact that we have only 10 training points, and the 9th order polynomial has 10 coefficents.
2. The model is more complex than the actual phenomenon being modeled. As a result, the model is not just fitting the underlying phenomenon, but also the noise in the data.

These suggest techniques we may use to avoid overfitting:

1. Increase the amount of training data. All else being equal, more training data is always better.
2. Limit the complexity of the model. Model complexity is often controlled via **hyperparameters**.
3. Use regularization – constrain the model to avoid overfitting.

Bias and Variance

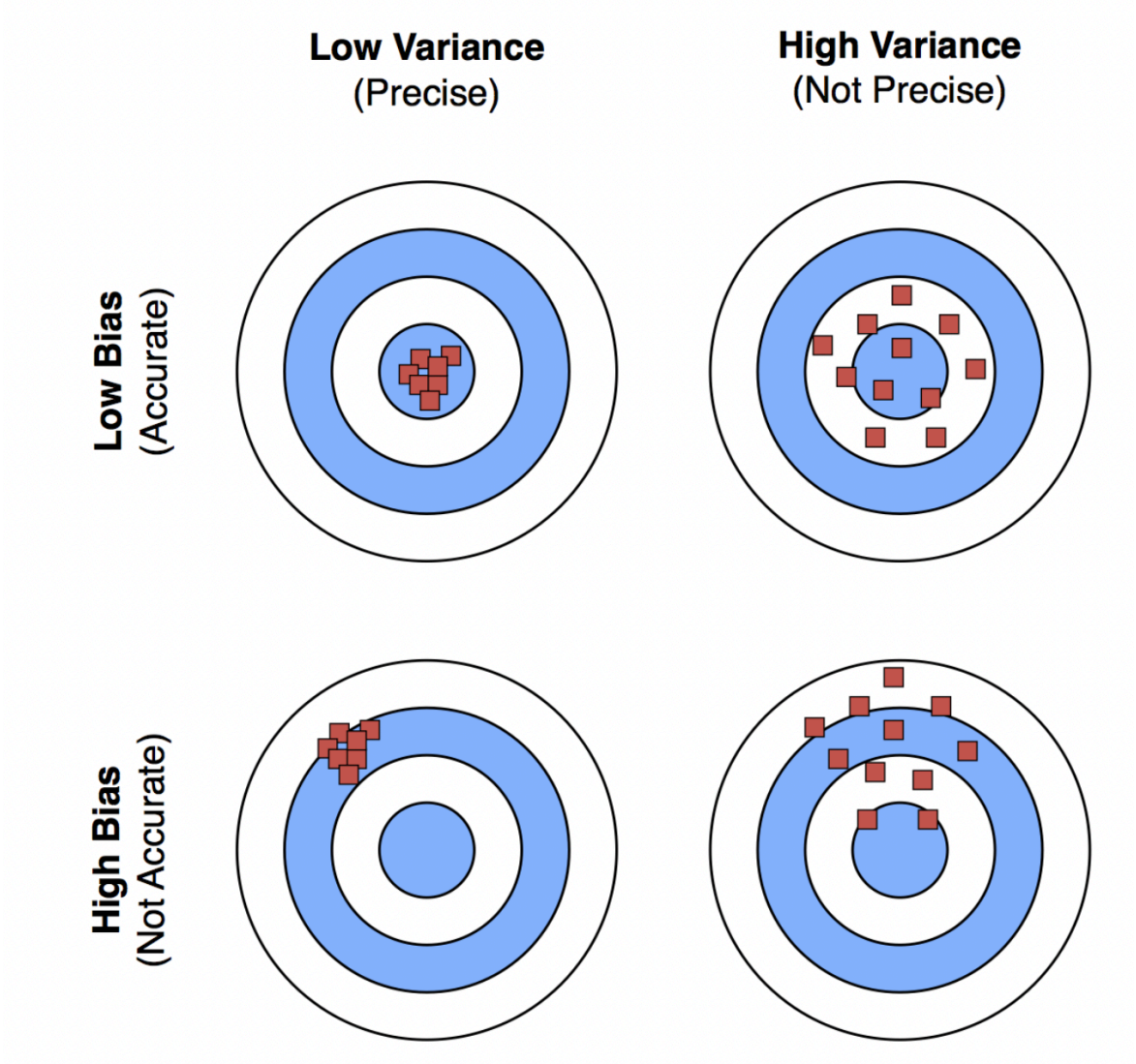
Overfitting is one of **two ways a model can be wrong.**

Bias

- Error due to an overly simplistic model.
- High bias: the model **underfits** – it misses the real pattern. (Our $k = 0$ or $k = 1$ polynomial.)
- Low bias: the model is flexible enough to capture the underlying pattern.

Variance

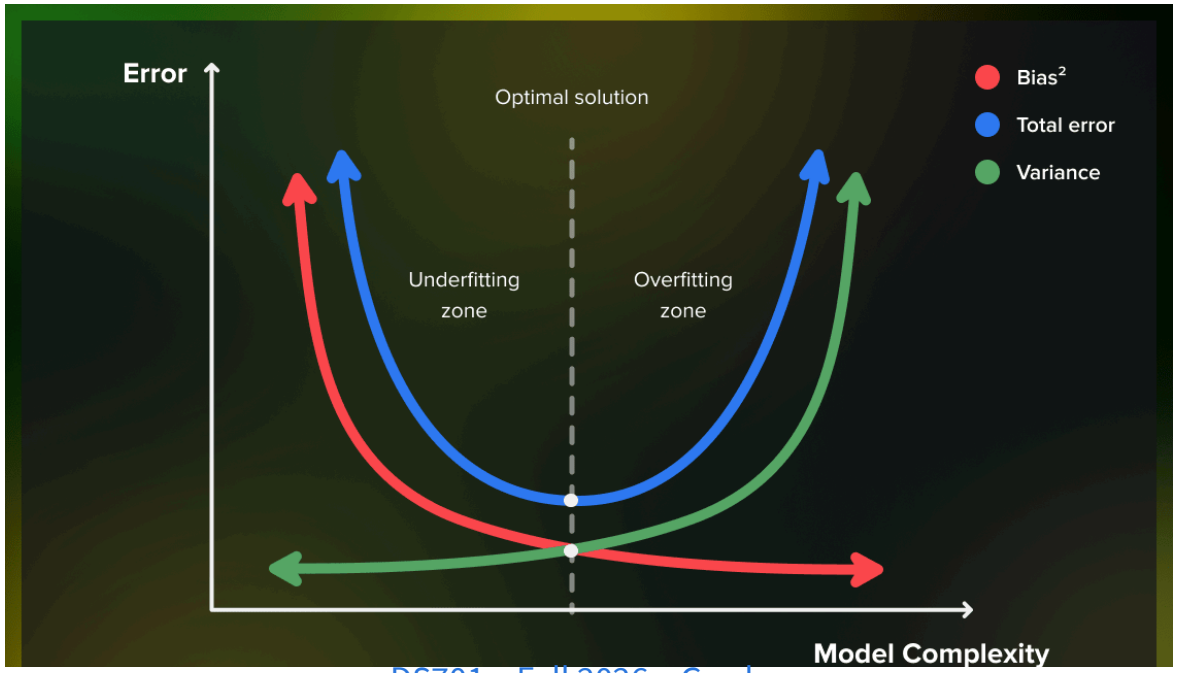
- Error due to an overly complex model.
- High variance: the model **overfits** – refit it on a different training sample from the same distribution and you get a very different answer. (Our $k = 9$ polynomial.)
- Low variance: predictions are stable across different training sets.


Low Variance
(Precise)
High Variance
(Not Precise)
Low Bias
(Accurate)
High Bias
(Not Accurate)

Bias-Variance Trade-Off

Goal: find the model complexity that minimizes **total** error.

- Low bias and low variance are both ideal, but hard to achieve simultaneously: making a model more flexible lowers bias and raises variance.
- The U-shaped test error curve we just saw is this trade-off in action – $k = 3$ is the sweet spot.



Parameters and Hyperparameters

- Notice that the model selection problem required us to choose a value k that specifies the order of the polynomial model.
- The values $w_0, w_1, \ldots, w_k$ are the **parameters** of the model; they are learned from the training data.
- In contrast, k is called a **hyperparameter.**

Holding Out Data

- We want to avoid overfitting, which occurs when a model fails to generalize – that is, when it has high error on data that it was not trained on.
- So: we will hold some data aside, and **not** use it for training the model, but instead use it for testing generalization ability.
- Let's assume that we have 20 data points to work with. `scikit-learn`'s `train_test_split()` splits them **randomly** into training and testing sets:

```
N = 20
x = np.linspace(0, 1, N)
y = np.sin(2 * np.pi * x) + default_rng(2).normal(size = N, scale = 0.20)

import sklearn.model_selection as model_selection

x_train, x_test, y_train, y_test = model_selection.train_test_split(
        x, y, test_size = 0.5, random_state = 0)

print(f'Number of items in training set: {x_train.shape[0]}, in testing set: {x_test.shape[0]}')
```

```
Number of items in training set: 10, in testing set: 10
```

Grid Search

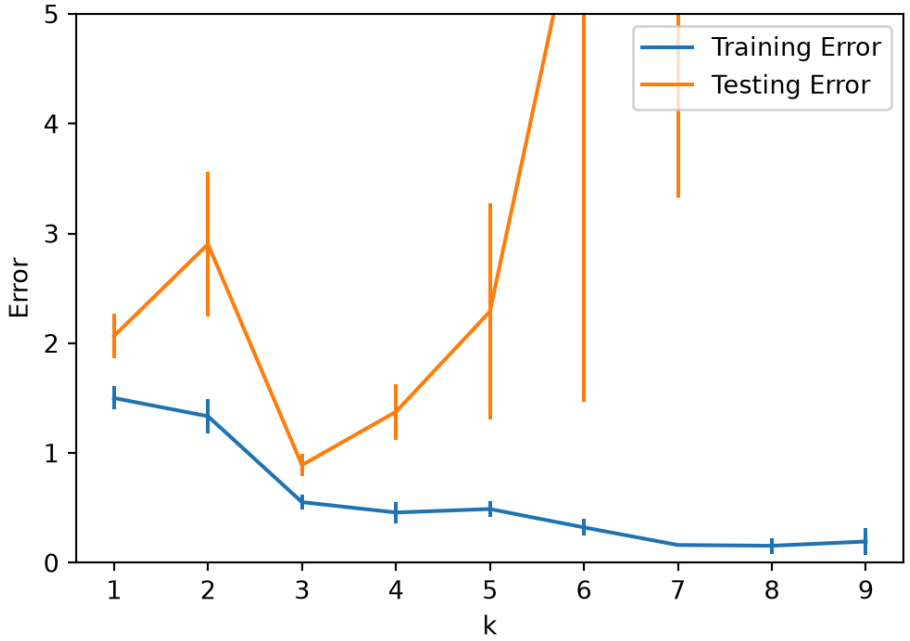
Our strategy will be, for each possible value of the hyperparameter k:

- randomly split the data 5 times
- fit the model on the training data
- test the model on the testing data
- compute the mean testing and training error over the 5 random splits

Trying all candidate values of the hyperparameter this way is called a **grid search**. (What are good candidate values? It depends on the problem, and may involve trial and error.)

Grid Search, cont.

Mean error for each value of k, with its standard error ($\sigma/\sqrt{n}$) over the 5 splits:



From this plot we can conclude that, for this dataset, a polynomial of degree $k = 3$ shows the best generalization ability.

Hold Out Strategies

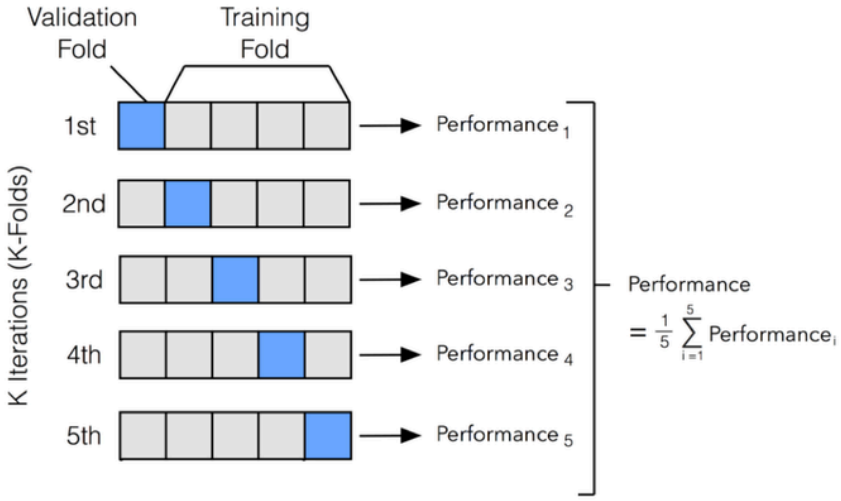
- Deciding how much, and which, data to hold out depends on a number of factors.
- In general we'd like to give the training stage as much data as possible to work with.
- However, the more data we use for training, the less we have for testing – which can decrease the accuracy of the testing stage.
- Furthermore, any single partition of the data can introduce dependencies – any class that is overrepresented in the training data will be underrepresented in the test data.

K-Fold Cross-Validation

In K-**Fold Cross-validation**, the data is partitioned *once*, and then each partition is used as the test data once.

This ensures that all the data gets equal weight in the training and in the testing.

- We divide the data into k "folds".
- The value of k can vary up to the size of the dataset.
- The larger k we use, the more data is used for training, but the more folds must be evaluated, which increases the time required.
- In the extreme case where k is equal to the data size, then each data item is held out by itself; this is called "leave-one-out".



How This Plays Out This Term

Every supervised lecture returns to today's ideas – the same trade-off, a different knob:

- Decision Trees: overfit via **depth**; random forests reduce variance by averaging.
- k-NN: choosing k – small k overfits, large k underfits.
- Regression: **regularization** penalizes complexity directly.
- Neural Networks: **early stopping** – halt training when held-out error starts to rise.

In every case, the hyperparameter is chosen by cross-validation, never by training error.

Conclusions

We have seen strategies that allow us to learn from data:

- Define a set of possible models
- Define an error function that tells us when the model is predicting well
- Using the error function, search through the space of models to find the best performer

We've also seen that there are some subtleties to this approach that must be dealt with to avoid problems:

- Simply using the model that has lowest error on the training data will **overfit**.
- Overfitting (variance) and underfitting (bias) are the two ways a model can be wrong; the goal is to balance them.
- We need to **hold out** data to assess the generalization ability of each trained model.
- We control model complexity using hyperparameters.
- We choose the best hyperparameters based on performance on held out data.