k-Nearest Neighbors

Ivan Corneillet

Data Scientist

GENERAL ASSEMBLY

Learning Objectives

After this lesson, you should be able to:

- Define and give examples of classification; implement a simple classifier by hand
- Explain the k-Nearest Neighbors algorithm; build a k-Nearest Neighbors model using *sklearn*
- Understand the fundamentals of evaluating and tuning classifiers; define error metrics for classification problems, goodness of fit, bias, and variance

Classification

k-Nearest Neighbors is a supervised learning algorithm for regression or classification (cont.)

	Regression (continuous predictions; i.e., how much or how many?)	Classification (categorical predictions; i.e., is this A, B or C?)
Supervised a.k.a., predictive modeling (generalization; make predictions)	k-Nearest Neighbors ✓	k-Nearest Neighbors ✓
Unsupervised *(representation; extract structure)*		

When ❻ BUILD*ing* **a model**, our data needs to be in the form of a **feature matrix $\boldsymbol{X}$** (i.e., the stimuli, e.g., *“ring bell”*) and a **response vector $\boldsymbol{y}$** (i.e., the response, e.g., *“dog salivates”*)

Feature Matrix X

	col0	col1	col2	col3
row0				
row1				
row2				
row3				

Response Vector y

	col
row0	
row1	
row2	
row3	

Response vector y (cont.)

Regression

Response vector $\boldsymbol{y}$

	col *e.g. price*
row0	\$1.1M
row1	\$.9M
row2	\$1.5M
row3	…

Classification

Response vector $\boldsymbol{c}$

(renamed from y to c for label classes)

	col *e.g., animal*
row0	“dog”
row1	“cat”
row2	“bird”
row3	…

Activity | Two simple classification problems

EXERCISE

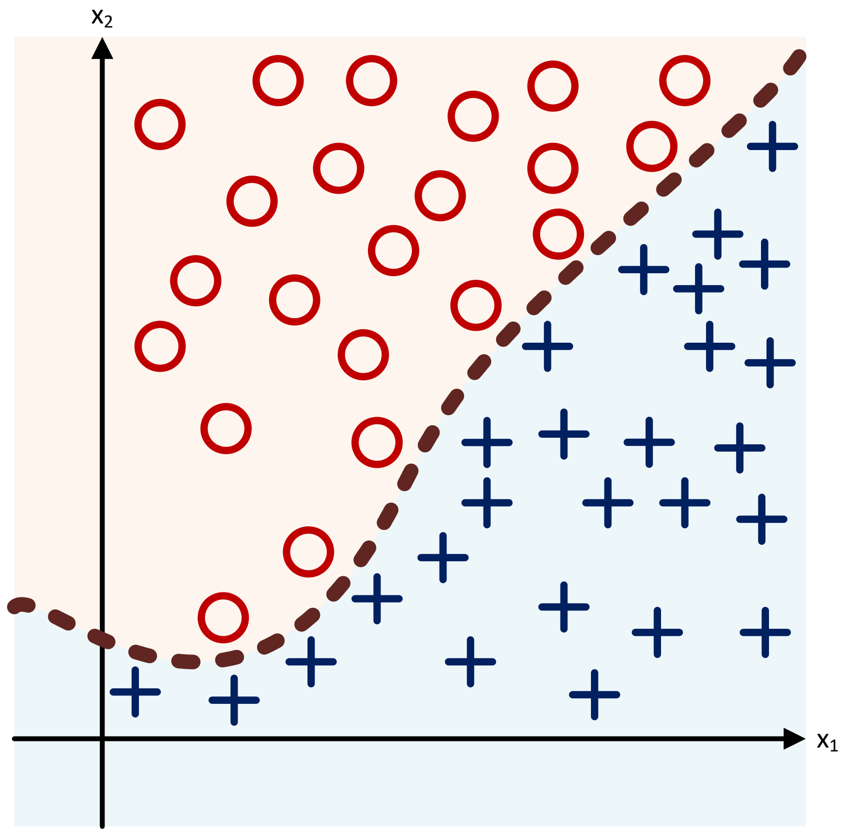
DIRECTIONS (5 minutes)

1. Consider the following datasets in the handouts:
   a. Feature matrix X with feature columns x_1 and x_2
   b. Response vector y with class labels O and +
   c. Hand-draw the boundary between the two class labels. What is the relationship linking x_1, x_2, O and +?
2. When finished, share your answers with your table

DELIVERABLE

Answers to the above questions

A classifier aims to isolate the response vector y's class label by splitting the feature space modeled by the feature matrix X



The Iris dataset: 3 class labels of iris plants (*Setosa*, *Versicolor*, and *Virginica*); 50 instances in each class label

CASE STUDY

Iris Setosa

Iris Versicolor

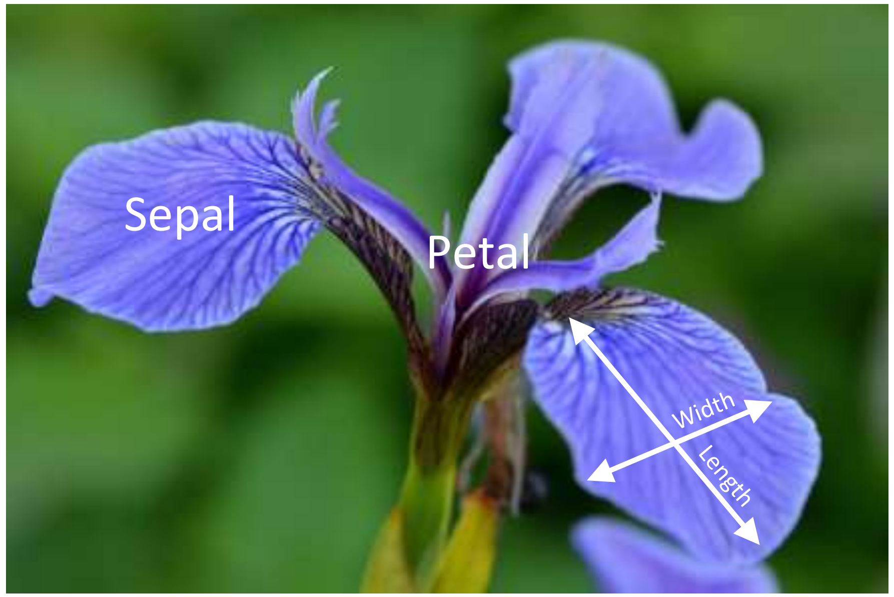
Iris Virginica

Source: Flickr

The Iris dataset (cont.)

CASE STUDY

- Can we teach a machine to identify the type of iris based on the following four attributes?
  - Sepal length and width
  - Petal length and width



Activity | Iris dataset: Exploratory Data Analysis

EXERCISE

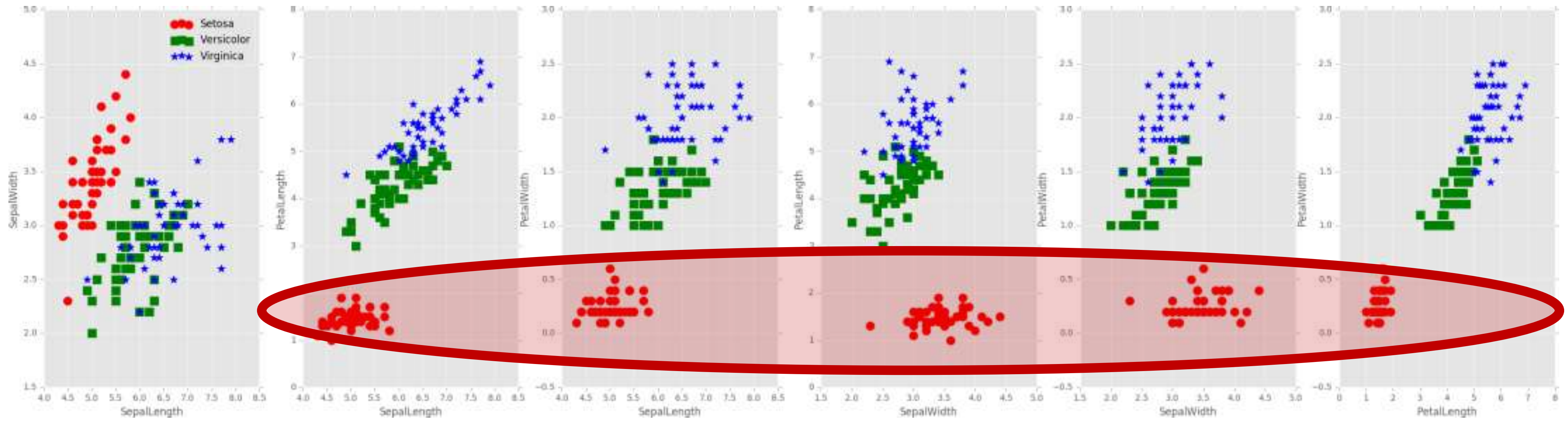
DIRECTIONS (10 minutes)

1. Using the Iris dataset (`dataset-05-iris.csv` in the `datasets` folder), perform exploratory analysis between $SepalLength$, $SepalWidth$, $PetalLength$, and $PetalWidth$ (the *feature* variables) and $Species$ (the *response* variable). How can you use these features to separate one species from the other two?
2. When finished, share your answers with your table

DELIVERABLE

Answers to the above questions

Activity | Iris dataset: Exploratory Data Analysis (cont.)



Activity | Iris dataset: hand-coded classifier

EXERCISE

DIRECTIONS (10 minutes)

1. Using the Exploratory Data Analysis, write a classifier in Python to separate the class labels (return '`Setosa`', '`Versicolor`', or '`Virginica`'). How would you measure how good is your classifier?
2. When finished, share your answers with your table

DELIVERABLE

Answers to the above questions

Accuracy and misclassification rate

- Accuracy (rate)
  - How many observations that we predicted were correct?
  - This is a value we want as high as possible

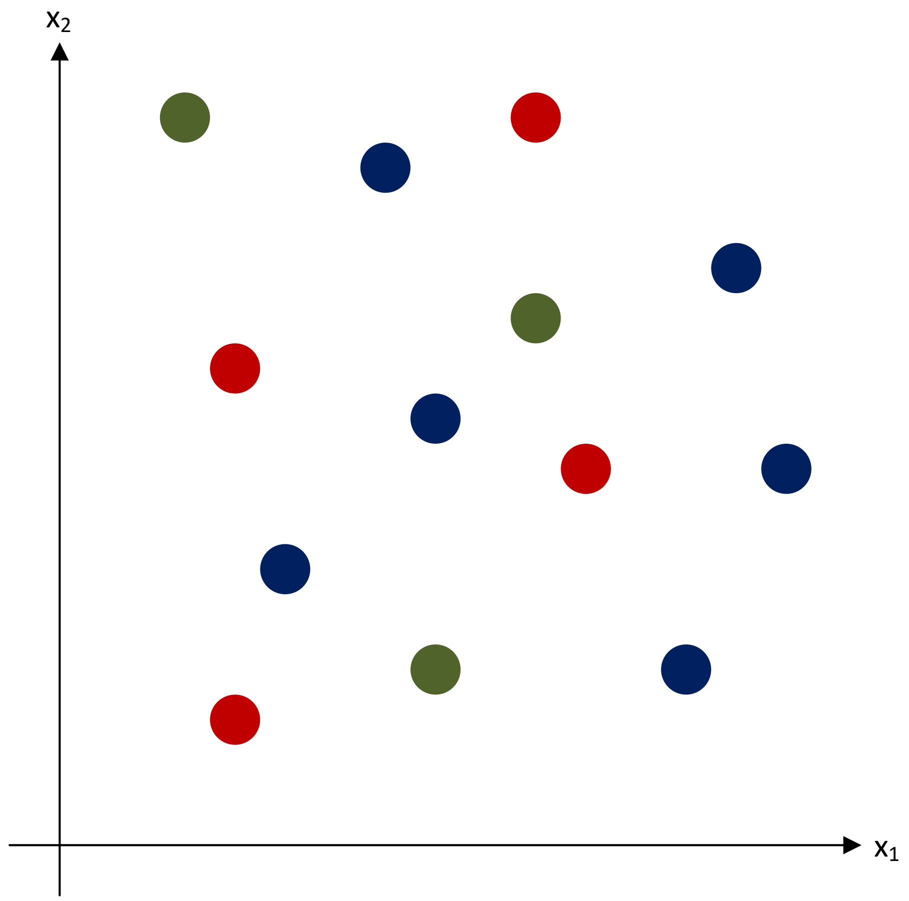
- Misclassification rate
  - Of all the observations we predicted, how many were incorrect?
  - This is a value we want as low as possible
  - Directly opposite of accuracy
    - (Pick one or the other; effectively they are the "same")

DS

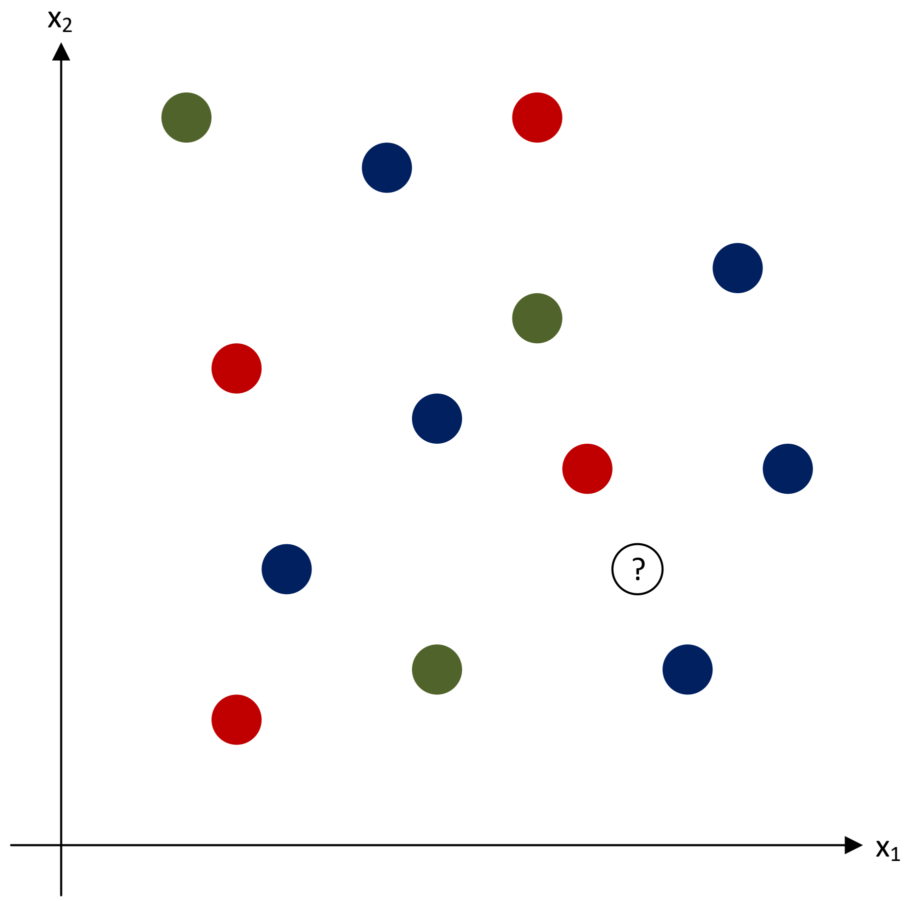
k-Nearest Neighbors

k-Nearest Neighbors

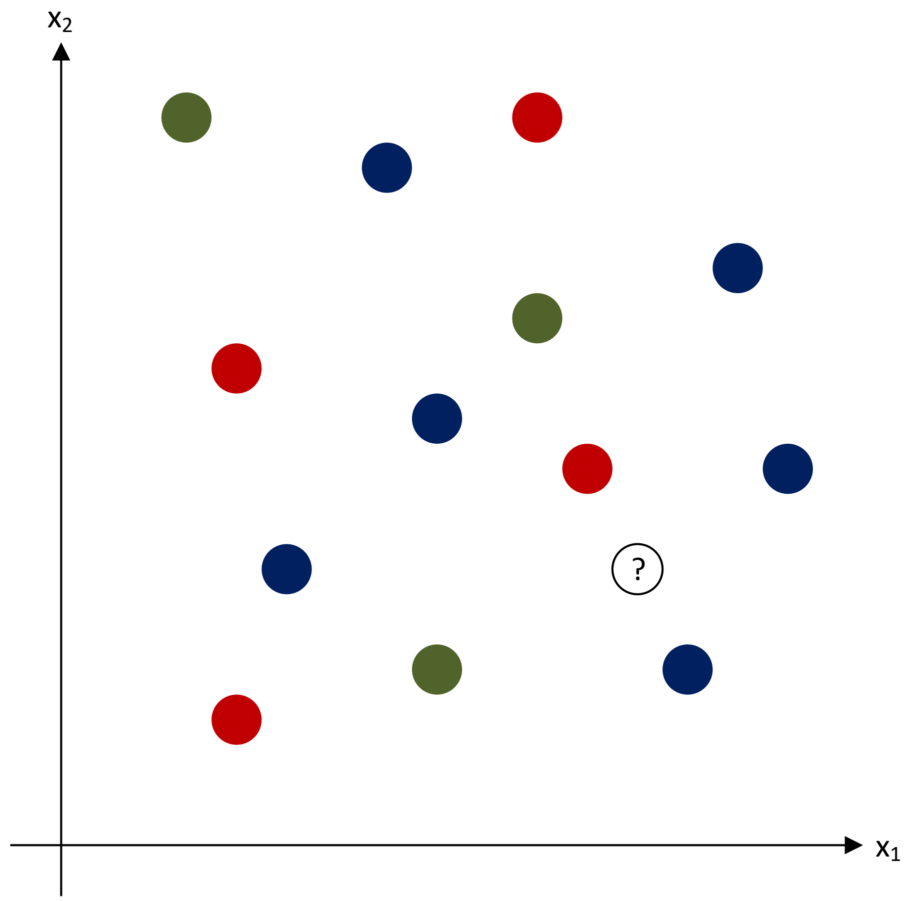
- k-Nearest Neighbors is a classification algorithm that makes a prediction based upon the closest data points



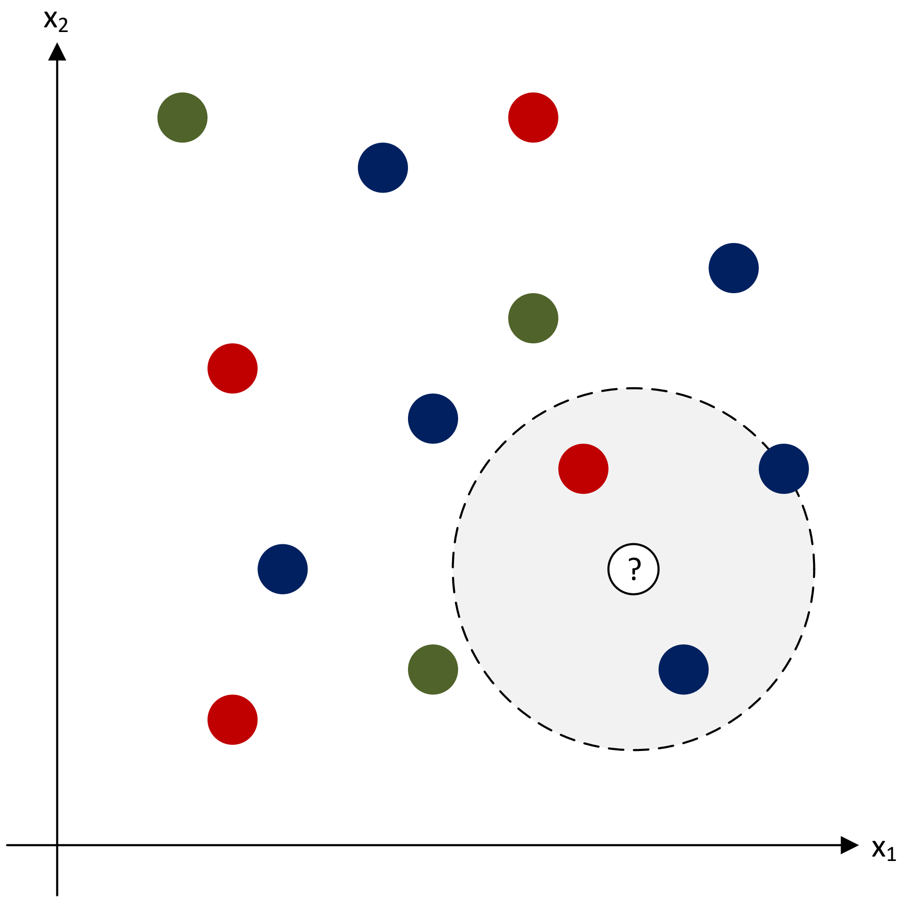
k-Nearest Neighbors | How would you predict the color of the "question mark" point?



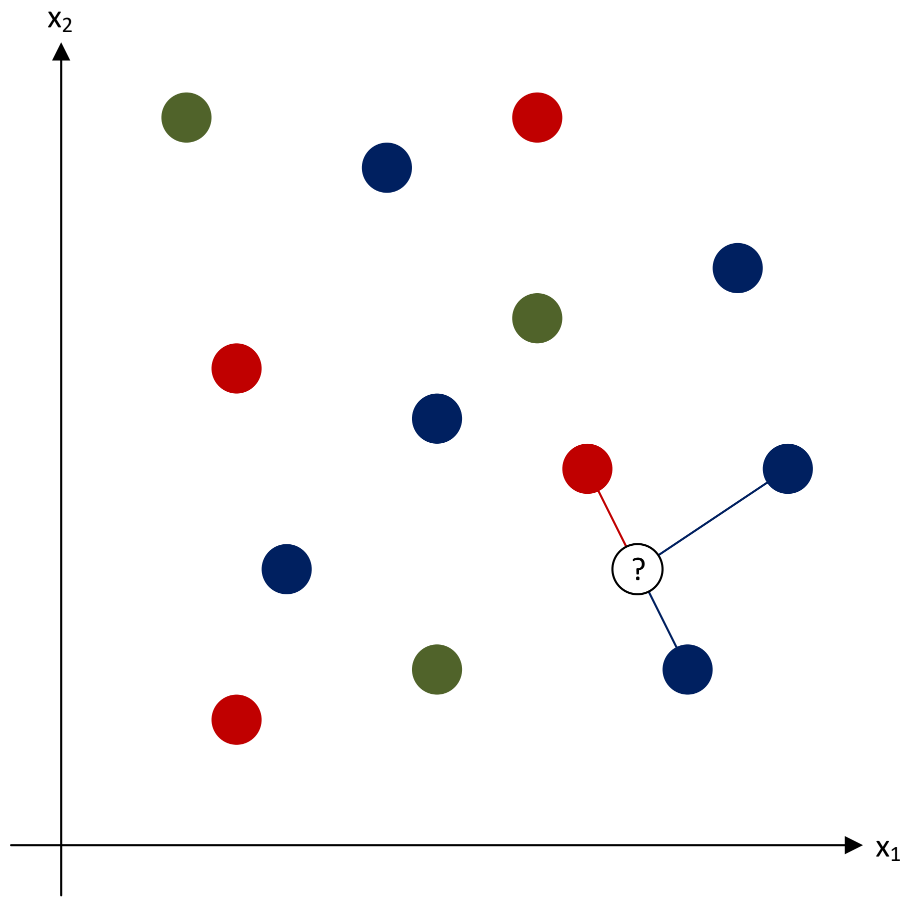
k-Nearest Neighbors | ❶ Pick a value for k, e.g., $k = 3$



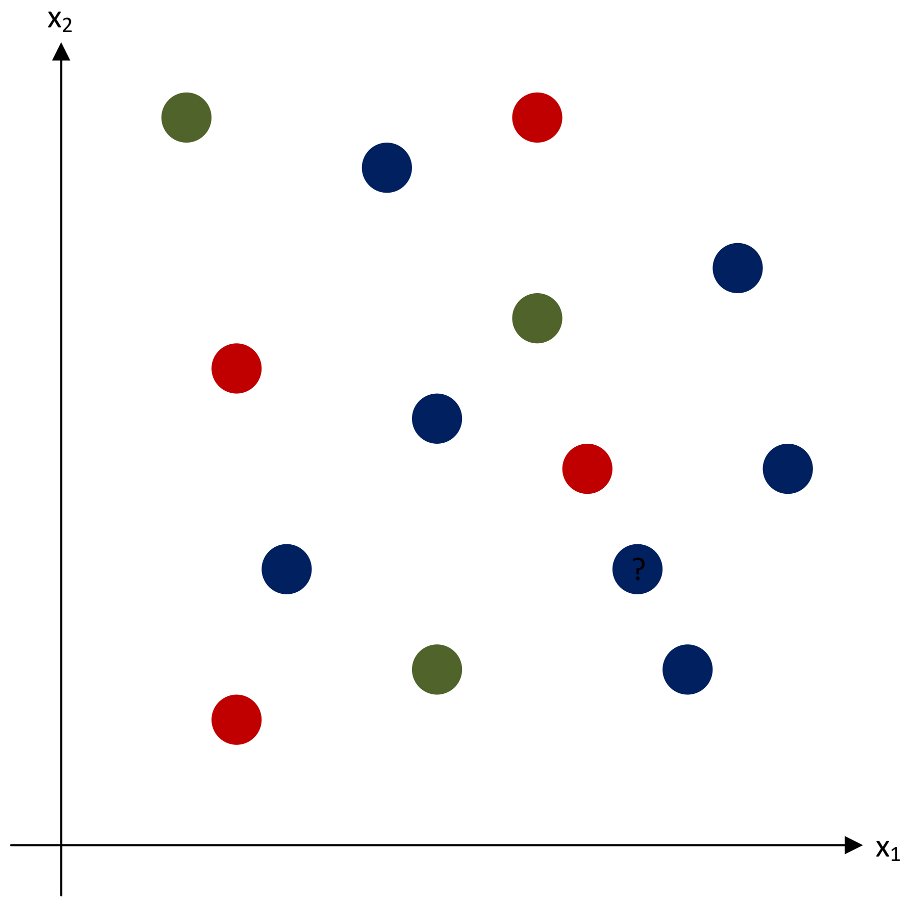
k-Nearest Neighbors | ❷ Calculate the distance to all other points; given those distances, pick the k closest points



k-Nearest Neighbors | ❸ Calculate the probabilities of each class label given those points: $\frac{1}{3}$ "red", $\frac{2}{3}$ "blue"

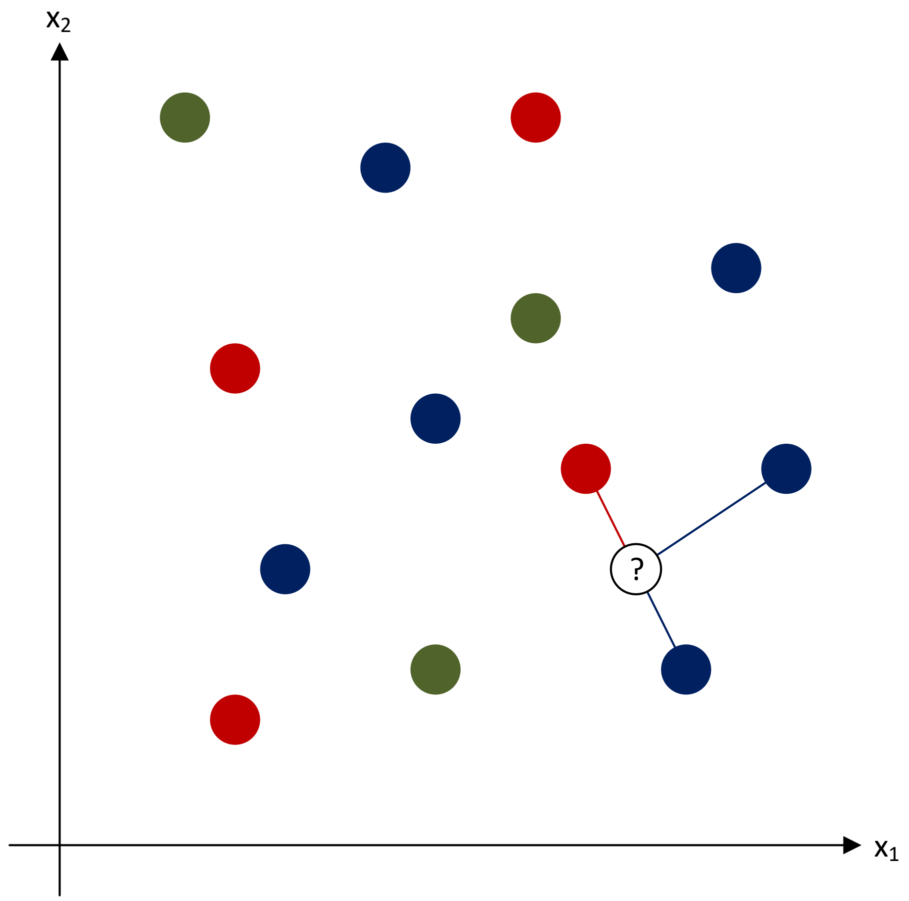


k-Nearest Neighbors | ❹ The original point is classified as the class label with the largest probability (“votes”): “blue”



k-Nearest Neighbors (cont.)

- k-Nearest Neighbors uses distance to predict a class label
- This application of distance is used as a measure of similarity between classifications
  - We are using shared traits to identify the most likely class label



Feature Normalization

Why is feature scaling important?

- K-Nearest Neighbors uses the Euclidean distance to find the closest neighbors:

$$d(A,B) = \sqrt{\left(x_1^A - x_1^B\right)^2 + \left(x_2^A - x_2^B\right)^2}$$

- Let's assume that x_1 and x_2 are not normalized so we have $x_2 \ll x_1$, e.g., x_1 in \$M and x_2 in \$

- In many cases the differences $x_1^A - x_1^B$ is also in \$B and $x_2^A - x_2^B$ in \$ so $\left|x_2^A - x_2^B\right| \ll \left|x_1^A - x_1^B\right|$ or $\left|\frac{x_2^A - x_2^B}{x_1^A - x_1^B}\right| \ll 1$

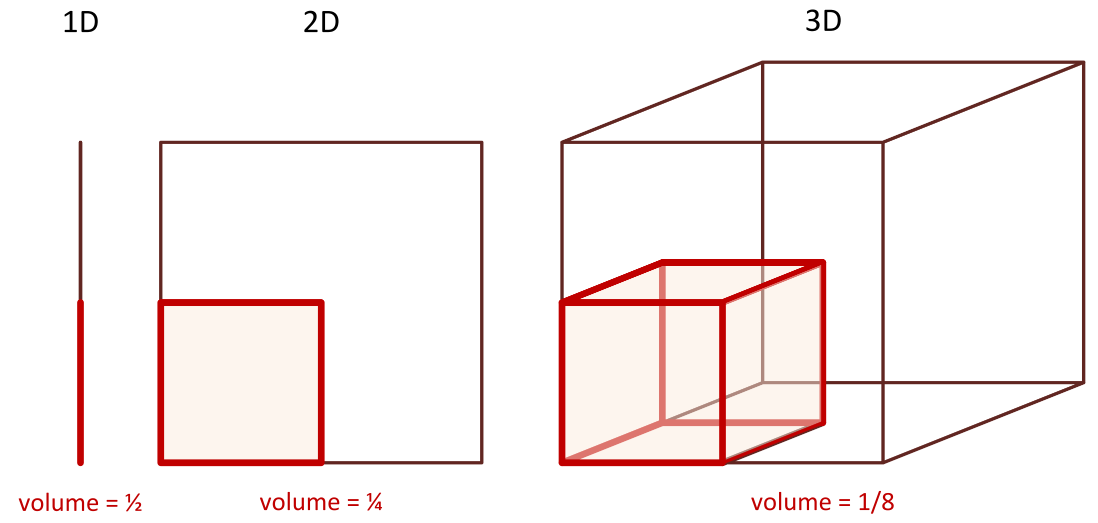
$$d(A,B) = \left|x_1^A - x_1^B\right| \cdot \sqrt{1 + \underbrace{\left(\frac{x_2^A - x_2^B}{x_1^A - x_1^B}\right)^2}_{\ll 1}} \approx \left|x_1^A - x_1^B\right|$$

(i.e., the distance between A and B is independent of the second feature vector x_2)

High Dimensionality

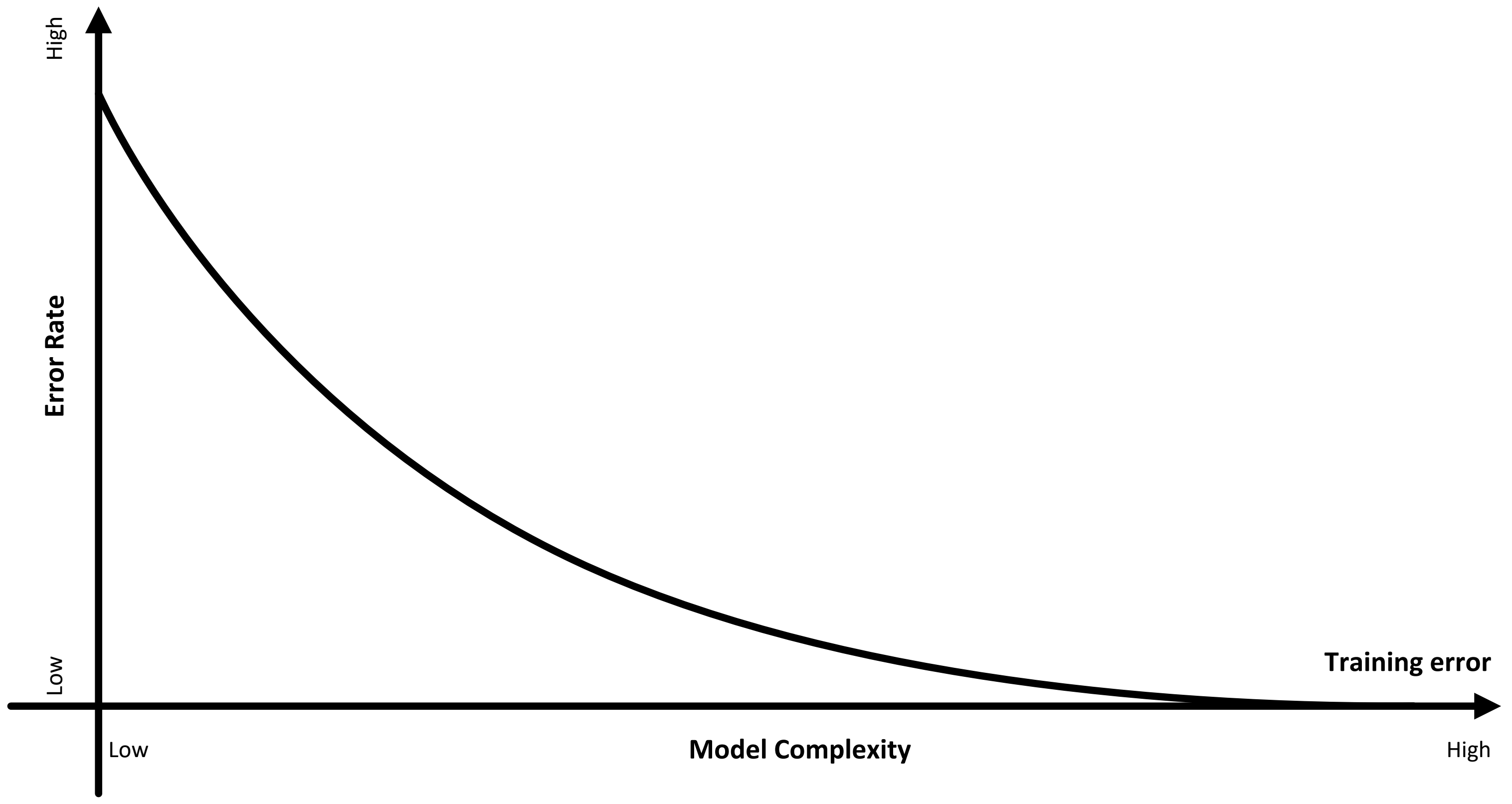
What happens in high dimensionality?

- When using (Euclidean) distance, higher dimensionality of data (i.e., more features) requires significantly more samples in order to have the same predictive power

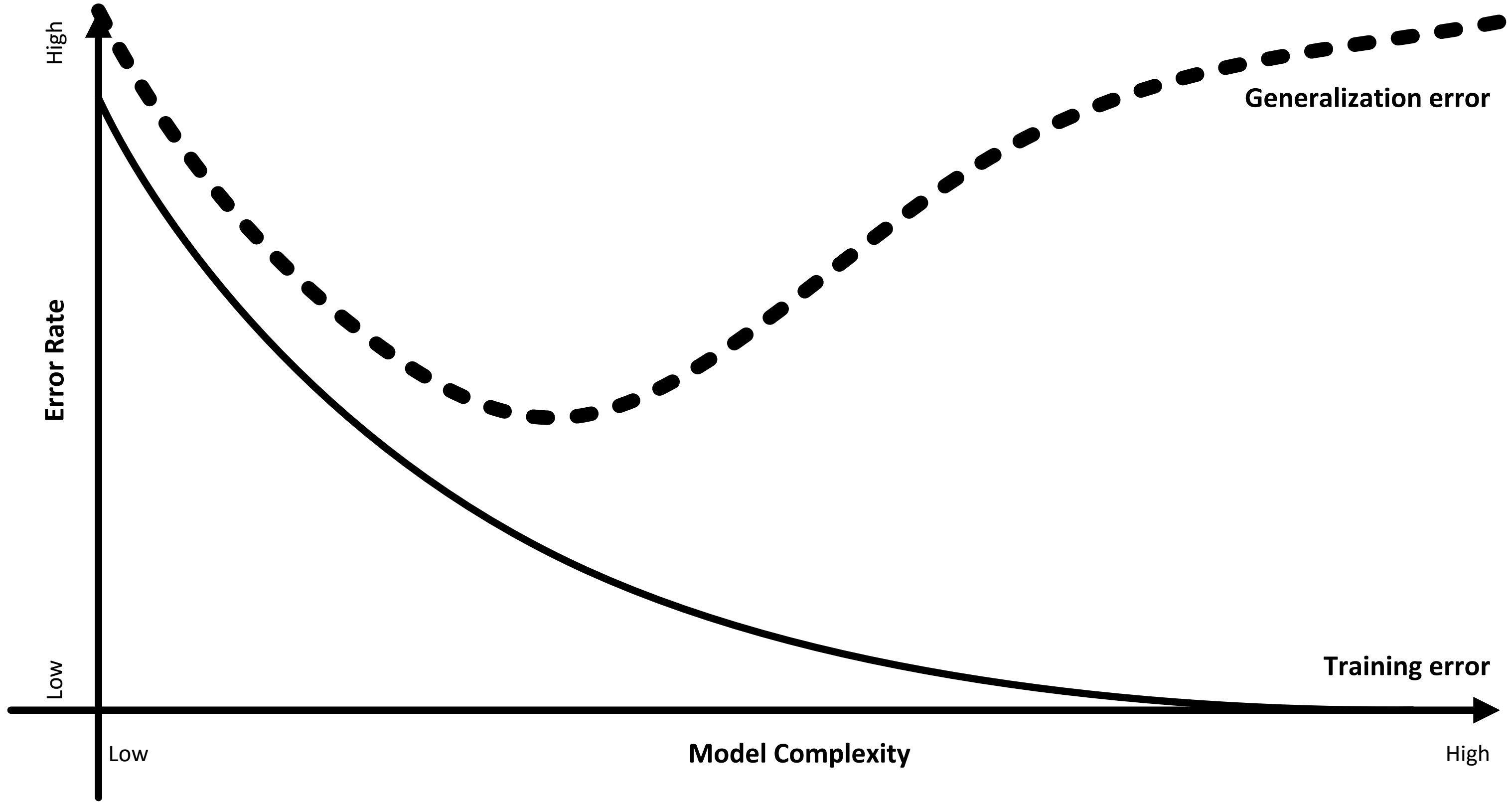


Model Fit

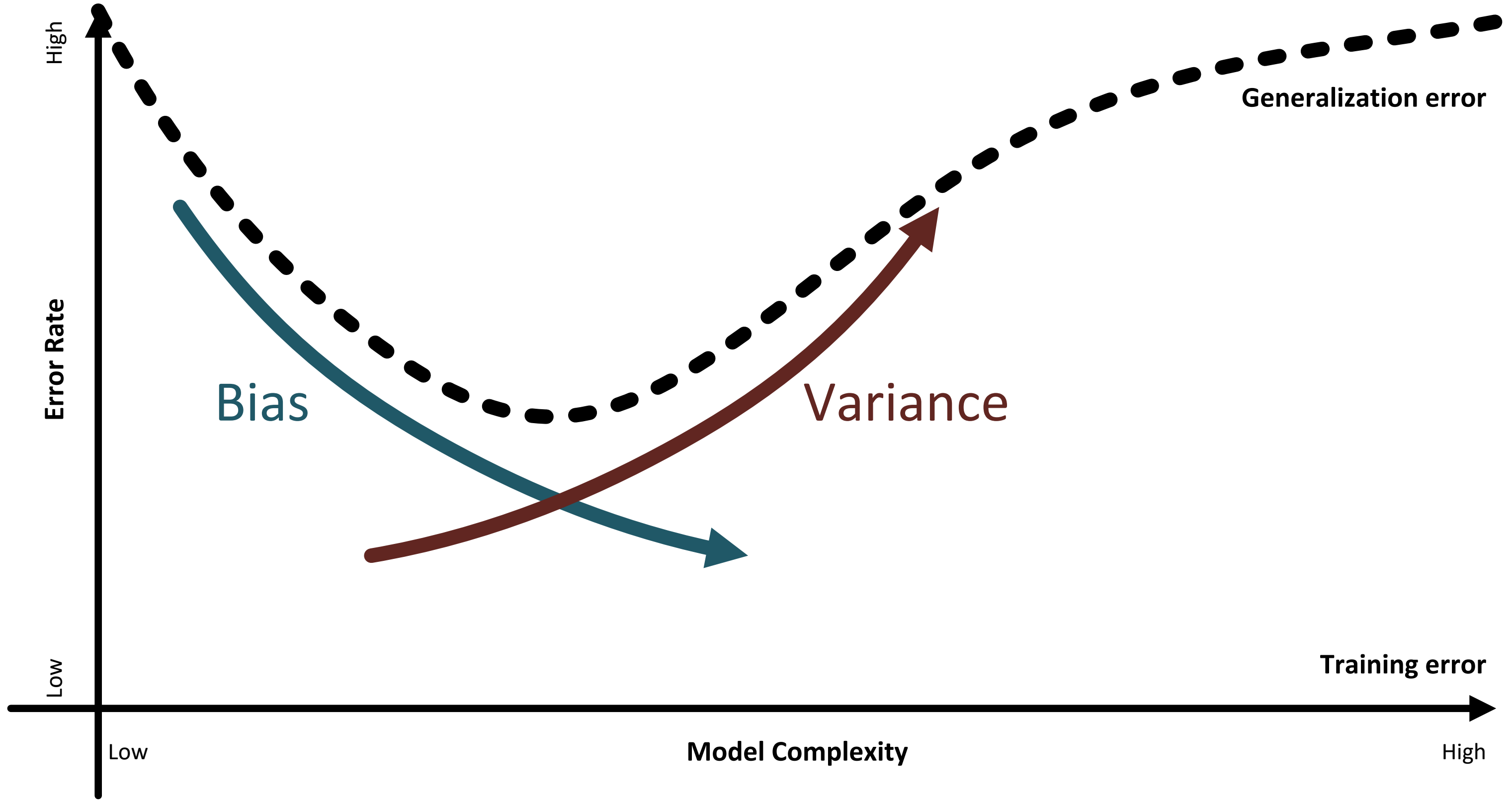
The training error can go down to zero (effectively memorizing the entire dataset)



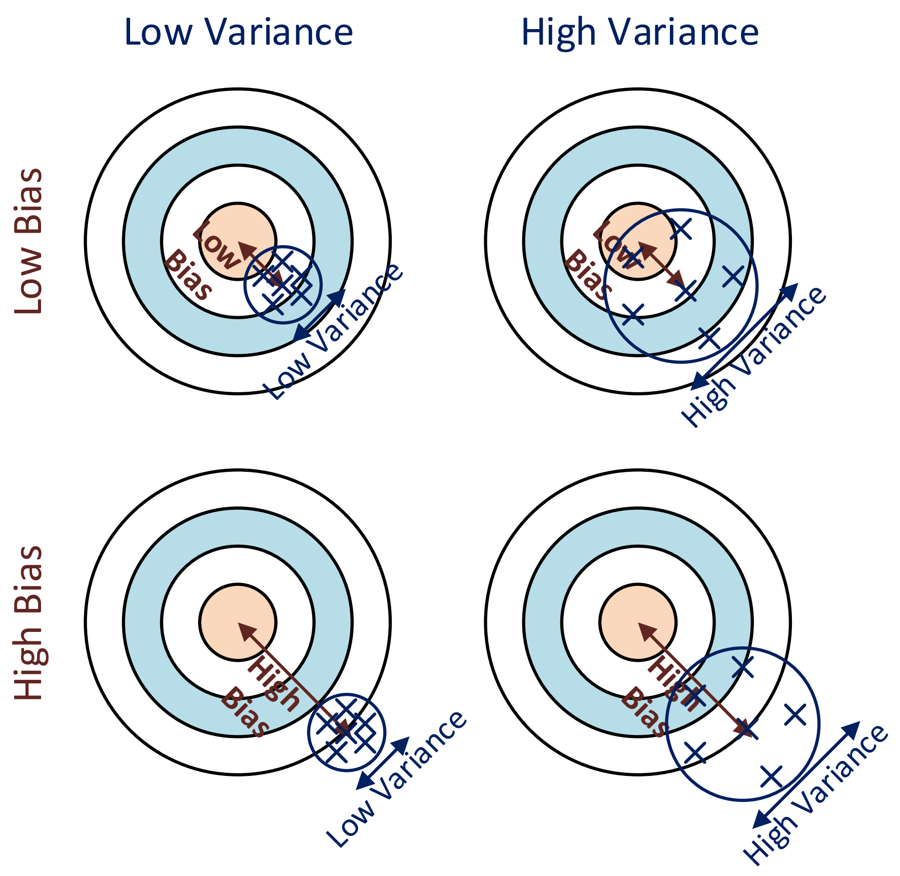
As the model gets more complex, the generalization error initially goes down; however, after reaching a minimum, it goes back up



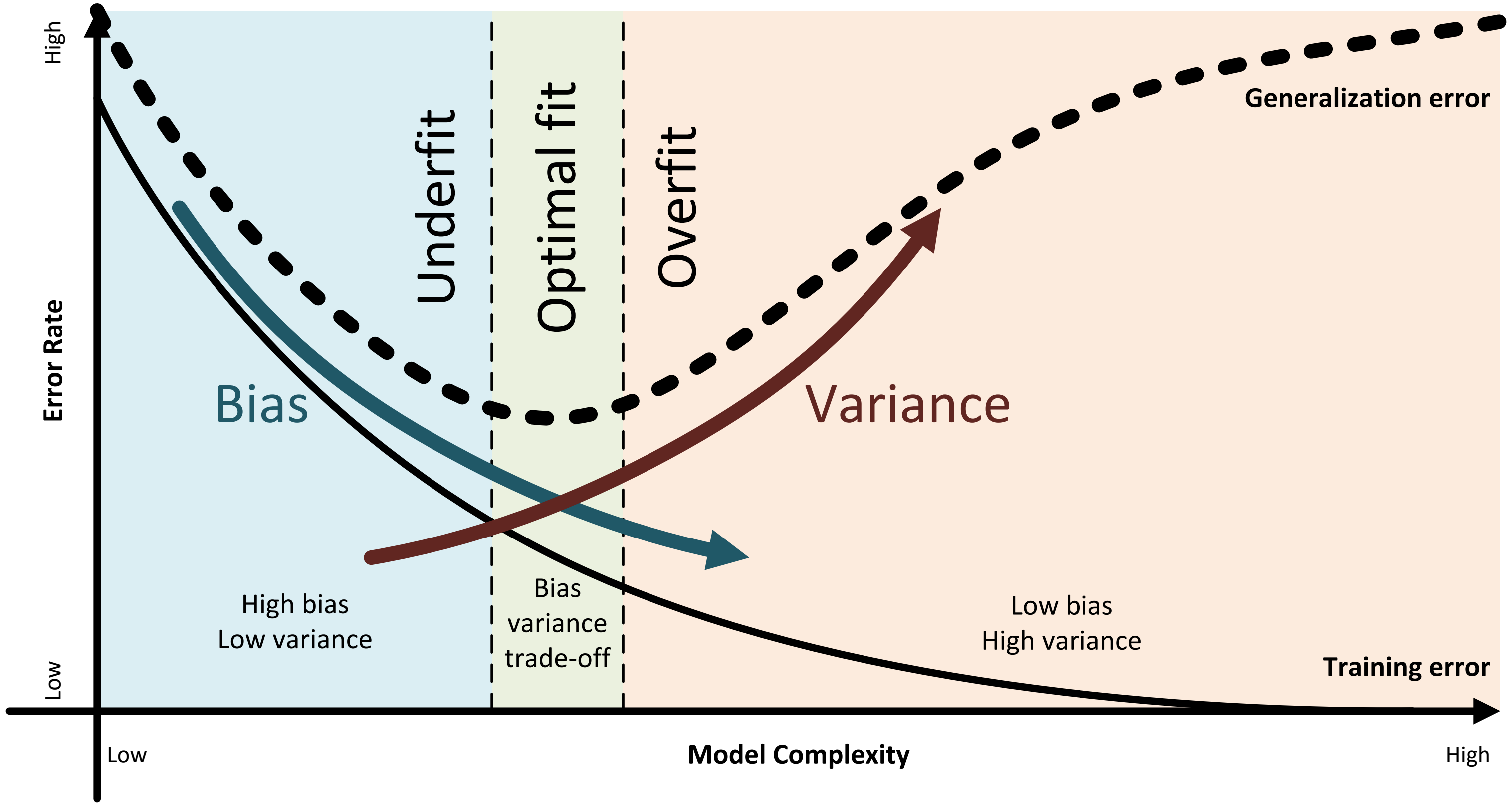
The generalization error is made of two components: bias and variance



The bias is a systematic, non-random error; the variance is an idiosyncratic, random error



Errors, fit, bias, and variance

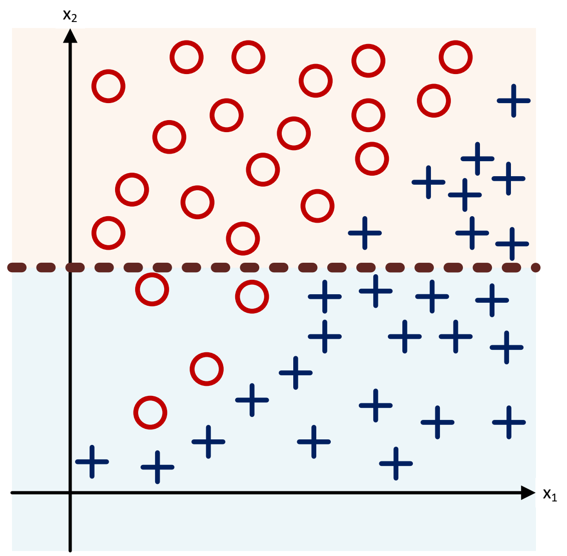


Errors, fit, bias, and variance (cont.)

EXAMPLE

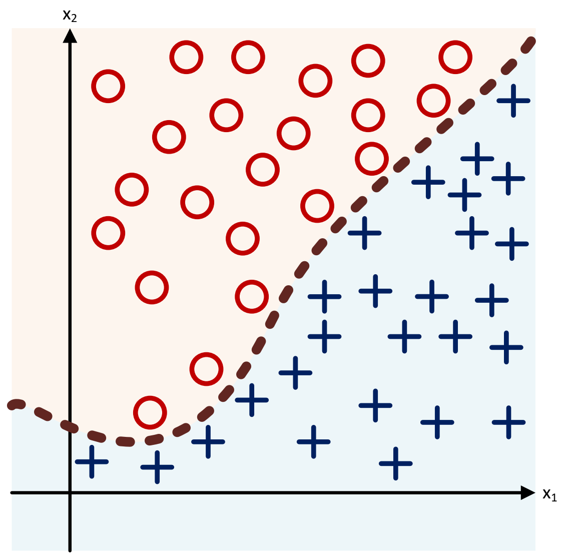
Underfit

high bias/low variance



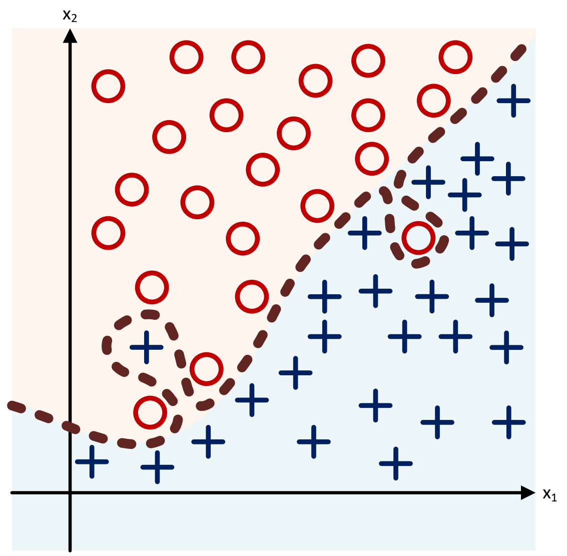
Optimal fit

bias-variance trade-off

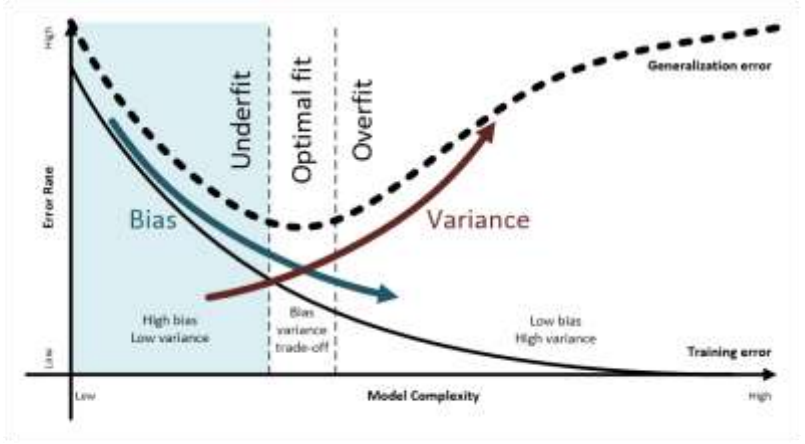


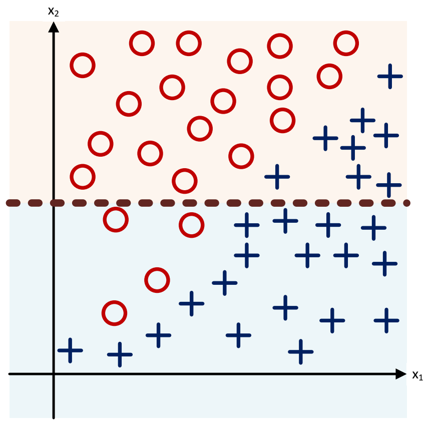
Overfit

low bias/high variance



Underfit

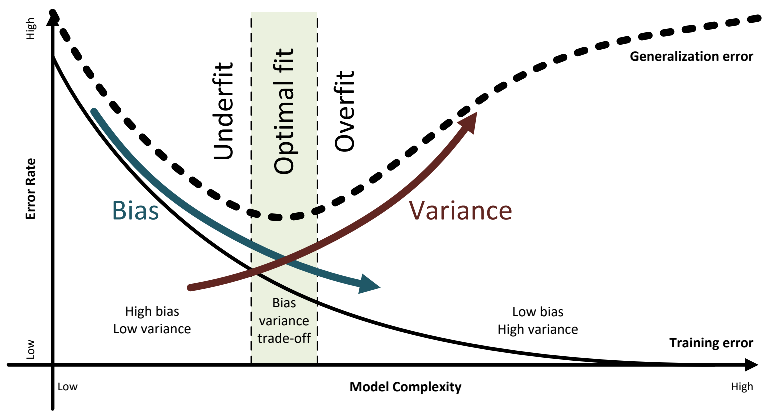


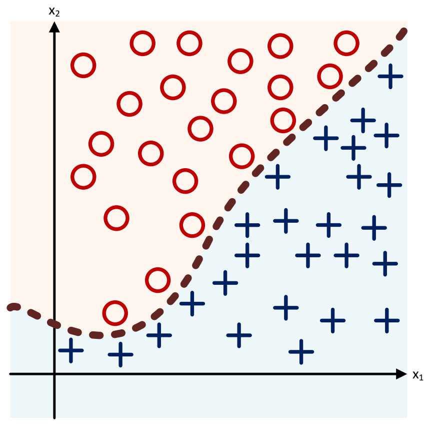


- Underfit
  - Model too simple
  - It cannot represent the desired behavior very well; both its training and generalization error are poor
  - High bias; low variance

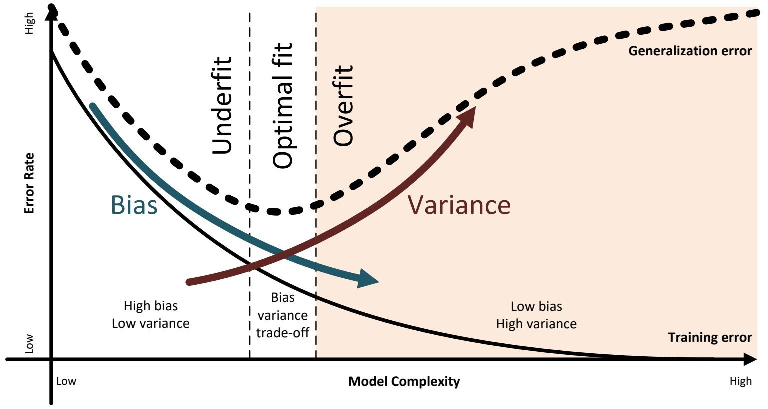
Optimal fit

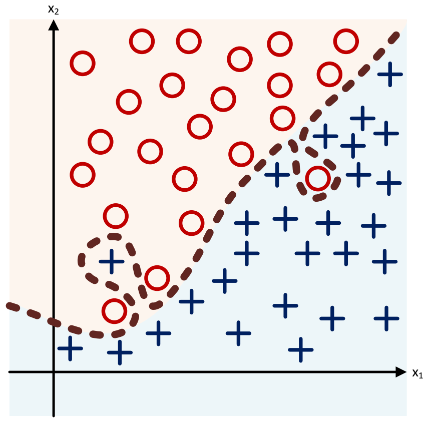
- Optimal fit
  - Model has the right level of complexity
  - It performs well on the training set (low training error) and generalize well to unknown data points (low generalization error)





Overfit

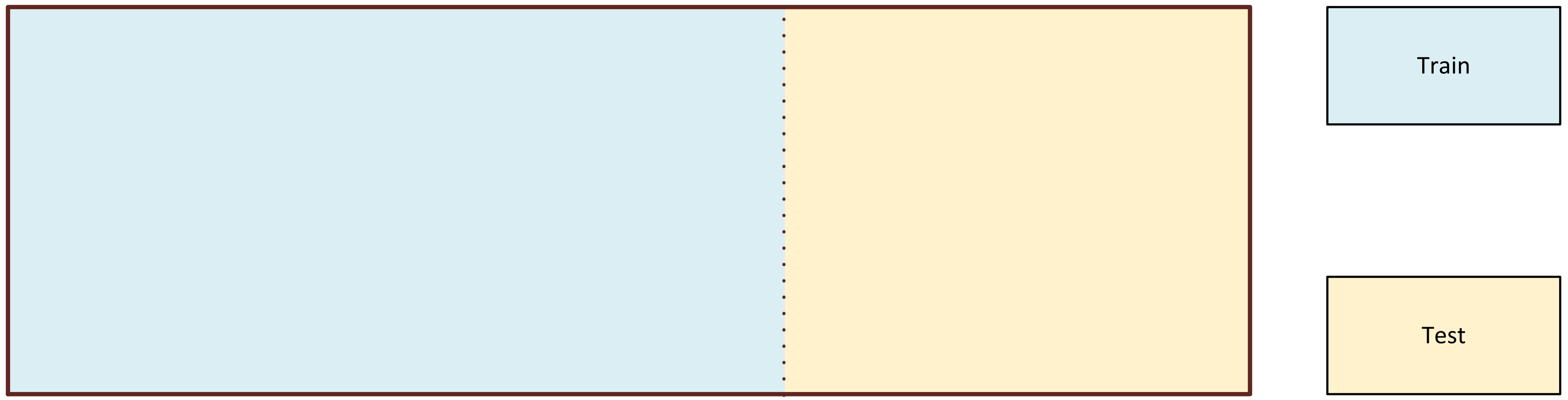




- Overfit
  - Model too complex
  - It performs very well on the training set (low training error) but does not generalize well to unseen data points (high generalization error)
  - Low bias; high variance

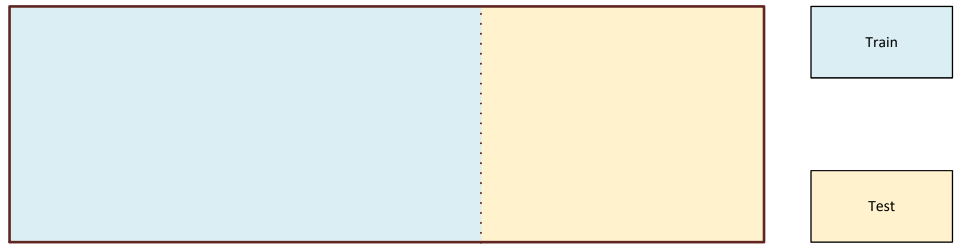
So far, we used the entire dataset to train the models.
Question: How can we estimate the generalization error?

Answer: Divide (randomly) the dataset into a Train set and a Test set

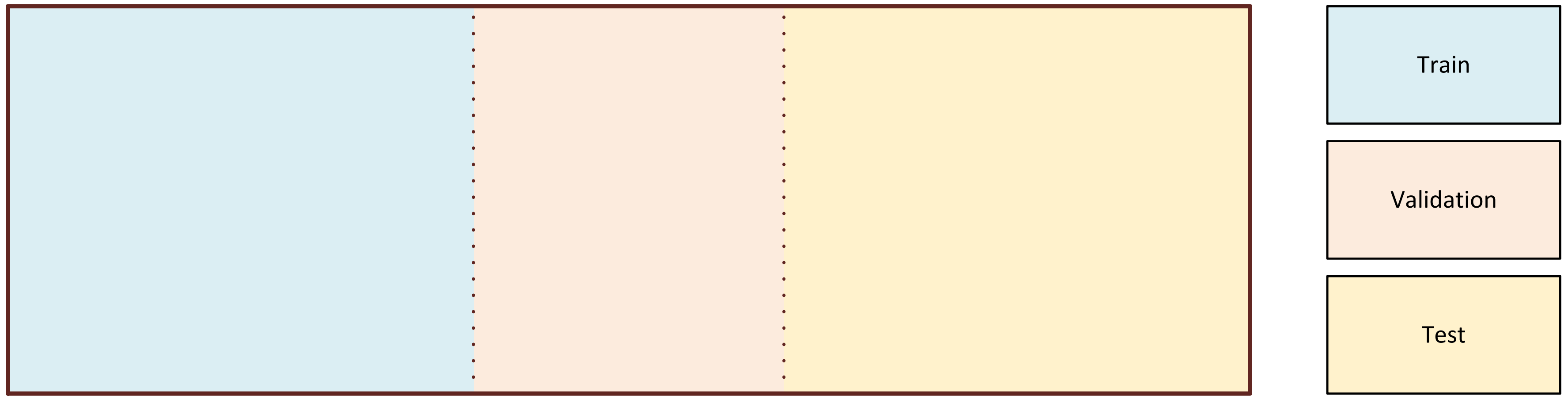


Train and Test sets

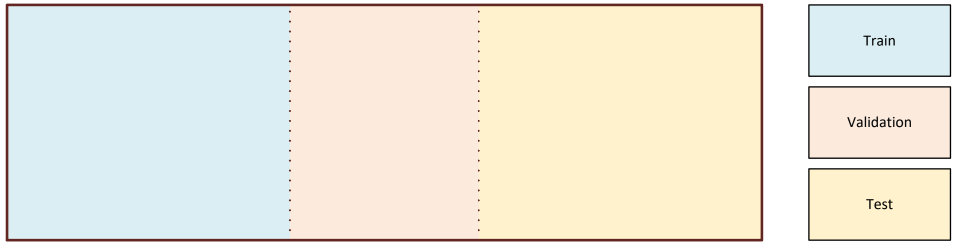
- Set aside the test set; don't look at it until the very end
- Train your model with the train set
  - Remodel as needed until you are satisfied with your model performance on the train set (low training error)
- Evaluate your model on the test set to compute the generalization error
  - Only then do you now know whether your model underfits, overfits, or seems ok
- If you need to go back and remodel you need a new test: as you incorporate knowledge from the test set back into your remodel, the test set's previously unseen data points are not longer unseen
  - Question: How can we really keep our test set aside until the very end



Answer: Divide (randomly) again your train set into a (new) Train set and a Validation set

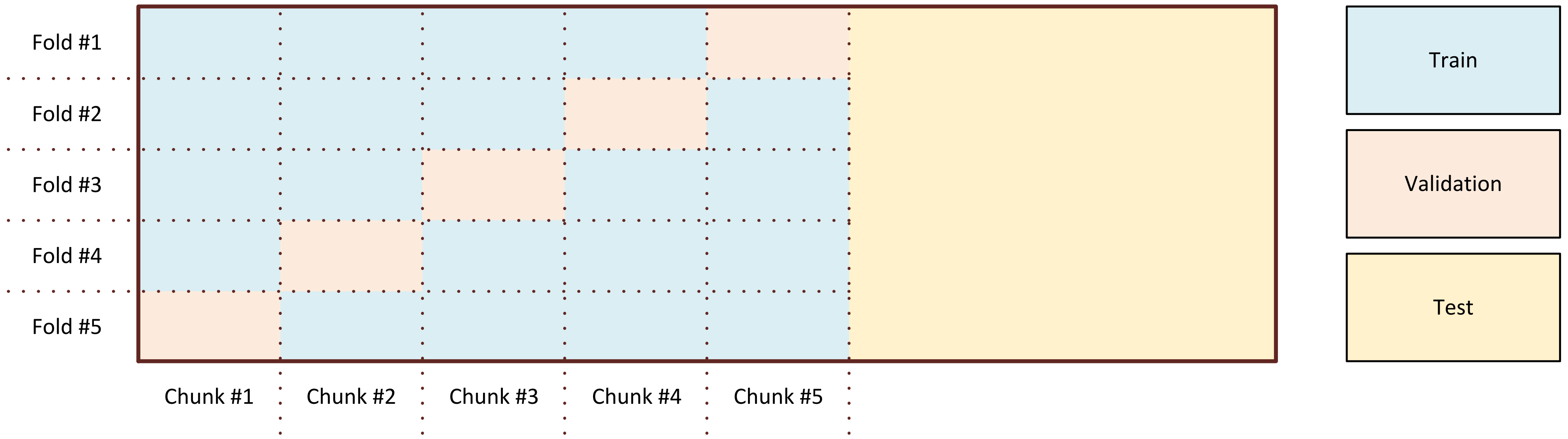


Train, Validation, and Test sets



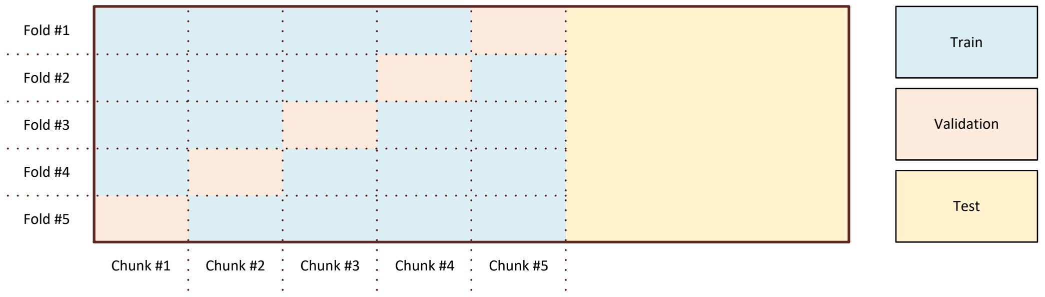
- You still train the model with the train set (model building) but now you use the cross-validation set, not the test set, to estimate the generalization error (model checking)
- After using the cross-validation set and before a new phase of remodeling, you should then reshuffle data between your train set and your cross-validation set
- Question: Reshuffling the train/cross-validation sets seems heavy work. Can we do better?

Answer: Yes we can with k-Fold Cross-Validation

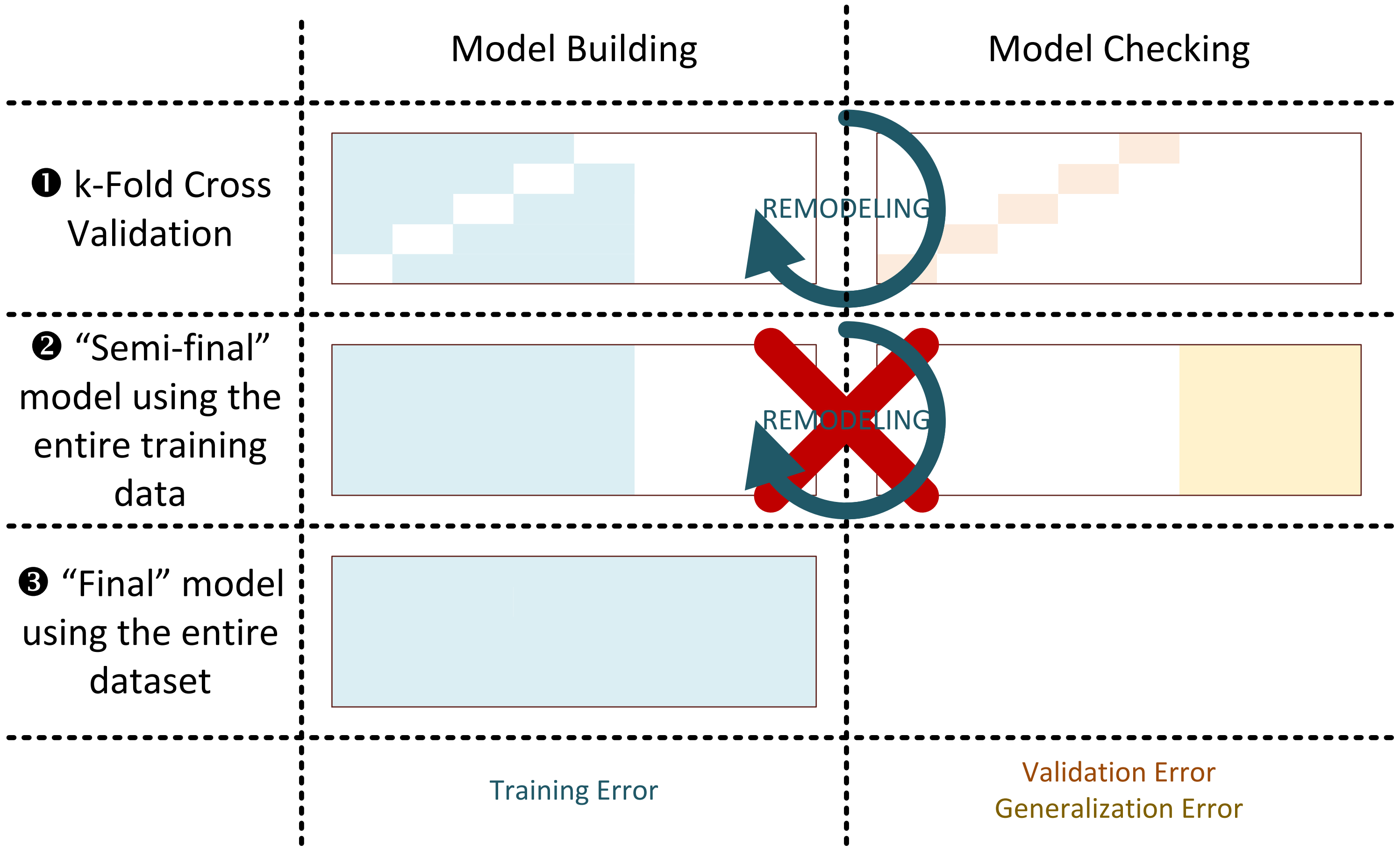


k-Fold Cross-Validation

- Typically, $k = 5$ or 10 with each sample being used both for training ($k - 1$ times) and validation (1 time)
- The training/validation errors are the average training/validation errors across all folds
- After selecting the model that minimize the validation error, you then build a final model that uses all the training data



Model building and model checking with k-Fold Cross-Validation


Model Building
Model Checking
❶ k-Fold Cross Validation
❷ "Semi-final" model using the entire training data
❸ "Final" model using the entire dataset
REMODELING
REMODELING
Training Error
Validation Error
Generalization Error

k-Nearest Neighbors | Pros and cons

- Pros
  - Intuitive and simple to explain
  - Training phase is fast
  - Non-parametric (does not presume a “form” of the decision boundary)
  - The decision boundary easily captures non-linearity

- Cons
  - Not interpretable
  - Prediction phase can be slow when n (number of observations) is large
  - Very sensitive to feature scaling; need to standardize the data
  - Sensitive to irrelevant features
  - Cannot be used if you have sparse data and feature space with dimension $p \geq 4$