

06 | k -Nearest Neighbors

Ivan Corneillet

Data Scientist

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

A black circle containing the white text "DS".

DS

Classification

k -Nearest Neighbors is a supervised learning algorithm for regression or classification

	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 ✓
<i>Unsupervised</i> (representation; extract structure)		

Response Vector y (or c) (cont.)

Regression

Response vector y

	col <i>e.g. price</i>
row0, <i>e.g., house #1</i>	\$1.1M
row1, <i>e.g., house #2</i>	\$.9M
row2, <i>e.g., house #3</i>	\$1.5M
	...

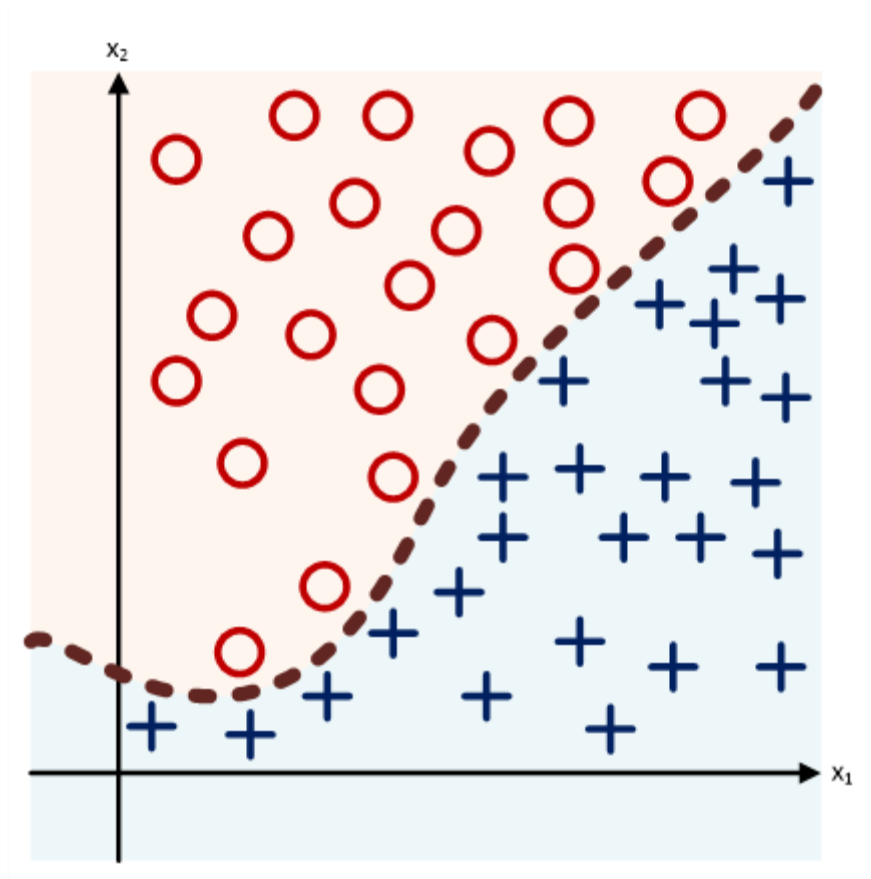
Classification

Response vector c

(renamed from y to c for label classes)

	col <i>e.g., animal</i>
row0, <i>e.g., image #1</i>	"dog"
row1, <i>e.g., image #2</i>	"cat"
row2, <i>e.g., image #3</i>	"bird"
	...

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



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

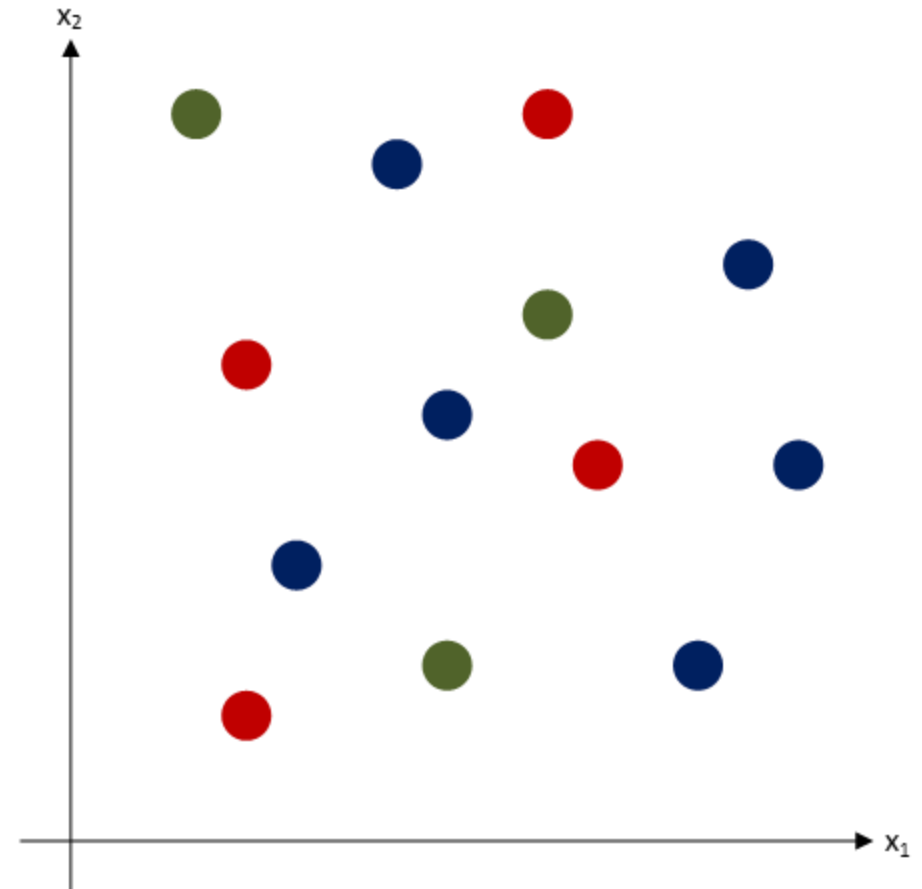
- Of all the observations we predicted, how many were incorrect?
- This is a value we want as low as possible
- Directly opposite of accuracy
 - (Pick one or the other; effectively they are the “same”)

DS

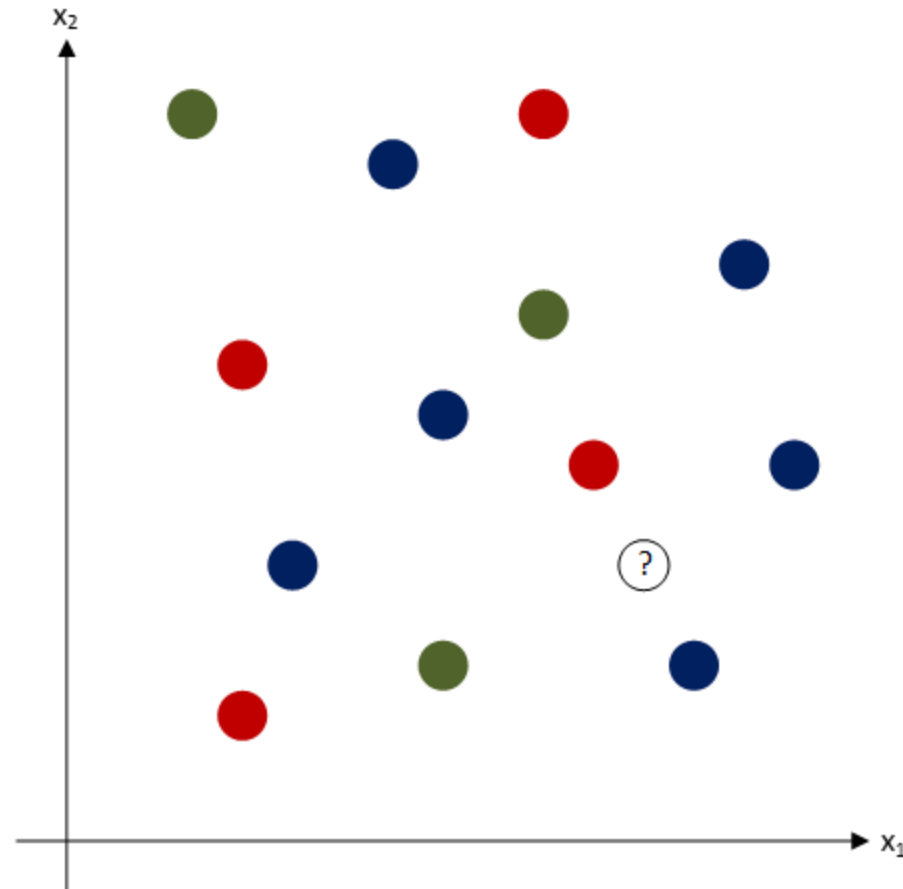
k -Nearest Neighbors

k -Nearest Neighbors

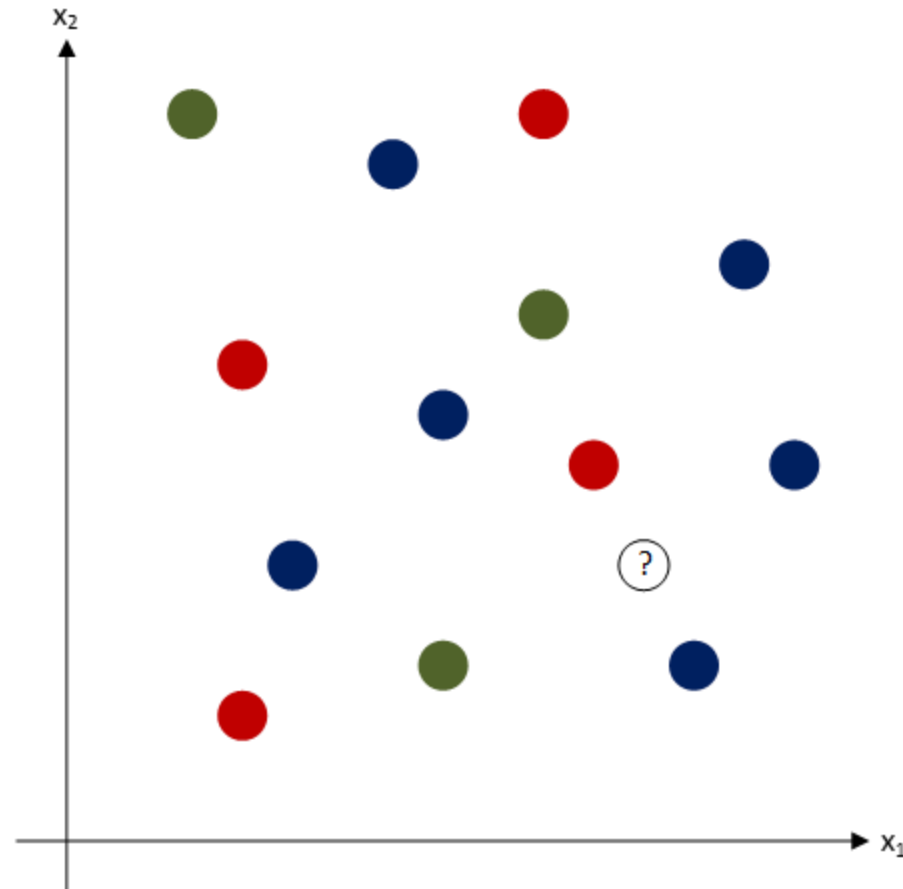
- k -Nearest Neighbors is a classification algorithm that makes a prediction based upon the closest data points



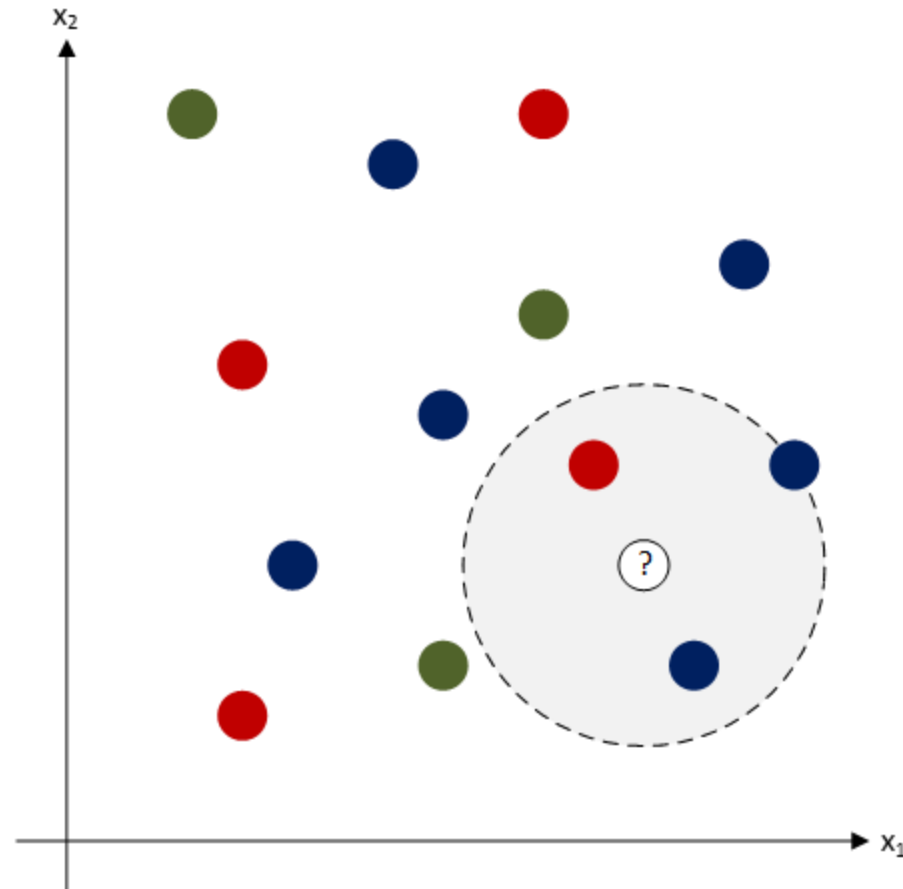
k -Nearest Neighbors | How would you predict the color of the “question mark” point?



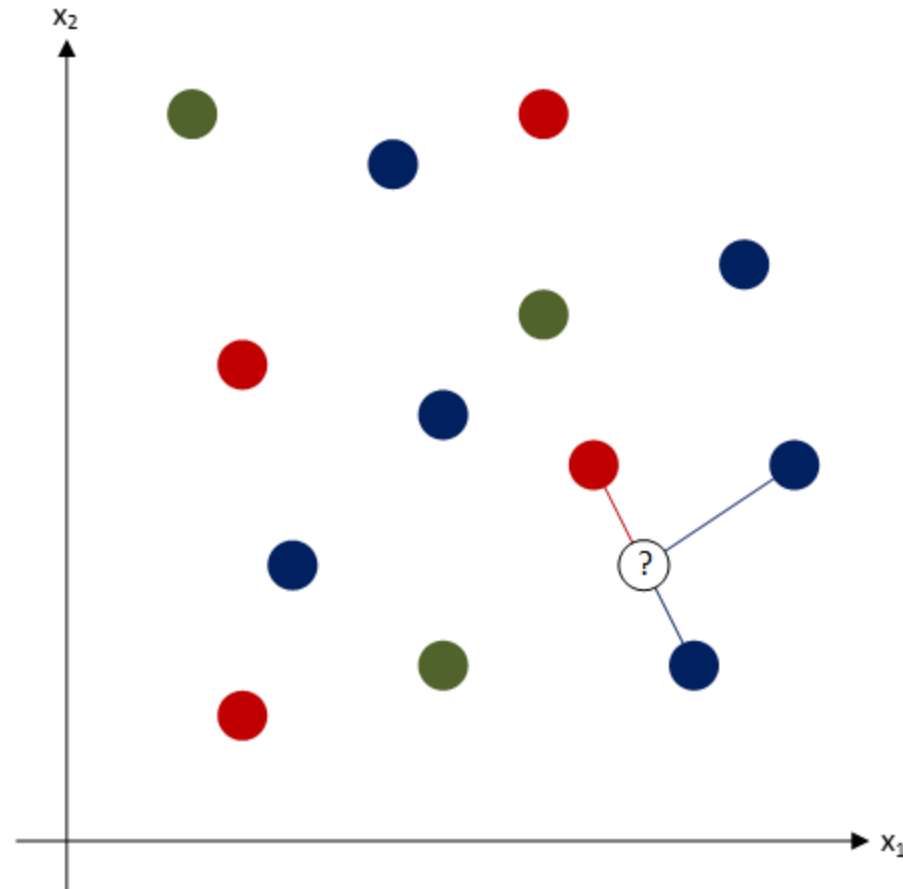
k -Nearest Neighbors | ❶ Pick a value for k , e.g., $k = 3$



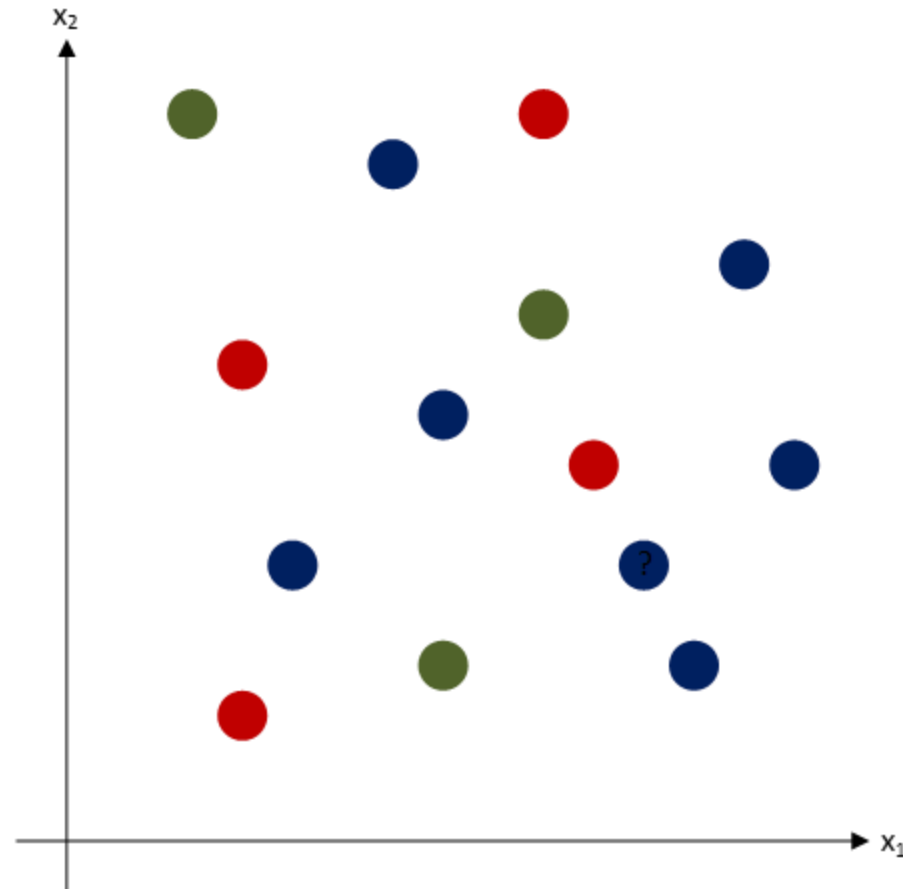
k -Nearest Neighbors | ② Calculate the distance to all other points; given those distances, pick the k closest points



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

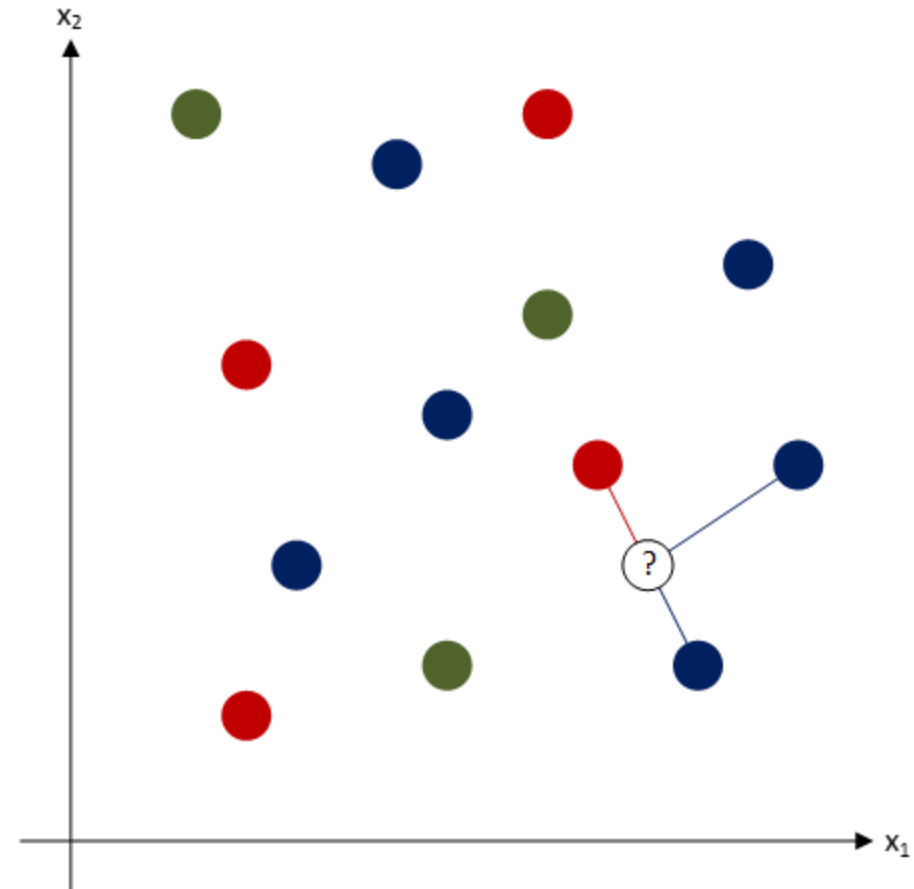


k -Nearest Neighbors | ④ The original point is classified as the class label with the largest probability (“votes”): “blue”



k -Nearest Neighbors (cont.)

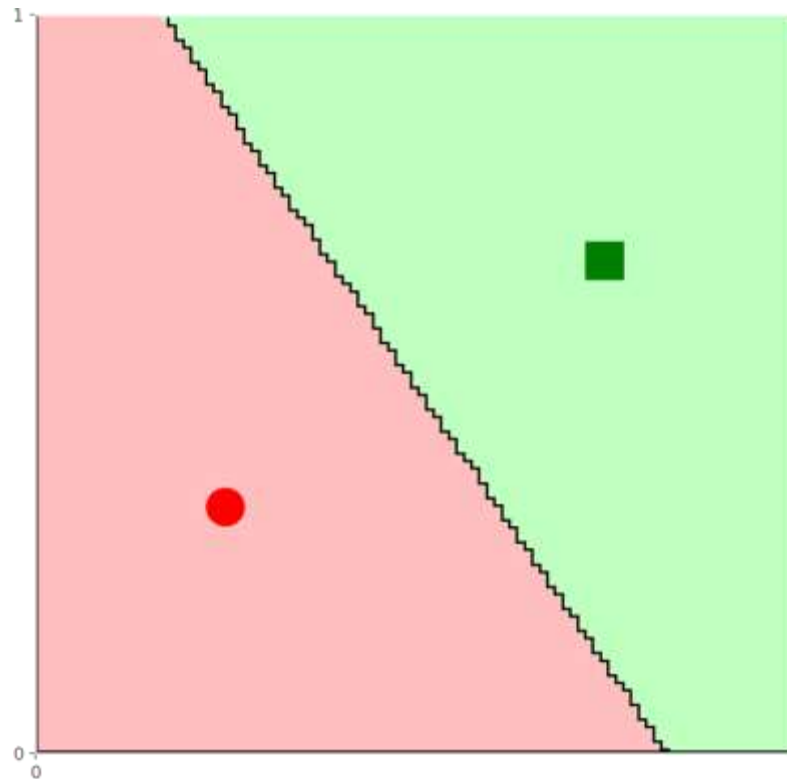
- k -Nearest Neighbors uses distance to predict a class label
- This application of distance is used as a measure of similarity between classifications
 - We are using shared traits to identify the most likely class label



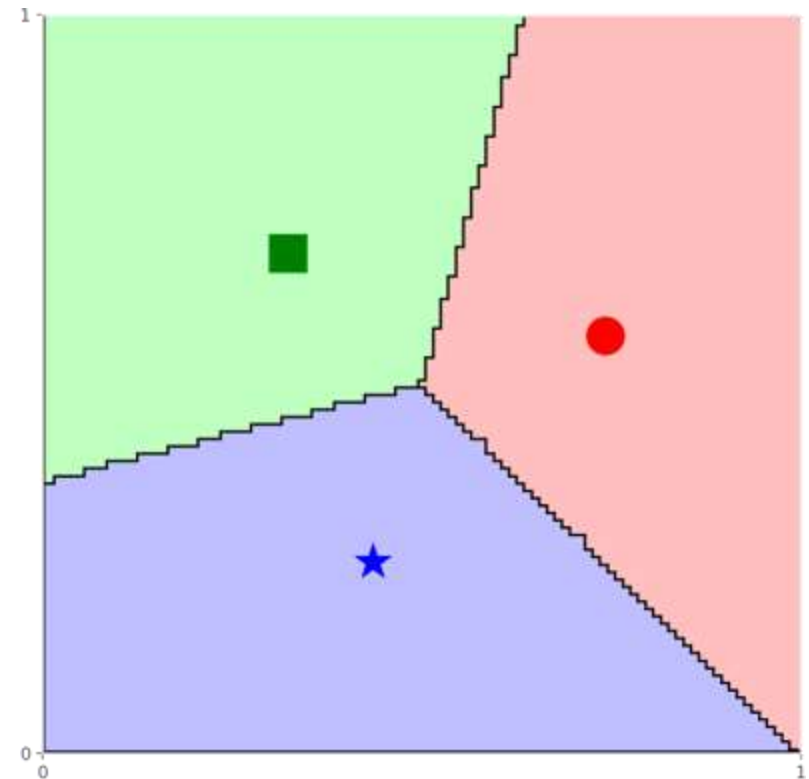
1-Nearest Neighbors

EXAMPLE

$n = 2$



$n = 3$

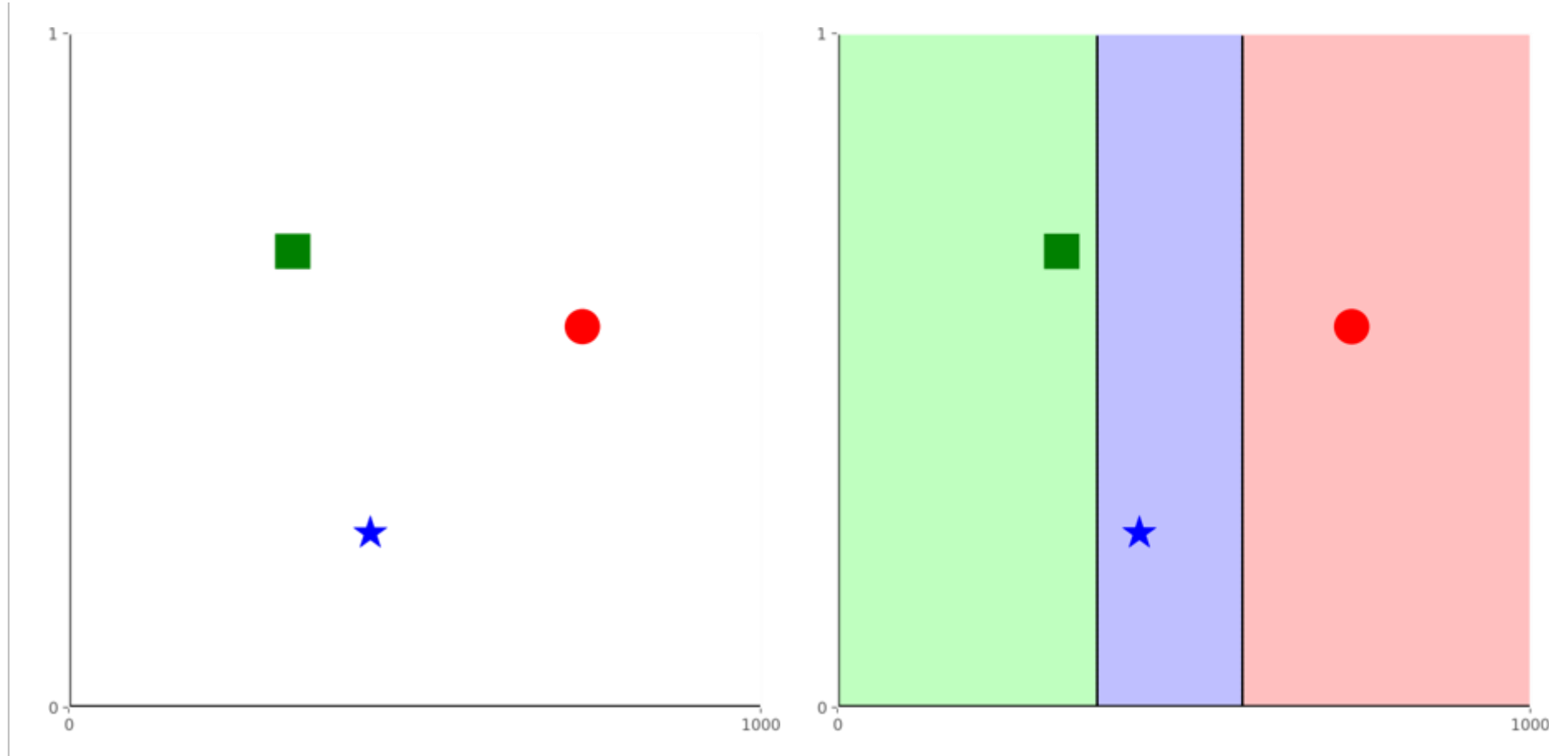


A black circle containing the white text "DS".

DS

Feature Normalization

Non-Normalized Features with k -Nearest Neighbors (cont.)





DS

High Dimensionality

The logo consists of a solid black circle containing the white letters "DS" in a bold, sans-serif font.

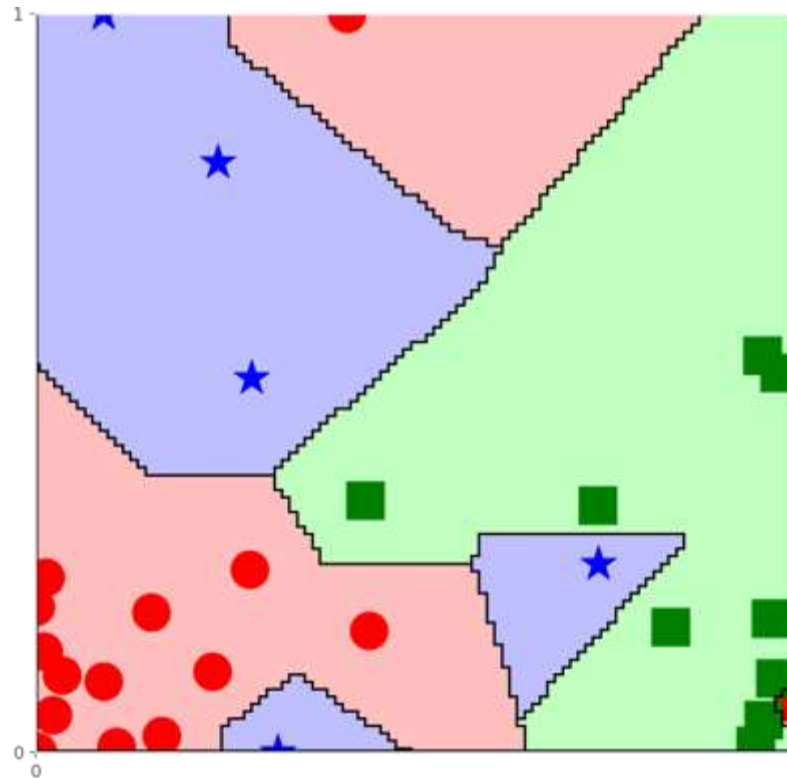
DS

Model Fit

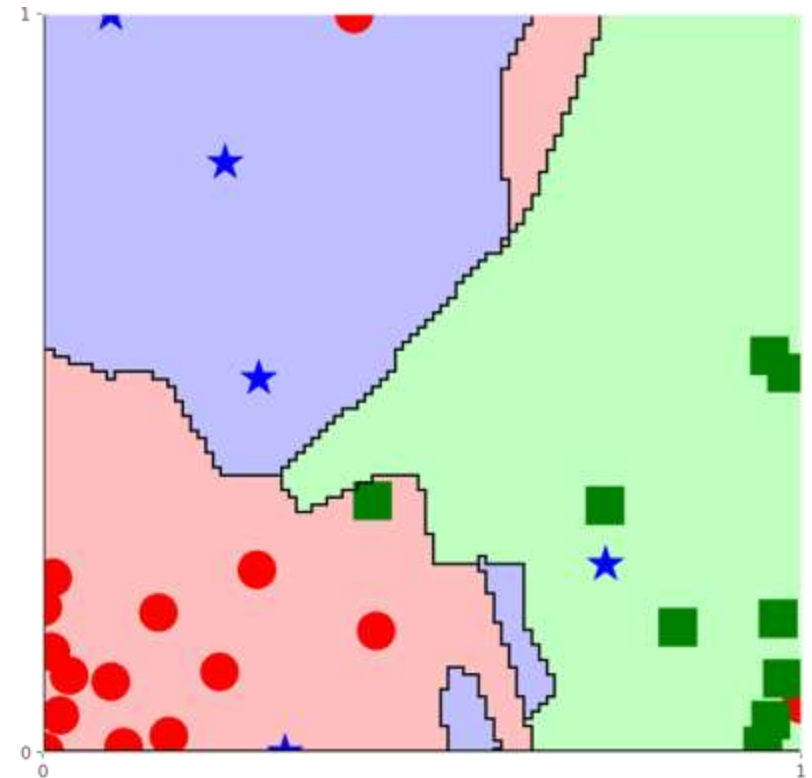
Model Fit | Motivating Example (cont.)

EXAMPLE

$k = 2$



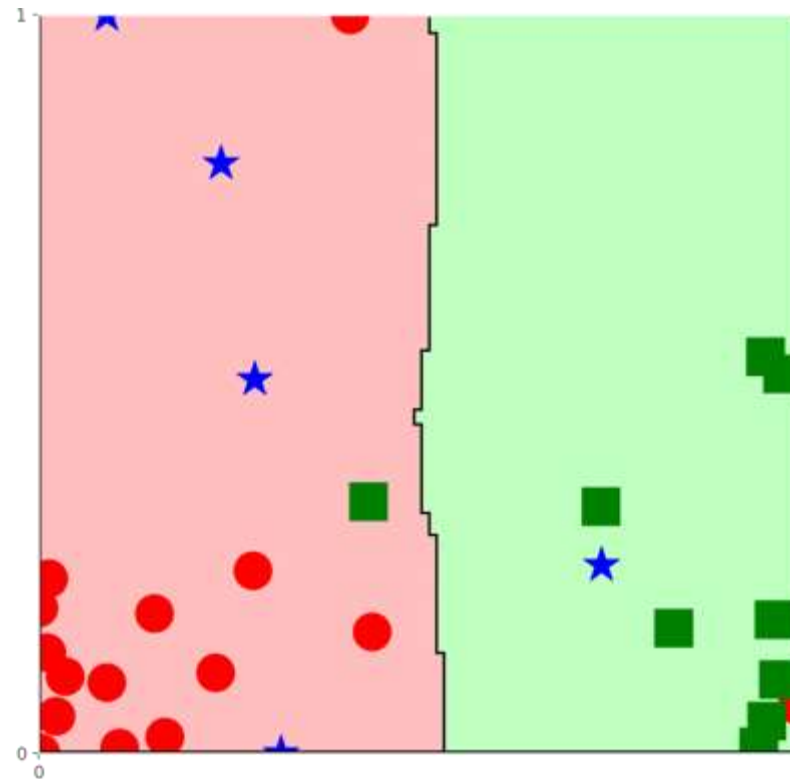
$k = 3$



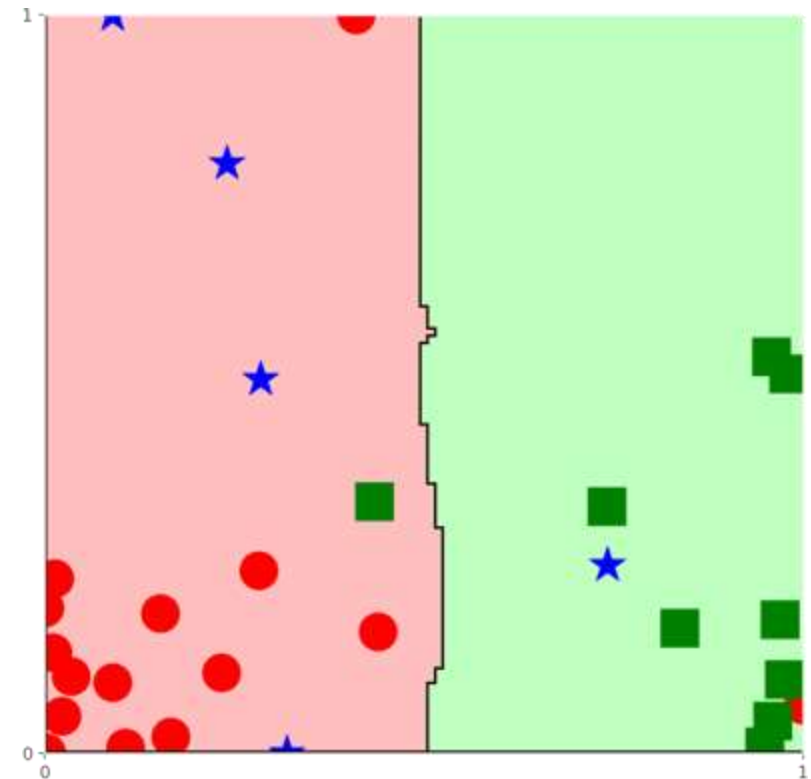
Model Fit | Motivating Example (cont.)



$k = 15$



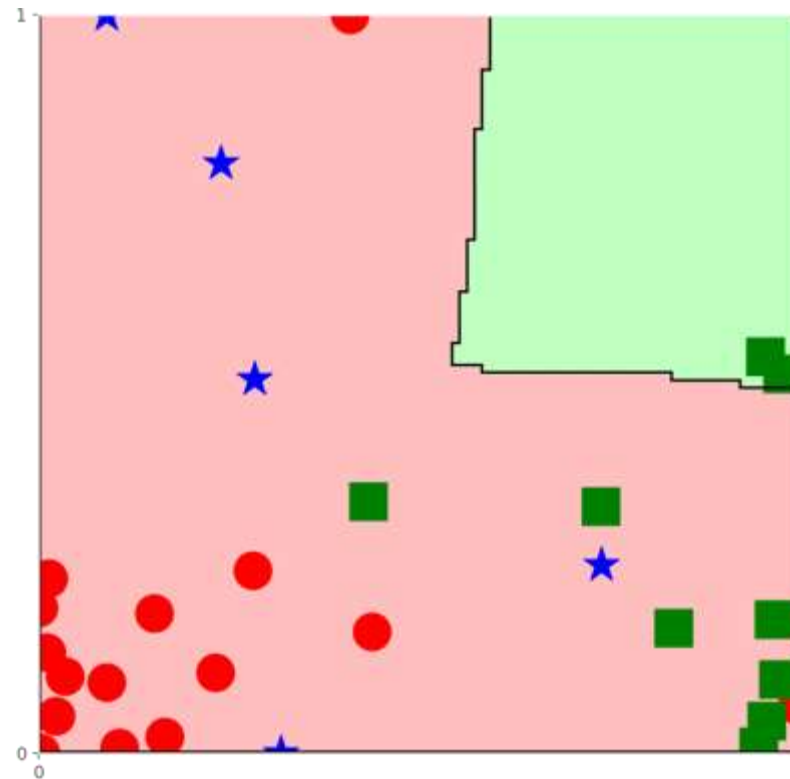
$k = 16$



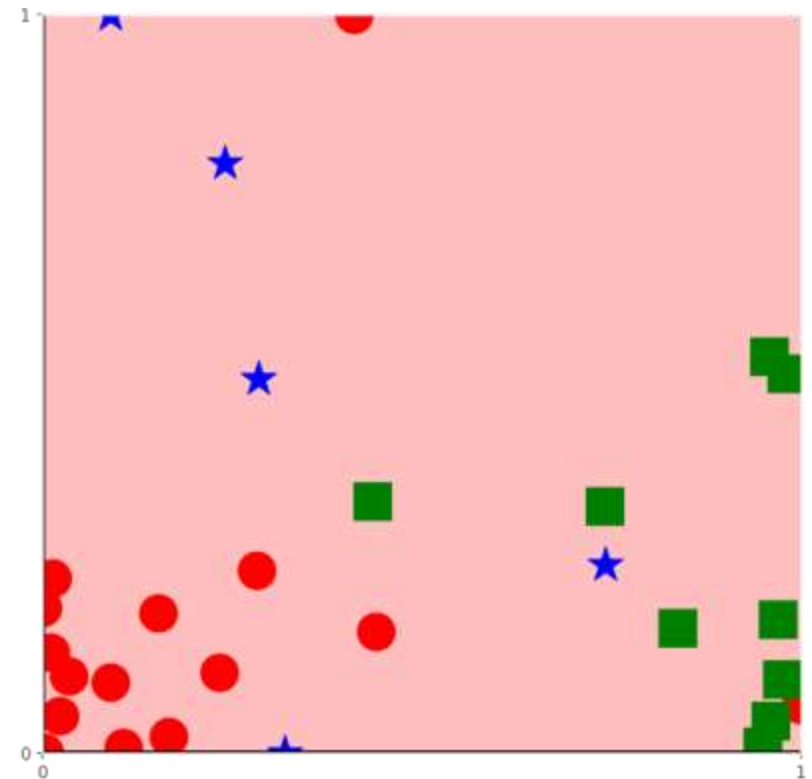
Model Fit | Motivating Example (cont.)



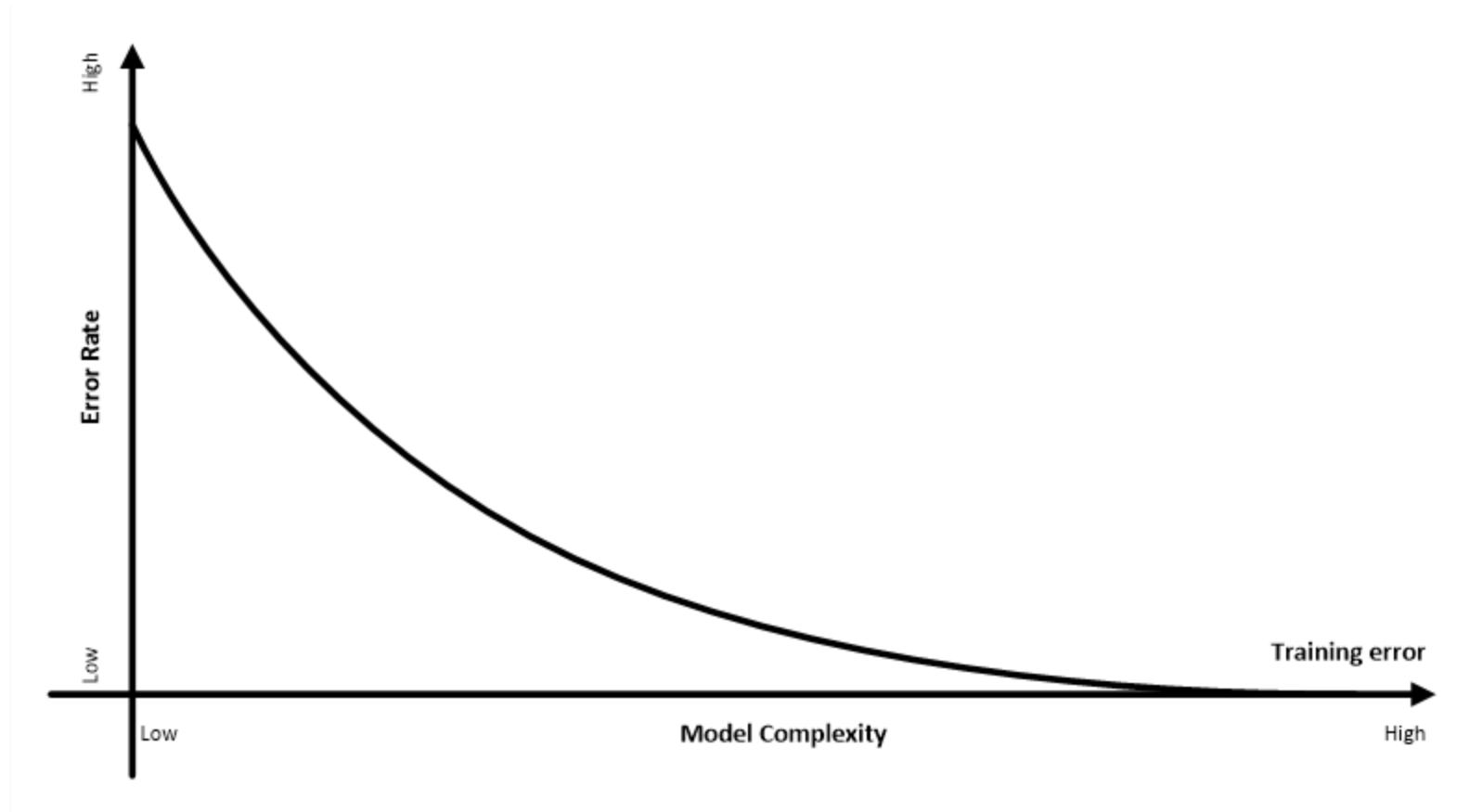
$k = 25$



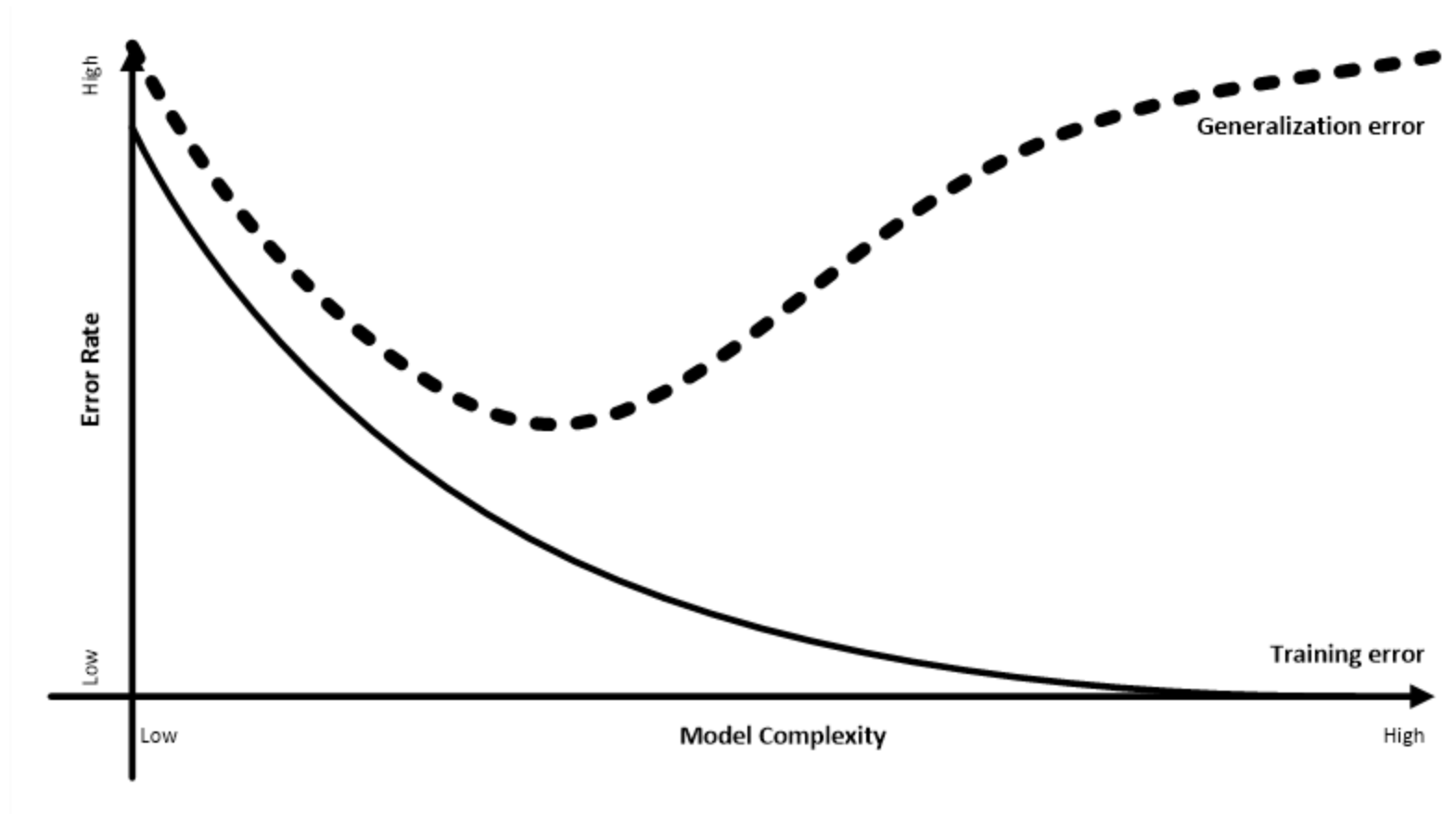
$k = 26$



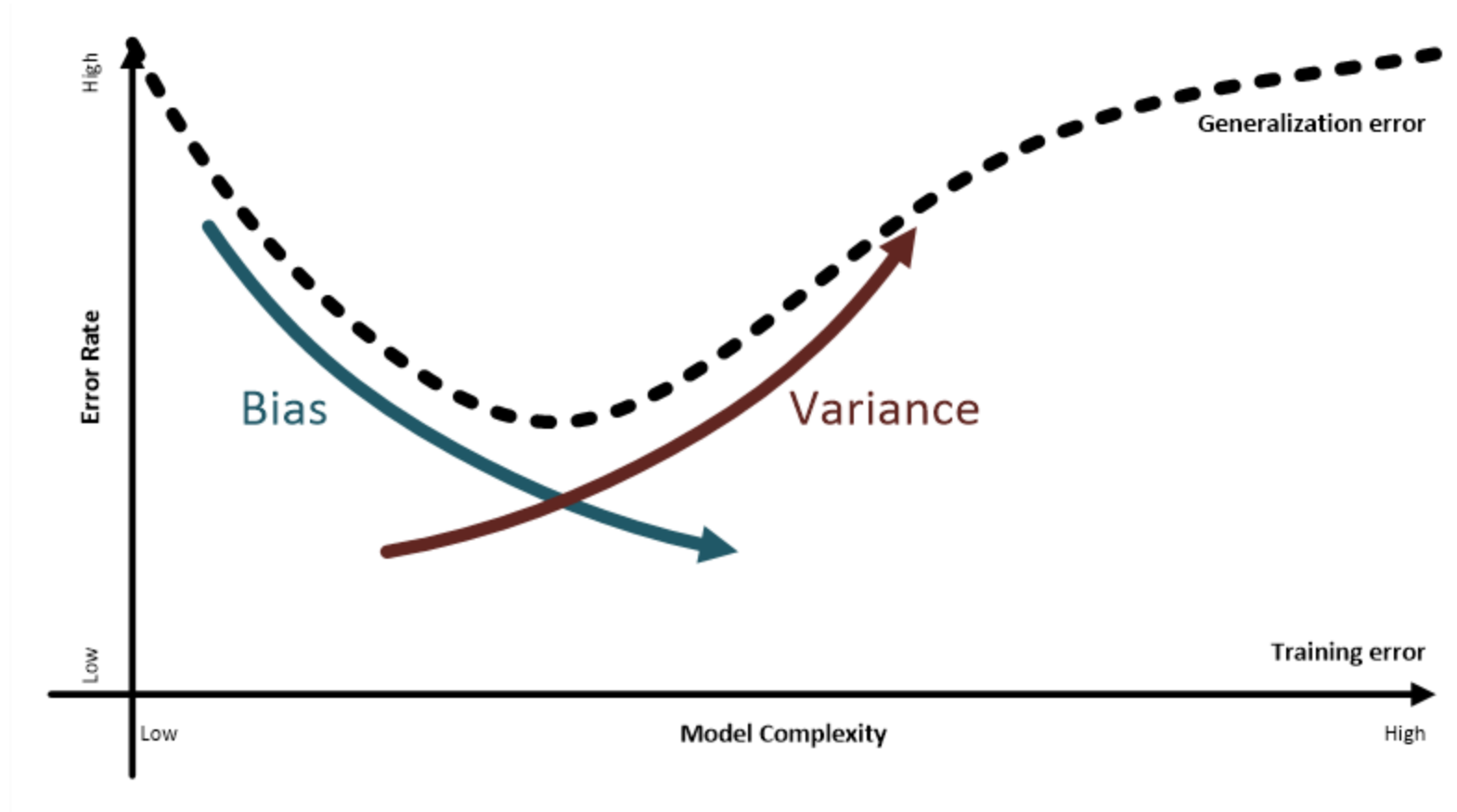
The Training Error can go down to zero (effectively memorizing the entire dataset)



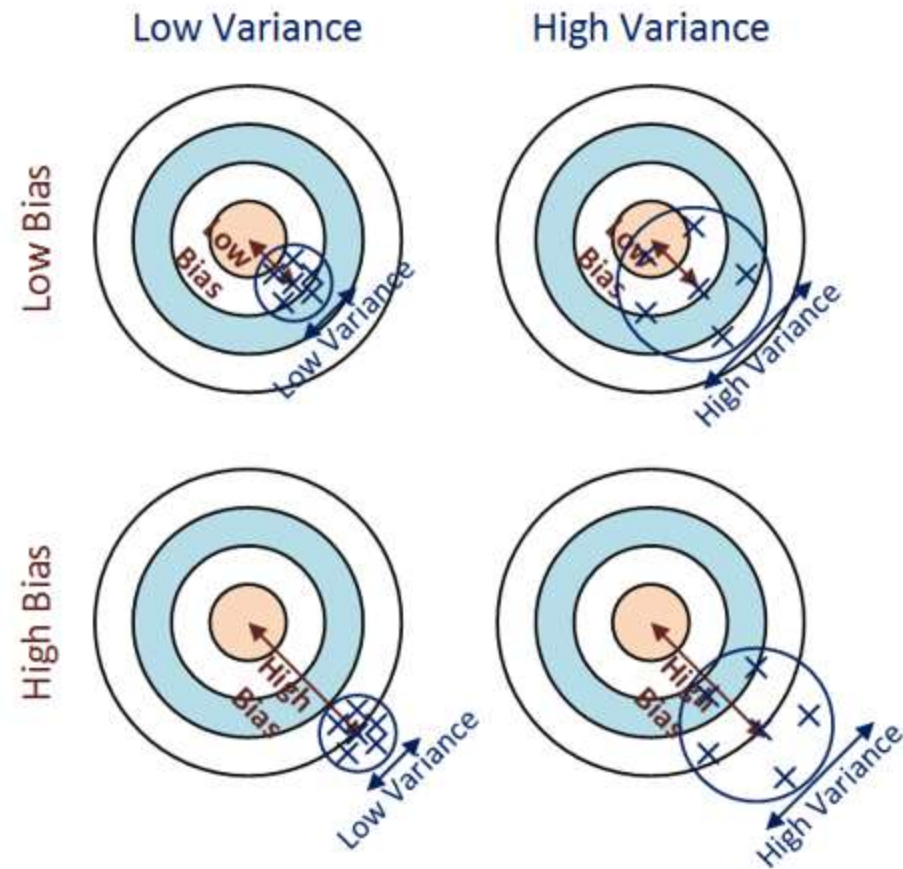
As the model gets more complex, the Generalization Error initially goes down; however, after reaching a minimum, it goes back up



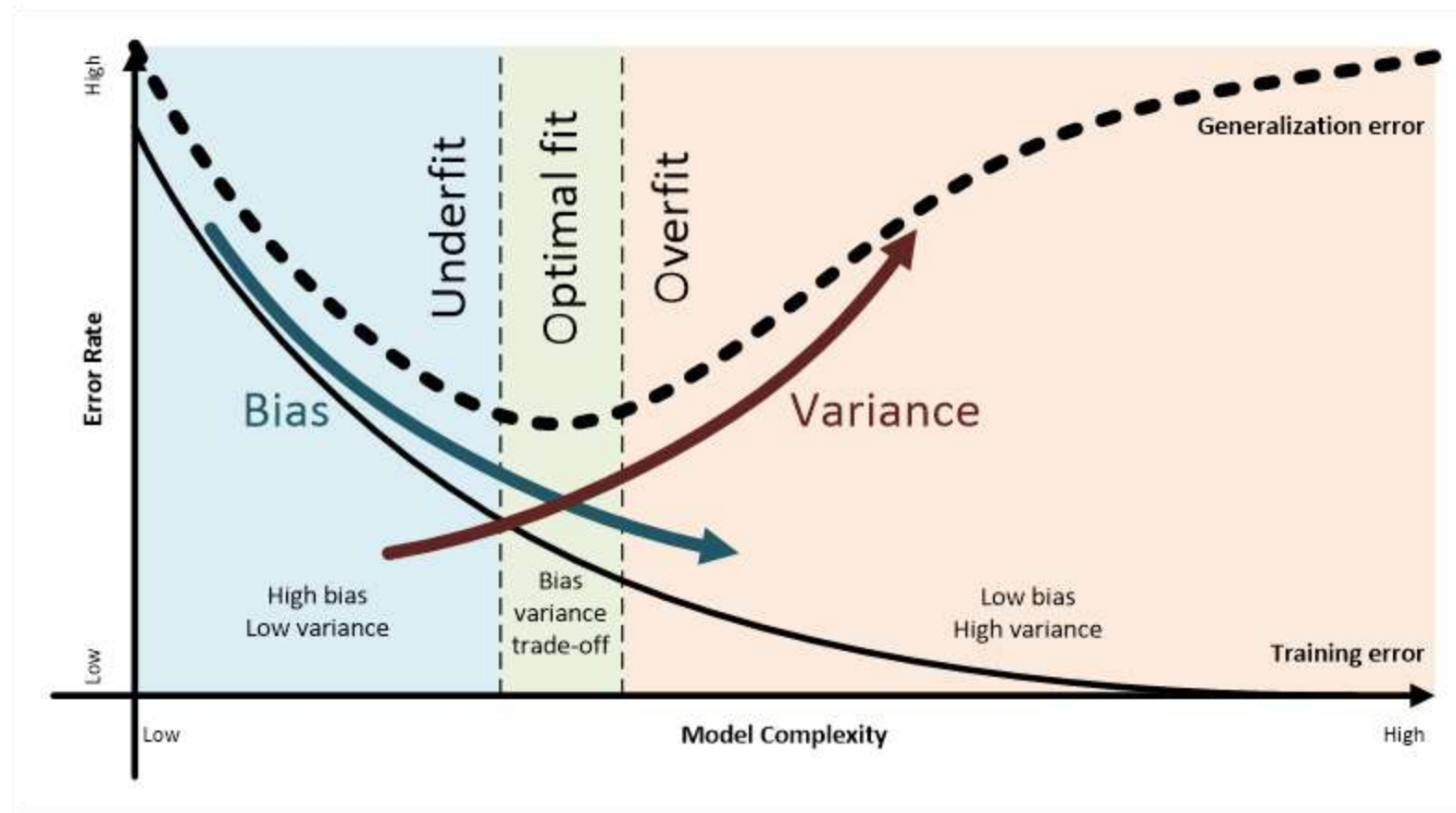
The Generalization Error is made of two components: Bias and Variance



The Bias is a systematic, non-random error; the Variance is an idiosyncratic, random error

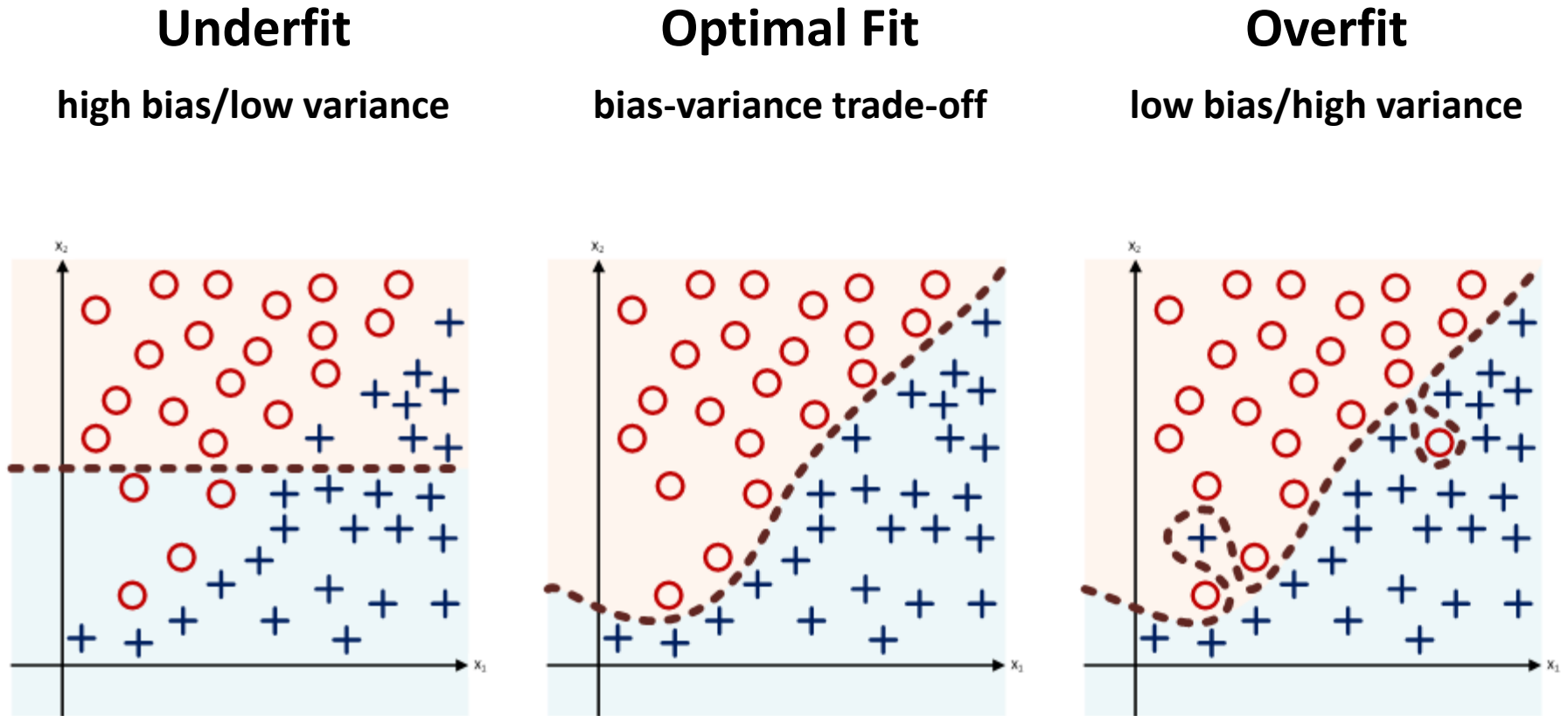


Errors, Complexity, Fit, Bias, and Variance

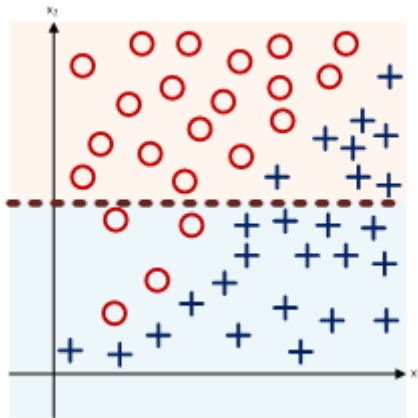
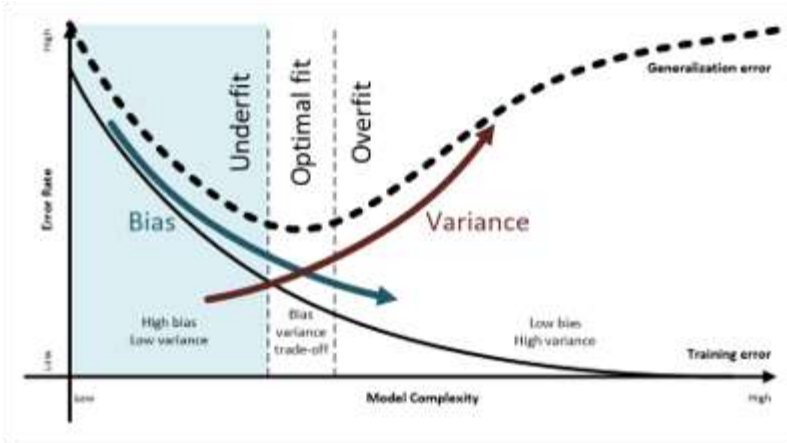


Errors, Complexity, Fit, Bias, and Variance (cont.)

EXAMPLE



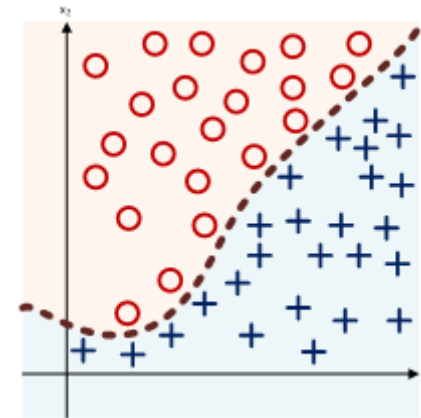
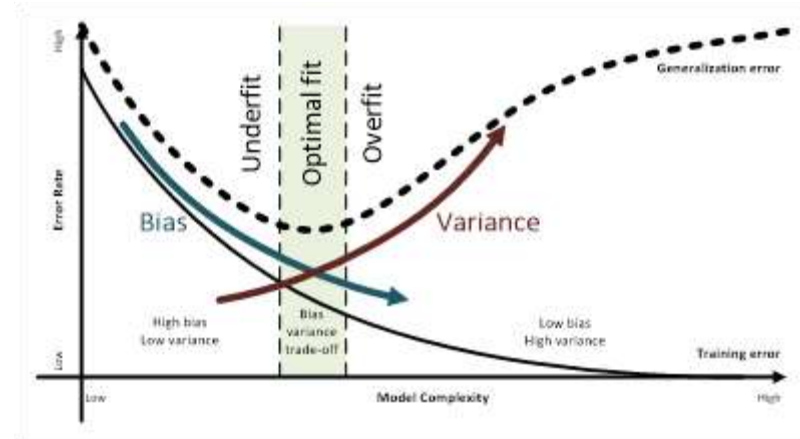
Underfit



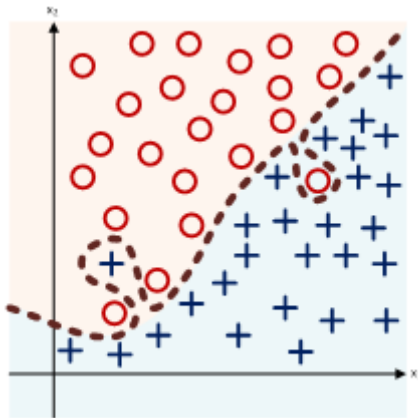
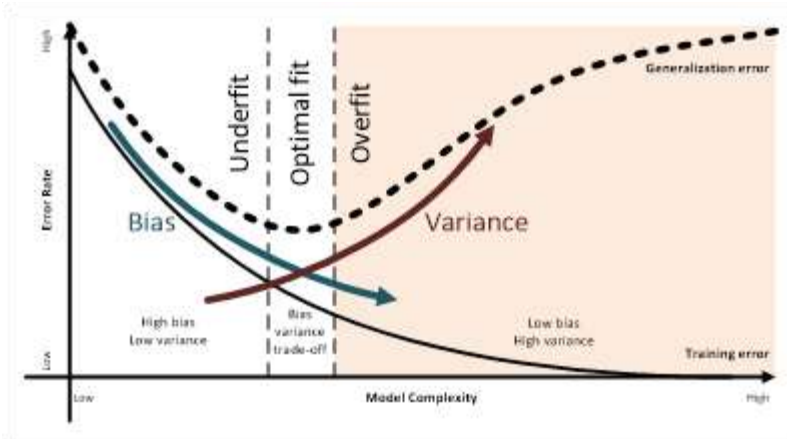
- Underfit
 - Model too simple
 - It cannot represent the desired behavior very well; both its training and generalization error are poor
 - High bias; low variance

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)

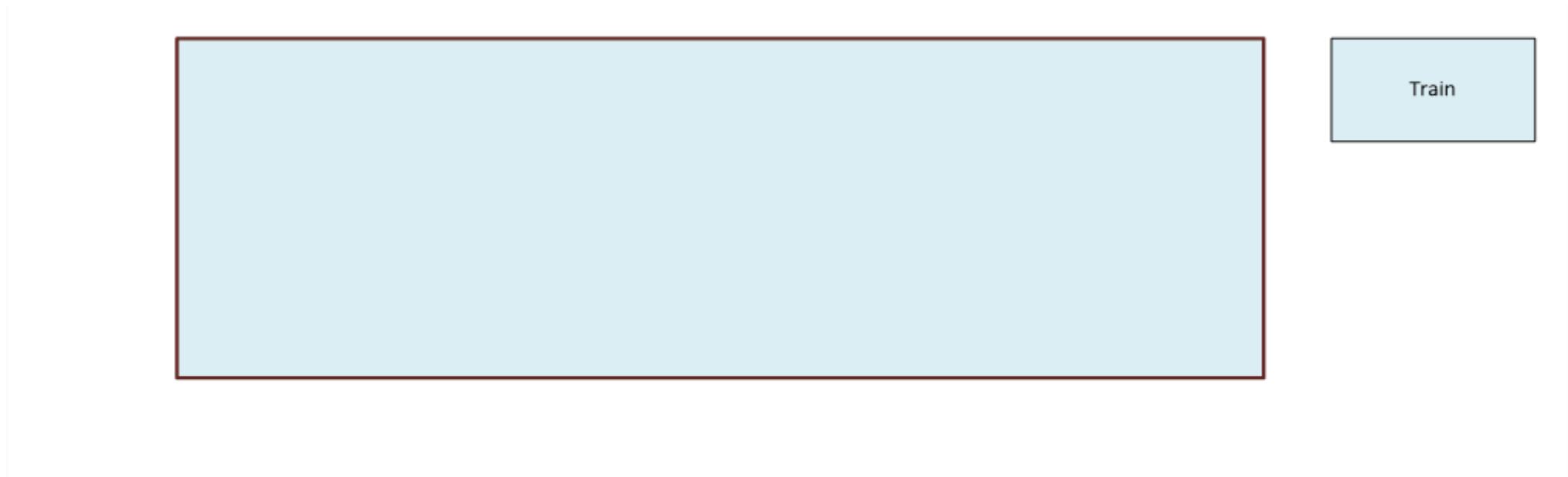


Overfit

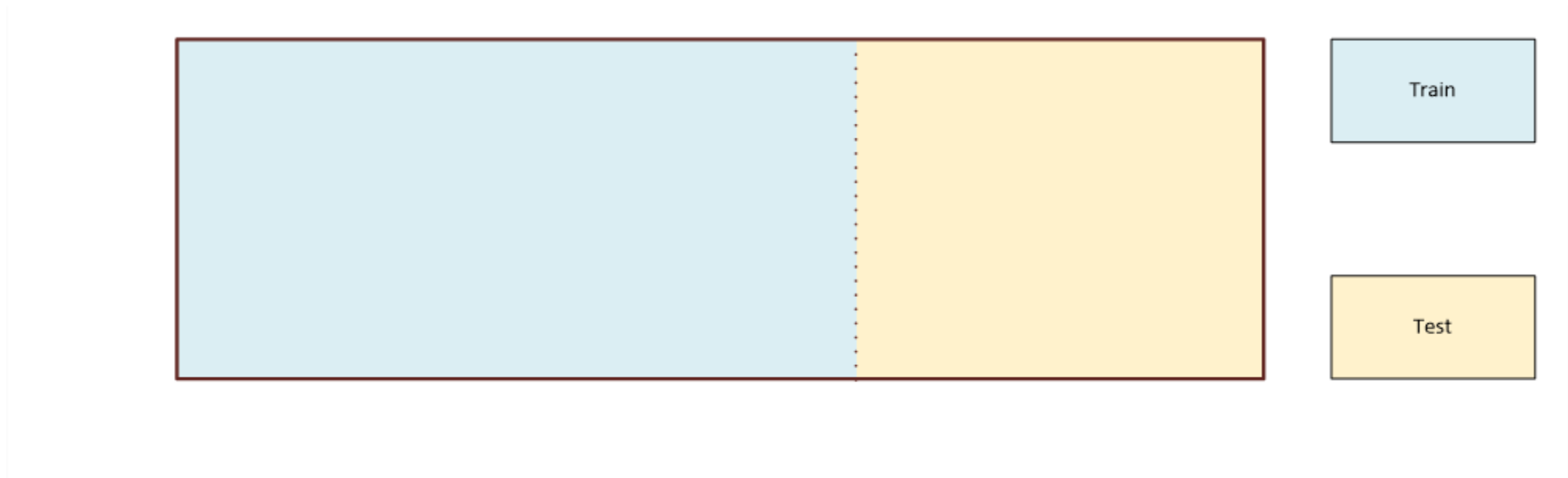


- Overfit
 - Model too complex
 - It performs very well on the training set (low training error) but does not generalize well to unseen data points (high generalization error)
 - Low bias; high variance

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

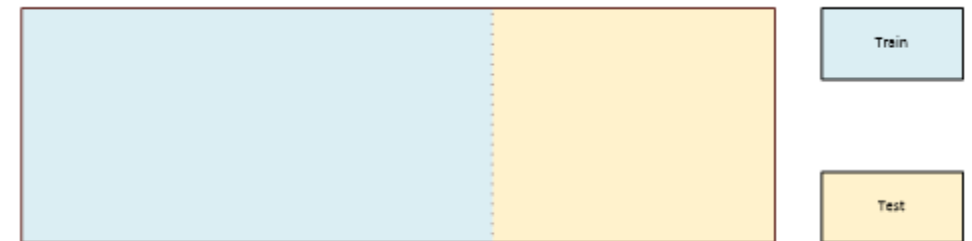


Answer: Divide (randomly) the dataset into a Train Set and a Test Set

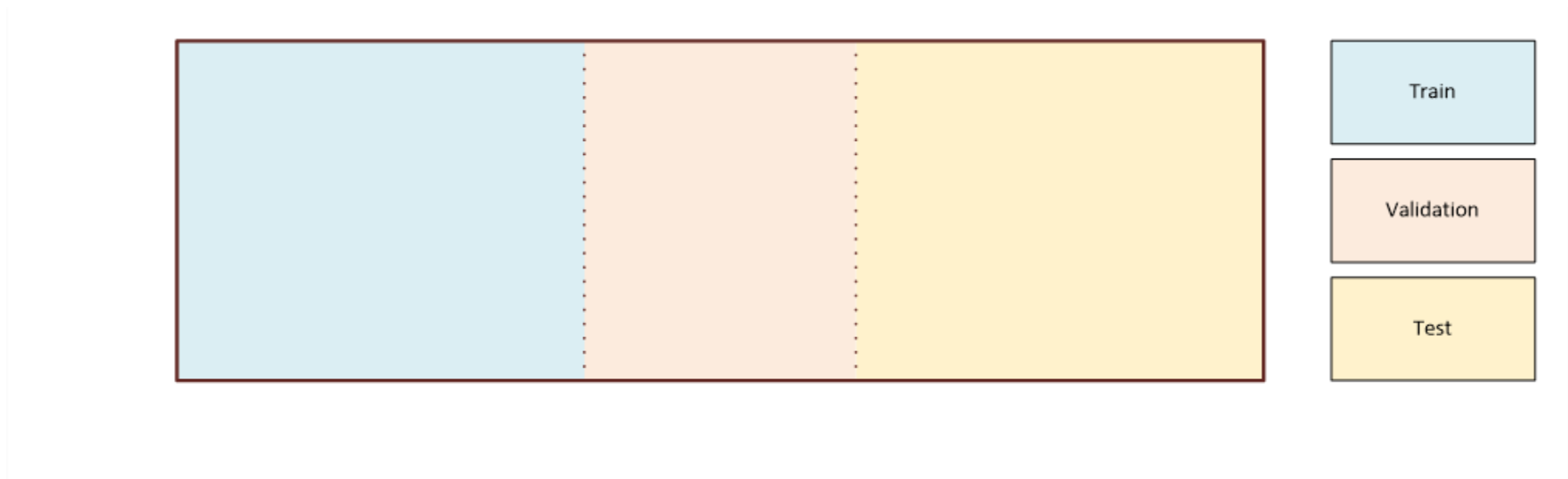


Train and Test Sets

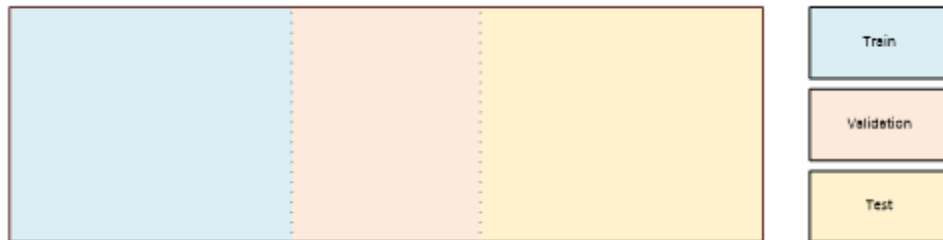
- Set aside the test set; don't look at it until the very end
- Train your model with the train set
 - Remodel as needed until you are satisfied with your model performance on the train set (low training error)
- Evaluate your model on the test set to compute the generalization error
 - Only then do 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

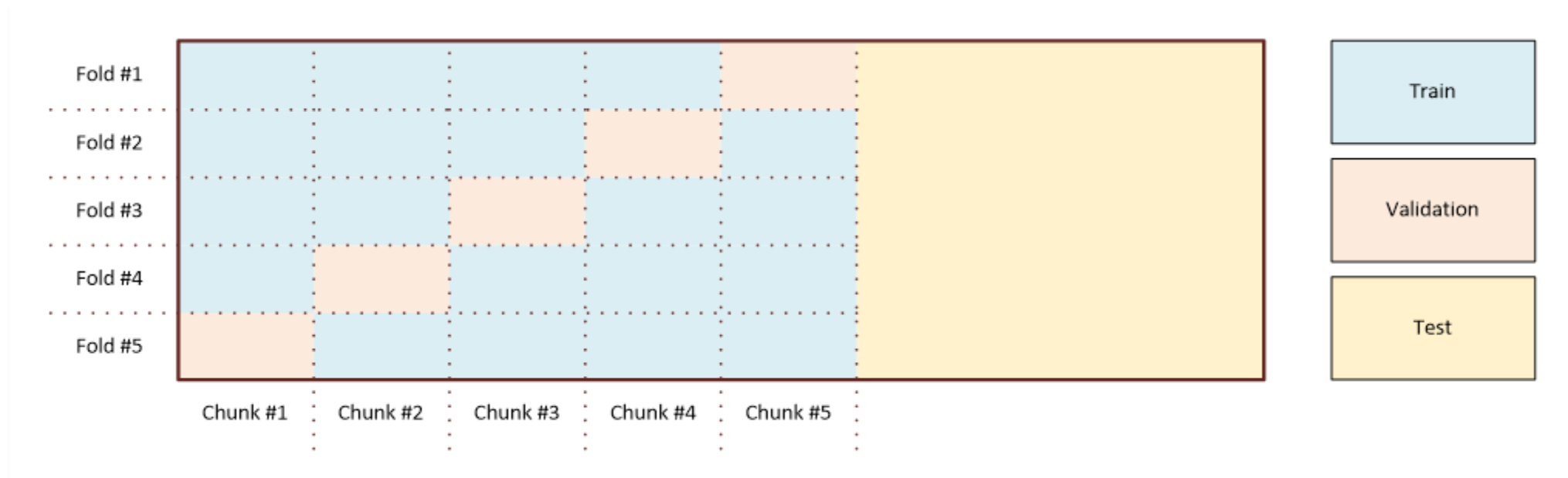


Train, Validation, and Test Sets



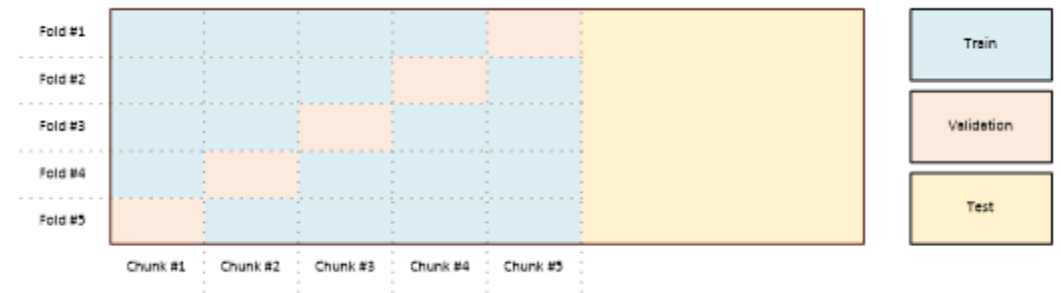
- You still train the model with the train set (model building) but now you use the cross-validation set, not the test set, to estimate the generalization error (model checking)
- After using the cross-validation set and before a new phase of remodeling, you should then reshuffle data between your train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

Answer: Yes, we can. Using k -Fold Cross-Validation

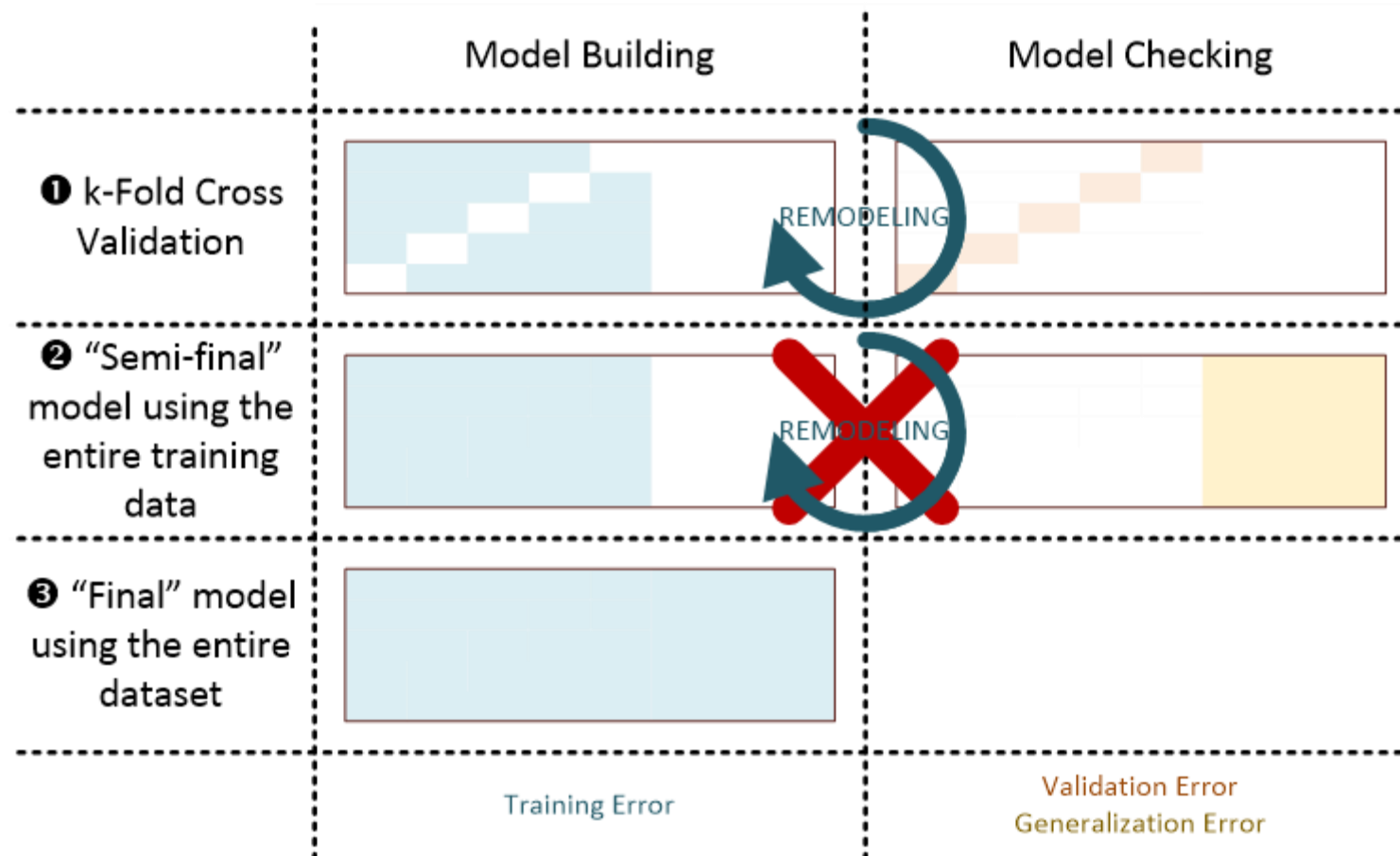


k -Fold Cross-Validation

- 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



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 $p \geq 4$

Slides © 2017 Ivan Corneillet Where Applicable
Do Not Reproduce Without Permission