

# Class 4- Machine Learning concepts

## Part I

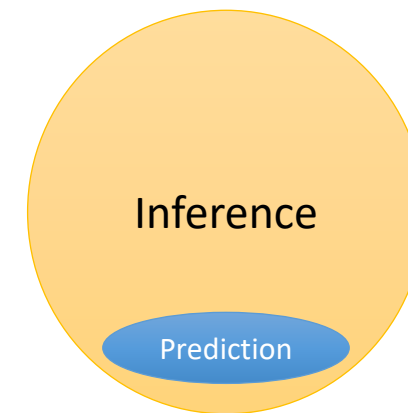
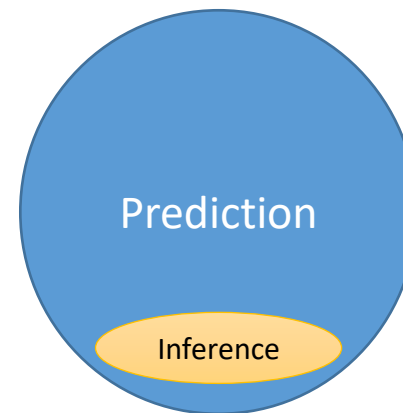




# Motivation

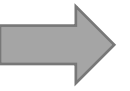
Machine learning fundamental concepts:

- Inference and prediction
- Part I: The Model
  - Parameters and hyperparameters
  - Parametric vs nonparametric ML models
- Part II: Evaluation metrics
- Part III: Bias-Variance tradeoff
- Part IV: Resampling methods
- Part V: How do machines learn?
- Part VI: Solvers/learners (GD, SGD, Adagrad, Adam, ...)



# Part I

## The Model



# The Model

---

$$y = f(X, \theta) + \epsilon = f(X_1, X_2, \dots, X_m, \theta_1, \theta_2, \dots, \theta_k) + \epsilon$$

$y$  : response, dependent variables, output, **Target**

$X$ : predictors, independent variables, input, **Features**

$\theta$ : estimates, specifications, **Parameters**

✓ It is all about estimating  $f$  by  $\hat{f}$  for two purposes:

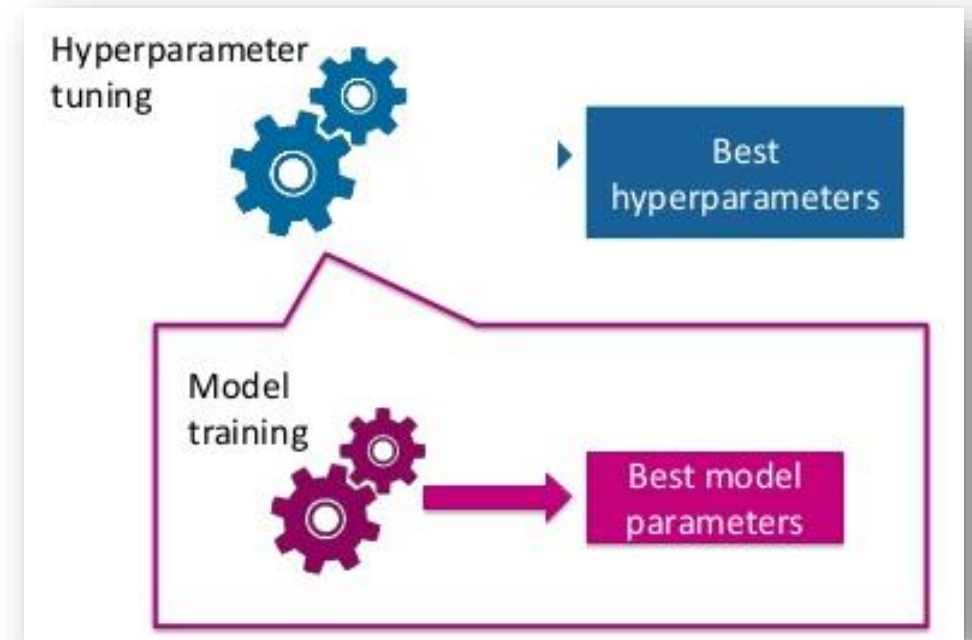
- 1) Inference (interpretable ML)
- 2) Prediction

# Parameters and Hyperparameters

$$y = f(X, \theta) + \epsilon = f(X_1, X_2, \dots, X_m, \theta_1, \theta_2, \dots, \theta_k) + \epsilon$$

Model **parameters** are estimated from data automatically and model **hyperparameters** are set manually (prior to training the model) and are used in processes to help estimate model parameters.

Example?





# Parametric Vs. Nonparametric models

$$y = f(X, \theta) + \epsilon$$

The true relationship,  $f(X)$  is **unknown** and the goal is to see which ML algorithm is better at **approximating** it. An algorithm learns/estimates  $f(X)$  from training data.

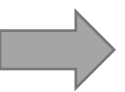
$f(X)$  is **assumed**. Examples:  
Linear regression, GLM,  
logistic regression, simple  
Neural networks, ....



	Pros 	Cons 
Parametric algorithms	<b>Simpler</b> Easier to understand and to interpret <b>Faster</b> Very fast to fit your data <b>Less data</b> Require "few" data to yield good perf.	<b>Limited complexity</b> Because of the specified form, parametric algorithms are more suited for "simple" problems where you can guess the structure in the data

# Part II

## Evaluation Metrics

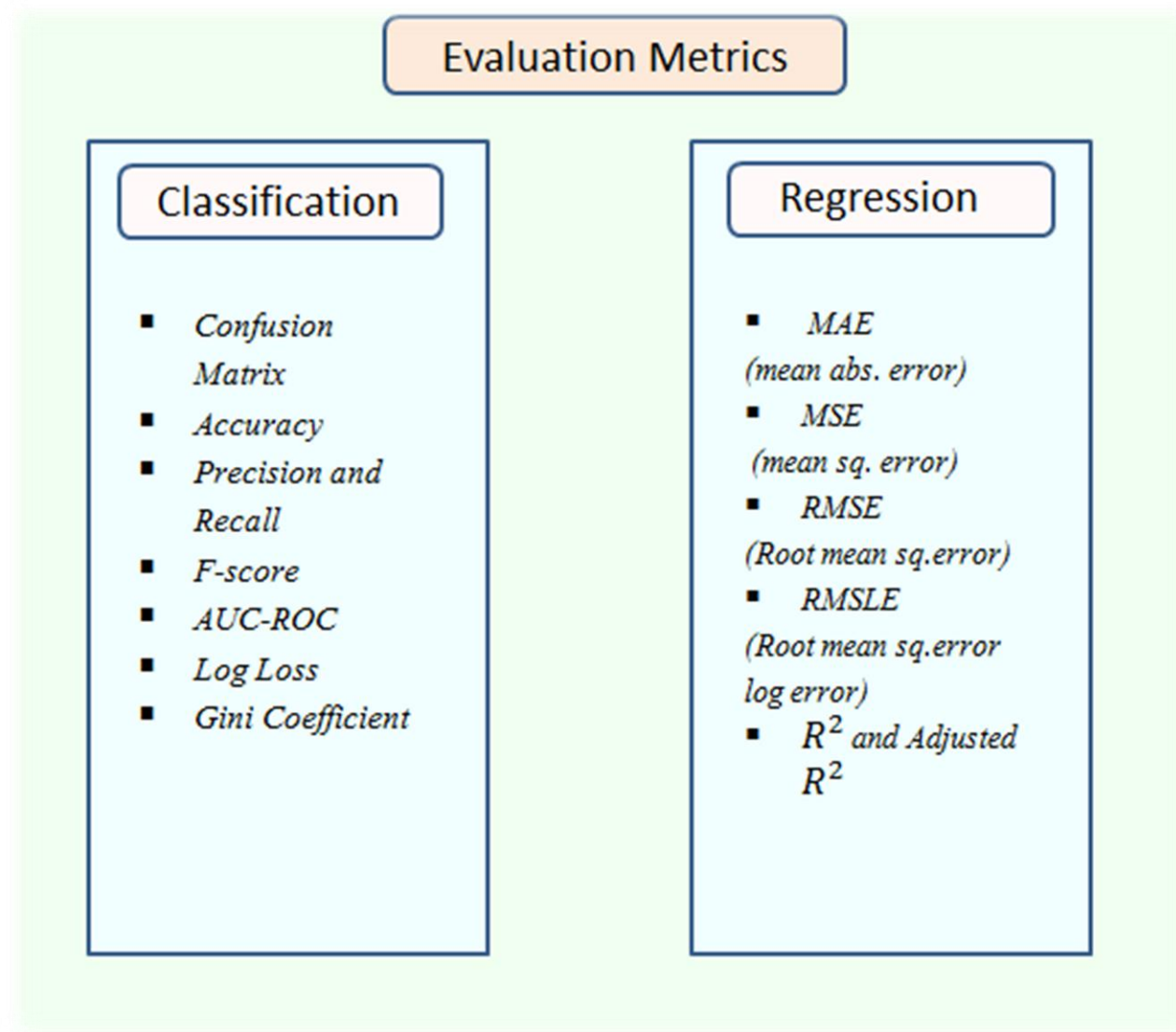


# Evaluation metrics

In general, we want to compare how close are the predictions to the actual numbers in the **test set**.

This is typically assessed using

- MSE for **quantitative** response
- Misclassification rate for **qualitative** response





# Part III

## Bias-Variance Tradeoff



# ML relative to statistical learning algorithms

- Advantages

- Ability to uncover complex interactions
- Process massive amount of data quickly
- Capture non-linear relationships
- Predict structural changes between features and target

- Disadvantages

- Can produce overly complex models
- Difficult to interpret
- Sensitive to noise
- Can overfit!

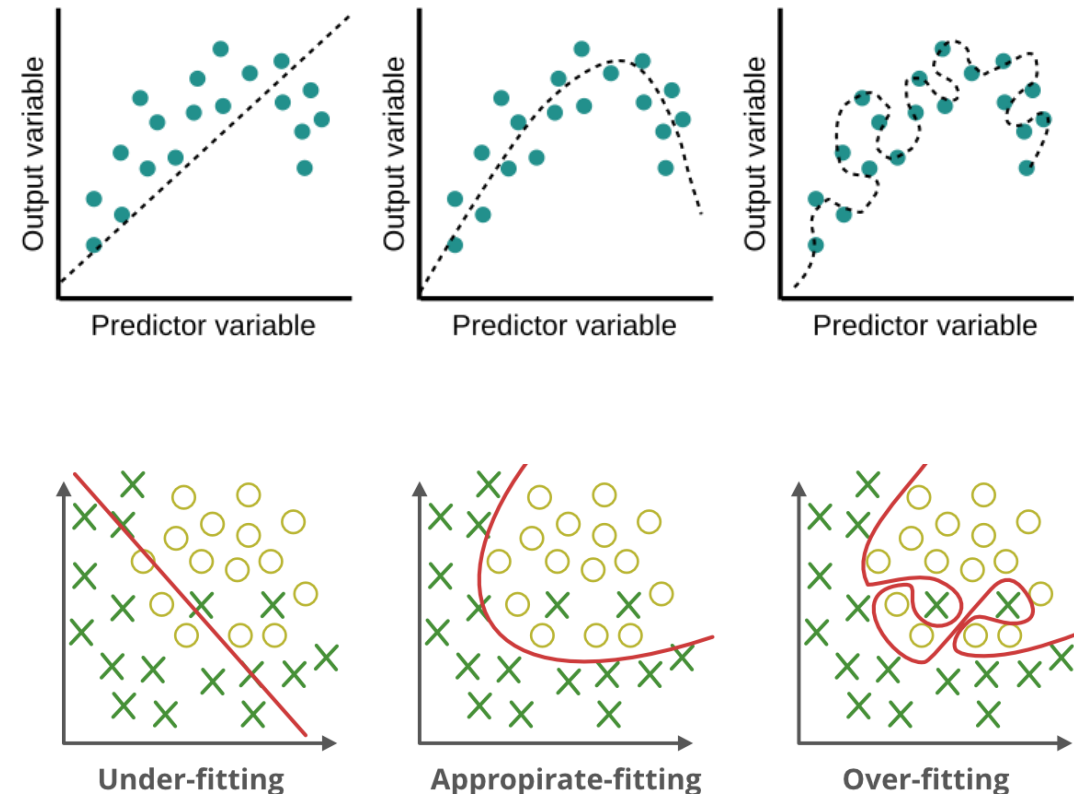
	Statistical Learning	Machine Learning
Focus	Hypothesis testing & interpretability	Predictive accuracy
Driver	Math, theory, hypothesis	Fitting data
Data size	Any reasonable set	Big data
Data type	Structured	Structured, unstructured, semi-structured
Dimensions / scalability	Mostly low dimensional data	High dimensional data
Model choice	Parameter significance & in-sample goodness of fit	Cross-validation of predictive accuracy on partitions of data
Interpretability	High	Low
Strength	Understand causal relationship & behavior	Prediction (forecasting and nowcasting)



# Overfitting

Overfitting happens when the fitted algorithm does **not generalize** well to new data:

- The model fits the training data **too** well while not predicts well in the new data
- The model **fits the noise** ( $\epsilon$ ) in training data (finds a pattern that does not exist)
- The algorithm has simply **memorized** the data, rather than **learned** from it!
- The model is too **complex**!



# → MSE decomposition

The **bias-variance** tradeoff is one of the core concepts in supervised learning.



Assume that the data is generated by a simple model!

$$y_i = f(\mathbf{x}_i) + \epsilon_i, \quad \mathbb{E}[\epsilon] = 0, \quad \mathbb{V}[\epsilon] = \sigma^2$$

The estimated model yields

$$\hat{y}_i = \hat{f}(X_i)$$

Let us decompose the mean squared error (**MSE**):

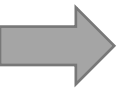
$$\begin{aligned} \mathbb{E}[\hat{\epsilon}^2] &= \mathbb{E}[(y - \hat{f}(\mathbf{x}))^2] = \mathbb{E}[(f(\mathbf{x}) + \epsilon - \hat{f}(\mathbf{x}))^2] \quad \dots = \underbrace{\mathbb{V}[\hat{f}(\mathbf{x})]}_{\text{variance of model}} + \underbrace{\mathbb{E}[(f(\mathbf{x}) - \hat{f}(\mathbf{x}))^2]}_{\text{squared bias}} + \sigma^2 \\ &= \underbrace{\mathbb{E}[(f(\mathbf{x}) - \hat{f}(\mathbf{x}))^2]}_{\text{total quadratic error}} + \underbrace{\mathbb{E}[\epsilon^2]}_{\text{irreducible error}} \end{aligned}$$

# → MSE decomposition

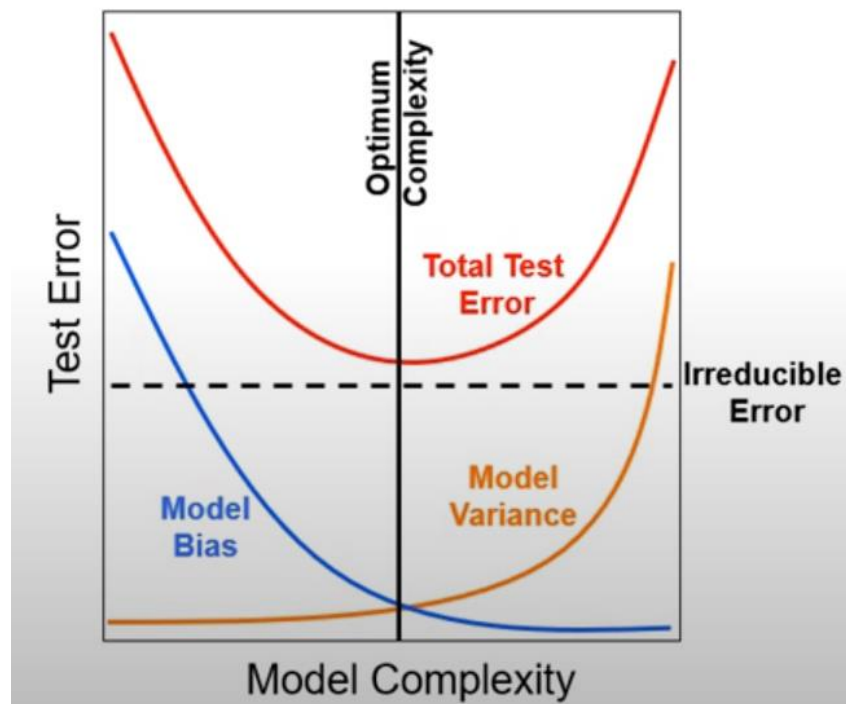
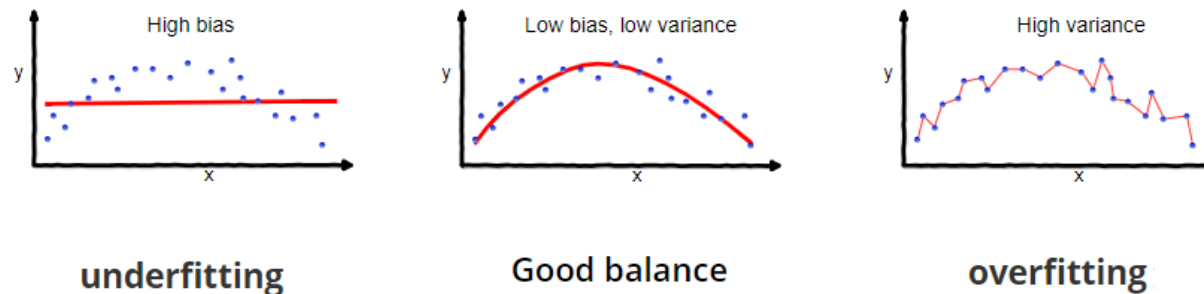
$$MSE = \text{model variance} + \text{model bias} + \text{irreducible error}$$

- 1) **Model variance** is the variance if we had estimated the model with a different **training set**
  - 2) **Model bias** is the error due to using an approximate model (model is too simple)
  - 3) **Irreducible error** is due to missing variables and limited samples. Can't be fixed with modeling
- The goal is to minimize the sum of **model variance** and **model bias**.
  - This is known as the bias-variance tradeoff because reducing one often leads to increasing the other.
  - Choosing the flexibility (complexity) of  $\hat{f}(X)$ , will amount to bias-variance tradeoff.



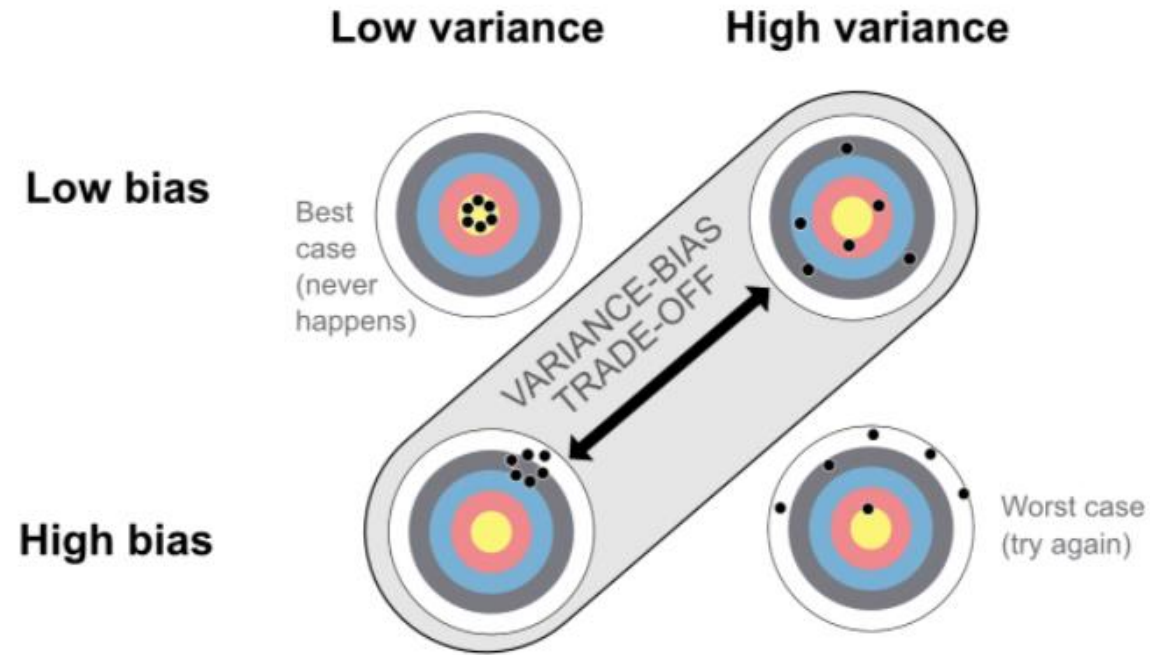


# Representations of the bias-variance tradeoff





# Other representations of the bias-variance tradeoff



# Part IV

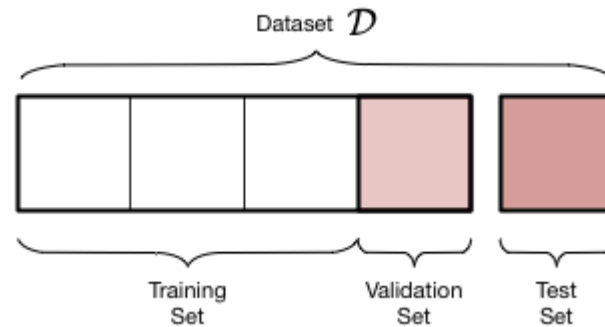
## Resampling methods



# → Partitioning of the dataset

The data set is typically divided into three non-overlapping samples:

- 1) **Training set** used to train the model
- 2) **Validation set** for validating and tuning the model
- 3) **Test set (holdout set)** for testing the model's ability to predict well on new data



To be valid and useful, any supervised machine learning model **must** generalize well beyond the training data.

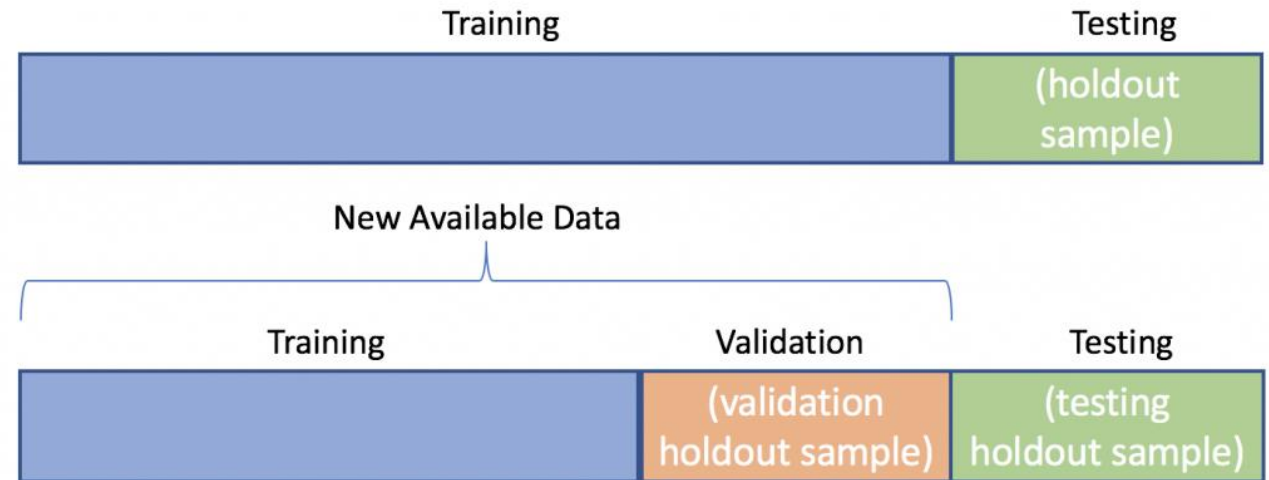
**Large dataset is needed! But what if we don't have it?**



# Resampling methods

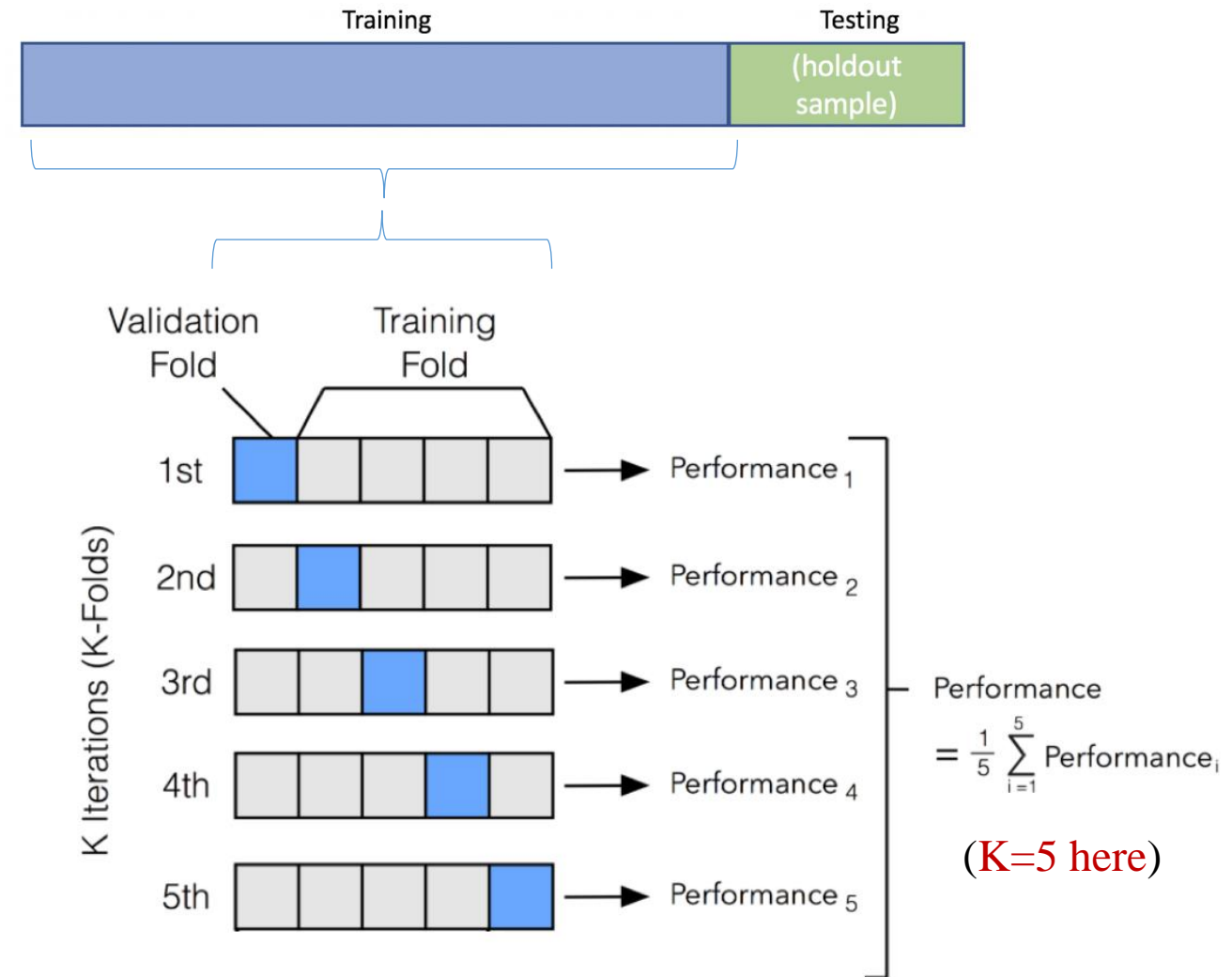
## Cross validation

- Sometimes we cannot afford to split the data in three because the algorithm may **not learn** anything from a **small training dataset**!
- **Small validation set** is also problematic because we cannot tune the hyperparameters properly!
- Solution: combining the training and validation sets!
- The goal is to obtain additional information about the fitted model!  
For example, to provide **estimates of test set prediction errors**.



# → K-fold Cross Validation

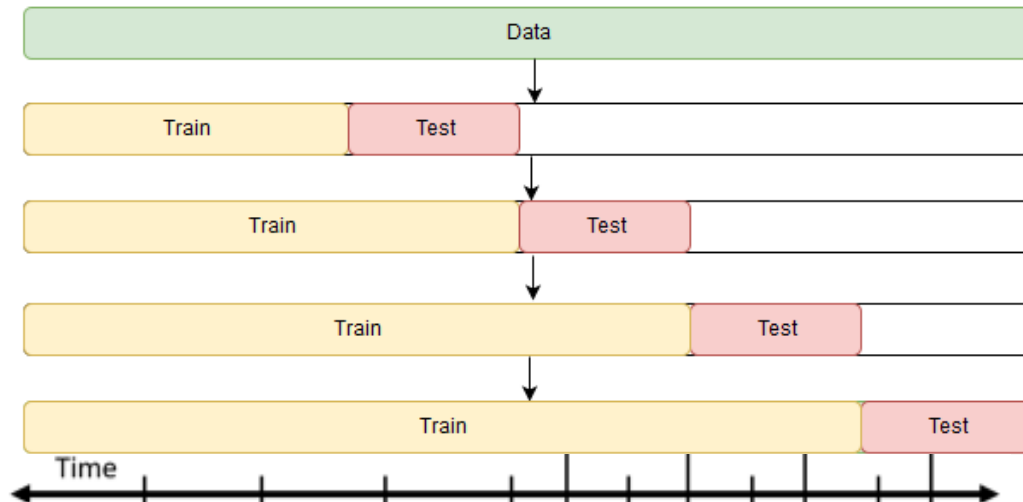
- 1) Divide the training data into  $K$  roughly equal-sized non-overlapping groups. Leave out  $k^{th}$  fold and fit the model to the other  $k - 1$  folds. Finally, obtain predictions for the left-out  $k^{th}$  fold.
- 2) Performance can be any of the evaluation metrics for regression or classification models. For example, MSE, accuracy, ...
- 3) This is done in turn for each part  $k = 1, 2, \dots, K$ , and then the results are combined.
  - Leave one out CV (LOOCV): if there is only 1 observation in each fold.



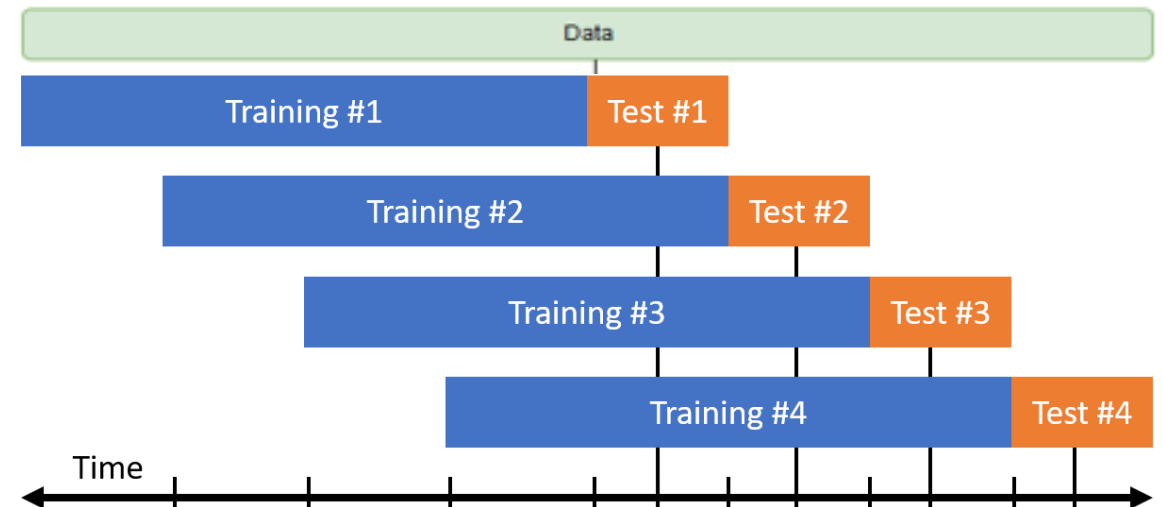
# Time Series Cross Validation

With time series data, we **cannot shuffle** the data! We also need to **avoid look-ahead bias**!

Walk forward cross validation  
**Expanding** windows



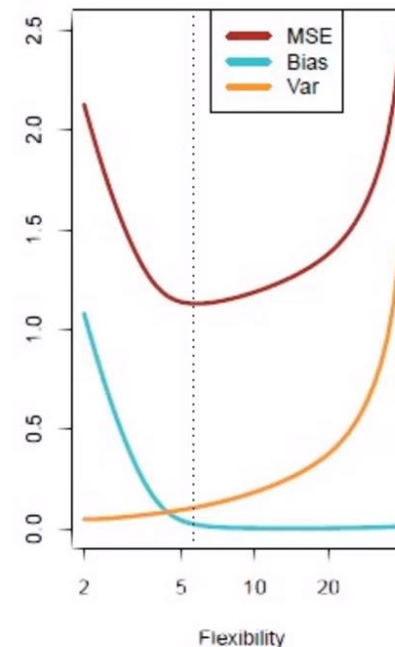
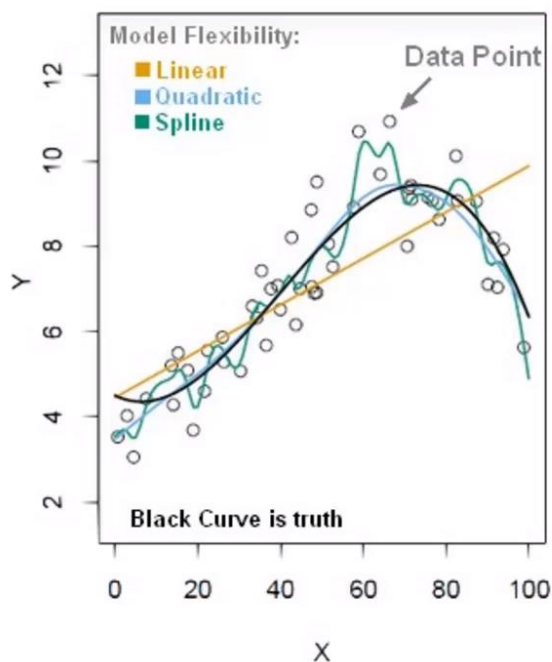
Walk forward cross validation  
**Rolling** windows



# ➔ Mitigate overfitting

The main techniques used to mitigate overfitting risk in a model construction are:

- 1) Complexity reduction (regularization)
- 2) Cross validation (estimate the test error)



# → Scaling the features

Let us use  $x_i$  for raw input and  $\tilde{x}_i$  for the transformed data. Common scaling practices include:

- Standardization:  $\tilde{x}_i = \left( \frac{x_i - \mu_x}{\sigma_x} \right)$

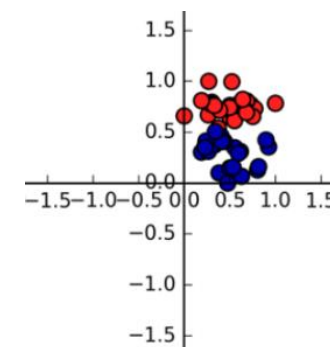
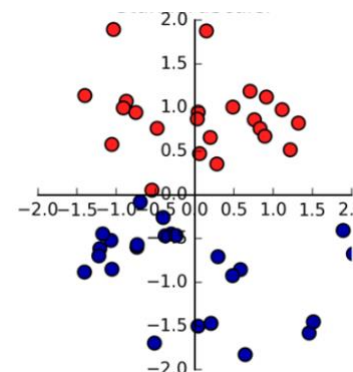
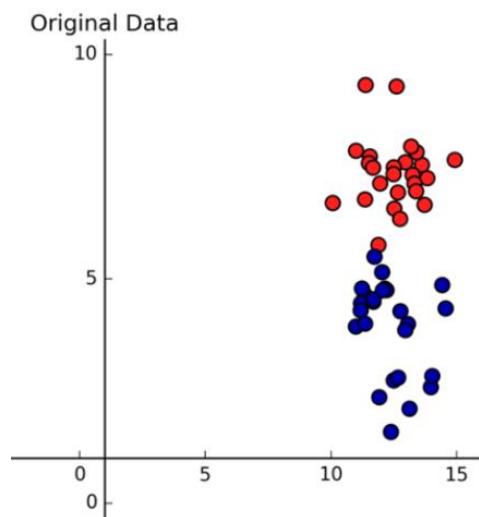
- Normalization:

- Min-Max scaler over [0,1]:

$$\tilde{x}_i = \left( \frac{x_i - \text{Min}(X)}{\text{Max}(X) - \text{Min}(X)} \right)$$

- Min-Max scaler over [-1,1]:

$$\tilde{x}_i = 2 * \left( \frac{x_i - \text{Min}(X)}{\text{Max}(X) - \text{Min}(X)} \right) - 1$$



# → Scaling the features

- Normalization is good to use when the distribution of the data does not follow a Normal distribution. (ideal for non-parametric algorithms like KNN)
- Standardization, can be helpful in cases where the data follows a Normal distribution. However, this does not have to be necessarily true.
- Unlike normalization, standardization does not have a **bounding range**. So, even if you have **outliers** in your data, they will not be affected by standardization.
- Be careful when scaling the **time series data**! Why?
- To avoid **data leakage**, It is a good practice to fit the scaler on the training data and then use it to transform the testing data.
- The choice of using normalization or standardization will **depend on** your **problem** and the **machine learning algorithm** you are using

# ➔ Question of the day!

---

