Practical ML for Engineers

Model Evaluation

Rappel

ML = Model space + Loss function (+ regularization) + optimization

Rappel: Model Space

- The model space defines the set in which we look for the best model for the data we have.
- Small model space => less complex models (=> underfitting)
- Large model space => more complex models (=> overfitting)
- In general, work with **parametrized family** of models:

$$\mathcal{M} = \{ f_{\boldsymbol{\theta}}(\boldsymbol{x}) \mid \boldsymbol{\theta} \in \boldsymbol{\Theta} \}$$

Example: m-dimensional Linear Regression Model

- Input is an m-dimensional feature vector: $\boldsymbol{x} = (x_1, \ldots, x_m)$.
- Model space is the set of all linear functions of the variables $x_1, \ldots, x_m$:

$$\mathcal{M} = \{ f(x) = a_1 \cdot \boldsymbol{x}_1 + a_2 \cdot x_2 + \ldots + a_m \cdot x_m + b \quad | \quad a_1, \ldots, a_m, b \in \mathbb{R}^{m+1} \}.$$

- Model is fully parametrized by the $(m+1)$-dimensional vector:

$$(b, a_1, \ldots, a_m) =: \boldsymbol{\theta} \in \Theta := \mathbb{R}^{m+1}.$$

Rappel: Loss Function

L2 Loss

$$L(y, \hat{y}) = (y - \hat{y})^2$$

- Penalizes large errors more than small ones
- Influenced by outliers

L1 Loss

$$L(y, \hat{y}) = |y - \hat{y}|$$

- More robust to outliers
- Less statistically well-behaved

Empirical Risk Minimization

Sum of the losses over the training dataset quantifies quality of each model, **as a function of the model parameters θ**:

$$\mathcal{J}(\boldsymbol{\theta}) = \sum_{i=1}^{n} L(y^{(i)}, f_{\boldsymbol{\theta}}(\boldsymbol{x})^{(i)})$$

Best model is the one that minimizes the loss:

$$\boldsymbol{\theta}^* = \arg\min_{\theta \in \Theta} J(\boldsymbol{\theta})$$

Rappel: Optimization

- Best parameter θ^* found by **minimizing the empirical risk** $\mathcal{J}(\theta)$
- In general, empirical risk minimized via iterative methods (gradient descent, …)
- For some class of problems with well-behaved loss functions and tractable model spaces, analytical solution exist for the optimal θ^* (linear models, …).

So this is it?

How can we get better?

Evaluate -> Understand -> Improve

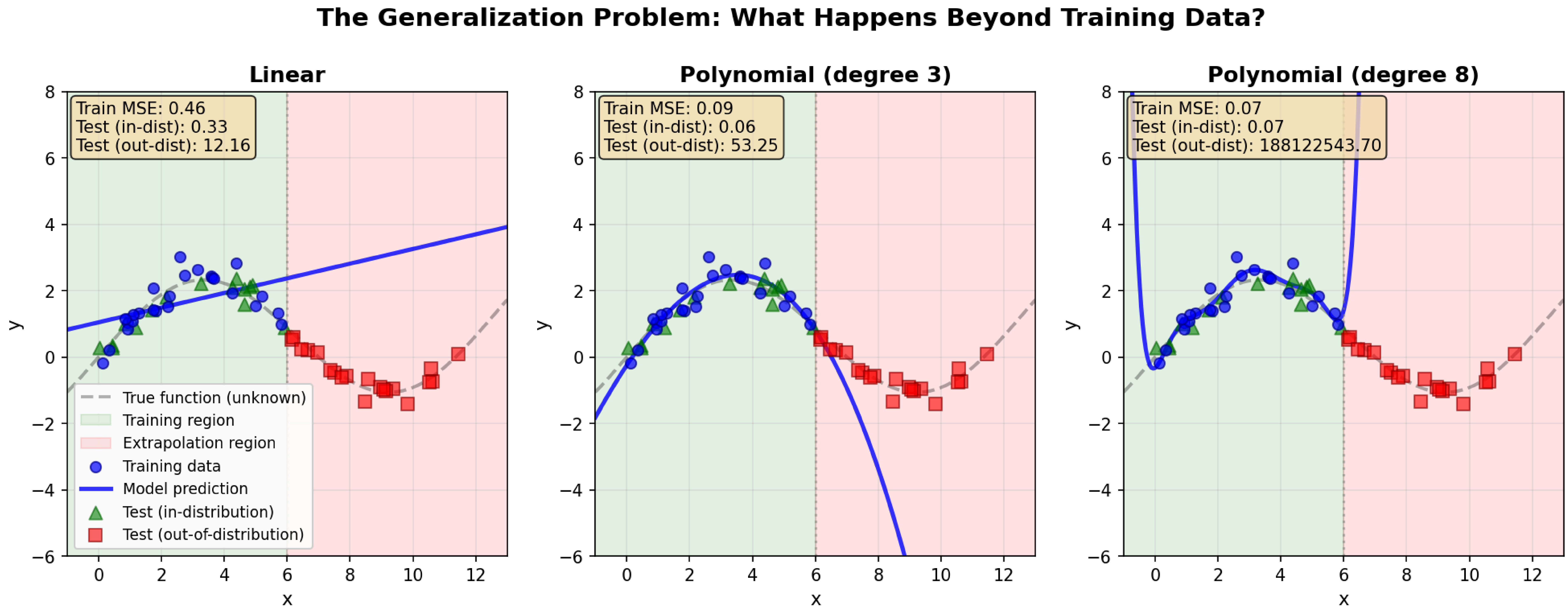
In ML, development is **iterative** and **experiment-driven**.

- => we have to try again
- => what we try has to be **guided by experiments**
- => a lot of ML is about carefuly designing experiments so that they provide us new knowledge

Part 1: Evaluation

Bias-Variance Tradeoff and the problem with Generalization

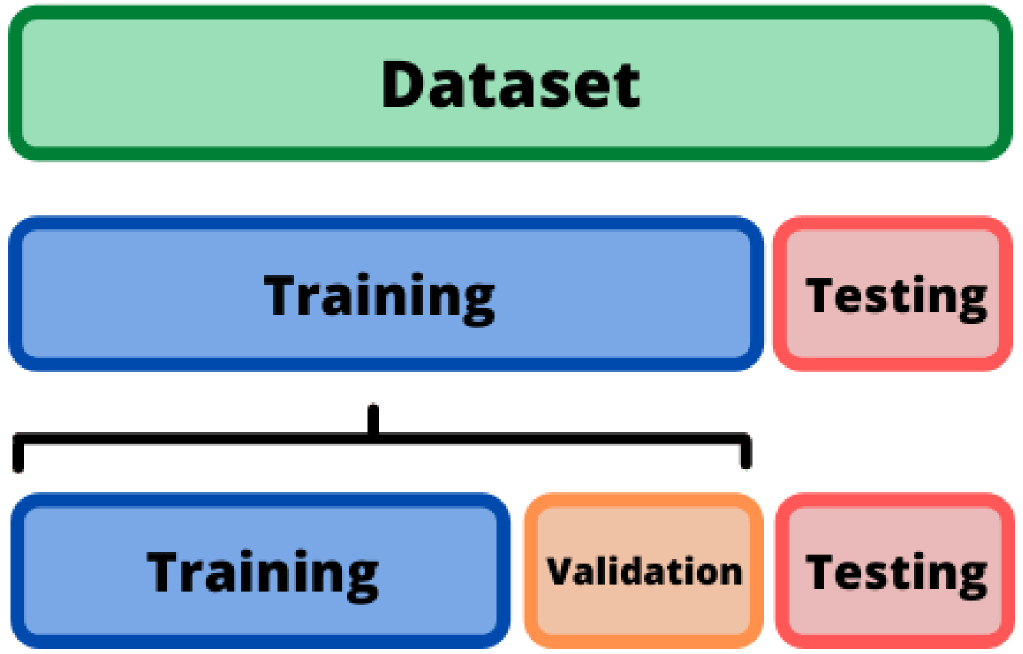
Fundamental Challenge: Limited Training Data



- **avoid extrapolation!!!**
- do not train on all data

Train-Test (-Val) Split

Idea: keep some data on the side (holdout) and **do not train on it.**



- Usual practice is to split the data in:
  - **Training data** (60%): used for model training
  - **Validation** data (20%): used for model selection
  - **Test** data (20%): used for final, honest performance evaluation.
- Test data is **never touched during model training and selection!**

Performance metrics (RMSE, MAE, R2, …) on test set used to estimate model performance on unseen data.

Model Selection

Why do we need a validation set?

Because the training process can be “too efficient”

- Model training is tailored to fit the data as closely as possible
- Can lead models to rely **too much on details** of the training data (noise) and not enough on the big picture

Overfitting

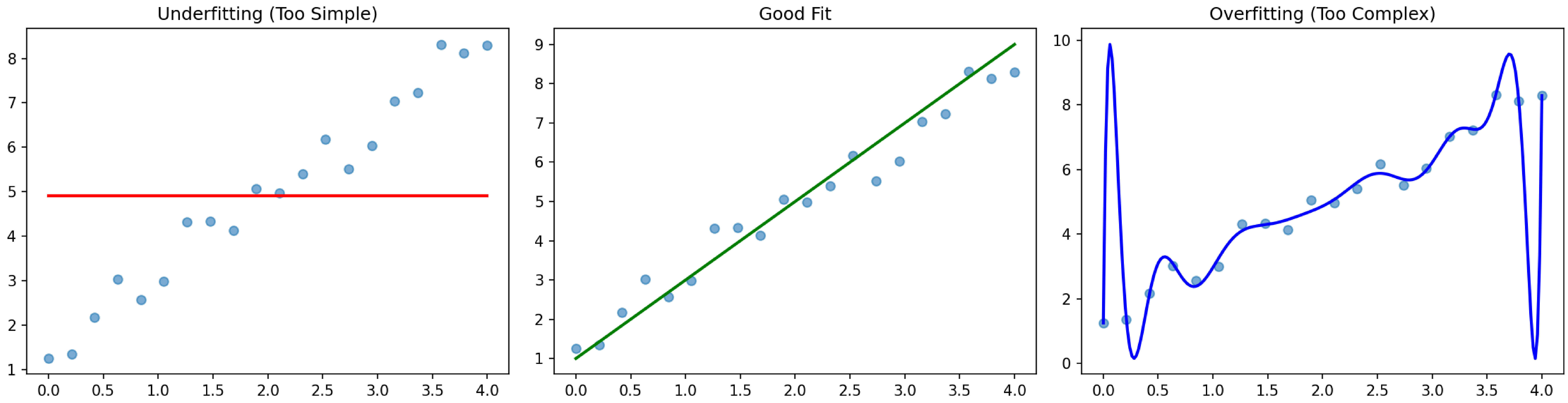
Overfitting and Underfitting

Overfitting

- Model too complex
- Low training error
- High validation error
- **Solution**: Regularization, more data, simpler model

Underfitting

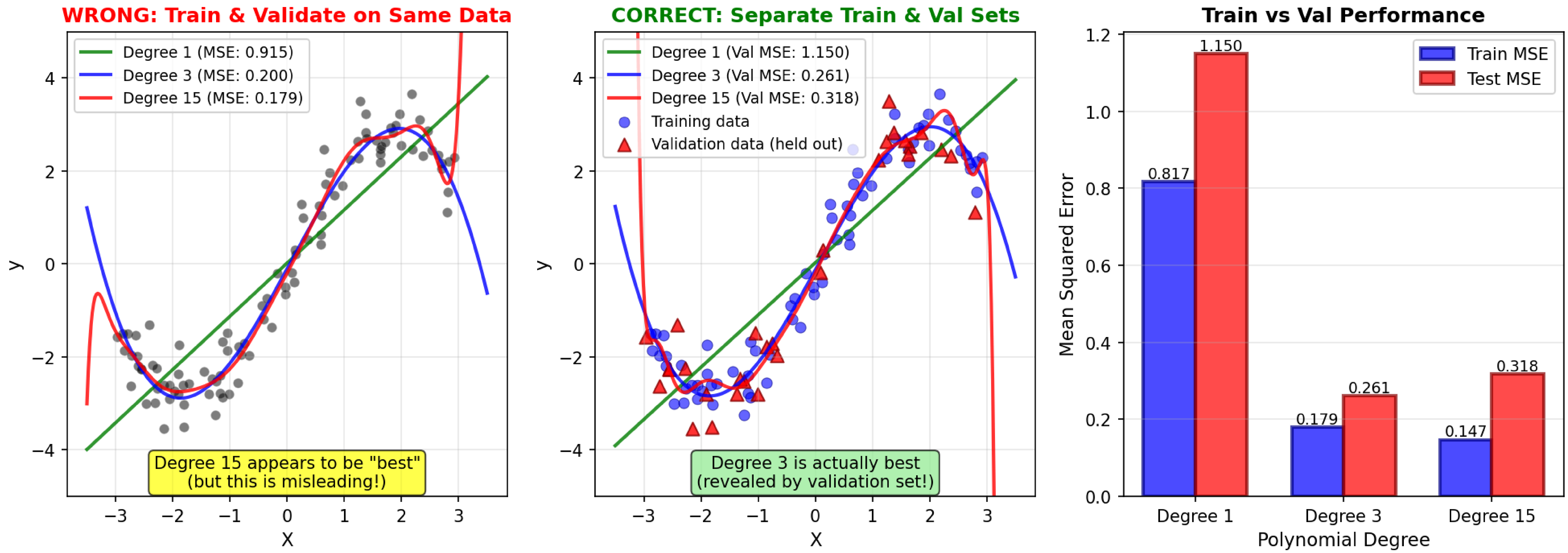
- Model too simple
- High training error
- High test error
- **Solution**: More complex model, more features



Preventing Overfitting: Model Selection

Model Selection

1. Design different models
2. Train all models on the train set
3. Select best model by comparing performances on validation set



Train-Test Split: Takeaways

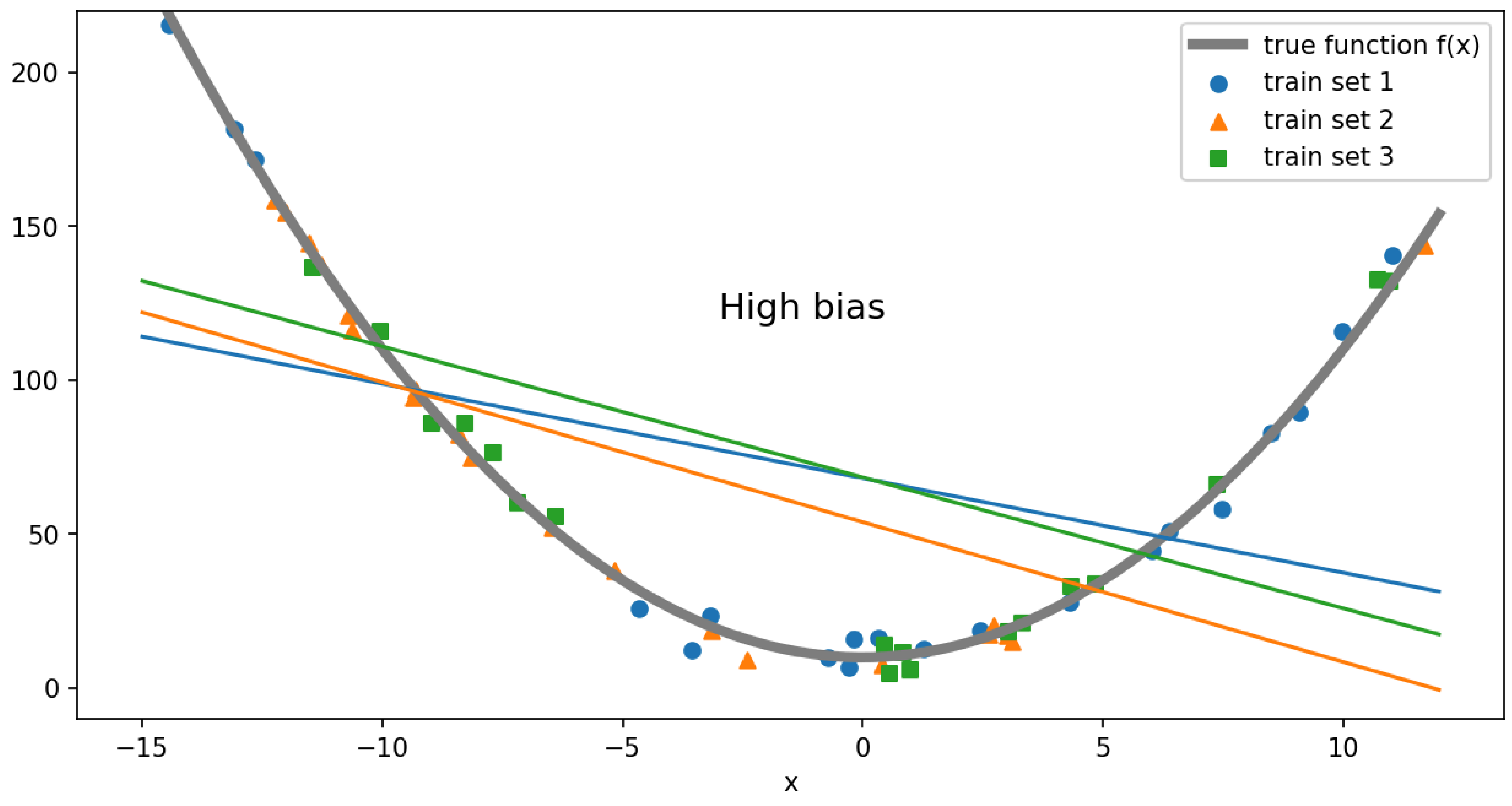
Train-val-test procedure

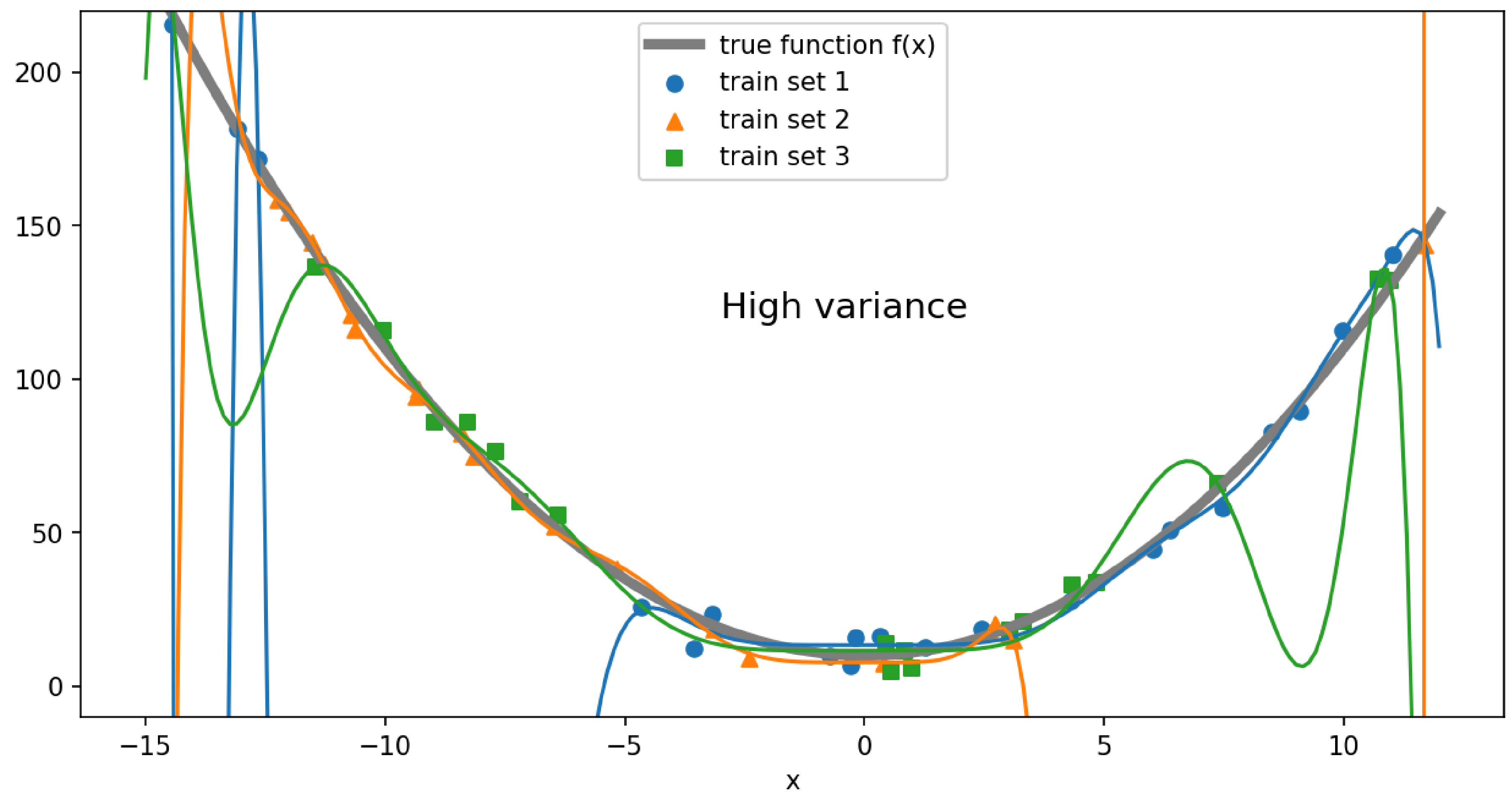
1. **Training set**: Learn patterns (60%)
2. **Validation set**: Select best model (20%)
3. **Test set**: Final reality check (20%)

Train → Validate (for model selection) → Test (final evaluation only)

- **Random splitting** - Ensures representative samples
- Chase **leakage** -> split **before** fiting anything on the data
- After model selection is done, final model can be trained on the combined train-val dataset
- **Test data is sacred**, never touch it until the end

Interlude: Bias-Variance Tradeoff





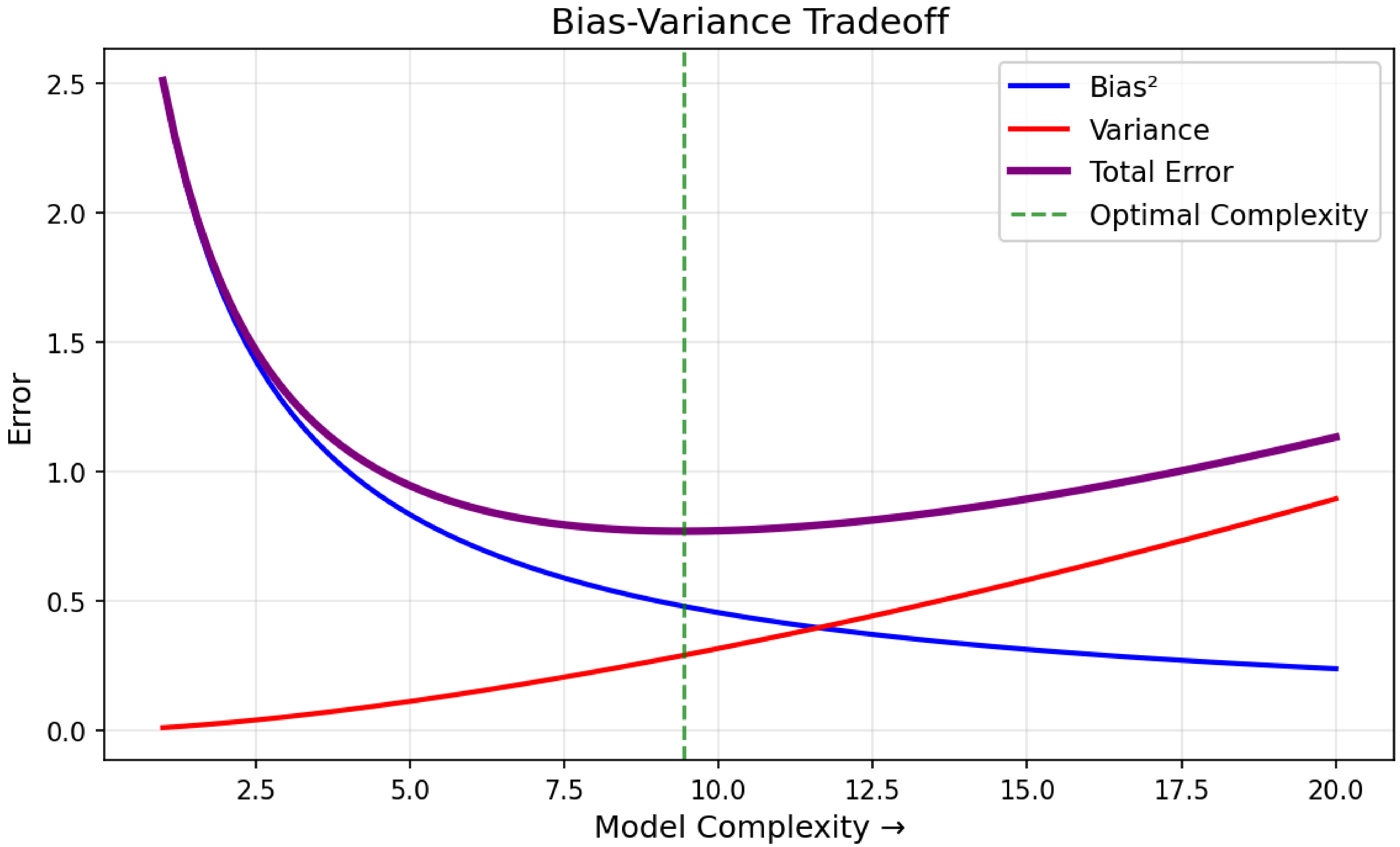
Bias

- unability of the model to fit the data
- due to lack of flexibility of the model
- linked to a too small model space (too low model complexity)

Variance

- high sensitivity of the model to (details of) the data
- due to an excess of flexibility of the model
- linked to a too large model space (too high model complexity)

The Bias-Variance Tradeoff



- **Bias**: Error from wrong assumptions (underfitting)
- **Variance**: Error from sensitivity to training data (overfitting)
- **Sweet spot**: Minimize total error

The No Free Lunch Theorem

"There is no such thing as a free lunch." (D. Wolpert and W. MacReady)

> All models are wrong but some are useful. (G. Box)

Part 2: Understanding

Once models have been trained and evaluated, it is time to try to understand **how they fail**

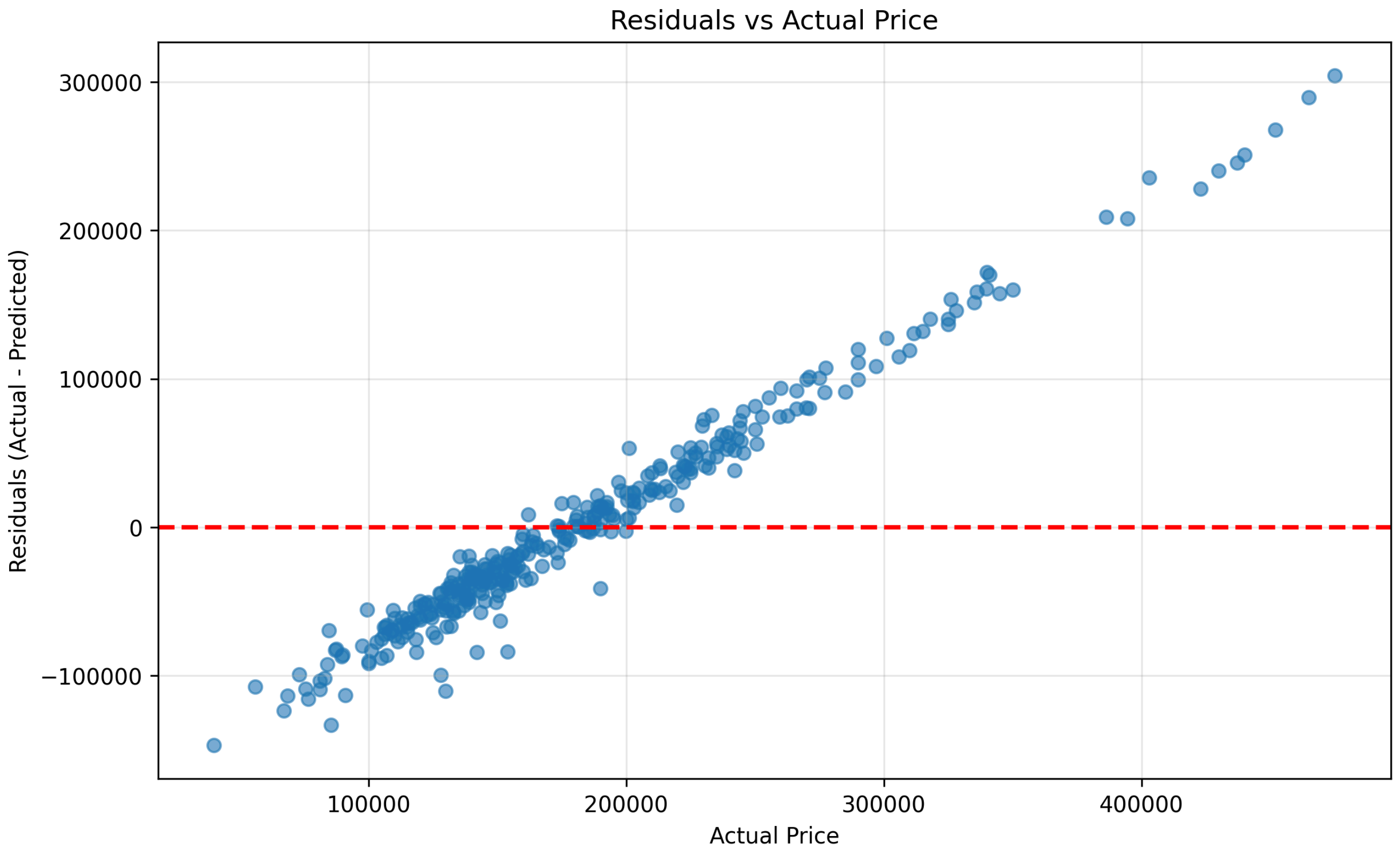
- **residuals analysis**
- detailed analysis of **worst predictions**

Residuals

Residuals are the part do the data that is not captured by the model (= error):

$$r_i := y_i - \hat{y}_i$$

In an indeal world, residuals should be **i.i.d Gaussian distributed**.



Residuals: What to check

- Plot residuals vs target variable -> is there structure
- Is the variance of the residuals constant (homoscedasticity)
- Are the residuals normally distributed (qq-plot)

(Pro tip) Worst Predictions Analysis

Part 3: Improving

- **feature engineering** (not covered here)
- **regularizazion**

Regularization

Regularization

Idea: add a penalty term to the loss function to constrain model complexity:

$$J_{regularized}(\theta) = J_{original}(\theta) + \lambda \cdot R(\theta)$$

- $J_{original}(\theta)$: Original loss (e.g., MSE)
- λ: Regularization strength (hyperparameter)
- $R(\theta)$: Penalty term on parameters

Key Insights

- Simpler models generalize better
- Can force model to be simple by making it pay for complexity during training

L2 Regularization (Ridge Regression)

$$J_{Ridge}(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} \theta_j^2$$

- Penalizes **squared** magnitude of coefficients
- Shrinks coefficients toward zero (but not exactly zero)

L1 Regularization (Lasso)

$$J_{Lasso}(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} |\theta_j|$$

- Penalizes **absolute** magnitude of coefficients
- Can shrink coefficients to **exactly zero**
- Performs a kind of automatic **feature selection**

Regularization: Procedure

Regularization procedure

- Validation set is used to choose regularization strength parameter λ
- In practice, **cross-validation** is generally used (next lecture)

Important

When using regularization, all features should be **standardized / normalized** (next lecture).

Sources

- https://dhavalpatel2101992.wordpress.com/2021/05/21/kaggle-titanic-dataset-cleaning-split-data-into-train-validation-and-test-set/