פשרה = trade-off

מטרה: לאפיין מודל

4. Model Characterization

Goal:
- Interpret cross-validation results.
- Identify fundamental trade-offs and/or relations between:
  - Model complexity and generalization performance.
  - Model complexity and the size of dataset.

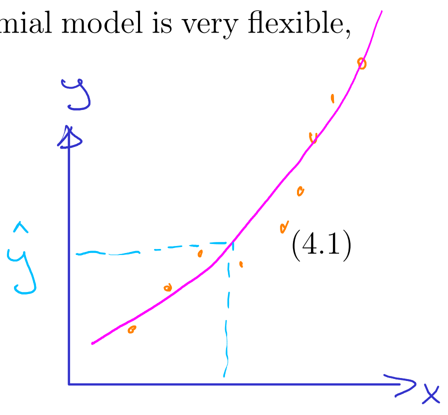
Contents

4.1. Uni-variate Polynomial Model

Goal: Extend a linear model "engine" to polynomial models. The polynomial model is very flexible, e.g. due to the Taylor expansion theorem.

The L-degree uni-variate polynomial regression model is

$$\hat{y} = f(x; \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_L x^L \\ = \sum_{j=0}^{L} w_j x^j \tag{4.1}$$



This problem is linear by the change of variables, $z_j = x^j,\ j = 0, \dots, L,$

$$\hat{y} = \sum_{j=0}^{L} w_j z_j \tag{4.2}$$

$$= w_0 + w_1 z_1 + \dots + w_L z_L$$

The corresponding prediction for the dataset $\{x_k, y_k\}_{k=1}^{M}$ can be easily written by using the matrix notation (also termed Vandermonde matrix),

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^L \\ 1 & x_2 & x_2^2 & \cdots & x_2^L \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_M & x_M^2 & \cdots & x_M^L \end{bmatrix} \qquad y = \begin{bmatrix} y_1 \\ \vdots \\ y_M \end{bmatrix} \tag{4.3}$$

The weights vector values are straightforward.

Hyper-parameter: The value of L is termed hyper-parameter of the model (further discussed in Sec. 7.1.2).

$$\mathbf{w}_{opt} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y} \leftarrow$$

מהרצאה קודמת

דוגמה למדד ביצועים - MSE

4.2. Generalization

Examples of questions of the significant importance:
- Is a model $\hat{y} = f(x; \mathbf{w})$ appropriate?
- How do we evaluate a model performance?
- How to select optimal hyper-parameter values, e.g. L of the polynomial model?

Goal: The goal is:

- Estimate some metrics of the model and understand the limitations on this estimate.
- Trial and error approach.
- Understand the limitations on the effectiveness of the model's ability to make inferences on unfamiliar data.

Let's assume that the dataset $(\mathbf{X}, \mathbf{y})$ are M samples drawn from some (unknown) joint probability distribution, $\mathcal{D}$. The theoretical model performance is given by some average model *metric* (not loss) over all (theoretically) possible points from $\mathcal{D}$,[1]

תאורטית - אינסוף מדידות

$$\bar{J}_{theory} = \lim_{M\to\infty} \frac{1}{M} \sum_{k=1}^{M} J(y_k, \hat{y}_k) \tag{4.4}$$

מעשית - מספר מדידות

מוגבל (עד מאוד)

Generalization: The difference between:

- Performance metric over dataset that is used to train the model, $\bar{J}_M$.
- Theoretical performance metric $\bar{J}_{theory}$ from (4.4).

Better generalization means smaller difference between model performance and theoretical performance.

The problem is that the distribution $\mathcal{D}$ is unknown in most of the practical applications. Moreover, in practice, the value of M is (very) limited and $\bar{J}_M$ can significantly differ from $\bar{J}_{theory}$.

The generalization gap has two main sources:

[1]Probabilistic formulation, $E_{\mathcal{D}}[J(y, \hat{y})]$

- **Data-related**: limited dataset size M, sampling bias, noise in the data, and non-representative samples from $\mathcal{D}$.
- **Model-related**: model complexity mismatch (underfitting or overfitting), inappropriate model assumptions, and insufficient regularization.

In the following, methods to reduce both data-related and model-related gaps such that $\bar{J}_M \approx \bar{J}_{theory}$ will be discussed.

The generalization concept is illustrated in Fig. 4.1.

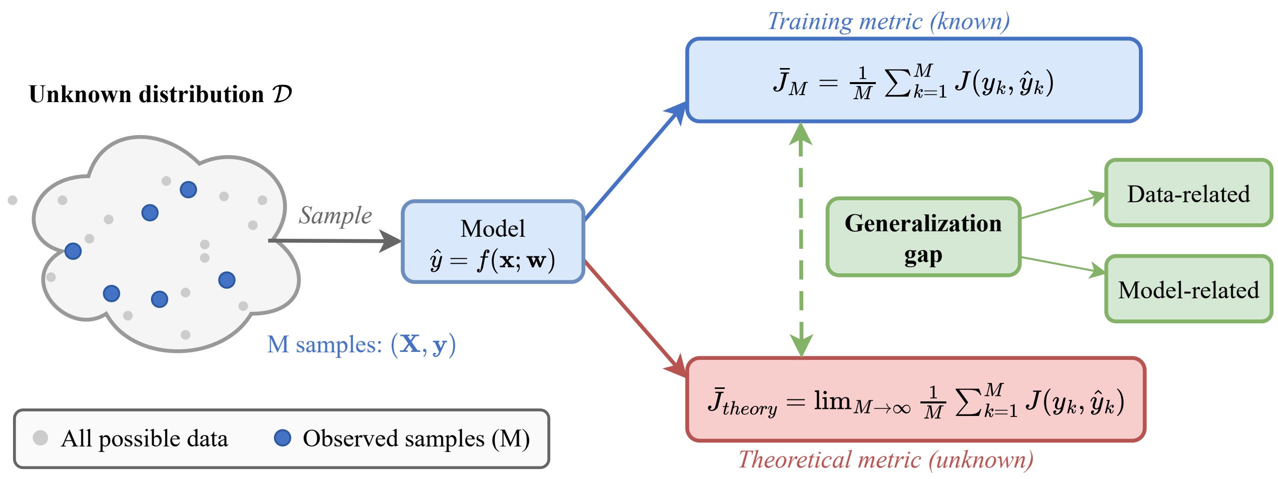

Unknown distribution $\mathcal{D}$
Sample
M samples: $(\mathbf{X}, \mathbf{y})$
Model
$\hat{y} = f(\mathbf{x}; \mathbf{w})$
Training metric (known)
$\bar{J}_M = \frac{1}{M}\sum_{k=1}^{M} J(y_k, \hat{y}_k)$
Generalization gap
Data-related
Model-related
$\bar{J}_{theory} = \lim_{M\to\infty} \frac{1}{M}\sum_{k=1}^{M} J(y_k, \hat{y}_k)$
Theoretical metric (unknown)
● All possible data
● Observed samples (M)

Figure 4.1.: Generalization: the gap between the training metric $\bar{J}_M$ (computed on M observed samples) and the theoretical metric $\bar{J}_{theory}$ (over the entire unknown distribution $\mathcal{D}$ of possible inputs). The gap has data-related and model-related sources.

4.3. Cross-validation

Goal: Trial and error approach to quantify generalization performance. The cross-validation is also termed performance assessment. Outcomes:

- Approximated generalization performance, $\bar{J}_M \approx \bar{J}_{theory}$.
- Hyper-parameters selection guideline.

The cross-validation methods are illustrated in Fig. 4.2.

Big dataset $(10^4 + \lesssim M)$: train/validation/test

First step is to resample the dataset into the **random order**. Then, split it into three *distinctive* datasets:

1) - *Training* dataset (50-80%): used for learning of model parameters, e.g. weights $\mathbf{w}$.
2) - *Validation* (10-25%): used for assessment of model hyper-parameters influence.
3) - *Test* (10-25%): performance assessment that is supposed to be sufficiently close to $\bar{J}_{theory}$.

2) → L בוחר → מריץ למידה של training * ⇒ L [illegible]

validation של בדיקה *

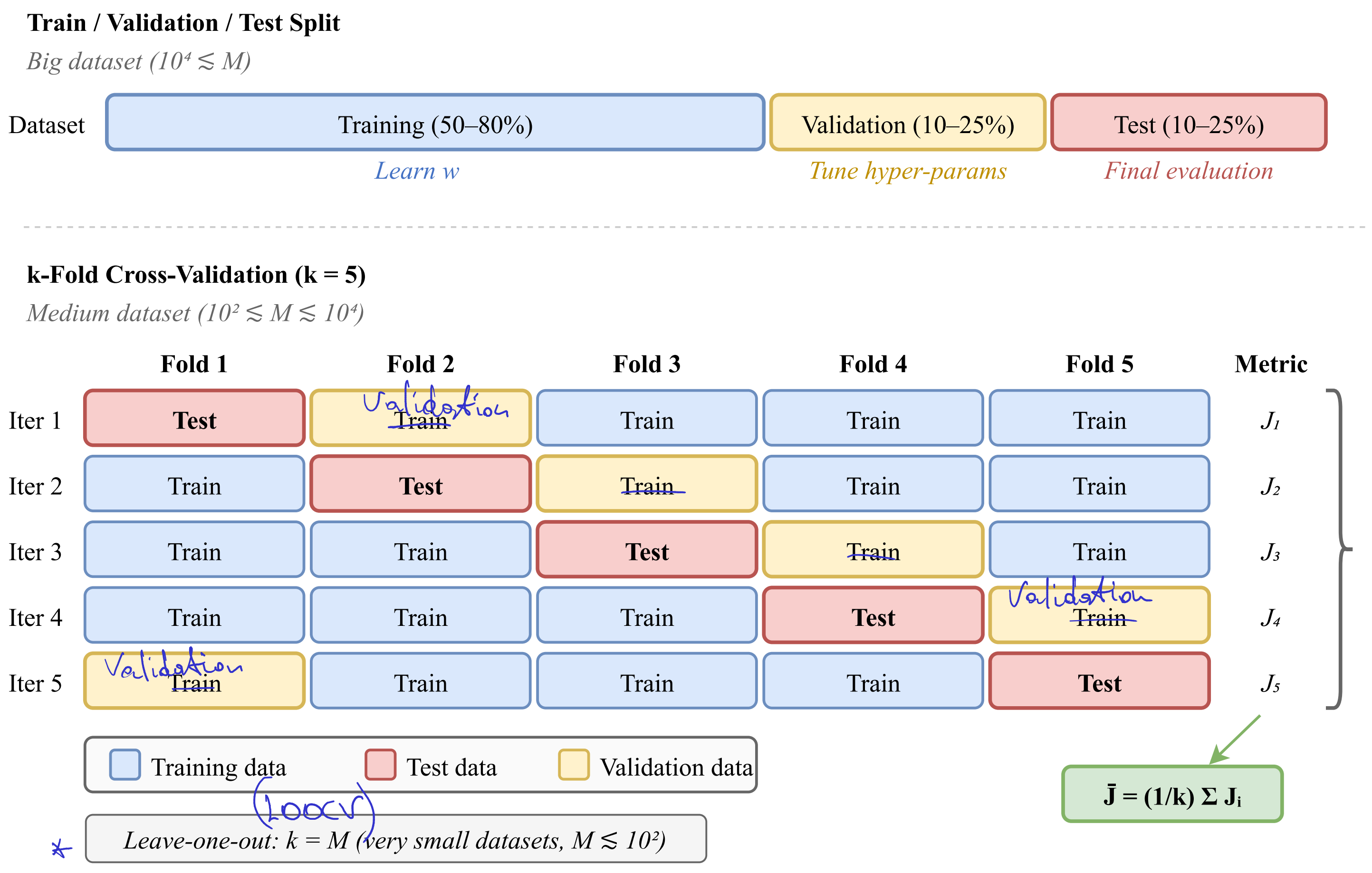


Figure 4.2.: Cross-validation illustration: train/validation/test split for big datasets (top) and nested k-fold cross-validation for medium datasets (bottom).

Medium datasets $(10^2 \lesssim M \lesssim 10^4)$: k-fold

First step is to resample the dataset into the **random order**. Then, apply the following steps:

- *Data Splitting*: First, the available dataset is divided into k subsets of approximately equal size. These subsets are often referred to as "folds".
- *Model Training and Evaluation*: The model is trained k times. In each iteration, one of the subsets is used as the test set, and the remaining $k - 1$ subsets are used as the training/validation sets. This means that in each iteration, the model is trained and validated on a different combination of training and test data.
- *Performance Evaluation*: After training the model k times, the performance of the model is evaluated by averaging the performance metrics obtained in each iteration.

Usually, k is defaulted to 5 or 10. When hyper-parameter tuning is required, the training partition in each iteration is further split into training and validation subsets (see nested cross-validation below).

Nested Cross-Validation

Goal: Simultaneously tune hyper-parameters and obtain an unbiased performance estimate.

The k-fold procedure described above uses a single loop. When hyper-parameters (e.g., polynomial degree L, regularization parameter λ) must be selected, a single CV loop conflates two objectives: hyper-parameter tuning and performance assessment. This leads to optimistic performance estimates because the test fold indirectly influences model selection.

Nested cross-validation separates these objectives into two loops (Fig. 4.2):

- *Outer loop* (k_{out} folds): holds out a test fold (red in Fig. 4.2) for unbiased performance evaluation. The remaining $k_{\text{out}} - 1$ folds form the outer training set.
- *Inner loop* (k_{in} folds): within each outer training set, a further k-fold split creates validation folds (yellow in Fig. 4.2) for hyper-parameter selection. For each candidate hyper-parameter value, train on the inner training folds (blue) and evaluate on the inner validation fold. Select the hyper-parameter value that maximizes the average inner validation metric.

After the inner loop selects the best hyper-parameter, retrain the model on the entire outer training set using that hyper-parameter, and evaluate on the outer test fold. The final performance estimate is the average over the k_{out} outer test metrics:

$$\bar{J} = \frac{1}{k_{\text{out}}} \sum_{i=1}^{k_{\text{out}}} J_i \tag{4.5}$$

The hyper-parameter selected may differ across outer folds. This is expected — each outer fold sees a slightly different training distribution. The purpose of nested CV is to produce an unbiased performance estimate, not a single set of hyper-parameters.

Very small datasets ($M \lesssim 10^2$): Leave-One-Out Cross-Validation (LOOCV) (*)

Uses k-fold with $k = M$, which means that each fold will contain only one data point.

LOOCV suffers from several drawbacks:

1. *High computational cost.* LOOCV requires training the model M times. For large datasets this is impractical; it is only feasible when M is small.

2. *High variance in performance estimates.* Because only one sample is left out, the M training sets are nearly identical to each other and have *low bias*. However, the resulting models are highly correlated, and the average of correlated estimates can have *higher variance* than 5-fold or 10-fold cross-validation.

3. *Sensitivity to outliers.* Each data point serves as the sole validation sample in exactly one fold. A single outlier or noisy point can strongly influence that fold's error, producing unreliable overall estimates compared to k-fold where outliers are diluted within larger folds.

4. *Potential for overfitting.* Because each training set contains $M-1$ points, the resulting models are very similar to the full-data model. In high-dimensional or noisy settings ($N \gg M$), LOOCV can favor overly complex models that fit the training data too closely.

Example 4.1: Consider $M = 80$ samples with $N = 2048$ features drawn entirely from i.i.d. $\mathcal{N}(0, 1)$ (pure noise) and binary labels with class sizes 51 vs. 29 (majority baseline ≈ 0.64). A pipeline of standardization $\rightarrow$ PCA(5) $\rightarrow$ RBF-SVM ($C = 0.05$) with LOOCV achieves 100% accuracy across 10 random seeds — despite the features containing no real signal. This illustrates how LOOCV combined with high dimensionality ($N \gg M$) can produce misleadingly optimistic performance estimates. The methods to identify of such a condition are discussed in Sec. 10.1.

LOOCV is generally recommended only for very small datasets where k-fold would leave too few samples per fold. For medium and large datasets, 5- or 10-fold cross-validation typically provides a better bias-variance trade-off in the performance estimate itself.

4.4. Overfitting and underfitting

השפעה של מספר פרמטרים של המודל

Goal: Understand underfitting and overfitting as two extremes of model complexity.

סעיף

Given a dataset split into *train* and *test* subsets (formalized in Sec. ?? ~~below~~), the model is fitted on the training data and evaluated on the test data. The relationship between model complexity and performance on these two subsets reveals two failure modes.

Overfitting is when the model is too complex, i.e. have too many parameters.

- Too many parameters relative to the number of observations. Theoretically, we expect at least an order of magnitude more observations than parameters.
- Follow the training data very closely.
- Fail to generalize well to unseen data.

Significant performance difference between train and ~~unseen~~ test data.

Underfitting happens when a model is too simple.

- Unable to capture the underlying pattern of the data and hence misses the trends in the data.
- Performs poorly on the training data and fail to generalize.

Poor performance on both train and test datasets.

Overfitting and underfitting are complementary and balancing between them is key to building robust machine learning models that perform well on new, unseen data, i.e. generalize well. This principle is illustrated in Fig. 4.3

The numerical example of underfitting and overfitting trade-off from the polynomial model (Sec. 4.1) is presented in Fig. 4.4.

4.5. Bias-Variance Trade-off

Goal: Inherent underfitting and overfitting trade-off.

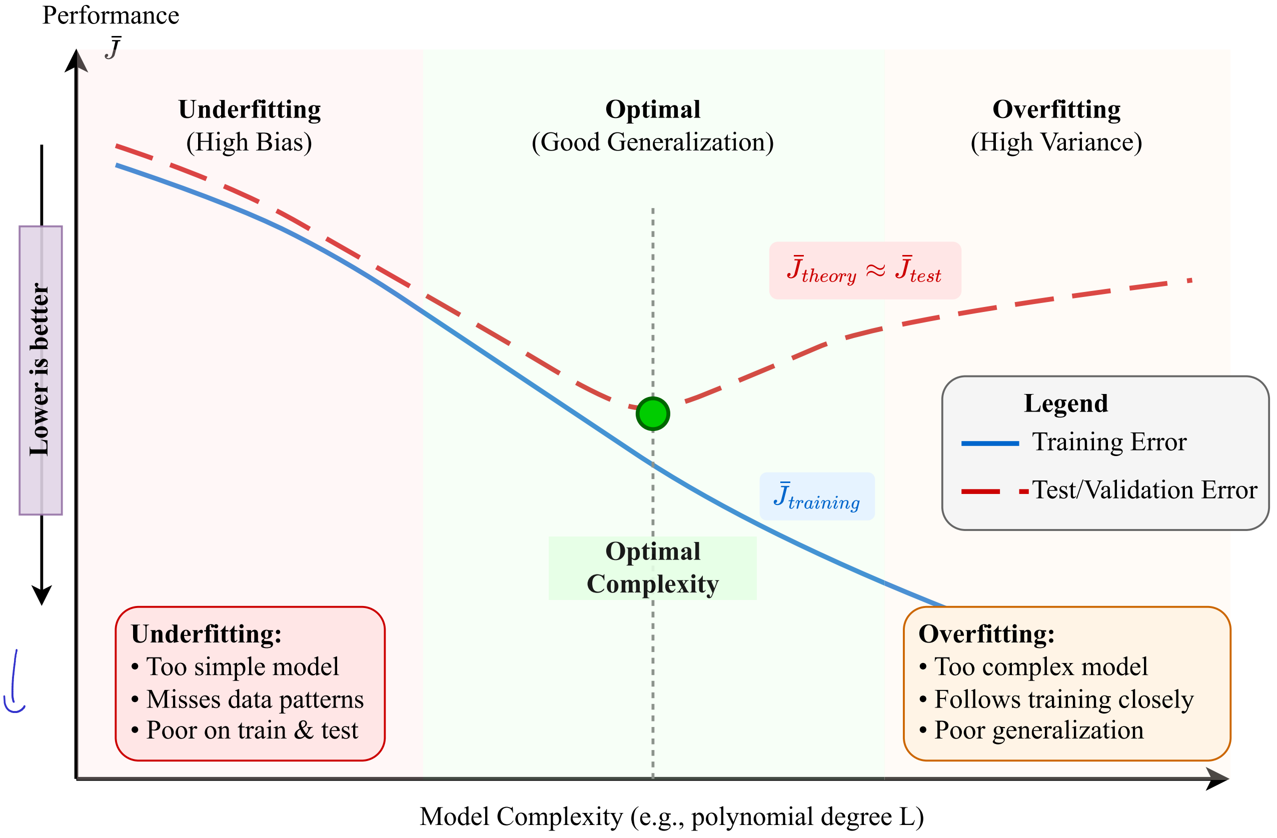
4.5.1. Noisy Deterministic Function Interpretation

The general model beneath the dataset model is

הנחת יסוד

$$\boxed{y_i = h(\mathbf{x}_i) + \epsilon_i,} \tag{4.6}$$

יעד של המודל: קירוב הכי טוב במידה האפשרי.



[illegible]

Figure 4.3.: Increasing model complexity improves the prediction error for training data, but from a certain point of transitioning from underfitting to overfitting, it increases the error for test data.

where $h(\mathbf{x})$ is some unknown function and ϵ is some random noise with unknown distribution, zero-mean and variance of σ^2 ($\mathbb{E}[\epsilon] = 0, \mathrm{Var}[\epsilon] = \sigma^2$).

A model predicts outputs via $\hat{y}_i = f(\mathbf{x}_i)$ with error $e_i = \hat{y}_i - y_i$.

Bias The bias of the model is the expected difference between predictions and the true function,

שגיאה ממוצעת

$$\mathrm{Bias}[\hat{y}] = \mathbb{E}[\hat{y}] - \mathbb{E}[y] = \mathbb{E}\left[f(\mathbf{x})\right] - h(\mathbf{x}) \tag{4.7}$$

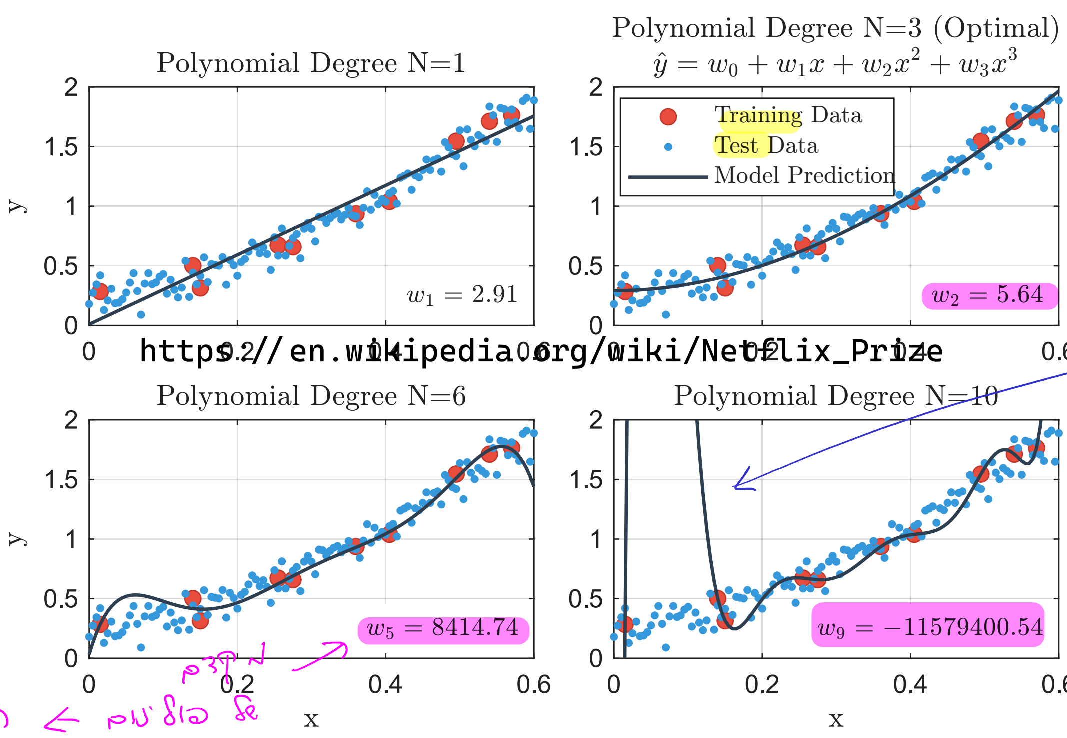
where the expectation is taken over all possible training samples.

For a specific dataset, the empirical bias can be estimated as:

$$\begin{aligned} \mathrm{Bias}_{\mathrm{empirical}} = \bar{\mathbf{e}} &= \frac{1}{M}\sum_{i=1}^{M} e_i \\ &= \frac{1}{M}\sum_{i=1}^{M} \hat{y}_i - \frac{1}{M}\sum_{i=1}^{M} h(\mathbf{x}_i) \qquad (\bar{\epsilon} = 0) \end{aligned} \tag{4.8}$$

Variance The variance of the model measures the variability of predictions across different training sets,

$$\mathrm{Var}[\hat{y}] = \mathbb{E}\left[(\hat{y} - \mathbb{E}[\hat{y}])^2\right] \tag{4.9}$$



https://en.wikipedia.org/wiki/Netflix_Prize

היקף
עובר
דרך
מני
train
באופן
מושלם

גרוע test
בימר

Figure 4.4.: Overfitting and underfitting polynomial example.

The bias and variance are illustrated in Fig. 4.5.

4.5.2. Bias-Variance Decomposition (*)

Probabilistic derivation The expected prediction error (MSE), taken over all possible training sets and noise realizations, can be decomposed as follows. Starting from:

$$\text{MSE} = \mathbb{E}\left[(\hat{y} - y)^2\right] = \mathbb{E}\left[\hat{y}^2\right] - 2\mathbb{E}[y\hat{y}] + \mathbb{E}\left[y^2\right] \tag{4.10}$$

We evaluate each term separately. For the first term, by the variance identity:

$$\mathbb{E}\left[\hat{y}^2\right] = \text{Var}[\hat{y}] + \mathbb{E}^2[\hat{y}] \tag{4.11}$$

For the cross-term, since $y = h(\mathbf{x}) + \epsilon$ and ϵ is independent of $\hat{y}$:

$$\mathbb{E}[y\hat{y}] = \mathbb{E}\left[(h(\mathbf{x}) + \epsilon)\hat{y}\right] = h(\mathbf{x})\mathbb{E}[\hat{y}] + \cancelto{0}{\mathbb{E}[\epsilon]}\mathbb{E}[\hat{y}] = h(\mathbf{x})\mathbb{E}[\hat{y}] \tag{4.12}$$

For the last term:

$$\mathbb{E}\left[y^2\right] = \mathbb{E}\left[(h(\mathbf{x}) + \epsilon)^2\right] = h^2(\mathbf{x}) + 2h(\mathbf{x})\cancelto{0}{\mathbb{E}[\epsilon]} + \mathbb{E}\left[\epsilon^2\right] = h^2(\mathbf{x}) + \sigma^2 \tag{4.13}$$

Combining all three terms:

$$\begin{aligned} \mathbb{E}\left[(\hat{y} - y)^2\right] &= \text{Var}[\hat{y}] + \mathbb{E}^2[\hat{y}] - 2h(\mathbf{x})\mathbb{E}[\hat{y}] + h^2(\mathbf{x}) + \sigma^2 \\ &= \underbrace{\left(\mathbb{E}[\hat{y}] - h(\mathbf{x})\right)^2}_{\text{bias}^2} + \underbrace{\text{Var}[\hat{y}]}_{\text{variance}} + \underbrace{\sigma^2}_{\text{noise}} \end{aligned} \tag{4.14}$$

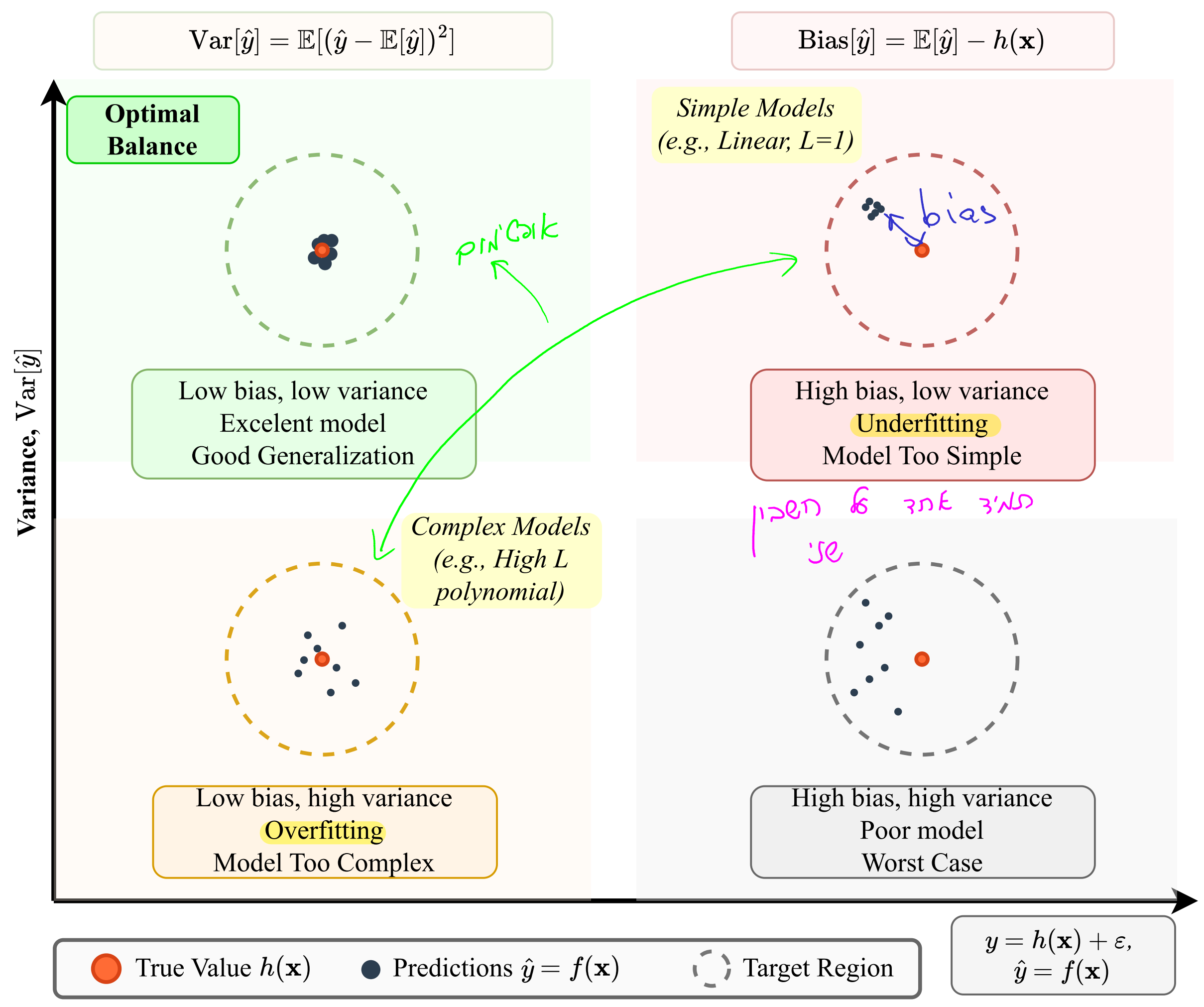

$\mathrm{Var}[\hat{y}] = \mathbb{E}[(\hat{y} - \mathbb{E}[\hat{y}])^2]$
$\mathrm{Bias}[\hat{y}] = \mathbb{E}[\hat{y}] - h(\mathbf{x})$
Optimal Balance
Simple Models (e.g., Linear, L=1)
bias
אופטימום
Low bias, low variance
Excelent model
Good Generalization
High bias, low variance
Underfitting
Model Too Simple
תמיד אחד על חשבון שני
Complex Models (e.g., High L polynomial)
Variance, $\mathrm{Var}[\hat{y}]$
Low bias, high variance
Overfitting
Model Too Complex
High bias, high variance
Poor model
Worst Case
True Value $h(\mathbf{x})$
Predictions $\hat{y} = f(\mathbf{x})$
Target Region
$y = h(\mathbf{x}) + \varepsilon$, $\hat{y} = f(\mathbf{x})$

Figure 4.5.: Illustration of bias and variance.

Empirical estimate For a specific dataset of M samples, the MSE can be similarly decomposed:

$$\mathrm{MSE} = \frac{1}{M}\sum_{i=1}^{M}(\hat{y}_i - y_i)^2 = \underbrace{\left(\bar{\hat{y}} - \bar{h}\right)^2}_{\text{bias}^2_{\text{emp}}} + \underbrace{\frac{1}{M}\sum_{i=1}^{M}(\hat{y}_i - \bar{\hat{y}})^2}_{\text{variance}_{\text{emp}}} + \underbrace{\frac{1}{M}\sum_{i=1}^{M}\epsilon_i^2}_{\text{noise}_{\text{emp}}} \tag{4.15}$$

where $\bar{\hat{y}} = \frac{1}{M}\sum_{i=1}^{M}\hat{y}_i$ and $\bar{h} = \frac{1}{M}\sum_{i=1}^{M}h(\mathbf{x}_i)$.

This decomposition shows that the total prediction error consists of three components: squared bias, variance, and irreducible noise. The inherent bias-variance trade-off is presented in Fig. 4.6. Underfitting is low variance and high bias, and overfitting is high variance and low bias. The best model performance has inherent bias-variance trade-off. However, some less appropriate models can have high bias and high variance simultaneously.

4.6. Normalization and Standardization

Goal: Data pre-processing to reduce possible overfitting.

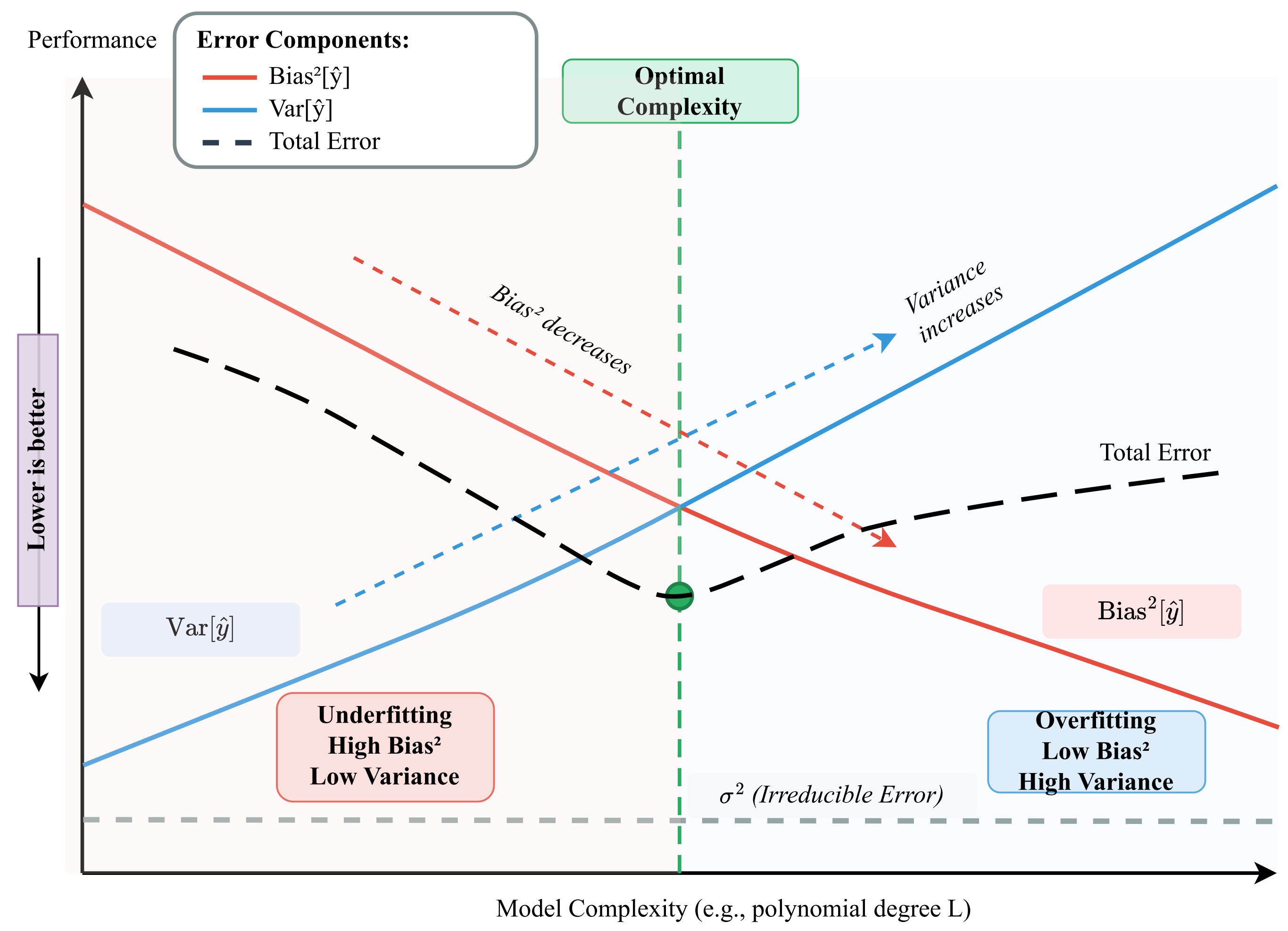


Figure 4.6.: Illustration of bias-variance trade-off.

Values in different columns in $\mathbf{X}$ (feature columns $\tilde{\mathbf{x}}_j$) may be different by orders of magnitudes, i.e. $\|\tilde{\mathbf{x}}_i\| \gg \|\tilde{\mathbf{x}}_j\|$. This results in:

- Some columns have significantly higher influence on $\hat{\mathbf{y}}$.
- Numerical instabilities.

בעיה : שוני (קיצוני) בין עמודות שונות
של X

Standardization (z-score normalization)

Mapping all the input values such that they follow a distribution with zero mean and unit variance.

$$z_{std} = \frac{\mathbf{x} - \bar{\mathbf{x}}}{\sigma_{\mathbf{x}}}, \qquad (4.16)$$

$\leftarrow$ ממוצע 0
שונות 1

where

$$\bar{\mathbf{x}} = \frac{1}{M}\sum_{j=1}^{M} x_j$$

$$\sigma_{\mathbf{x}}^2 = \frac{1}{M}\sum_{j=1}^{M} (x_j - \bar{\mathbf{x}})^2$$

Implementation steps:

1. On **train** dataset, evaluate $\bar{\mathbf{x}}$ and $\sigma_{\mathbf{x}}$.
2. Apply normalization on **train** dataset, using $\bar{\mathbf{x}}$ and $\sigma_{\mathbf{x}}$.
3. Apply normalization on **test** dataset, using **same** $\bar{\mathbf{x}}$ and $\sigma_{\mathbf{x}}$ (no recalculation).

When normalization is applied to $\mathbf{y}$, the output of the model is transformed back, $\hat{\mathbf{y}} = \hat{\mathbf{y}}_{std}\sigma_{\mathbf{y}} + \bar{\mathbf{y}}$.

Example: `zscore` command in Matlab and `sklearn.preprocessing.StandardScaler` in Python.

Normalization

2) תמונה בין 0,1 או לפעמים [-1,1]

Mapping all values of a feature to be in the range $[0, 1]$ by the transformation[2]

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{4.17}$$

Implementation steps for normalization are similar to standardization.

Beware, normalization and standardization are used interchangeably.

Pros:

- Improves convergence speed of gradient-based optimization.
- Prevents features with large magnitudes from dominating distance-based models.
- Standardization of both $\mathbf{X}$ and $\mathbf{y}$ results in $w_0 = 0$ and removes the requirement for $\mathbf{1}$ column in $\mathbf{X}$.

Cons:

- Min-max normalization is sensitive to outliers.
- Scaling parameters must be computed on the training set and applied consistently to the test set.

4.7. Dataset Size

Goal: Dataset size influence on underfitting-overfitting trade-off. Understanding when more data helps and when it does not.

By the law of large numbers, the generalization gap $\bar{J}_M - \bar{J}_{theory}$ shrinks as the number of samples M grows. However, the benefit of additional data depends on whether the model is overfitting or underfitting.

Overfitting (complex models) More data reduces variance. The training error rises (the model can no longer memorize every sample) while the test error drops — both converge toward the theoretical performance $\bar{J}_{theory}$. This is illustrated in Fig. 4.7.

[2] In Python: `sklearn.preprocessing.MinMaxScaler`

Underfitting (simple models) More data does *not* help since the model lacks the capacity to capture the underlying pattern regardless of M. Both training and test errors remain high and close to each other.

Therefore, increasing the dataset size is a **data-related** method (see Sec. 4.2) that primarily reduces *variance*, not *bias*.

Beyond a certain dataset size, performance improvement may be almost negligible or non-existent.

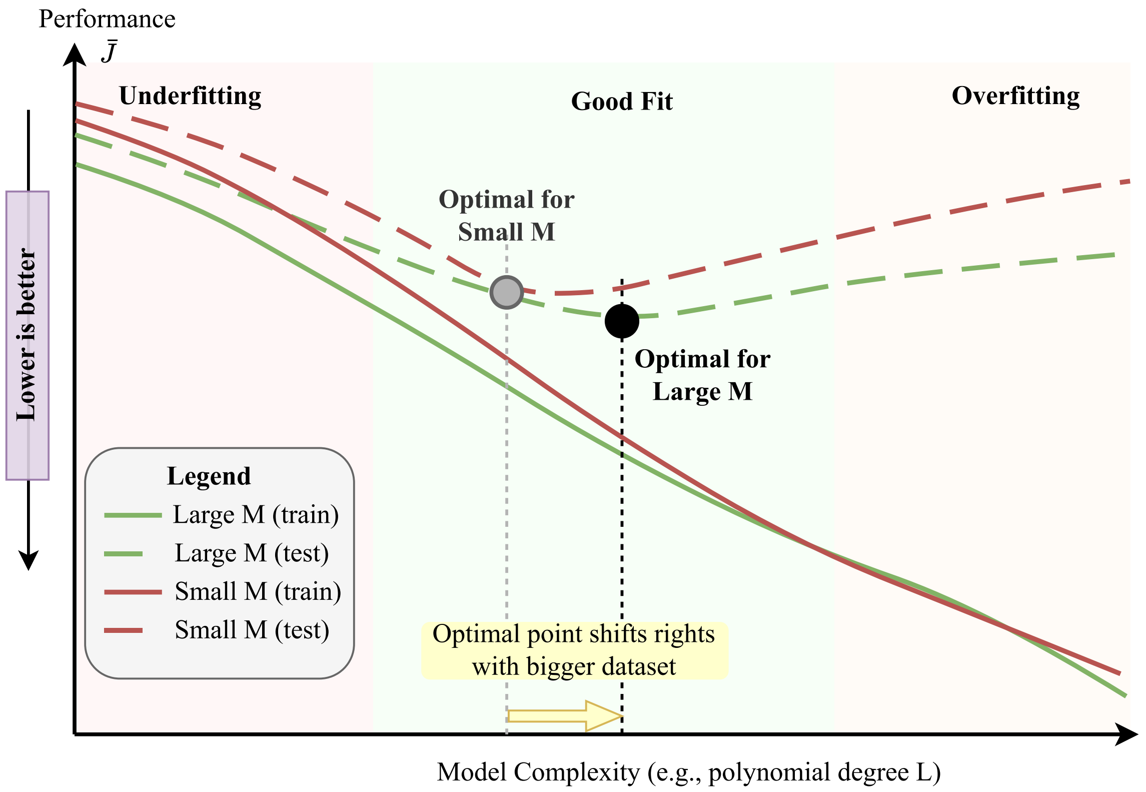


Figure 4.7.: Increase in dataset size improves the over-complex model performance.

4.8. Regularization

Overfit רוב המודלים:
inherent מובנה

Goal: Loss function tweak that influences on underfitting-overfitting trade-off.

Regularization: Penalty to the loss function

$$\mathcal{L}_{\text{new}} = \mathcal{L} + \lambda g(\mathbf{w}) \tag{4.18}$$

where λ is termed *regularization parameter*.

L_2**-regularization**: Special case of

$$g(\mathbf{w}) = \frac{1}{2M}\|\mathbf{w}\|^2 = \frac{1}{2M}\sum_{i=1}^{N} w_i^2 \tag{4.19}$$

where vector of weights $\mathbf{w}$ does not include w_0 weight. Moreover, when normalization is used, no w_0 weight is needed. Two special cases are:

- $\lambda \to 0$ gets the original loss function (overfitting).
- $\lambda \to \infty$ makes loss function independent on $\mathbf{w}$ (underfitting).

Polynomial regression example The resulting coefficients w_i will be significantly higher for higher i (Fig. 4.4). Reducing them results in smooth $\hat{y}$ prediction. The illustration of λ influence is presented in Fig. 4.8.

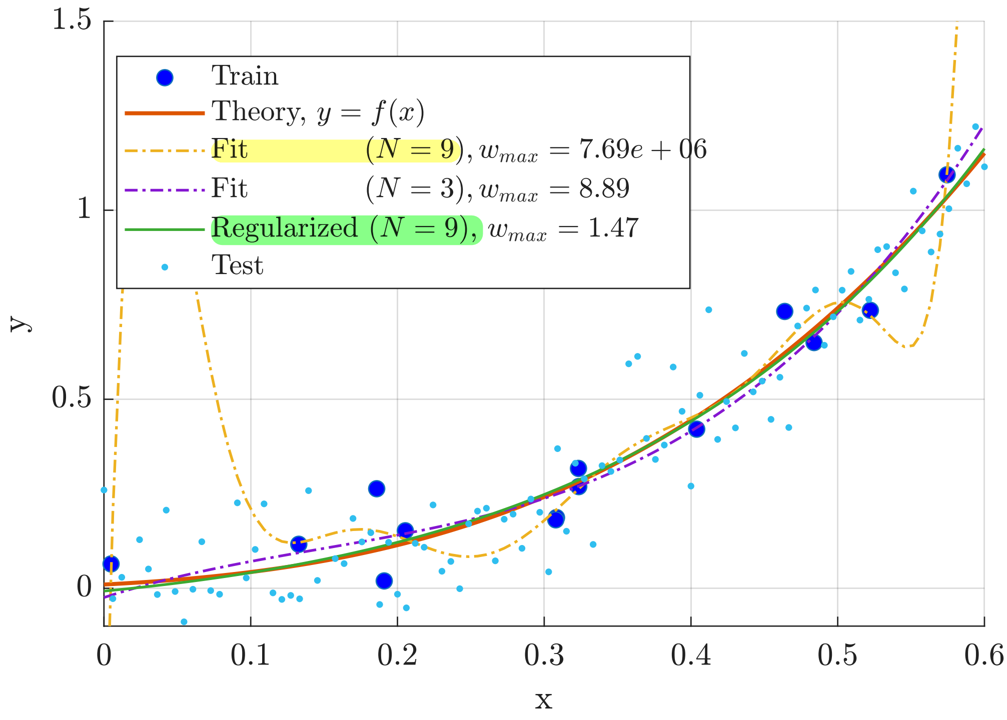


Figure 4.8.: Illustration of regularization influence on the polynomial model.

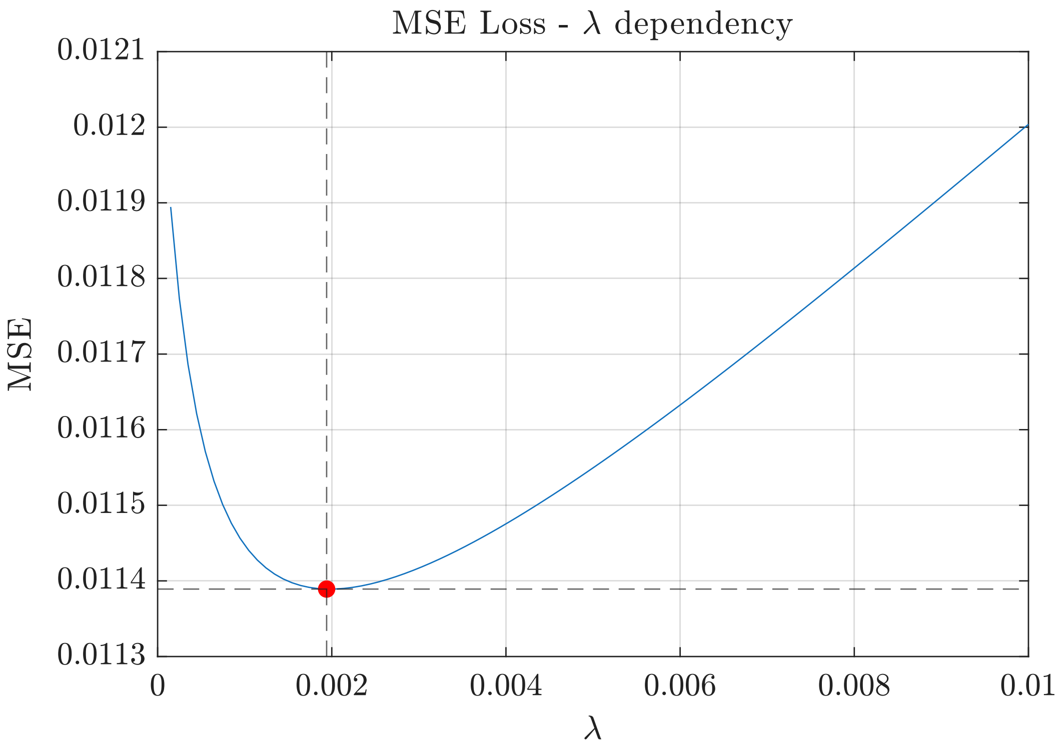


Figure 4.9.: Dependence of MSE on λ for regularization in Fig. 4.8 (polynomial regression). Overfitting is on the left ($\lambda \to 0$) and undefitting on the right ($\lambda \to \infty$).

The corresponding underfitting-overfitting λ-related trade-off is presented in Fig. 4.9.

4.8.1. Ridge Regression

Ridge regression: L_2-regularized linear regression.

$$\begin{aligned}\mathcal{L}(\mathbf{w}) &= \frac{1}{2M}\left\|\mathbf{y}-\mathbf{X}\mathbf{w}\right\|^2 + \frac{\lambda}{2M}\underbrace{\left\|\mathbf{w}\right\|^2}_{\sum_{i=1}^N w_i^2} \\ &= \frac{1}{2M}\left(\mathbf{y}-\mathbf{X}\mathbf{w}\right)^T\left(\mathbf{y}-\mathbf{X}\mathbf{w}\right) + \frac{\lambda}{2M}\mathbf{w}^T\mathbf{w}\end{aligned} \tag{4.20}$$

Derivative

$$\nabla\mathcal{L}(\mathbf{w}) = \frac{1}{M}\left(-\mathbf{X}^T\left(\mathbf{y}-\mathbf{X}\mathbf{w}\right)+\lambda\mathbf{w}\right) = 0 \tag{4.21}$$

Solution

$$\begin{aligned}&-\mathbf{X}^T\left(\mathbf{y}-\mathbf{X}\mathbf{w}\right)+\lambda\mathbf{w} = -\mathbf{X}^T\mathbf{y}+\mathbf{X}^T\mathbf{X}\mathbf{w}+\lambda\mathbf{w} = 0\\ &\mathbf{X}^T\mathbf{y} = \mathbf{X}^T\mathbf{X}\mathbf{w}+\lambda\mathbf{w}\end{aligned}$$

Finally, the regularized weights are given by

$$\mathbf{w} = \left(\lambda\mathbf{I}+\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y} \tag{4.22}$$

Can be viewed as special case of linear regression, of the form

$$\begin{gathered}\tilde{y} = \tilde{\mathbf{X}}\mathbf{w}\\ \begin{bmatrix}\mathbf{y}\\0\\\vdots\\0\end{bmatrix} = \begin{bmatrix}- & \mathbf{X} & -\\ \sqrt{\lambda} & & 0\\ & \ddots & \\ 0 & & \sqrt{\lambda}\end{bmatrix}\begin{bmatrix}w_1\\\vdots\\w_N\end{bmatrix} + w_0\end{gathered} \tag{4.23}$$

Slope interpretation Higher value of λ reduces (mostly) the highest slopes (weights w_i) $\Rightarrow$ the dependency on the parameters with these weights is reduced.

Eigenvalues interpretation This calculation limits the smallest eigenvalues to $> \frac{1}{\lambda}$ and thus improve the numerical stability.

Bias-Variance Trade-off The bias-variance trade-off for ridge regression is illustrated in Fig. 4.10.

GD Solution

Tailored GD (Sec. 3.4) solution is

$$\begin{aligned}\mathbf{w}_{n+1} &= \mathbf{w}_n - \alpha\nabla\mathcal{L}(\mathbf{w})\\ &= \mathbf{w}_n - \frac{\alpha}{M}\left[\mathbf{X}^T\left(\mathbf{X}\mathbf{w}_n-\mathbf{y}\right)+\lambda\mathbf{w}_n\right]\\ &= \mathbf{w}_n\left(1-\frac{\alpha}{M}\lambda\right) - \frac{\alpha}{M}\mathbf{X}^T\left(\mathbf{X}\mathbf{w}_n-\mathbf{y}\right)\end{aligned} \tag{4.24}$$

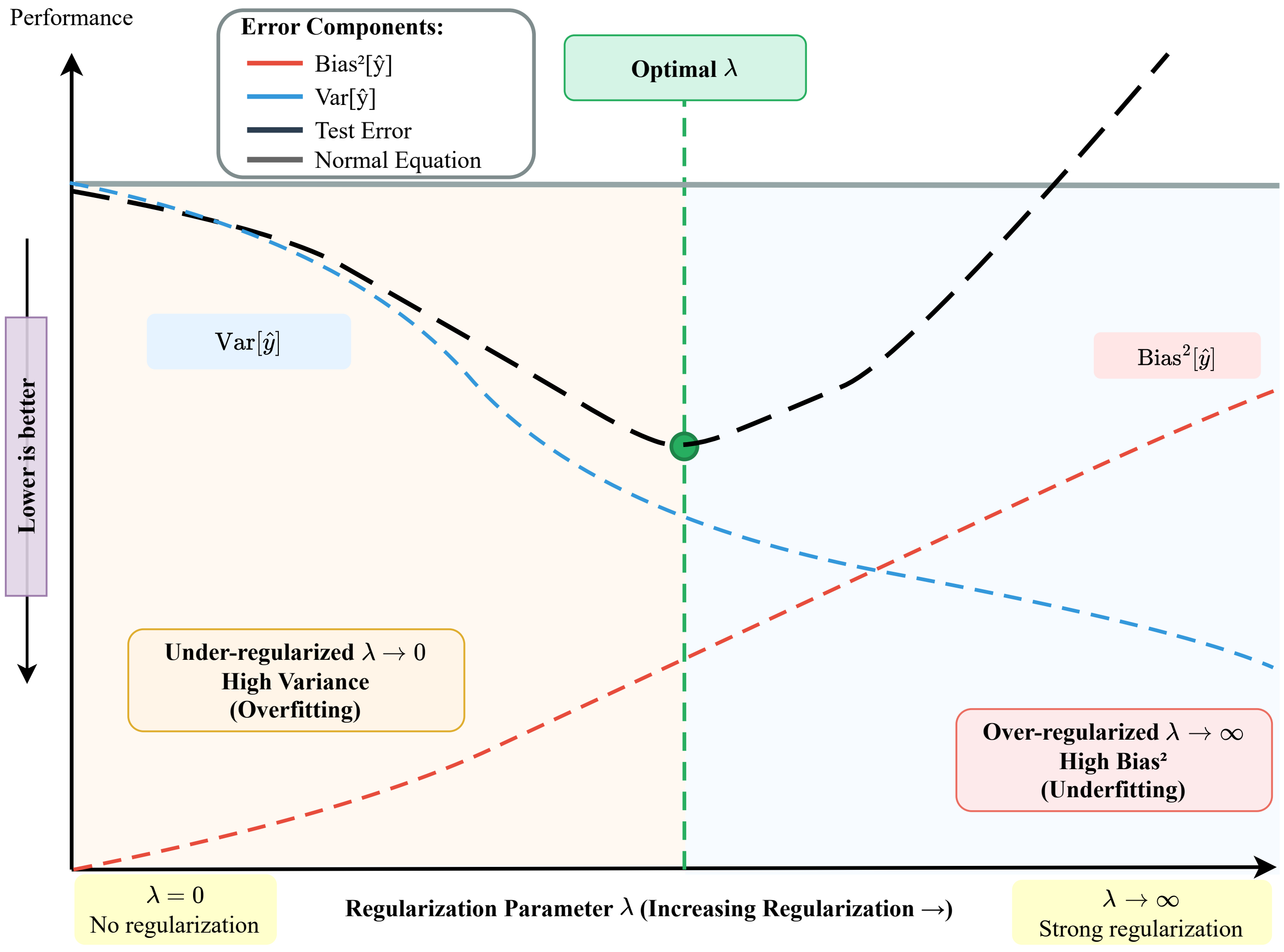


Figure 4.10.: Regularization trade-off for ridge regression: Test error $\bar{J}_{\text{test}}$ is minimized at optimal λ. Note, the trends are opposite to Fig. 4.6.

The α value is typically chosen such that

$$0.99 \gtrsim \left(1 - \frac{\alpha}{M}\lambda\right) \gtrsim 0.95 \tag{4.25}$$

4.9. Summary

Table 4.1 summarizes the methods discussed in this chapter and their influence on the generalization gap. Each method primarily addresses either data-related or model-related sources of the gap (Sec. 4.4).

- **Cross-validation** mitigates data-related issues by evaluating model performance across different data splits, and guides model-related decisions such as hyper-parameter selection.
- **Normalization and standardization** are data pre-processing steps that reduce the influence of feature scale imbalance and improve numerical stability.
- **Increasing dataset size** directly reduces the data-related gap by making $\bar{J}_M$ closer to $\bar{J}_{theory}$, and reduce overfitting in complex models.
- **Regularization** constrains model complexity via the penalty parameter λ, controlling the bias-variance trade-off from the model side.

Table 4.1.: Chapter methods and their generalization influence.

Method	**Data-related**	**Model-related**
Cross-validation	✓	✓
Normalization / Standardization	✓	
Dataset size increase	✓	
Regularization (λ)		✓
Hyper-parameter tuning (L, . . .)		✓

- **Hyper-parameter tuning** (e.g., polynomial degree L, regularization λ) adjusts model complexity to balance underfitting and overfitting.